Fast and Accurate Triangle Counting in Graph Streams Using Predictions: Extended Version

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 a sequence of graph streams is 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*—[change]component, formatting, style, styling, insert**

I. Introduction

Counting the number of triangles is a fundamental primitive in the analysis of graphs, with applications ranging from community detection [4] and anomaly detection [3], [19] to molecular biology [21]. 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. [35] 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. Recently Shin et al. [30], [32] introduced the notion of *waiting room sampling*, that allows to take advantage of the temporal locality of triangles observed in most applications by keeping the most recent edges in the stream in a waiting room. While such algorithms provide 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 [23].

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 [add link]. Our algorithms use reservoir sampling 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 [7], 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 runs in time comparable to 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 [12], [20], [37], [38], in many different settings [24]–[27], [36]. 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 survey by Latapy [16] [update with more recent survey?][which one?] 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. [6] presented a suite of sampling algorithms, resulting in $(1+\epsilon)$-approximation of the global triangles. Pavan et al. [28] 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. [11] 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. [35] 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 [8], to maintain a bounded-size uniform random sample of the edges in a fully dynamic stream. Shin et al. [30], [32] 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. [31] presented *ThinkD*$_{acc}$, for handling fully dynamic graph streams with random pairing providing, that provides highly accurate estimates. [Is there any difference in approach between Triest and ThinkD? It is not clear from the previous sentences.] Singh et al. [33] combined some of the ideas above to design an hybrid (i.e., partially fixed memory and partially fixed probability) approach. None of these works incorporates predictors, which is instead the key feature of our algorithm Tonic, allowing it to focus on the most relevant edges and to adapt to different data distributions.

The use of predictions about the input has been recently formalized in the *algorithms with predictions* framework (see the survey by Mitzenmacher and Vassilvitskii [23]), and several algorithms with predictions for well-studied problems have been proposed [10], [13], [22]. The problem of counting global triangles in a *insertion-only* graph stream has been recently studied in such framework by Chen et al. [7]. 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 [7] 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 [7] 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 [add link].

For $t = 1, ..., m$, we denote with $G^{(t)} = (V, E^{(t)})$, where $E^{(t)} = \{e^{(1)}, \ldots, e^{(t)}\}$, 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 $t = 1, 2, \ldots$, 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. [30], [32], 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. [30], [32] 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 most heavy edges observed up to time t.

The intuition for using such predictions is that in most cases triangles are not distributed equally across edges, and 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 triangle counts.

C. Reservoir Sampling

The last component of our algorithm Tonic is the sampling of edges through *reservoir sampling* [39]. 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 not in the waiting room $\mathcal{W}$ or 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 [39] 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.

[Here.]

IV. Tonic: Counting Triangles with Predictions

We now describe our algorithm Tonic for counting triangles in graph streams with predictions. 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 the extended version [ADD REF]. 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 not 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 $\{v, w\}$ 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 7). Otherwise, if $\mathcal{W}$ is full, it replaces the oldest edge $\{x, y\}$ in $\mathcal{W}$ with $\{u, v\}$ (lines 9-10), and, if $\mathcal{H}$ is not full, it inserts $\{x, y\}$ in $\mathcal{H}$ (lines 11-12). Otherwise, let e' be the *lightest* edge in $\mathcal{H}$, i.e., $e' = \arg\min_{e \in \mathcal{H}} O_H(e)$ (lines 14-15). Let $e_{\max}$ be the heaviest edge between $\{x, y\}$ and e', and let $e_{\min}$ be the lightest 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 16-19). `Tonic` reports the final estimates $\hat{T}$ and $\hat{T}_u$ at the end of the stream (line 38).

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 α; fraction of heavy edges space β; 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* Σ **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 $\{x, y\} \leftarrow$ 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 $\{u', v'\} \leftarrow$ 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** $\{u', v'\} \leftarrow \{x, y\}$; ;
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 $u \in \mathcal{S}$;

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 edge is sampled by the algorithm and uses such probability to update the relevant counts (see Section XX in the extended version in Appendix 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.

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 [30], [32]. For lack of space, all proofs are in the extended version [add ref].

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 ℓ

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 $p_{uvw} = \min\left(1,\ \frac{k(1-\alpha)(1-\beta)}{\ell} \times \frac{k(1-\alpha)(1-\beta)-1}{\ell-1}\right)$;
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 ℓ in the stream

1 $p_{sampling} = \frac{k(1-\alpha)(1-\beta)}{\ell}$;
2 **if** *FlipBiasedCoin*$(p_{sampling})$ == *HEAD* **then**
3 $\{\bar{u}, \bar{v}\} \leftarrow$ edge sampled uniformly at random from $\mathcal{S}_L$;
4 $\mathcal{S}_L \longleftarrow \mathcal{S}_L \setminus \{\{\bar{u}, \bar{v}\}\} \cup \{\{u, v\}\}$;

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:*

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

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

1) Time Complexity: In this subsection, we are going to analyze the time and space complexity of `Tonic`. In the analysis we also account for how the sets of edges stored by `Tonic` is 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 the common neighbors for nodes u and v is much smaller than k, i.e., our memory budget.

2) Comparison with WRS for global triangles count: We now prove that our algorithm leads to better estimates than WRS [30], [32] 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 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 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 $T^L = \sum_{l\in\mathcal{S}_L} \Delta(l)$ 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* $\mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] = \mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)]$.

Even more strikingly, our experimental evaluation shows that our algorithm Tonic provides estimates of quality similar to WRS even when an *adversarial* predictor is used (see Section V). [If we do not say much about adversarial predictor in the Experiments, refer to extended version of the paper instead.]

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. Similarly, 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 defined a simple predictor `MinDegreePredictor` that predicts, as measure of heaviness for $\{u, v\}$, the minimum between the degree $\delta(u)$ of u and the degree $\delta(v)$ of v, that is $O_H = \min\{\delta(u), \delta(v)\}$. Note that the degree $\delta(u)$ 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 involving fully dynamic streams are in the Extended Version of the paper [2]. The goals of our experiments are: i) to assess the dependency of `Tonic`'s performance 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 the estimates from `Tonic` during the evolution of an input graph stream.

Experimental Setup. We implemented `Tonic` in C++17. The code and the scripts to reproduce all experiments are available anonymously at https://anonymous.4open.science/r/Tonic-F2C4/. All the code was compiled under `gcc` 9.4.0 and ran on a machine with 2.20 GHz Intel Xeon CPU, 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 reported 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, defined as $|\hat{T} - T|/T$.

Datasets. We considered both single graph and sequence of graph streams as our datasets of interest, representing social and citation networks, and autonomous system (AS) relationships, which have been downloaded from [14], [18], [29]. Recall that, from each dataset, we removed self-loops and multiple edges, deriving a stream of insertion-only edges, for consistency with previous works.

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 [7];
- *SOC Youtube Growth* includes a list of all of the user-to-user links in Youtube video-sharing social network;
- *Cit US Patents* [9] represents the citation graph between US patents, where each edge $\{u, v\}$ indicates that patent u cited patent v (used also in [32]);
- *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;
- *SNAP 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* [5], [15] comprises 4 single graphs of the Twitter following/followers network. In all our experiments, we are going to consider each of the 4 single networks independently, and the larger network obtained by merging the 4 single networks (referred as *Twitter-merged*) used in [35].

Datasets' statistics are summarized in Table I. 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.

TABLE I
Datasets statistics

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)	$29.9M$	$373M$	$4.4B$

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

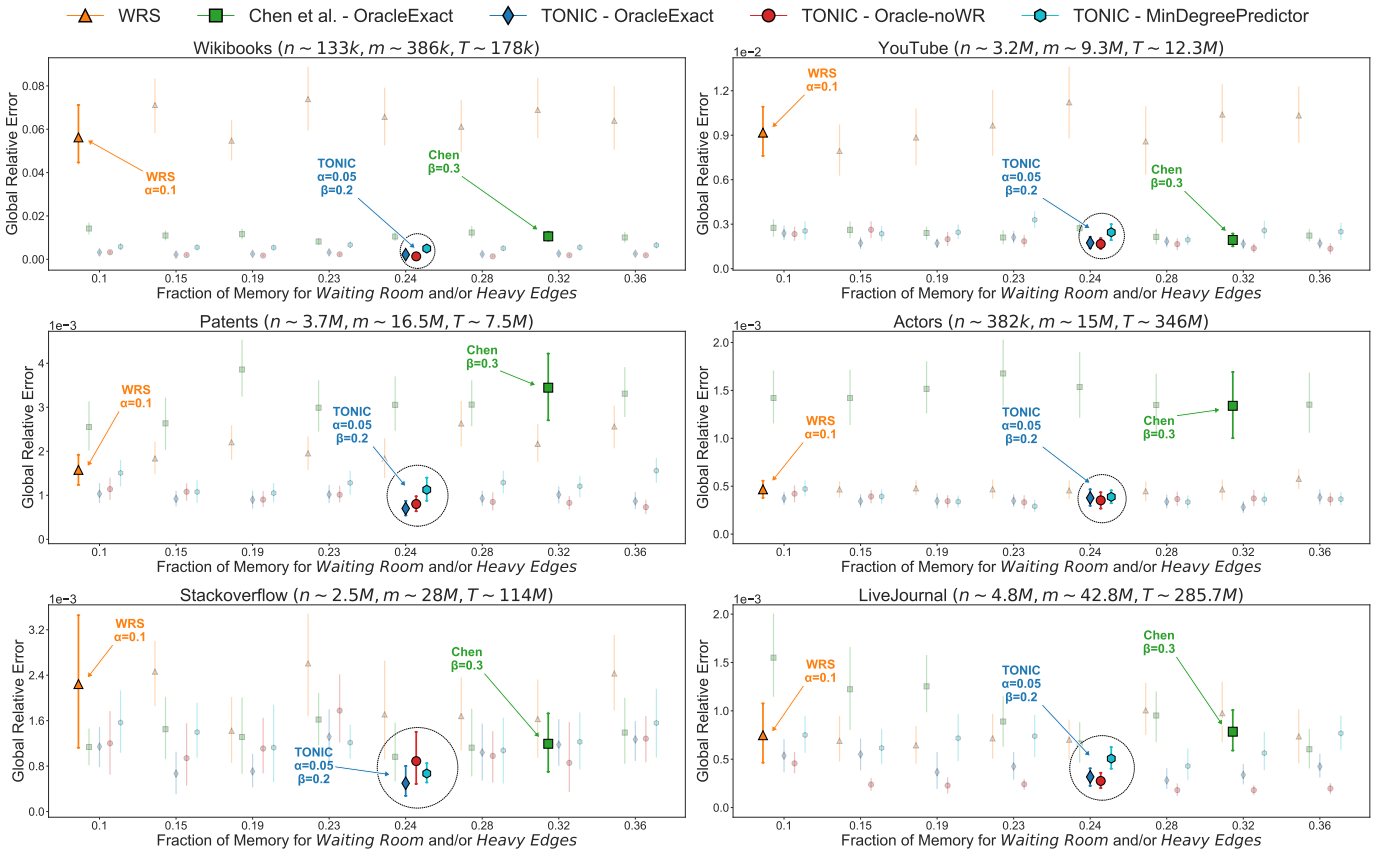

Fig. 1. Error vs fraction of memory budget used for waiting room and/or heavy edges. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and 95% confidence interval over 50 repetitions are shown. The chosen configuration for `Tonic` and the configurations suggested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

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 each predictor, we only retain 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 $O_H(e) = 0$. Furthermore, for `MinDegreePredictor`, starting from the edge representation which contains $m/10$ entries of type: $((u, v); min(deg_u, deg_v))$, we computed the number of unique nodes $\bar{n}$ in such predictor, and we retain the $\bar{n}$ highest degree nodes from the graph, obtaining the node-based representation of the predictor, in which each entry is of type $(u; deg_u)$. Then, when we query a node-based `MinDegreePredictor` 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 $\bar{n}$ distinct nodes in the edge-based predictor do not necessarily correspond to the same nodes taken from the $\bar{n}$ highest-degree nodes in the graph: for example, think about an high-degree node that has all low-degree neighbors,that is, it would not appear inside the unique nodes of edge-based predictor, but it is inside the highest-degree nodes of the graph (i.e., it has an entry in the node-based predictor). However, we empirically noticed that this is rarely the case. Node-based oracles are in someway encoding edge features in a much more succinct representation. Hence, the resulting overhead is significantly lower than the one used in edge-based representation. Further details can be found on the Extended Version [2].

For sequences $\Sigma_1, \Sigma_2, \ldots$ 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

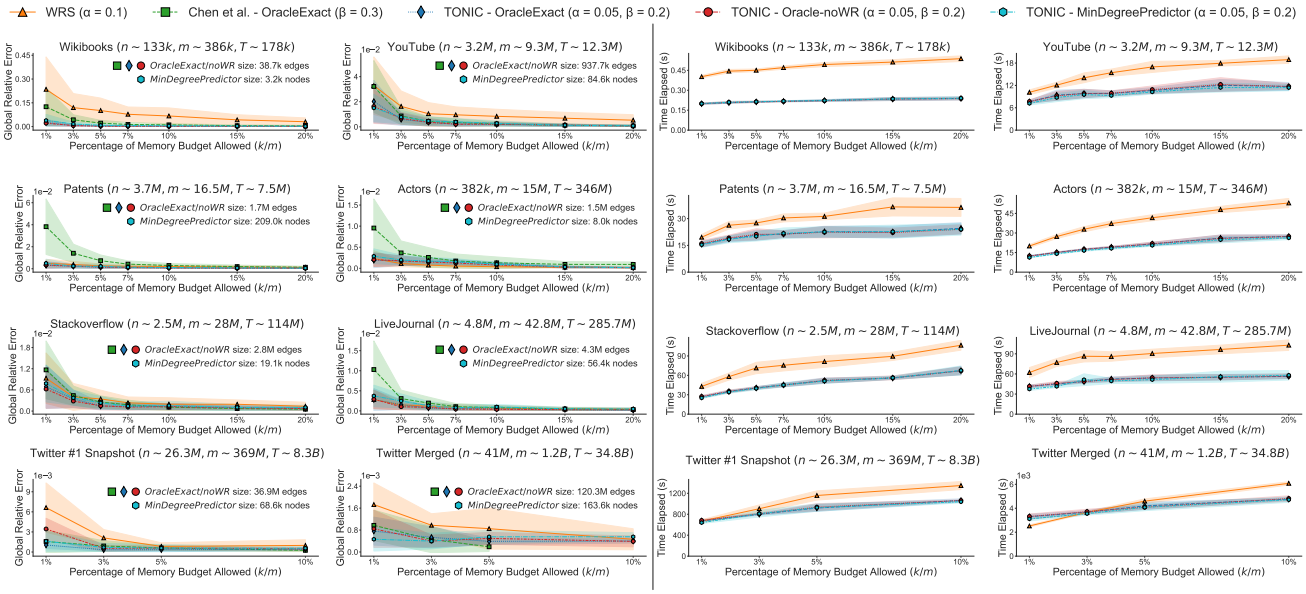

Fig. 2. Error (left) and runtime (right) vs memory budget. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` and `Chen` they are fixed as in the respective publications; for `Tonic` they are as chosen in Fig. 1).

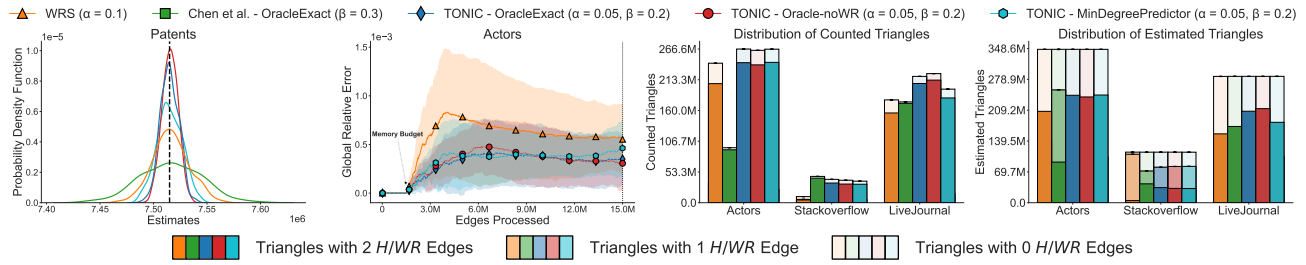

Fig. 3. (Left) Distribution of estimates for Patents dataset. (Center-left) Estimation error as time progresses on Actors dataset. (Center-right) Number and type of triangles counted by each algorithm on three datasets. (Right) Fraction of each type of triangles counted by each algorithm on three datasets.

streams, edges e adjacent to vertices not in Σ_1 always have $O_H(e) = 0$. Here, notice that the node-based representation of `MinDegreePredictor` of Σ_1 allows for all the possible combination of edges including the nodes in the predictor for all the graphs besides the training one, and therefore is not limited to edges which are present only in Σ_1. 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 `WRS` [32] (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 [7] (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 $\texttt{WRS}_{\texttt{del}}$ [32] (considering the version for FD streams) and with $\texttt{ThinkD}_{\texttt{acc}}$ [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 $\texttt{WRS}_{\texttt{del}}$ 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. $\texttt{ThinkD}_{\texttt{acc}}$ has no tunable parameters.

Results. First, we ran `Tonic` for various values of the parameters α and β (see Extended Version [2] for a more complete overview). We also ran `WRS` and `Chen` with various

[1]While the algorithm described in [7] 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.

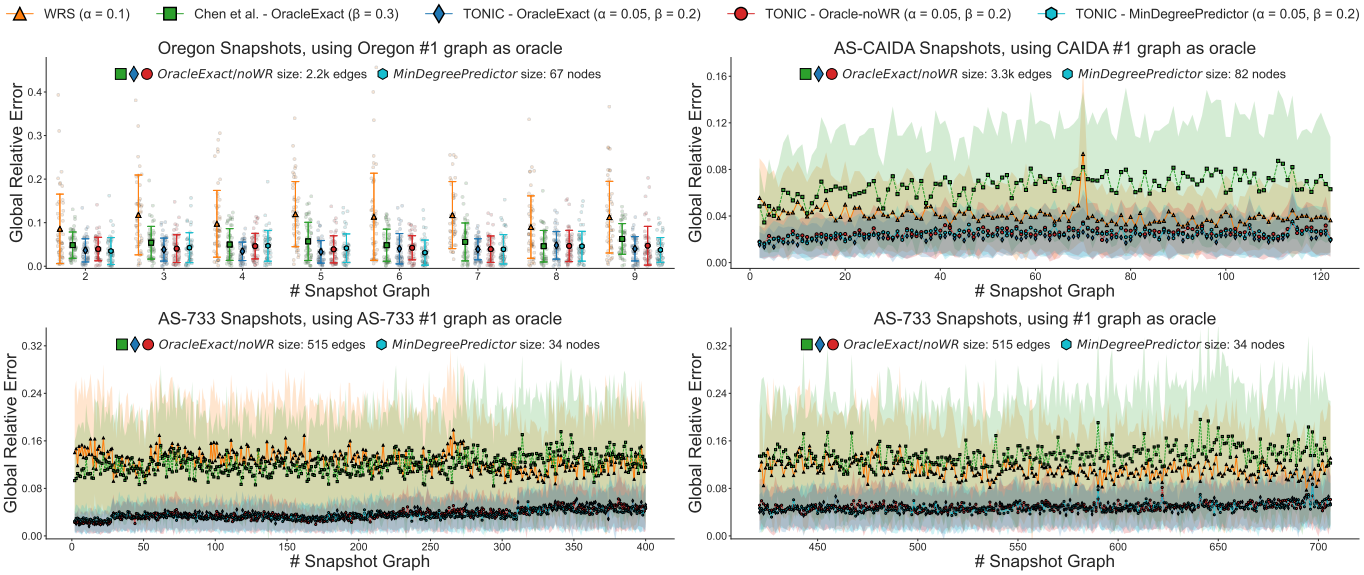

Fig. 4. Error with snapshot networks with sequence of graph streams. The bottom plots are for the first 400 streams (left) and the remaining streams (right) of AS-733. In all cases the predictors are trained only on the first graph stream of the sequence (with results not shown on such graph stream). 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).

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`). 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 $k = f \cdot m$, with $m = |E|$, for $f = 0.01, 0.03, 0.05, 0.1$. 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` (with predictor `OracleExact`) and `WRS`, since the runtime of `Chen` is always at least 4 times larger than `Tonic` runtime. We see that `Tonic` is always faster than `WRS` (excluding $0.01 \cdot m$ 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`.

We then assessed the quality of the approximations reported by `Tonic` and `WRS` in terms of unbiasedness, variance, number of counted and estimated triangles, and quality estimates at any time of the stream. Fig. 3 shows the results for some representative dataset (see Extended Version of the paper for the all results). We observe that, as expected, all three algorithms have unbiased estimates, but `Tonic` has a much lower variance (Fig. 3 left), confirming our theoretical analysis (Section IV-A2). 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`).

[Decide to keep snapshots experiments as last one or not.] Finally, 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 stream, and ran each algorithm on each stream of the sequence. 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 when used on the subsequent graph streams. 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 there is a slight deterioration of results 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`.

In the following/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. For the complete version of results, we refer again to [2]. More specifically, we create fully-dynamic streams considering 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 this way, we are able to maintain the statistics of each snapshot inside the full FD stream, that is if we consider the FD stream up to some correct timestamps, since we added removals inside the window, we obtain the corresponding graph snapshots. Hence, at the end of the sequence, we end with our full FD stream for each different network of snapshots. In the following, we refer to our algorithm for FD streams as `Tonic-FD` for which we described the general workflow in the Extended Version [2]. 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. Figure 5 shows the estimation error as time progresses during Oregon FD stream. The title contains the following dataset's statistics: number of unique nodes $\bar{n}$, number of unique edges $\bar{m}$, maximum number of edges m_{max}, total number of edges m, and number of global triangles at the end T_m, derived from the FD stream. All the algorithms have been run with memory budget $k = m_{max}/10$ (which is highlighted by the black arrow in the stream) and reporting average and standard deviation over 50 independent repetitions.

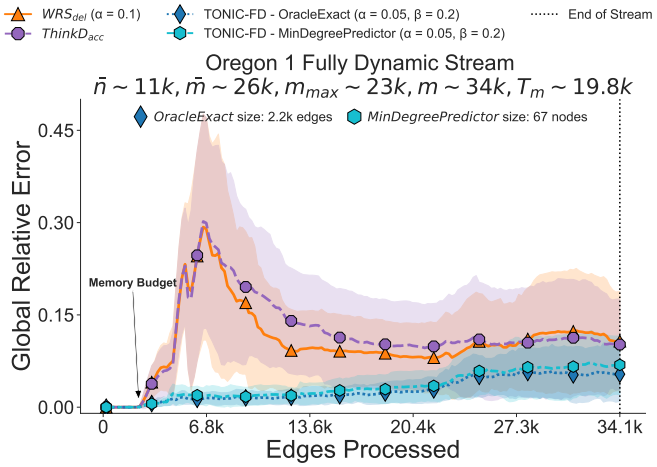

Fig. 5. Estimation error as time progresses during Oregon fully dynamic stream. For each combination of algorithm and parameter (including predictor for `Tonic-FD`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` they are fixed as in the respective publication; for `Tonic-FD` they are as chosen in Fig. 1.

VI. Conclusion

In this paper we presented `Tonic`, the first practical algorithm with predictions for fast and accurate triangle counting in 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 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 and of algorithms for fully dynamic graph streams.

References

[1] M. Al Hasan and V. S. Dave, "Triangle counting in large networks: a review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1226, 2018.

[2] Anonymous. Fast and accurate triangle counting in graph streams using predictions - extended version. [Online]. Available: https://anonymous.4open.science/r/Tonic-F2C4/TonicAdditionalMaterial.pdf

[3] 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)*, vol. 4, no. 3, pp. 1–28, 2010.

[4] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips, "Tolerating the community detection resolution limit with edge weighting," *Physical Review E*, vol. 83, no. 5, p. 056119, 2011.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," in *Proceedings of the 20th international conference on World Wide Web*, 2011, pp. 587–596.

[6] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2006, pp. 253–262.

[7] 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 preprint arXiv:2203.09572*, 2022.

[8] R. Gemulla, W. Lehner, and P. J. Haas, "Maintaining bounded-size sample synopses of evolving datasets," *The VLDB Journal*, vol. 17, pp. 173–201, 2008.

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

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

[11] M. Jha, C. Seshadhri, and A. Pinar, "A space efficient streaming algorithm for triangle counting using the birthday paradox," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 589–597.

[12] 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, Warwick, UK, July 9-13, 2012, Proceedings, Part II 39*. Springer, 2012, pp. 598–609.

[13] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[14] J. Kunegis, "KONECT – The Koblenz Network Collection," in *Proc. Int. Conf. on World Wide Web Companion*, 2013, pp. 1343–1350. [Online]. Available: http://dl.acm.org/citation.cfm?id=2488173

[15] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

[16] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical computer science*, vol. 407, no. 1-3, pp. 458–473, 2008.

[17] 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.

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

[19] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 685–694.

[20] M. Manjunath, K. Mehlhorn, K. Panagiotou, and H. Sun, "Approximate counting of cycles in streams," in *Algorithms–ESA 2011: 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings 19*. Springer, 2011, pp. 677–688.

[21] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[22] M. Mitzenmacher, "A model for learned bloom filters and optimizing by sandwiching," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[23] M. Mitzenmacher and S. Vassilvitskii, "Algorithms with predictions," *Communications of the ACM*, vol. 65, no. 7, pp. 33–35, 2022.

[24] R. Pagh and C. E. Tsourakakis, "Colorful triangle counting and a mapreduce implementation," *Information Processing Letters*, vol. 112, no. 7, pp. 277–281, 2012.

[25] H.-M. Park and C.-W. Chung, "An efficient mapreduce algorithm for counting triangles in a very large graph," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 539–548.

[26] H.-M. Park, S.-H. Myaeng, and U. Kang, "Pte: Enumerating trillion triangles on distributed systems," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1115–1124.

[27] H.-M. Park, F. Silvestri, U. Kang, and R. Pagh, "Mapreduce triangle enumeration with guarantees," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, pp. 1739–1748.

[28] A. Pavan, K. Tangwongsan, S. Tirthapura, and K.-L. Wu, "Counting and sampling triangles from a graph stream," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 1870–1881, 2013.

[29] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com

[30] K. Shin, "Wrs: Waiting room sampling for accurate triangle counting in real graph streams," in *ICDM*. IEEE, 2017, pp. 1087–1092.

[31] 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*. Springer, 2018, pp. 141–157.

[32] 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)*, vol. 14, no. 2, pp. 1–39, 2020.

[33] P. Singh, V. Srinivasan, and A. Thomo, "Fast and scalable triangle counting in graph streams: The hybrid approach," in *International Conference on Advanced Information Networking and Applications*. Springer, 2021, pp. 107–119.

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

[35] 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 Transactions on Knowledge Discovery from Data (TKDD)*, vol. 11, no. 4, pp. 1–50, 2017.

[36] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *Proceedings of the 20th international conference on World wide web*, 2011, pp. 607–614.

[37] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 837–846.

[38] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller, "Triangle sparsifiers." *J. Graph Algorithms Appl.*, vol. 15, no. 6, pp. 703–726, 2011.

[39] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

APPENDIX

Generate bibitems: [1]

In the following, we present and describe more accurately some given results, alongside with theoretical proofs. Moreover, we show more extensively the results of all our experiments for each dataset, including also local triangles ones. [TO DO: make a brief summary of this note].

NOTATION AND SYMBOLS

Table II resumes the notation and symbols used in the main paper, and in the supplementary material.

TABLE II
TABLE OF NOTATION AND SYMBOLS USED IN OUR WORK TO DO: CHECK

Symbol	Definition
Notations for graph streams	
$G^{(t)} = (V,\ E^{(t)})$	Graph G at time t
$\{u, v\}$	(Unordered) edge between u and v
$\{u, v, w\}$	Triangle with nodes u, v and w
$e^{(t)}$	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	
k	Memory budget, i.e., maximum number of edges that can be stored
$\mathcal{W}$	Waiting room
α	Fraction of the size of the waiting room, i.e., $\lvert\mathcal{W}\rvert/k = \alpha, \alpha \in [0, 1]$
$\mathcal{H}$	Set of heavy edges
β	Fraction of the size of the set of heavy edges, i.e., $\lvert\mathcal{H}\rvert/k(1-\alpha) = \beta,\ \beta \in [0, 1]$
$\mathcal{S}_L$	Sample set of light edges of size $\lvert\mathcal{S}_L\rvert = s_\ell = k(1-\alpha)(1-\beta)$
$\mathcal{S} = (\hat{V}, \hat{E})$	Overall 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 $\{u, v, w\}$

ADDITIONAL ALGORITHMS

[TO DO: check coherence with algo's description in the main paper (e.g., use emph for "light")]

In `SampleLightEdge` (Algorithm 4), we present the subroutine for sampling light edges [remove if we decide to keep this subroutine in the main paper] with probability computed as s_ℓ/ℓ, where $s_\ell = k\,(1-\alpha)\,(1-\beta)$ is the fixed size of the sample $\mathcal{S}_L$ of light edges, while ℓ is the number of light edges observed in $\Sigma^{(t)}$ (remember that at time t an edge of $\Sigma^{(t)}$ is *light* if it is not in $\mathcal{W}^{(t)}$ nor in $\mathcal{H}^{(t)}$). Also, we assume that *FlipBiasedCoin(p)* flips *HEAD* with probability p.

Algorithm 4: `SampleLightEdge`$(\{u, v\}, \mathcal{S}_L, \ell)$

Input : edge $\{u, v\}$; current sample $\mathcal{S}_L$ of light edges; number of observed light edges ℓ in the stream

1 $p_{sampling} = \frac{k(1-\alpha)(1-\beta)}{\ell}$;
2 **if** *FlipBiasedCoin*$(p_{sampling})$ == *HEAD* **then**
3 $\quad \{\bar{u}, \bar{v}\} \leftarrow$ edge sampled uniformly at random from $\mathcal{S}_L$;
4 $\quad \mathcal{S}_L \longleftarrow \mathcal{S}_L \setminus \{\{\bar{u}, \bar{v}\}\} \cup \{\{u, v\}\}$;

A. Random Pairing

In our algorithm `Tonic-FD` we sample edges through *random pairing* [8], 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 [17], [31], [35]. 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 $|\mathcal{S}_L| = k$. Anyway, at any time t in the stream, `Tonic-FD` stores in $\mathcal{S}_L$ a set of $k' \leq k$ edges sampled uniformly at random among the *light edges* of $G^{(t)}$, that are edges of $G^{(t)}$ that are not in the waiting room $\mathcal{W}^{(t)}$ or kept in the set $\mathcal{H}^{(t)}$ of (predicted) heavy edges (see Section IV). 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. The algorithm works as follows: when receiving an edge deletions, it increments the counter d_b for bad deletions and removes the edge from the sample $\mathcal{S}_L$ if present, 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 an 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 [8] 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 Predicitons in Fully Dynamic Streams

In `Tonic-FD` (Algorithm. 5), we provide the pseudocode for the extension of our algorithm to fully dynamic graph streams (i.e., allowing also deletions of edges). The general workflow of `Tonic-FD` is very similar to the insertion-only version presented in the main paper. Since we are dealing with both edge insertion and deletions, we make use of random pairing [8], as described in Section A.

The algorithm works as follows: given the current edge $\{u, v, \eta\}$ in the stream, first it checks whether the edge if an edge insertion or deletions. If we have an edge insertion (line 6), we proceed as Algorithm 3 in the main paper, but paying attention to the number d_g and d_b of respectively good and bad deletion in the stream observed so far. Intuitively, the number d_g represents the number of edge deletions that involves edges that are not stored in the sample $\mathcal{S}_L$ (i.e., good

deletions), while the number d_b is the number of deletions of edges that are currently stored in our sample $\mathcal{S}_L$ (i.e., bad deletions), that causes the decrement of the size of our stored subgraph. If such numbers are compensated (i.e., their sum is equal to zero, line 19), `Tonic-FD` resumes to the standard reservoir sampling method. Otherwise, we need to account for such uncompensated number of deletions, and sample the incoming edge with probability $d_b/(d_b + d_g)$ (line 25). Also, depending on such probability, we re-adjust the number of deletions by decrementing the respective counter (lines 27, 28).

If the current edge in the stream is an edge deletion (6), `Tonic-FD` checks if the edge is in the waiting room $\mathcal{W}$ or in the heavy edges set $\mathcal{H}$ (line 30); in such case, the algorithm removes the edge from the respective set. Otherwise, we are dealing with a light edge. We first need to decrease the number ℓ of light edges seen in the stream so far (line 18), 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 (line 34). Algorithm 5 removes such edge and accounts for the bad deletion. (ii) the current edge is not stored in our subgraph (line 37); `Tonic` ignores the edge and accounts for the good deletion.

When returning the final estimates (line 38), we return always non-negative triangle counts: in this way, we trade for unbiasedness [specify also in unbiasedness theorem] of the algorithm by achieving better accuracy of estimates.

In `CountTriangles-FD` (Algorithm 6), we report the procedure for counting triangles in the current subgraph in the fully dynamic setting. Such algorithm is slightly different from the subroutine presented in the main paper since it takes into consideration the number 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 triangle countings.

Algorithm 5: `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 α; fraction of heavy edges space β; 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* Σ **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 $\{x, y\} \leftarrow$ 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 $\{u', v'\} \leftarrow$ 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** $\{u', v'\} \leftarrow \{x, y\}$;
19 **if** $d_g + d_b == 0$ **then**
 `// deletions compensated`
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**
25 **if** *FlipBiasedCoin*$\left(\frac{d_b}{d_b + d_g}\right)$ == *HEAD* **then**
26 $\mathcal{S}_L \longleftarrow \mathcal{S}_L \cup \{\{u', v'\}\}$;
27 $d_b \longleftarrow d_b - 1$;
28 **else** $d_g \longleftarrow d_g - 1$;
29 **else if** $\eta == -$ **then**
30 **if** $\{u, v\} \in \mathcal{W} \cup \mathcal{H}$ **then**
 $\mathcal{W} \cup \mathcal{H} \longleftarrow \mathcal{W} \cup \mathcal{H} \setminus \{u, v\}$;
31 **else**
32 $\ell \longleftarrow \ell - 1$;
33 **if** $\{u, v\} \in \mathcal{S}_L$ **then**
 `// bad deletion`
34 $\mathcal{S}_L \longleftarrow \mathcal{S}_L \setminus \{\{u, v\}\}$;
35 $d_b \longleftarrow d_b + 1$;
36 **else**
 `// good deletion`
37 $d_g \longleftarrow d_g + 1$;
38 **return** `max`$(0, \hat{T})$, `max`$(0, \hat{T}_u)$ for each node $u \in \mathcal{S}$;

Proofs

In this section, we are going to provide theoretical proofs for the claims of the main paper. We are going to describe the correctness of the algorithm (probabilities computation), unbiasedness, variance, time and space complexity, and the variance comparison with `WRS` algorithm.

C. Probabilities Computation

In the following, we are going to prove that the probability p_{uvw} computed and used by procedure Algorithm 2 `CountTriangles` in the main paper (resp. `CountTriangles-FD`, Algorithm 6) within `Tonic` (`Tonic-FD`), corresponds to the probability that triangle $\{u, v, w\}$ is counted by `Tonic` (`Tonic-FD`).

In general, given a triangle $\{u, v, w\}$ discovered when $\{u, v\}$ 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 number of predicted light edges seen so far, and the number of good and 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. The latter corresponds also to the edge arrived on the stream at the moment in which we discovered the triangle $\{u, v, w\}$. We have the following results.

Lemma A.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 $\{u, v, w\}$*

Algorithm 6: `CountTriangles-FD`$(\{u,v,\eta\},\mathcal{S},\ell)$

Input : edge $\{u,v,\eta\}$; subgraph $\mathcal{S}=\left(\hat{V},\ \hat{E}\right)$; number ℓ of predicted light edges in the stream $\Sigma^{(t)}$; counter for good deletions d_g; counter for bad deletions d_b

for *each node* w *in* $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ **do**
  initialize $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ to zero if they have not been set yet;
  $p_{uvw}=1$;
  if $\{w,u\}\in\mathcal{S}_L$ *AND* $\{v,w\}\in\mathcal{S}_L$ **then**
    $p_{uvw}=\min\left(1,\ \frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\times\frac{k(1-\alpha)(1-\beta)-1}{\ell+d_g+d_b-1}\right)$;
  else if $\{w,u\}\in\mathcal{S}_L$ *OR* $\{v,w\}\in\mathcal{S}_L$ **then**
    $p_{uvw}=\min\left(1,\frac{k(1-\alpha)(1-\beta)}{\ell+d_g+d_b}\right)$;
  if $\eta==+$ **then**
    increment $\hat{T}$, $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ by $1/p_{uvw}$;
  else if $\eta==-$ **then**
    decrement $\hat{T}$, $\hat{T}_u$, $\hat{T}_v$, $\hat{T}_w$ by $1/p_{uvw}$;

with last edge $\{u,v\}$*, the probability* p_{uvw} *that* $\{u,v,w\}$ *is discovered 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 A.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 Algorithm `Tonic` (Algorithm 1 of main paper) always stores edges in the waiting room and in the heavy edge set (lines [TO DO: check lines]), then the triangle is counted 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 $\ell\le k(1-\alpha)(1-\beta)$, 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:

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

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

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

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

Lemma A.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 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 $\mathcal{H}^{(t-1)}\cup\mathcal{W}^{(t-1)}\cup\mathcal{S}_L^{(t-1)}=\mathcal{S}^{(t-1)}$. In the following, we proceed with the proof of the computation of the probabilities by cases as in Lemma A.2:

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` (Algorithm 5) always stores edges in the waiting room and in the heavy edge set (lines 7, 10, 12, and 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 $\ell\le k(1-\alpha)(1-\beta)$, we have sampled both edges deterministically, hence the discovered triangle is counted or deleted with probability 1. Otherwise, when $\ell>k(1-\alpha)(1-\beta)$, 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:

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

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

D. Unbiasedness of Tonic

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

Theorem A.3. *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, 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 A.1, A.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:

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

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

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

which proves Eq. 2. □

E. Time Complexity

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

Theorem A.4. *Given an input graph stream Σ of insertion-only edges, 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 Algorithm 2 of main paper, line 1 of Algorithm 6) and the insertion, retrieval or deletion of the lightest edge in $\mathcal{H}$ (line ?? of Algorithm 1 in main paper, line 15 of Algorithm 6). Also, for Tonic-FD such cost is paid if the deletion of current edge occurs in the heavy edge set $\mathcal{H}$ (line 30). 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. □

F. 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, for local counting, the analysis above 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).

G. Tonic vs WRS: Variance Comparison

in the following section, we prove Theorem/Proposition X of the main pepr. 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 (see line 19 of Algorithm 1 in the main paper) 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. 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).

In our analysis we need to 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 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 4. *Let* $\mathbf{Var}[\hat{T}_{WRS}(p)]$ *be the variance of the estimate* $\hat{T}_{WRS}$ *obtained by* WRS *when edges are sampled independently with probability* p*, and let* $\mathbf{Var}[\hat{T}_{Tonic}(p')]$ *be the variance of the estimate* $\hat{T}_{Tonic}$ *obtained by* Tonic *when* light *edges are sampled with probability* p'*. Then* $\mathbf{Var}[\hat{T}_{WRS}(p)] \geq \mathbf{Var}[\hat{T}_{Tonic}(p')]$ *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 $\mathcal{T}^{S_1,S_2}$. Also, 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:

$$\begin{aligned}
&\mathbf{Var}\left[\hat{T}^{S1,S2}_{algo}\right] = \mathbf{Var}\left[\sum_{\{u,v,w\}\in\mathcal{T}^{S1,S2}_{algo}} \delta_{\{u,v,w\}}\right] \\
&= \sum_{\{u,v,w\}\in\mathcal{T}^{S1,S2}_{algo}} \mathbf{Var}\left[\delta_{\{u,v,w\}}\right] + \mathbf{Cov}\left[\hat{T}^{S1,S2}_{algo}, \hat{T}^{S1,S2}_{algo}\right] \\
&= \sum_{\{u,v,w\}\in\mathcal{T}^{S1,S2}_{algo}} \left(\mathbb{E}\left[\delta^2_{\{u,v,w\}}\right] - \left(\mathbb{E}\left[\delta_{\{u,v,w\}}\right]\right)^2\right) + \\
&\quad + \mathbf{Cov}\left[\hat{T}^{S1,S2}_{algo}, \hat{T}^{S1,S2}_{algo}\right] \\
&= T^{S1,S2}_{algo}\left(1/q^2 - 1\right) + \mathbf{Cov}\left[\hat{T}^{S1,S2}_{algo}, \hat{T}^{S1,S2}_{algo}\right]
\end{aligned}$$

where $\delta_{\{u,v,w\}}$ is defined as Eq. 3 (but in a fixed probability context), and q can be p, p^2, p', p'^2 depending on the algorithm for which we are computing the variance and on the set of triangles $\mathcal{T}^{S1,S2}$ 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 we have $q = p'$ 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:

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

We can express the covariance terms as:

$$\begin{aligned}
&\mathbf{Cov}\left[\hat{T}^{x,y},\hat{T}^{w,z}\right] = \sum_{\Delta_i\in\mathcal{T}^{x,y},\Delta_j\in\mathcal{T}^{w,z}} \mathbf{Cov}\left[\delta_i^{x,y},\delta_j^{w,z}\right] \\
&= \sum_{\Delta_i\in\mathcal{T}^{x,y},\Delta_j\in\mathcal{T}^{w,z}} \mathbb{E}\left[\delta_i^{x,y},\delta_j^{w,z}\right] - \mathbb{E}\left[\delta_i^{x,y}\right]\mathbb{E}\left[\delta_j^{w,z}\right] \\
&= \sum_{\substack{\Delta_i,\Delta_j\in\\ \mathcal{T}^{x,y}\cap\mathcal{T}^{w,z}}} \frac{1}{p_i}\,\frac{1}{p_j}\,\mathbb{P}\left[\Delta_i,\Delta_j\in\hat{T}\text{ and }\Delta_i\cap\Delta_j\text{ sampled}\right] - 1 \\
&= \sum_{\Delta_i,\Delta_j\in\mathcal{T}^{x,y}\cap\mathcal{T}^{w,z}} \frac{1}{p_i}\,\frac{1}{p_j}\,\frac{p_i\;p_j}{q} - 1 \\
&= |\mathcal{T}^{x,y}\times\mathcal{T}^{w,z}|\left(\frac{1}{q}-1\right)
\end{aligned} \tag{4}$$

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 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 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:

$$\begin{aligned}
&\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] - \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')] = T^{\mathcal{H},\mathcal{W}}(1/p-1) \\
&+ T^{\mathcal{H},\mathcal{H}}(1/p^2-1) + T^{\mathcal{H},L}(1/p^2-1/p') + 2S_{\mathcal{H}}(1/p-1) \\
&+ T^{L,\mathcal{W}}(1/p-1/p') + T^{L,L}(1/p^2-1/p'^2) + \\
&+ 2\bar{S}_{\mathcal{H}}(1/p-1/p').
\end{aligned} \tag{5}$$

In Eq. 5, 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. 5 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 the following:

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

We have that

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

since $\Delta(h)\geq\rho/c$ by the assumptions on the predictor. Analogously, we have

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

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

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

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 5. *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* $\mathbf{Var}[\hat{T}_{\texttt{WRS}}(p)] = \mathbf{Var}[\hat{T}_{\texttt{Tonic}}(p')]$.

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. □

Additional Experimental Results

H. Further Experimental Evaluation

[TO DO: present experiments of the section. We may decide to further split the "subsubsections"]

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. Moreover, by considering a bigger set of heavy edges, we observed that overlap between the top 10% of edges from `MinDegreePredictor` and the top 10% of edges from `OracleExact` (i.e., predictors used in our experimental evaluation) ranges from 40% to 65%, for the considered datasets.

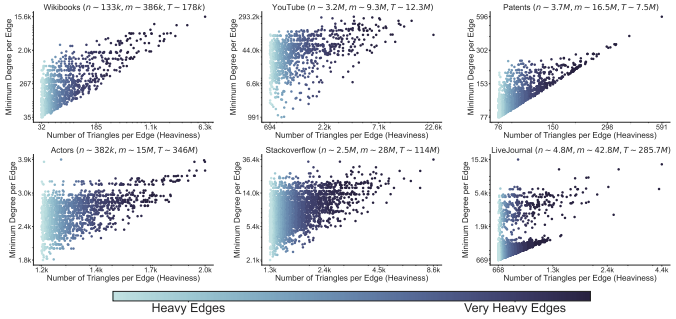

Fig. 6. Relation between Number of Triangles (i.e., heaviness) per edge and Minimum Degree per edge through the top m ‰ heaviest edges in the graph stream (for Wikibooks dataset, the top 1000 heaviest edges have been considered). Both axis are represented using a log scale.

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`. Moreover, since for `Tonic` some permutation of parameters could correspond to the same deterministic space used, we reported in Fig. 1 the best of such combinations, besides our supposed choice of parameters $\alpha_{\texttt{Tonic}} = 0.05$ and $\beta_{\texttt{Tonic}} = 0.2$ that is 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.

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 in the figure), 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. We collected such statistics for 50 independent trials, and then we reported the mean bars with standard deviation (black error bar). 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 $0.2k$.

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 $\sum_{u \in \mathcal{V}} T_u = 3T$. For assessing the performance of local triangles estimates, 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:

- *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* [34] 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

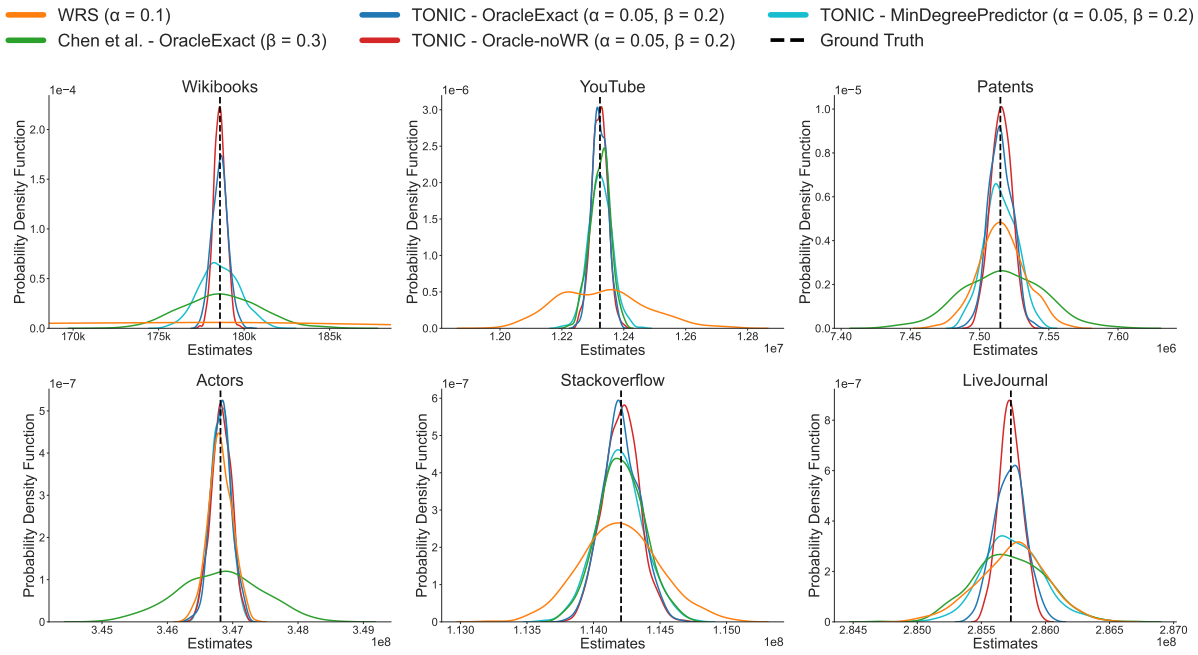

Fig. 7. Probability Density Distribution of estimations. For each combination of algorithm and parameter (including oracle for `Tonic`), the distribution of estimates over 500 independent repetitions is shown. The algorithms parameters are as in legend (for `WRS` and `Chen` they are fixed as in the respective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

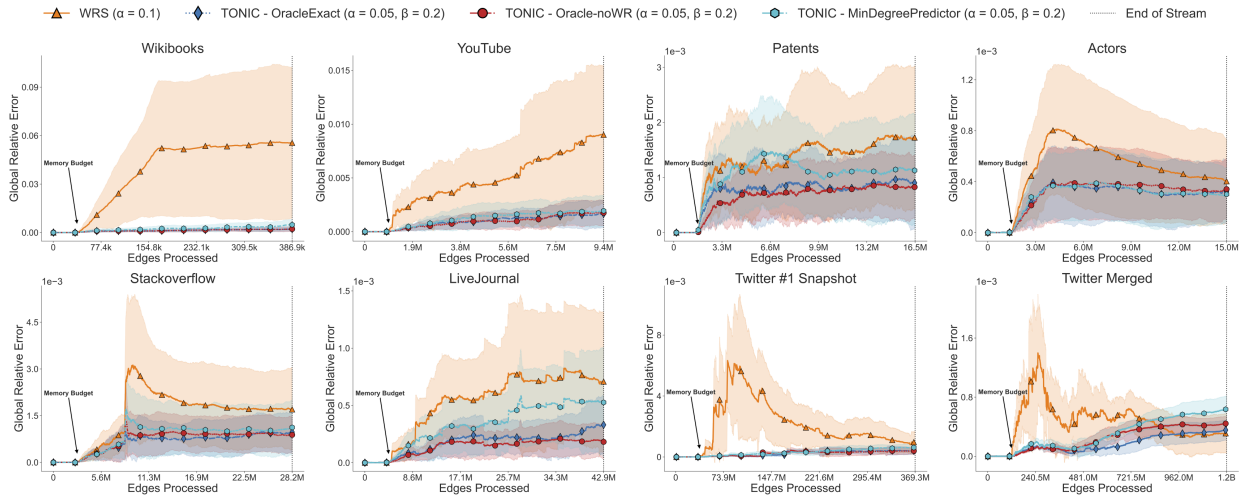

Fig. 8. Estimation error as time progresses during the input graph stream. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown (except for Twitter datasets, where 10 repetitions have been considered). The 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. 2, we provided both algorithms with memory budget $k = f \cdot m$, 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) Scalability to Big Datasets: in this section we provide experimental analysis also for the second, third and fourth snapshot of the network, assessing `Tonic` scalability performance with respect to the competitors. First snapshot and the merged version of Twitter have been already discussed in Fig 2. Here, in Fig. 11, we show results for accuracy and time 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 4 times larger than `Tonic` runtime. We see that `Tonic` is always faster than `WRS` (excluding $0.01 \cdot m$ memory budget for Twitter #2 snapshot, for which the runtimes are comparable). Such results confirm that `Tonic`, in practice, is able to scale

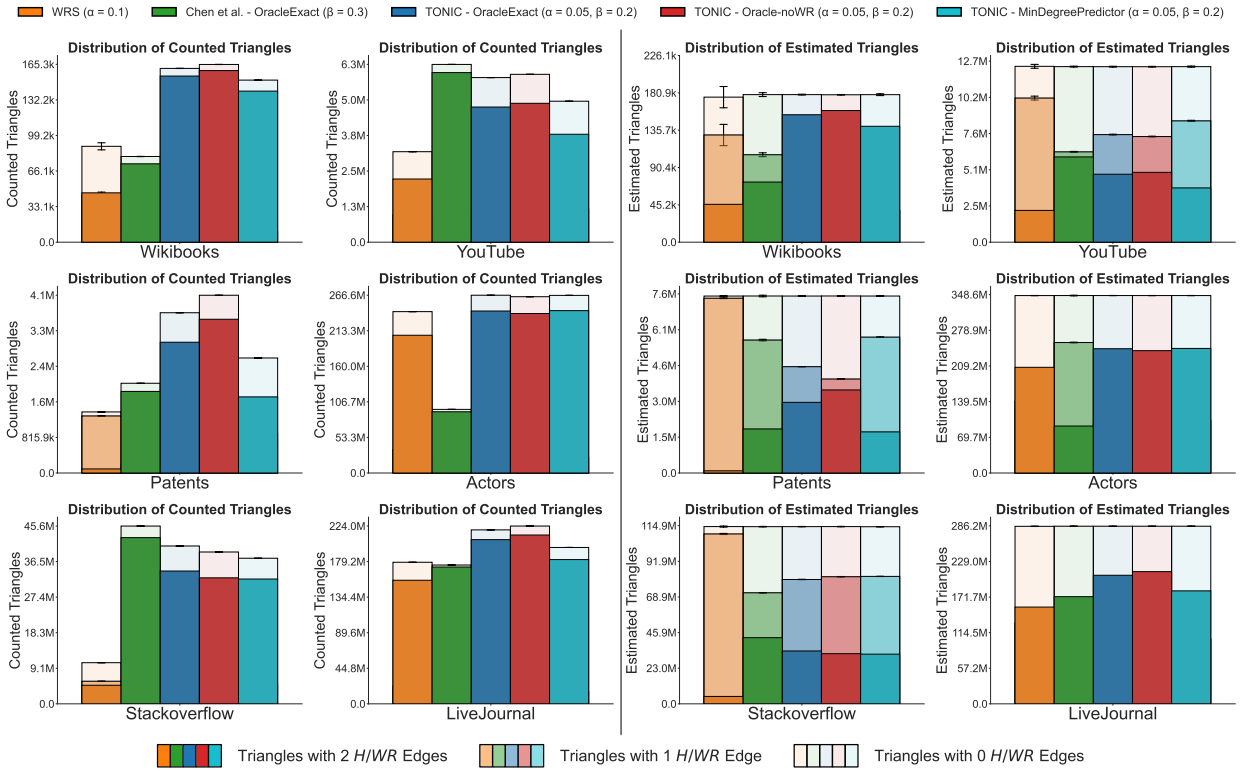

Fig. 9. Distributions of Counted (left) and Estimated (right) Triangles for each dataset. For each combination of algorithm and parameter (including oracle for `Tonic`), the average and standard deviation over 50 repetitions are shown. The algorithms parameters are as in legend (for `WRS` and `Chen`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

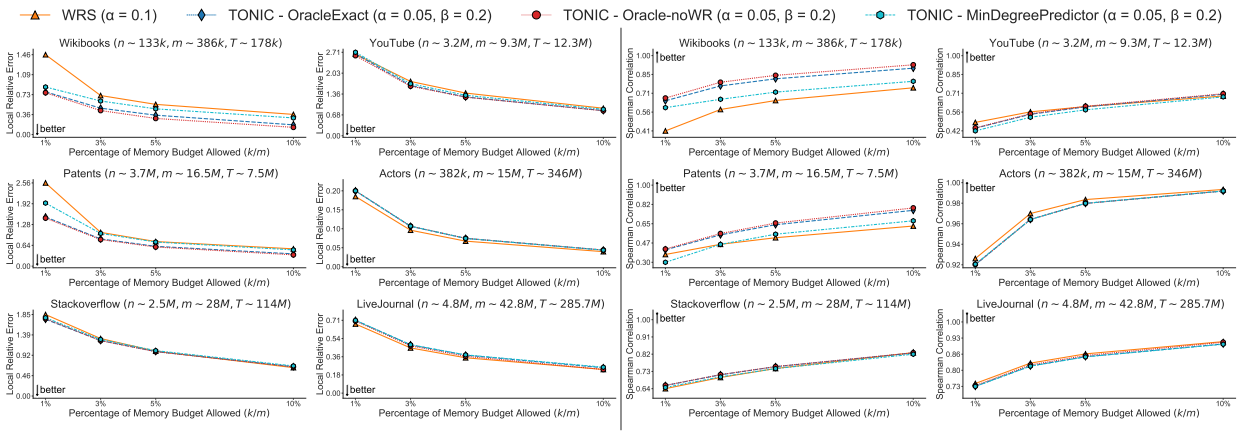

Fig. 10. Error (left) and Spearman Rank Coefficient (right) vs memory budget. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 10 repetitions are shown. The algorithms parameters are as in legend (for `WRS` they are fixed as in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

better with respect to worst-case analyses of the running time. [Comments on accuracy? We may point that Twitter snaps are adjacency list streams... Still waiting for last results of Chen.]

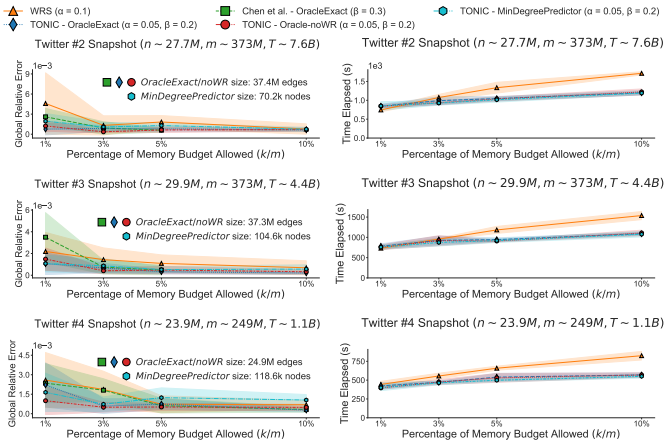

Fig. 11. Error (left) and runtime (right) vs memory budget. For each combination of algorithm and parameter (including predictor for Tonic), the average and standard deviation over 10 repetitions are shown. The algorithms parameters are as in legend (for WRS they are fixed as in the respective publication; 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 best and adversarial oracles and predictors. Again, we want to emphasize that the node-degree predictor in this case 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 graphs in the sequence, as reported in the caption in 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, the results show 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. In this way, we are able to maintain the statistics of each snapshot inside the full FD stream, that is if we consider the FD stream up to some correct timestamps, since we added removals inside the window, we obtain the corresponding graph snapshots. Hence, at the end of the sequence, we end with our full FD stream for each different network of snapshots. In the following, we refer to our algorithm for FD streams as Tonic-FD for which we described the general workflow in Algorithm 5.

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 and runtime vs memory budget, (ii) error during the evolving of the FD stream. The choice of the memory budget to provide to the algorithms is a bit challenging since the streams could vary greatly considering the number of total edges (insertions and deletions), and the number of unique edges. To this end, we consider as reference the number of maximum edges m_{max} contained in the FD stream at some time. Oracles and predictors for Tonic-FD are trained *only* on the *first* snapshot of the sequence, as described for snapshot experiments in Section V. In Fig. 13 we report results where we provide the three algorithms with memory budget $k = f \cdot m_{max}$, where the range of f is reported on the x-axis for each dataset (subplot). For each subplot, we report in the title the following dataset's statistics: number of unique nodes $\bar{n}$, number of unique edges $\bar{m}$, maximum number of edges m_{max}, total number of edges m, and number of global triangles at the end T_m. Moreover, we show oracle and predictors sizes in the legend of each subplot. As one can see..... Then, Fig. 14 shows the estimation error as time progresses during the input FD stream. All the algorithms have been run with memory budget $k = m_{max}/10$, which is highlighted by the black arrow in the stream for each subplot. For all the experiments, the average and standard deviation over 50 independent repetitions are shown, except for Twitter FD stream, where we show results over 10 repetitions due to time unfeasibility.

I. Details on Oracles and Predictors

[Description of this section, if this structure is left]

1) Overhead of node-based oracles: [The following par is already present in the main paper at the moment] in this section, we provide further details on how we build MinDegPredictor. More specifically, sorting the edges of the training stream by their respective decreasing minimum degree, we retain the top 10% 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-valued

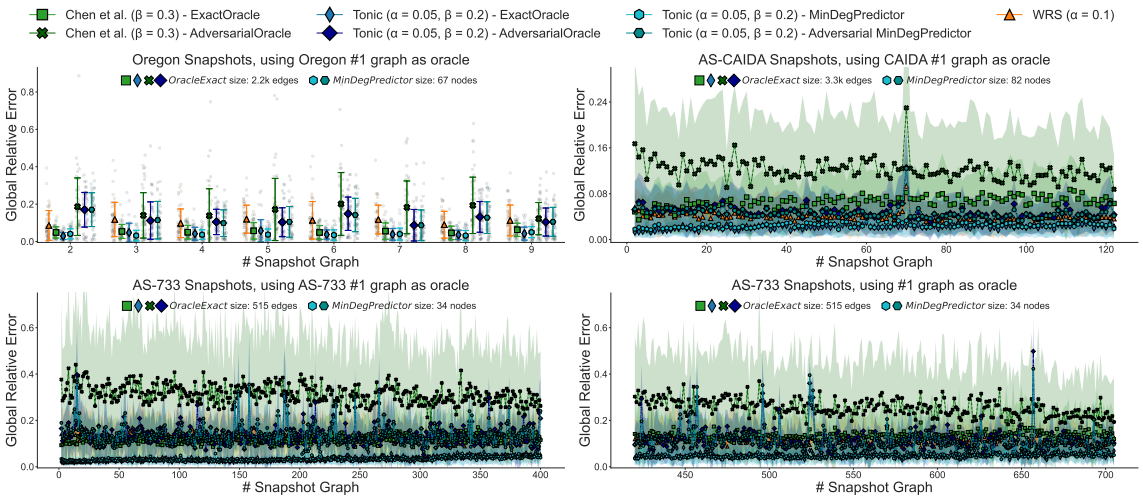

Fig. 12. Error with snapshot networks with sequence of graph streams. In all cases the predictors are trained only on the first graph stream of the sequence (with results not shown on such graph stream). For each combination of algorithm and parameter (including predictor for `Tonic`), the average and 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).

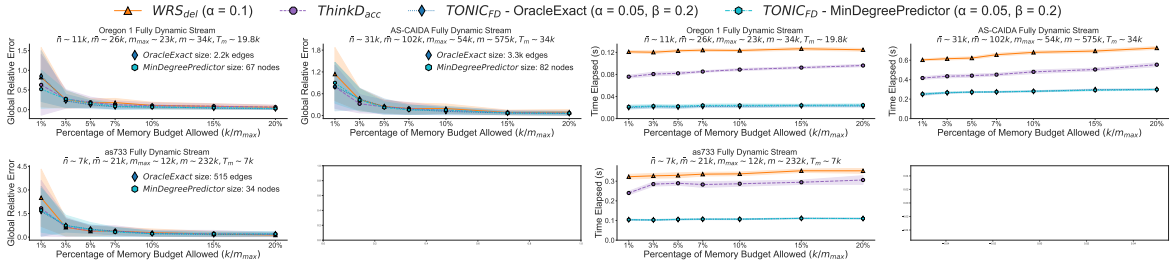

Fig. 13. Error (left) and runtime (right) vs memory budget. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown (except for Twitter, where 10 repetitions have been considered). The algorithms parameters are as in legend (for `WRS` they are fixed as in the respective publications; for `Tonic` they are as chosen in Fig. 1).

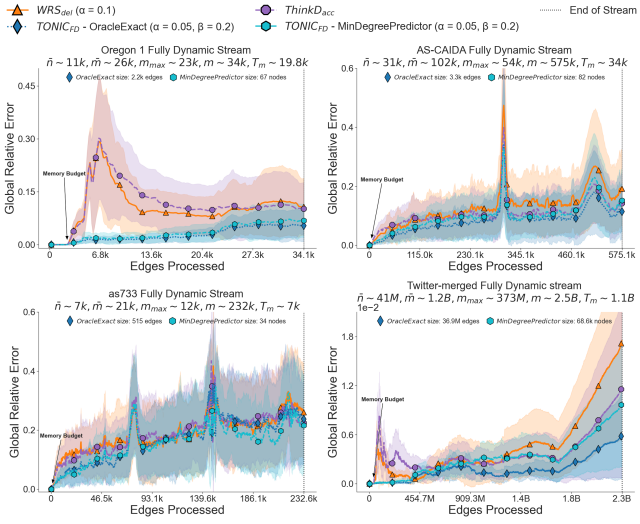

Fig. 14. Estimation error as time progresses during the input fully dynamic graph stream. For each combination of algorithm and parameter (including predictor for `Tonic`), the average and standard deviation over 50 repetitions are shown (except for Twitter, where 10 repetitions have been considered). 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).

entries. Starting from 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 `Tonic` and `Tonic-FD`, 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. However, we noticed that such "approximation" of nodes is very accurate and the great majority of nodes involved in the highest minimum degree edges are the ones with highest degrees. Node-based oracles are in someway encoding edge features in a much more succinct representation. Hence, the resulting oracles' overhead is significantly lower than the one used in edge-based derived oracles. 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.

In this way, we are able to obtain huge savings in terms of space to store and time to read the predictor. To be more precise, since in node-based representation each entry is made by two numbers in the lookup table (each entry is $(u; deg(u))$) instead of three numbers in the one for edge-based representation (each entry is $((u, v); f(u, v))$, where f can be a generic function, e.g., heaviness or min degree), the space saving in terms of byte in memory is even bigger. 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. Then, we the Jaccard Similarity 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. As one can see, the nodes contained in the two different predictors are almost the same. To give some additional numbers, for Twitter merged (last row of Table III) 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 $\approx 0.12s$, confirming the huge amount of overhead that we could gain using a node-based `MinDegreePredictor`.

TABLE III
STATISTICS ABOUT USING NODE-BASED MINDEGREEPREDICTOR INSTEAD OF EDGE-BASED MINDEGREEPREDICTOR. THE FIRST THREE COLUMNS REPRESENT RESPECTIVELY THE RATIO OF THE NUMBER OF ENTRIES, THE SIZE IN MEMORY (IN BYTES) AND THE TIME FOR READING FROM FILE OF NODE-BASED PREDICTOR OVER EDGE-BASED PREDICTOR. TIMES FOR READING THE COLUMN HAVE OBTAINED OVER 50 REPETITION, REPORTING MEAN AND STANDARD DEVIATION. 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

2) Time to build predictors/oracles: in the following, we present results of comparison between building `OracleExact` and *node-based* representation of `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 (see Table I), 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`. [TO DO: check if this applies also for `Tonic-FD`, and in that case mention times for wrsDel and thinkDacc]

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