Hypergraph Ego-networks and Their Temporal Evolution

Cazamere Comrie, Jon Kleinberg
Cornell University
{clc348, kleinberg}@cornell.edu

2021 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-2398-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDM51629.2021.00019

***Abstract*—Interactions involving multiple objects simultaneously are ubiquitous across many domains. The systems these interactions inhabit can be modelled using hypergraphs, a generalization of traditional graphs in which each edge can connect any number of nodes. Analyzing the global and static properties of these hypergraphs has led to a plethora of novel findings regarding how these modelled system are structured. However, less is known about the localized structure of these systems and how they evolve over time.**

In this paper, we propose the study of hypergraph ego-networks, a structure that can be used to model higher-order interactions involving a single node. We also propose the temporal reconstruction of hypergraph ego-networks as a benchmark problem for models that aim to predict the local temporal structure of hypergraphs. By combining a deep learning binary classifier with a hill-climbing algorithm, we will present a model for reconstructing hypergraph ego-networks by incorporating structural patterns found across multiple domains.

I. Introduction

Interactions involving multiple objects simultaneously are ubiquitous across many domains: academic papers tend to have multiple co-authors, emails are sent to multiple recipients, and bills in congress are co-sponsored by multiple members. A growing body of research has been dedicated to understanding the structure of these higher-order interactions [4]–[6], [9], [11], seeking to explore these interactions by analyzing the macroscopic trends of the interactions within these systems.

These systems can be modelled using hypergraphs [7]. As generalizations of graphs, hypergraphs are composed of nodes and hyperedges, where each edge can contain any number of nodes. For example, a hypergraph modelling a co-authorship network would represent each author as a node, and each hyperedge would represent a paper co-authored by a set of authors. Whereas a typical graph could only model the pairwise interactions among this set of authors, a hypergraph allows for authors to interact with multiple other authors simultaneously, and thus captures these higher-order interactions effectively.

There are two dichotomies worth considering when discussing the structure of hypergraphs. The first is whether a hypergraph is static or temporal. Static hypergraphs have been studied extensively, where researchers have analyzed the structure of the hypergraph at a specific moment in time. However, very little is known about temporal hypergraphs, which can be viewed as an ordered, timestamped sequence of hypergraphs. Kook et al. [11] have recently considered aggregate patterns found in real-world temporal hypergraphs.

The second dichotomy is whether a hypergraph property is global or local. Analyzing the global properties of hypergraphs has lead to a plethora of novel findings regarding how these modelled systems are structured. For example, [9] observes that large real-world hypergraphs at the macroscopic level have similar well-known properties to real-world dyadic graphs, such as a giant connected component and a heavy-tailed degree distribution. However, little is known about the localized structure of higher-order systems.

In this work we aim to fill these gaps in knowledge by studying the local, temporal structure of hypergraphs. We proceed by drawing an analogy to corresponding structures in traditional (pairwise) graphs, where work has been done on analyzing all the interactions involving a single node [1]–[3]. These interactions centering on one node are commonly modelled using *ego-networks*, the network of pairwise interactions among the neighbors of a single node. Ego-networks in dyadic graphs are used to understand not only the structure of local interactions, but how these local interactions influence the behavior of the global system they inhabit.

What would be the analogue of an ego-network in the hypergraph context? To better contextualize this, as an example consider the complete history of all papers written by a single author, where each paper is represented as a hyperedge containing all authors of each paper. This in turn forms the author's ego-network. It is then interesting to ask whether there are any recurring patterns to the order in which hyperedges appear in the ego-network, and if there are fundamental properties of temporal higher-order interactions that lead to these patterns. By better understanding the temporal nature of these group interactions, we can gain new insights on how ego-networks grow across different domains, and how these local patterns inform global properties of hypergraphs.

Yet, simply proposing models for understanding the temporal evolution of localized hypergraphs is not enough. Without some sort of benchmark problem, it is very difficult to know which models are better than others. Any effective model that attempts to understand the temporal evolution of these aforementioned systems should be inherently predictive. Therefore, there is significant value in the creation of a benchmark problem where one can evaluate their evolutionary models.

In this paper, we propose the study of hypergraph ego-networks, a structure that can be analogously used to model higher-order interactions involving a single node and its neighbors. We propose the temporal reconstruction of hypergraph

978-1-6654-2398-4/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDM51629.2021.00019

ego-networks as a benchmark problem for models that aim to predict the local temporal structure of hypergraphs. We can convey the problem as follows. Suppose we are given a node v that is an element in m hyperedges, and a list of hyperedges $e_1, e_2, ..., e_m$ not sorted by time. How accurately can we predict the order in which the hyperedges arrived? Is there an algorithm that can significantly beat simple baselines derived from random ordering? Here we present a model that outperforms this baseline by incorporating structural patterns found in hypergraph ego-networks across multiple domains.

We demonstrate the effectiveness of our model in three different datasets, capturing higher-order structure in three distinct domains. The first is coauth-DBLP, a publication dataset where nodes represent authors and hyperedges represent a paper co-authored by a set of authors. Next is email-Avocado, a collection of emails where each node represents an email account, and each hyperedge contains all nodes who received the same email together with the sender of the email. The third dataset is threads-ask-ubuntu, where each node represents a user, and each hyperedge contains all users answering a question on a forum. These datasets are particularly useful to study for this problem as they contain entire lifetimes of user activity. The entire history of every interaction for each user is recorded in each dataset, allowing us to accurately see when nodes are first added to ego-networks, and to analyze the temporal structure of each user's ego-network.

Before we describe our model, we will define a few basic constraints for our temporal reconstruction problem. Firstly, a hyperedge is defined to be *non-trivial* if it contains at least two nodes, and is *trivial* otherwise. The hypergraph ego-networks we will be analyzing will exclude all trivial hyperedges, as these do not capture information regarding a node's higher-order interactions with others. Secondly, the *length* or *size* of an ego-network is defined as the total number of hyperedges in the ego-network. We only analyze ego-networks of at least some minimum length (typically 20 or greater for coauth-DBLP and 10 or greater for email-Avocado and threads-ask-ubuntu). This is so that our model can observe a sufficient number of higher-order interactions before it makes any predictions. We will only analyze hypergraph ego-networks that grow through the addition of hyperedges, so no hyperedges will be removed from a growing ego-network. We leave the temporal analysis of shrinking ego-networks for future work. Finally, we will ignore all ego-networks where the majority of hyperedges are identical and where the ego-network has less than 10 neighbors (except for email-Avocado), as any model that attempts to sort these ego-networks will do well.

Our approach is as follows. Firstly, we propose a supervised deep learning method to learn if a given ego-network is correctly ordered in time. We do this by training on ego-networks datasets where half of the ego-networks are correctly ordered. Our method requires only a few crucial combinatorial features in order to perform significantly well, and generalizes across our datasets without any changes in learning parameters. We also find that temporal features are the most significant for prediction. On all datasets, our model significantly outperforms baselines derived from random ordering.

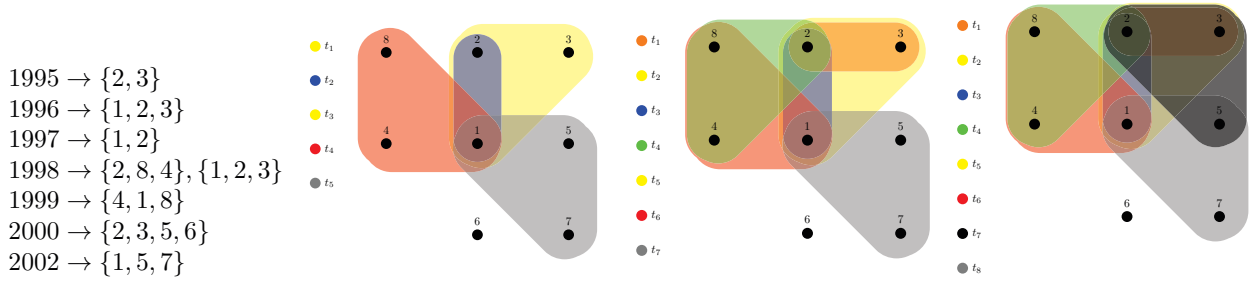
Next, we define a hill climbing algorithm that is given our learning method and a shuffled ego-network as input. Each iteration of the algorithm swaps every pair of hyperedges and applies the supervised method to each ordering. Out of all the orderings that have increased likelihood of being a correct ordering, the algorithm chooses one at random. This then becomes the input ordering for the next iteration. This process is repeated until no swaps improve likelihood, in which case the algorithm saves the current ordering, and repeats the above process on another randomized ordering. After a certain amount of further attempts, the algorithm then returns the stored ordering with highest likelihood. Experimental results show that our model significantly outperforms multiple baselines on each dataset. Thus, the model is able to accurately reconstruct hypergraph ego-networks across domains.

II. Basic Definitions

Hypergraphs: *Hypergraphs* are generalizations of traditional pairwise graphs where an edge can connect any number of nodes, whereas an edge in a graph only connect two nodes together [7]. We refer to these edges as hyperedges. More formally, a hypergraph $G = (V, E)$ consists of a set of *nodes* V and a set of subsets of V known as E (the set of *hyperedges*). Each hyperedge $e \in E$ contains a number of nodes $|e|$, which we refer to as its *size*. Each node $v \in V$ can belong to multiple hyperedges, and the number of hyperedges a particular node belong to is known as the *degree* of the node.
Simplex: In this paper, we will be looking at temporal hypergraphs where each hyperedge is associated with a particular timestamp. From this point on, we will be referring to these hyperedges as *simplices*, and use S to denote the set of all simplices. We define the *size* of a simplex to be the number of nodes in the simplex. We also define a simplex to be *trivial* if the simplex is of size less than 2, and is *non-trivial* otherwise.
Hypergraph Ego-network: The *hypergraph ego-network* E of a node u is the set of simplices that represent the interactions among u's neighbors. We refer to u as the *user node* or the *ego*, and we refer to u's neighbors as *alters*. For the sake of brevity, we will mostly refer to hypergraph ego-networks from this point on as just *ego-networks*. We define the *length* of an ego-network to be the number of simplices in it. Because there are multiple natural ways to represent the interactions among a node u's neighbors — for example, whether all interactions must involve u or whether some can take place only among the neighbors — we propose definitions for three natural, distinct types of hypergraph ego-networks: *star*, *radial*, and *contracted ego-networks*. We define them here, and give an example of the three types in Figure 1.
Star Ego-network: If S is the set of all simplices, and u is the user node, the *star ego-network* $T(u)$ is defined as follows:

$$T(u) = \{s : (u \in s)\}, \forall s \in S$$

In other words, the star ego-network is composed of all simplices that include the user node. We refer to simplices of this type as *user simplices*. This is the simplest type



(a) Simplex Dataset (b) Star (c) Radial (d) Contracted

Fig. 1. **Example of three different hypergraph ego-networks of user node 1 in the domain of co-authorship. (a) Higher-order network consisting of eight timestamp papers modelled as simplices on eight authors (nodes). Note that two papers were published in the year 1998, which we will add one by one in the horizontal order they appear in the dataset. (b) Star ego-network of user node 1. The star ego-network consists of all simplices in our dataset that include the user node. Therefore, in ordinal time, the star ego-network of 1 will be: $t_1 \to \{1,2,3\}$, $t_2 \to \{1,2\}$, $t_3 \to \{1,2,3\}$, $t_4 \to \{4,1,8\}$, $t_5 \to \{1,5,7\}$. (c) Radial ego-network of 1. Since the radial ego-network additionally includes all simplices from the dataset where all nodes are alters, the two simplices $\{2,3\}$ and $\{2,8,4\}$ are added into the ego-network. As a result, the radial ego-network is: $t_1 \to \{2,3\}$, $t_2 \to \{1,2,3\}$, $t_3 \to \{1,2\}$, $t_4 \to \{2,8,4\}$, $t_5 \to \{1,2,3\}$, $t_6 \to \{4,1,8\}$, $t_7 \to \{1,5,7\}$. (d) Contracted ego-network of 1. The contracted ego-network consists of the intersection of every simplex in the dataset with the set $\{A(u) \cup u\}$, where u is the user node (in this case 1) and $A(u)$ is the set of all alters of u. In our example, this means that we will additionally take the intersection of the simplex $\{2,3,5,6\}$ and $\{A(u) \cup u\}$. As the node 6 is not an alter of the user node 1, we will add the simplex $\{2,3,5\}$ instead, removing 6 from the set. Therefore, the contracted ego-network is: $t_1 \to \{2,3\}$, $t_2 \to \{1,2,3\}$, $t_3 \to \{1,2\}$, $t_4 \to \{2,8,4\}$, $t_5 \to \{1,2,3\}$, $t_6 \to \{4,1,8\}$, $t_7 \to \{2,3,5\}$, $t_8 \to \{1,5,7\}$.**

of ego-network we will be working with in this paper, as it does not model any interactions between the alter nodes except for those that involve the user node u. Note that the richness of this structure really manifests itself only in structures with hyperedges on at least three nodes; it is not nearly as interesting in the dyadic case, where pairwise edges only connect a user node to its neighbors. Star hypergraph ego-networks are still able to model interactions among alters, provided those interactions take place within a user simplex.

Radial Ego-network: If S is the set of all simplices, u is the user node, and $A(u)$ is the set of all alters of u, the *radial ego-network* $R(u)$ is defined as follows:

$$R(u) = \{s : s \subseteq \{A(u) \cup \{u\}\}\}, \forall s \in S$$

The radial ego-network is composed of all simplices where every node in the simplex is either the user node or an alter of the user node. With the radial ego-network, we are able to include interactions among alters, as some of these simplices may not include the user node at all. These simplices that are composed entirely of alters we define as *alter simplices*.

Contracted Ego-network: If S is the set of all simplices, u is the user node, and $A(u)$ is the set of all alters, the *contracted ego-network* $C(u)$ is defined as follow:

$$C(u) = \{s \cap \{A(u) \cup \{u\}\}\}, \forall s \in S$$

The contracted ego-network is the intersection of each simplex in S with the set $\{A(u) \cup \{u\}\}$. The contracted ego-network captures more alter simplices than the radial ego-network, as it includes subsets of simplices that interact with nodes outside of u's ego-network.

The above definitions lead to a key structural property of the three ego-network types. For some user node u:

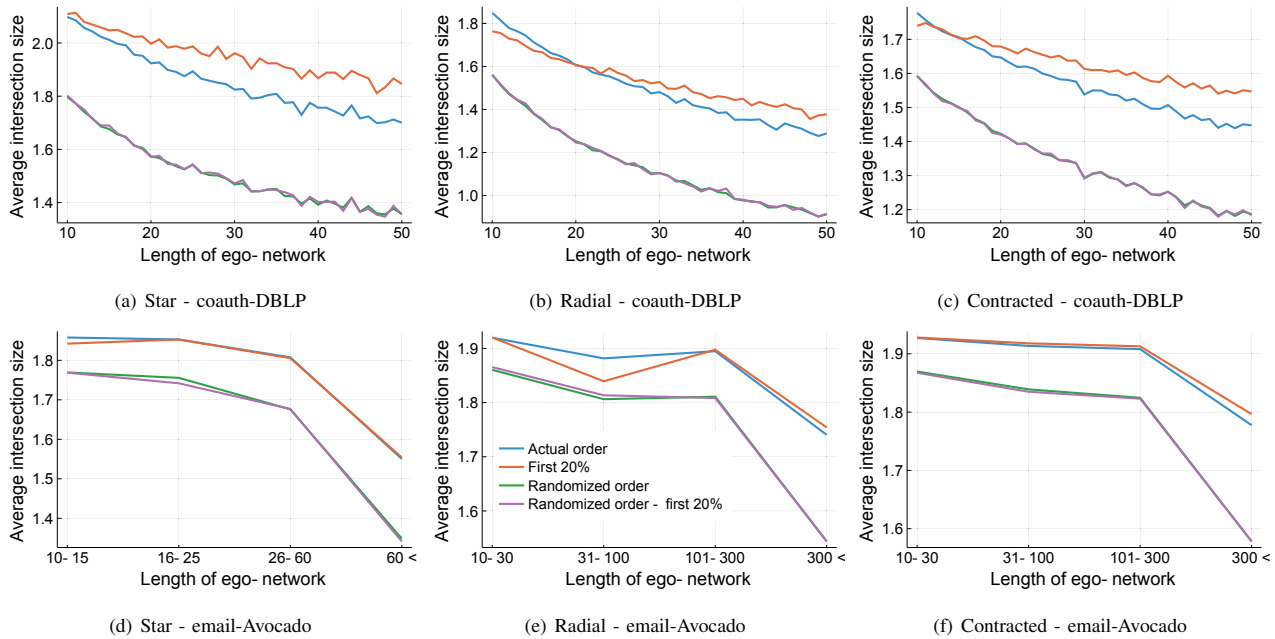
$$T(u) \subseteq R(u) \subseteq C(u)$$

It is important to note the scope of the information available to an ego-network. Each ego-network only has access to the local information of each of the alters, which includes all the interactions of that alter in the ego-network. The ego-network does not have access to any of their alter's interactions that take place outside of the ego-network. In other words, when we eventually apply our supervised learning model to an ego-network, the model will not have access to information such as the structure of every alter's individual ego-network. We leave the incorporation of such information for future work.

III. Datasets

We proceed by describing the three datasets used in this paper. Each dataset consists of a set of timestamped simplices, and from this set, we are able to build star, radial, and contracted ego-networks for each user node. We collected data from three different domains, emails, online threads, and publications. The three datasets used are useful to study for this problem as they contain entire lifetimes of user activity. For example, in email-Avocado, every email that a given ego has ever been a recipient of or has sent has been recorded from the first to the last. This allows us to accurately analyze the temporal structure of each user's ego-network. Our code uses the ScHoLP library to extract ego-network information from higher-order datasets.

- **coauth-DBLP: 1569217 nodes**. Each node is a researcher, and each simplex corresponds to a set of authors on a publication. The timestamp of each simplex corresponds to the year the paper was published.
- **email-Avocado: 28244 nodes**. Each node is an email address, and each simplex corresponds to a set of recipients and the sender of an email. The timestamp of each simplex corresponds to the second the email was sent.
- **threads-ask-ubuntu: 33853 nodes**. Each node is a user, and each simplex corresponds to a set of users answering



(a) Star - coauth-DBLP (b) Radial - coauth-DBLP (c) Contracted - coauth-DBLP

(d) Star - email-Avocado (e) Radial - email-Avocado (f) Contracted - email-Avocado

Fig. 2. **Average intersection size. From the plots, we observe that adjacent simplices in correctly ordered ego-networks across all types and domains intersect each other far more an average that adjacent simplices in randomly shuffled ego-networks. This key finding will be significant for our prediction task, acting as a key differentiator between correctly ordered and randomly ordered ego-networks.**

a question on a forum. The timestamp of each simplex corresponds to the second the question was posted.

It is important to note that for a given ego-network in coauth-DBLP, multiple simplices can exist at the same timestamp. This does not happen in email-Avocado and threads-ask-ubuntu because of the granularity of data collection mentioned above. We deal with this in two ways. Firstly, we convert the timestamps for each simplex from real time to *ordinal time*, which is a consecutive ordering of the simplices as $1, 2, 3...$ in order of their arrival times. Then, for each set of simplices that have the same ordinal time label, we add each simplex iteratively into the ego-network as illustrated in Figure 1.

In order to capture enough higher-order interaction for our prediction task, we will only examine ego-networks with a length of at least 20 or greater for coauth-DBLP and 10 or greater for email-Avocado and threads-ask-ubuntu. We will only be analyzing non-trivial simplices in each ego-network. Additionally, simplices can never be removed from an ego-network. We also will ignore all ego-networks where the majority of simplices are identical and ego-networks that have less than 10 alters (except for email-Avocado), as any model that attempts to sort these ego-networks will do well.

IV. Key Observations And Measures

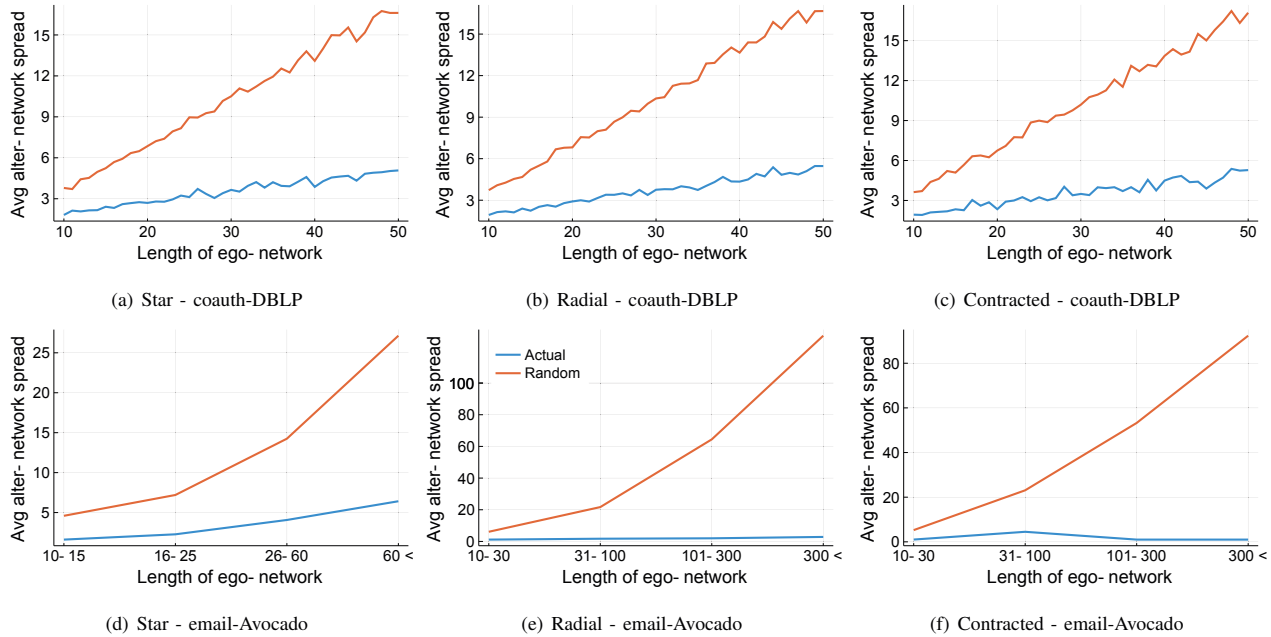
In this section, we examine a sequence of key observations that we identify about the temporal growth of ego-networks in the domains we analyze. First, we discuss an underlying locality principle, that the same nodes tend to reappear in neighboring simplices in the temporal order. We next observe that each ego-network can be thought of as a union of star-shaped sub-networks, one for each alter, and that analyzing the temporal structure these sub-networks can give us insights into the overall evolution of the ego-network. We demonstrate a relationship between the time at which a user node arrives in their own radial and contracted ego-network and the size of their ego-network. Then, we highlight the typical placement of high-degree nodes in ego-networks. Finally, we discuss how nodes that have never been seen in an ego-network tend to enter the ego-network at a near-constant rate. For the sake of brevity, in this section we will only analyze ego-networks found in coauth-DBLP and email-Avocado (the observations we will discuss still hold for threads-ask-ubuntu), but we will use all three datasets in our prediction task.

A. Intersection Size

We first observe that contiguous simplices in an ego-network tend to have relatively large intersections with each other, suggesting that temporally adjacent higher-order interactions have high similarity. In other words, the same nodes tend to appear in neighboring simplices. We define the average intersection size of a given ego-network E with m simplices $\{s_1, ..., s_m\}$ as

$$I(E) = \frac{\sum_{i=1}^{m-1} |s_i \cap s_{i+1}|}{m-1}$$

Figure 2 shows the average intersection size of all star, radial, and contracted ego-networks against the size of the ego-network for each dataset. We also plot the average intersection



(a) Star - coauth-DBLP (b) Radial - coauth-DBLP (c) Contracted - coauth-DBLP

(d) Star - email-Avocado (e) Radial - email-Avocado (f) Contracted - email-Avocado

Fig. 3. **Average alter-network spread for all three ego-networks in coauth-DBLP and email-Avocado. We find that, on average, simplices in alter-networks are temporally localized, yet this is not the case if we randomly order the ego-network that each alter-network is a part of. This finding supports the importance of the alter-network spread feature used in our binary classification method.**

size in the first 20% of each ego-network and the average intersection size of randomly shuffled ego-networks.

From Figure 2, we first observe that adjacent simplices in randomly shuffled ego-networks do not intersect as much as correctly ordered ego-networks. This remains true across all ego-network types in both coauth-DBLP and email-Avocado. Therefore, higher-order interactions in real-world ego-networks tend to be similar to temporally adjacent interactions. Secondly, we find that in coauth-DBLP, the first few simplices in both correctly-ordered star ego-networks and large radial and contracted ego-networks are highly similar, with large intersection size across many pairs of adjacent simplices. However, this is not the case with randomly shuffled ego-networks or with small radial and contracted ego-networks, where there is no significant difference between the average intersection size of the first 20% of simplicies of a shuffled ego-network compared to the entire ego-network. These observations are consistent with prior work studying sequences of higher-order interactions [6].

A key idea here is that if we observe the average intersection size between adjacent simplices in a traditional pairwise graph, it becomes difficult to assess how similar two adjacent interactions involving more than two nodes are, as both interactions must be broken down into several pairs of nodes. By representing these interactions with hypergraphs, we are able to better capture ideas of intersecting edges.

Prior work studying the evolution of dyadic ego-networks finds that ego-networks tend to expand a great deal towards the beginning of their lifetime [1]. Our findings show that in the higher-order case, the opposite is true for coauth-DBLP, where the first few interactions in a hypergraph ego-network's lifetime are highly similar, with many of the same nodes reappearing in contiguous simplices. These results are particularly significant as we also find that the average size of an incoming simplex for an ego-network increases over time (the details of which we have omitted), implying that despite the fact that these earlier simplices are relatively small, they still intersect a great deal. We conclude that both observations mentioned above are important concepts for any model that attempts to capture the temporal structure of ego-networks.

B. Alter-networks

We now analyze the temporal spacing between similar simplices in ego-networks and extend some of the ideas previously seen when discussing intersection density. Every alter a of a user u has their own star-shaped collection of all simplices in u's ego-network $E(u)$ that include a. We define this set of simplices as a's *alter-network*, $Alt(a, E(u))$. The alter-network of a can also be thought of as the intersection of a's star ego-network with u's ego-network:

$$Alt(a, E(u)) = E(u) \cap T(a) = \{s : (a \in s)\}, \forall s \in S$$

where S is the set of all simplices in $E(u)$. Every ego-network of any type can be thought of as a union of its alter's interactions. Stated differently, for the ego-network $E(u)$ of a user node u and the set of alters $A(u) = a_1, ..., a_n$:

$$E(u) = \bigcup_{i=1}^{n} Alt(a_i, E(u))$$

Knowledge of the temporal structure of the alter-networks in a given ego-network proves to be very useful when predicting the evolution of the ego-network. We define the *alter-network spread* of a given alter-network to be the average ordinal time difference between two adjacent simplices in the alter-network. Stated differently, let $t(s)$ be the timestamp at which simplex s arrives in the ego-network $E(u)$ of some user u. Then, the alter-network spread of an alter-network of size m composed of simplices $\{s_1, ..., s_m\}$ is:

$$\frac{t(s_m) - t(s_1)}{m - 1}$$

For each alter-network in coauth-DBLP and email-Avocado, we measure the average alter-network spread and plot the average value against the size of the ego-network, as shown in Figure 3. We compare against a random model that calculates the average alter-network spread for a shuffled alter-network. From this, we observe that the spacing between neighboring simplices in a given alter-network is relatively small. This is not the case for randomized alter-networks, which have a higher average alter-network spread. This proves to be a vital feature for our learning model, as the existence of dense alter-networks for a given ego-network is strong evidence that the ego-network is correctly sorted.

The intuition behind the alter-network spread is that it captures the period of time in which a user is interacting with a particular alter, which above we confirm to be temporally local. However, an interesting question to ask is whether or not the rate at which users' interact with their alters is constant. Using the example of co-authorship, it is intuitive to think that the first and final few papers a user co-authors with a frequent collaborator are less temporally consistent than the papers they publish in between. It is natural to conjecture that this middle section is the most dense, and therefore would have the smallest alter-network spread. For large alter-networks, which we will define to be of size at least 10, we also measure spread at different sections of the alter-network. To do this, we split alter-networks into thirds, and calculated the average alter-network spread in each third. We observe that the middle third of large alter-networks are in fact much denser on average than the beginning and final third. Thus, on large alter-networks, our model should capture this pattern.

When comparing average alter-network spread to average intersection size, these two features are in no sense identical, but they are certainly related. We would expect to find an ego-network with high average intersection size to have low alter-network spread, as intersecting adjacent simplices implies adjacent simplices in the alter-networks of alters that are common across both simplices. This relationship can be seen by comparing Figures 2 and 3. As the length of ego-networks increase, the average intersection size decreases while the average alter-network spread increases.

Finally, we observe that high-degree alters on average tend to appear earlier on in ego-networks than low-degree alters (we omit the details in this paper). In the context of co-authorships, this observations means that if w is one of some user node u's most frequent co-authors, then edges containing w are more likely to appear earlier than later. This finding is intuitive and especially important for models attempting to solve the problem of temporal reconstruction, as the first few simplices of any correctly ordered ego-network are likely to contain high-degree nodes.

C. An Anthropic Principle for Ego-Networks

In an ego-network, the user node u occupies a privileged position at the center, but many of the higher-order interactions in the radial and contracted ego-networks do not involve u, and in fact might have pre-dated the first interaction that does involve u. Thus, the ego-network is defined by u, but parts of it pre-dated u's arrival into the system. We refer to this tension as an *anthropic principle for ego-networks*, by analogy with the collection of anthropic principles in physics and philosophy [8] that involve a similar duality: that your position as an observer is privileged, but that the system you are observing existed before you were there to observe it.

We now consider these ideas in more detail. As an example in terms of co-authorship, two alters in some user node u's ego-network may have written papers together before ever meeting u (Figure 1(c)). This is only the case for radial and contracted ego-networks, as the star ego-network does not include interactions among alters that omit u. In a star ego-network, the user node will always arrive at the very first timestamp. In contrast, the radial and contracted ego-network of a particular user node may have begun a long time before the user node ever entered it.

An interesting question to then consider is: across real-world radial and contracted ego-networks, at what timestamp does the user node typically arrive in their own ego-network? For coauth-DBLP, we plot the average timestamp at which a user node arrived in their correctly ordered and randomly shuffled radial and contracted ego-networks against the size of the ego-network, as shown in Figure 4. Here, we observe a linear relationship between the timestamp at which a user node arrives and the size of their contracted ego-network. User nodes in randomly shuffled contracted ego-networks tend to arrive far earlier, implying that alter simplices tend to dominate the earlier sections of real-world contracted ego-networks.

However, we observe a sublinear relationship for radial ego-networks, as the user node usually enters their ego-network around or before the fifth timestamp. An interesting observation is that the user node tends to arrive almost at the very first simplex in expectation in randomly shuffled radial ego-networks, implying not only that radial ego-networks are largely made up of user simplices, but also that the fraction of simplices in radial ego-network that include the user node is near-constant as the size of the ego-network increases. We conclude that there is an element of predictability when observing the time at which user nodes enter their radial and contracted ego-networks in co-authorship datasets, and this arrival time becomes a useful feature for our learning model.

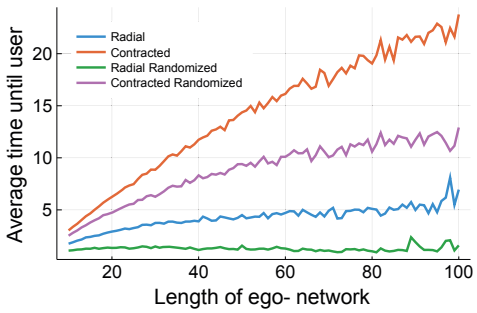


Fig. 4. **For coauth-DBLP, the user node tends to arrive into its radial ego-network before or around the fifth timestamp on average. For contracted ego-networks, there is a linear relationship between the length of the ego-network and the timestamp at which the user node arrives in the ego-network. This becomes a key feature for our classifier on radial and contracted ego-networks in coauth-DBLP.**

D. Novelty Rate

We now measure the rate at which novel nodes enter ego-networks in order to understand when and how often new nodes typically arrive. We define the *novelty* of a simplex to be the number of nodes in the simplex that have never been contained in any previous simplices. For example, in Figure 1(a), simplex $t_5 \rightarrow \{1, 5, 7\}$ has a novelty of 2 since the nodes 5 and 7 do not appear in simplices $t_1, ..., t_4$.

Figure 5 reports the average novelty at each timestamp for ego-networks of different types and sizes. For each plot, we omit the average novelty of the first simplex in each ego-network. We do this because this value will have novelty equal to the average simplex size of the first simplex, since each node in the simplex is novel. From Figure 5, we observe that for star and radial ego-networks, the number of novel nodes that enter an ego-network at each timestamp slowly and gradually decrease, until a certain point where novel nodes begin to enter the ego-network again. This agrees with our earlier finding that low-degree nodes tend to enter ego-networks later on in the ego-network's lifetime. It is also interesting to note that for star ego-networks, the average novelty of a simplex is always above 1 regardless of the ego-network's size, implying that the average incoming simplex has at least a single novel node. For contracted ego-networks, the rate at which novel nodes enter the ego-network is always decreasing but very slowly. These observations imply that co-authorship ego-networks grow at a near-constant rate for the majority of their lifetimes.

V. Prediction Task

We now discuss the problem of temporal reconstruction of hypergraph ego-networks and detail our solution. We are guided by two related goals in formulating and studying this problem: first, for applications where we might have the structure of a hypergraph ego-network but lack information about how it evolved; and second, as a way to evaluate and gain insight into the mechanisms of ego-network evolution in hypergraphs, and how the findings in the previous section might be used for their analysis. Our solution can be broken down into two tasks: first, we will describe the supervised learning task of classifying a given ego-network as correctly sorted or randomly shuffled, and after we will detail a local search algorithm used to iteratively sort a shuffled ego-network.

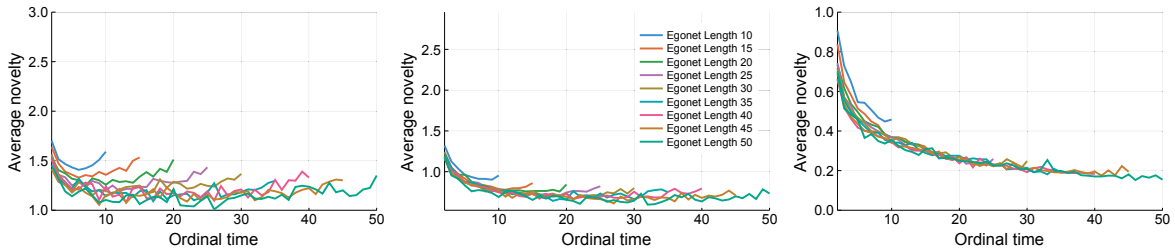
A. Learning Task

We define a supervised binary classification task, where we predict whether a given ego-network is correctly or randomly ordered. Half of our training examples will be correctly sorted ego-networks, and the other half will be shuffled. Therefore, random guessing achieves an accuracy of 50%. The lengths of radial and contracted ego-networks for email-Avocado and threads-ask-ubuntu are usually large (many of which have length greater than 100), and therefore we will only train our model on star ego-networks. For each ego-network type for coauth-DBLP, we will train and test our model on a sample of 50000 ego-networks of length 20 to 50. For email-Avocado and threads-ask-ubuntu we will sample approximately 1000 and 1800 star ego-networks of length at least 10 respectively.

We will construct a set of features from each ego-network. For star ego-networks, these will include features based on the main findings of the previous section: intersection density (the average intersection size of an ego-network divided by average simplex size of the ego-network) and average alter-network spread. We will also use the length of the ego-network, the number of future simplices the first simplex in the ego-network is a subset of, and the number of prior simplices the last simplex is a superset of. According to [6], early sets in a sequence of sets tend to be subsets of future sets, and later sets tend to be supersets of prior sets. For radial and contracted ego-networks, we will additionally include the timestamp at which the user node entered the ego-network.

We trained several classification models on our data, namely deep neural networks, logistic regression, random forests, and naive Bayes. As the neural network performed the best, we will report accuracies from the neural network. We trained three different and fully connected neural networks, one for each dataset. For coauth-DBLP and threads-ask-ubuntu, our network had two hidden layers of sizes 100 and 24, and the network for email-Avocado had two layers of sizes 12 and 6. Each network had an initial learning rate of 0.001 with minibatch size of 200. For each model, we performed 10-fold cross validation and will report the mean classification accuracy. All models were trained using the sci-kit learn library. Training and testing each model took less than a minute using a Lambda 72-core GPU server (1536 GB RAM).

Table I reports the classification accuracy of our models. In general, our models perform well across all datasets and ego-network types. Due to the limited number of ego-networks in email-Avocado and threads-ask-ubuntu to sample from, models trained on these datasets report high standard deviation values. When comparing our method to a random guessing baseline, we significantly outperform this baseline on all datasets and ego-network types.



(a) Star Ego-network (b) Radial Ego-network (c) Contracted Ego-network

Fig. 5. **Average novelty across the three ego-network types.**

TABLE I
CLASSIFIER RESULTS

	Star	Radial	Contracted
coauth-DBLP	0.93 ± 0.01	0.91 ± 0.01	0.85 ± 0.01
email-Avocado	0.84 ± 0.09	N/A	N/A
threads-ask-ubuntu	0.72 ± 0.05	N/A	N/A

We now quantitatively analyze feature importance across datasets. For all datasets, we find that alter-network spread is by far the most significant feature, achieving accuracies of 89%, 84%, and 75% when used as a single feature across star, radial, and contracted ego-networks respectively in coauth-DBLP. Intersection density is also a significant feature but less powerful. Finally, we also decided to test the prediction model derived from star ego-networks in coauth-DBLP on star ego-networks in email-Avocado and threads-ask-ubuntu. Our model performed better than random guessing, achieving a prediction accuracy of 83% on email-Avocado and 72% on threads-ask-ubuntu. This implies that our model has found principles that apply generally across different domains.

B. Reconstruction Algorithm

We now get to our temporal reconstruction algorithm. In brief, we define a hill climbing algorithm that iteratively swaps all pairs of simplices, applies our supervised method to each resultant ordering, and selects one of the orderings that is more likely to be correctly sorted than the current ordering. This is repeated until no pairwise swaps improve likelihood, in which case this process repeats itself on another randomized ordering. Once we have a selection of improved orderings, the algorithm selects the ordering with highest likelihood.

The accuracy of our model is measured by how many pairs of simplices in the predicted order are out of order with respect to the true answer. That is, of all $\binom{m}{2}$ pairs of simplices (e_i, e_j), what fraction of these pairs appear in the correct time order in the predicted ordering? If we were to guess at random, we would expect to find half of all pairs to be in order; so if a predicted ordering had significantly more than half of its $\binom{m}{2}$ pairs in order, this suggests that the prediction algorithm is capturing significant patterns regarding the temporal structure of hypergraph ego-networks.

TABLE II
RECONSTRUCTION ALGORITHM RESULTS. WE SAMPLE 100 EGO-NETWORKS FOR EACH DATASET AND EGO-NETWORK TYPE, AND REPORT THE OUT OF ORDER ACCURACY FOR EACH. WE SET T = 10.

	Star	Radial	Contracted
coauth-DBLP	0.65 ± 0.08	0.56 ± 0.05	0.65 ± 0.08
email-Avocado	0.63 ± 0.11	N/A	N/A
threads-ask-ubuntu	0.70 ± 0.07	N/A	N/A

We provide the pseudocode of our temporal reconstruction algorithm that is described in greater detail in Algorithm 1. The algorithm only requires three parameters: *ego-network* π_0, *supervised model* M, and *number of total iterations* T. It then repeats the following steps T times, using a counter variable i initialized to 1 in order to iterate the algorithm until T.

1) Set $\pi_0(i)$ to a randomly shuffled version of π_0.
2) Set $\pi := \pi_0(i)$.
3) Swap every pair of simplices in π and apply M to each resultant ego-network.
4) For each resultant ordering, if the probability that the resultant ordering is correctly sorted is higher than that of the previous ordering, we will save this ordering.
5) For all resultant orderings with improved likelihood, select one at random, and set it to π.
6) Repeat steps 2-5 until no pair of swapped simplices will improve likelihood, in which case we save the current ordering, increase i by one, and go to step 1.

This process is repeated T times until we have a set of improved orderings, at which point we select the ordering with the highest probability of being correctly ordered.

Table II shows the results of our algorithm on each dataset on a sample of 100 simplices of length between 10 and 15 for email-Avocado, of length 10 for threads-ask-ubuntu, and 20 for coauth-DBLP. As stated previously, a random guessing baseline would expect to find half of all pairs to be out of order, and as a result would achieve an accuracy of 50%. Another naive but intuitive baseline that uses a more principled heuristic than random ordering would be to sort all ego-networks by increasing simplex size. This baseline achieves results of approximately 50% across all datasets and ego-network types. Our algorithm's results show a non-trivial improvement over both baselines. We attribute this to the set

of powerful features used by our learning method, which take advantage of key ideas such as alter-networks, the intersection size between adjacent simplices, and the time at which user nodes typically arrive in radial and contracted ego-networks.

C. Theoretical Bounds

It is interesting to ask whether any theoretical guarantees can be made for the quality of hypergraph orderings found by this type of local search. In general, it is difficult to say anything formal about local search using the trained model, given that we do not have a succinct description of the model. However, given that maximizing average intersection size (among consecutive simplices in order) serves as an effective heuristic for the ordering problem, we can achieve some insight into the power of local search by proving an analogous result for local search to maximize average intersection size.

We'll quantify the performance by two parameters of the instance: c, equal to the maximum size of any simplex in the instance; and d, equal to the maximum number of simplices that any one node occurs in. We also preprocess the instance by deleting all elements that occur in at most one simplex, since none of these elements can contribute to the average intersection size. We can then delete any simplices that become empty as a result. In what follows, we will therefore assume without loss of generality that all simplices are non-empty, and each element occurs in at least two simplices.

We consider an arbitrary local search algorithm that swaps pairs of simplices as long as the swap strictly increases the objective function of average intersection size. A local optimum is an ordering at the end of this process, when no swap strictly increases the objective. We now prove an approximation guarantee for local search.

Theorem 1: For any local optimum, the average intersection size is at least $1/(2c^2d)$ times the average intersection of the globally optimal solution.

Proof. Since each simplex has size $\leq c$, the optimal solution (with order π) has average intersection size at most

$$\frac{1}{m-1}\sum_{i=1}^{m-1}|s_{\pi(i)} \cap s_{\pi(i+1)}| \leq \frac{1}{m-1}\sum_{i=1}^{m-1} c = c.$$

Now consider a locally optimal solution; assume for notational simplicity that it orders the simplices as $s_1, s_2, \ldots, s_m$. We partition the indices $1 \leq i \leq m-1$ into two sets: the set A consisting of indices i such that $|s_{i-1} \cap s_i|$ and $|s_i \cap s_{i+1}|$ are both 0, and the set B consisting of all other indices i.

Next, let B' be the set of all indices i such that $|s_i \cap s_{i+1}| > 0$. We have $B' \subseteq B$; also, if $i \in B$, then at least one of $i-1$ or i is in B', from which it follows that $|B| \leq 2|B'|$. Since each index $i \in B'$ contributes at least 1 to the total intersection size, the average intersection size in our locally optimal solution is at least $\frac{1}{m-1}|B'| \geq \frac{1}{2(m-1)}|B|$.

Now we come to the key step. For each $i \in A$, consider an arbitrary $u_j \in s_{i+1}$. The simplex s_i does not contain u_j (since otherwise we would have $|s_i \cap s_{i+1}| > 0$, contradicting $i \in A$). But u_j occurs in at least two simplices; let s_h be another simplex in which it occurs. We cannot have $h \in A$, since then swapping s_h and s_i would strictly increase the average intersection size, contradicting local optimality. Thus $h \in B$.

We now charge index $i \in A$ to index $h \in B$. s_h has $\leq c$ elements, and each can be charged $\leq d-1$ times, so h can be charged $\leq c(d-1)$ times; hence $|A| \leq c(d-1)|B|$. We also have $|A| + |B| = m-1$, so $|B| \geq \frac{m-1}{1+c(d-1)} \geq \frac{m-1}{cd}$.

Thus the locally maximum solution has average intersection size $\geq \frac{|B|}{2(m-1)} \geq \frac{1}{2cd}$ while the optimum solution has total intersection size at most c. The optimum solution therefore has average intersection size at most $2c^2d$ times that of the locally optimal solution, completing the proof. ■

Algorithm 1: Temporal Reconstruction hill climbing algorithm for sorting unordered hypergraph ego-networks

Input: ego-network π_0, supervised model M, and number of total iterations T
Output: a predicted ordering $\max_{j=1}^{T} L(j)$

```
L ← ∅
i ← 1
while i ≤ T do
    π_0(i) ← random(π_0)
    π ← π_0(i)
    do
        U ← all orders π' obtained by one swap from
          π where M(π') > M(π)
        π ← random choice from U
    while U ≠ ∅;
    L(i) ← π
    i ← i + 1
end
```

VI. Related Work

A large volume of past work has been done on the evolution of global dyadic graphs [14]–[16]. The evolution of dynamic systems that model higher-order interactions using hypergraphs has also been previously investigated [4], [5], [9], [11]. [4] studies the temporal evolution of global hypergraph datasets in the context of simplicial closures and link prediction, and also looks at predicting system domain using higher-order ego-networks. In contrast, our paper attempts to understand the temporal evolution of local hypergraph ego-networks, extracting higher-order ego-network features to predict evolution rather than system domain.

[4] also mentions the projected graph, the encoding of higher-order information as a traditional dyadic graph. Significant information loss occurs when hypergraphs that model higher-order interactions are converted into the projected graph. Thus, we do not use the projected graph in our paper to analyze higher-order networks. [11] examines temporal properties of global hypergraphs in order to realistically generate hypergraphs. In contrast, our paper instead focuses on modelling local hypergraph structure.

Dyadic ego-networks have been used to model local interactions across various fields, including social and co-authorship

ego-networks [2]. Temporal dyadic ego-network evolution has also been frequently studied [1], [3]. There is also past work on using machine learning methods to understand the structural patterns in pairwise ego-networks [17]. In [1], dyadic ego-networks from social media datasets are analyzed in order to describe common patterns found regarding their growth. [1] also finds that dyadic ego-networks tend to rapidly expand at the beginning of their lifetimes, adding many nodes in a short period of time. In our paper, we show the opposite to be true in the co-authorship case, where user nodes typically strengthen their ties with early alters rather than finding new nodes.

Benson et al analyze repeat behavior in sequences of sets [6] via a formalism they term the Correlated Repeated Unions (CRU) model. In our context, we could model a hypergraph ego-network as a sequence of sets, with each set being a simplex. Both our paper and [6] identify a recency bias, but we consider different questions, with our paper focusing on orderings rather than set composition.

VII. Discussion

In this paper, we have proposed the study of hypergraph ego-networks, a structure that can be used to model higher-order interactions involving a single node and its alters. We define three ego-network types: star, radial, and contracted ego-networks, each modeling different higher-order interactions surrounding the user node. We have examined higher-order interactions across three domains: co-authorship (via a collection of publications *coauth-DBLP*), communication (via a collection of emails *email-Avocado*), and online threads (via a collection of users participating in a thread *threads-ask-ubuntu*). The coauth-DBLP and threads-ask-ubuntu datasets can be found at: https://www.cs.cornell.edu/~arb/data/, and email-Avocado can be found at https://catalog.ldc.upenn.edu/LDC2015T03. Our code can be found at https://github.com/Cazamere/hypergraph-assembly.

Our work introduces several key observations that subsequently inform a set of prediction algorithms to accurately reconstruct hypergraph ego-networks. The most powerful of these is the *alter-network spread*, the average temporal distance between adjacent simplices in each alter-network of an ego-network. We find that alter-networks possess strong temporal locality, as they tend to occupy very defined sets of proximate timestamps in the ego-network they belong to. This notion of temporal locality within a collection of sets suggests interesting connections to other contexts where similar locality phenomena arise. Perhaps most directly, it would be interesting to consider connections to the well-known principle of locality of reference in computer systems, which is also based on the idea that occurrences of particular elements in long access patterns are clustered in time [18]. And as a slightly more distant but intriguing connection, notions of locality in set systems form the underpinning for fundamental combinatorial questions about high-dimensional polyhedra, where the ordering of vertices by breadth-first search obeys a form of locality with respect to the facets that contain them [10].

Interestingly, we also find that radial ego-networks in coauth-DBLP contain a large and constant fraction of user simplices, regardless of size, as user nodes arrived into their radial ego-networks around or slightly before the fifth timestamp. However, this is not the case for contracted ego-networks, where an approximately linear relationship is observed between the time a user node arrives into their ego-network and the size of the ego-network.

Finally, we also propose the temporal reconstruction of hypergraph ego-networks as a benchmark problem for models that aim to predict the local temporal structure of hypergraphs. As a solution, we propose a supervised deep learning method to learn if a given hypergraph ego-network is correctly ordered in time. Next, we define a hill climbing algorithm that is given our learning method and a shuffled ego-network. Our model significantly outperforms several baselines by incorporating various structural patterns found in hypergraph ego-networks across multiple domains such as intersection density and alter-network spread. We envision that our model may act as a foundation for future work that aims to further understand the local, temporal structure of higher-order interactions.

References

[1] L. M. Aiello and N. Barbieri. Evolution of ego-networks in social media with link recommendations. In *ACM WSDM Conf.*, 2017.

[2] V. Arnaboldi, M. Conti, A. Passarella, and R. I. Dunbar. Online social networks and information diffusion: The role of ego networks. *Online Social Networks and Media*, 1:44–55, 2017.

[3] V. Arnaboldi, R. I. Dunbar, A. Passarella, and M. Conti. Analysis of co-authorship ego networks. In *Intl. Conf. Network Science*, 2016.

[4] A. R. Benson, R. Abebe, M. T. Schaub, A. Jadbabaie, and J. Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48):E11221–E11230, 2018.

[5] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

[6] A. R. Benson, R. Kumar, and A. Tomkins. Sequences of sets. In *Proc. SIGKDD. Conf. Knowledge Discovery & Data Mining*, 2018.

[7] C. Berge. *Hypergraphs: combinatorics of finite sets*. Elsevier, 1984.

[8] B. Carter. Large number coincidences and the anthropic principle in cosmology. In *Confrontation of cosmological theories with observational data*. 1974.

[9] M. T. Do, S.-e. Yoon, B. Hooi, and K. Shin. Structural patterns and generative models of real-world hypergraphs. In *Proc. ACM SIGKDD Intl. Conf. on Knowledge Discovery & Data Mining*, 2020.

[10] F. Eisenbrand, N. Hähnle, and T. Rothvoß. Diameter of polyhedra: limits of abstraction. In *Proc. ACM Symp. Comp. Geom.*, 2009.

[11] Y. Kook, J. Ko, and K. Shin. Evolution of real-world hypergraphs: Patterns and models without oracles. *arXiv preprint 2008.12729*, 2020.

[12] G. Lee, M. Choe, and K. Shin. How do hyperedges overlap in real-world hypergraphs?-patterns, measures, and generators. In *Proceedings of the Web Conference 2021*, pages 3396–3407, 2021.

[13] G. Lee, J. Ko, and K. Shin. Hypergraph motifs: Concepts, algorithms, and discoveries. *arXiv preprint arXiv:2003.01853*, 2020.

[14] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: an approach to modeling networks. *Journal of Machine Learning Research*, 11(2), 2010.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. ACM SIGKDD Intl. Conf. on Knowledge discovery in data mining*, 2005.

[16] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. KDD*, 1(1), 2007.

[17] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NeurIPS*, volume 2012, pages 548–56, 2012.

[18] A. Tanenbaum and H. Bos. *Modern operating systems*. Pearson, 2015.

A Dynamic Algorithm for Network Propagation

Barak Sternberg
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
barakolo@gmail.com
https://orcid.org/0000-0002-1803-6437

Roded Sharan[1]
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
roded@post.tau.ac.il

Abstract

Network propagation is a powerful transformation that amplifies signal-to-noise ratio in biological and other data. To date, most of its applications in the biological domain employed standard techniques for its computation that require $O(m)$ time for a network with n vertices and m edges. When applied in a dynamic setting where the network is constantly modified, the cost of these computations becomes prohibitive. Here we study, for the first time in the biological context, the complexity of dynamic algorithms for network propagation. We develop a vertex decremental algorithm that is motivated by various biological applications and can maintain propagation scores over general weights at an amortized cost of $O(m/n^{1/4})$ per update. In application to real networks, the dynamic algorithm achieves significant, 50- to 100-fold, speedups over conventional static methods for network propagation, demonstrating its great potential in practice.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Network propagation, Dynamic graph algorithm, protein-protein interaction network

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.7

Supplement Material `https://github.com/barakolo/dygraph_bio`

Funding R. S. was supported a grant from the Ministry of Science, Technology and Space of the State of Israel and the Helmholtz Centers, Germany.

1 Introduction

Network propagation has become a central technique in biology, as in other domains, to rank the relevance of genes to a process under investigation [7]. However, its complexity is becoming a bottleneck in dynamic settings where the network is subjected to multiple changes. In the biological domain, dynamic computations are essential not only because the network is updated with time but also because certain applications involve the systematic evaluation of propagation results under many network modifications. For example, in [12] the propagation over a network with n nodes is compared to n other propagations that are performed on modifications of the network where each time a different vertex is removed (simulating a knockout). Another application of the dynamic setting is when working with tissue-specific networks. As an example, in [14] multiple tissue-specific networks are formed from a given protein-protein interaction network by removing vertices with low expression, and propagation computations are applied to each.

[1] Corresponding author

While the computation of network propagation requires matrix inversion, a common and more efficient alternative utilizes the power iteration method [6]. This method approximates the propagation scores to within some additive constant at a cost of $O(m)$. An alternative local approach for obtaining approximate propagation was also suggested [3], yielding $O(m)$ time for a single propagation at worst case.

Focusing on a dynamic setting in which vertices are removed one by one [12], the total complexity of maintaining the propagation vectors after each removal becomes $O(mn)$ for n vertices when computing each propagation afresh. There is relatively scarce prior work regarding the computation of network propagation in a dynamic fashion which can be applied to the above setting. Specifically, Zhang et al. [5] and Ihsaka et al. [10] provide a fully dynamic propagation algorithm whose expected time per edge update is $O(1)$. However, both algorithms are limited to unweighted graphs and expected time analysis and might yield $O(mn)$ time under the settings considered here (of n vertex removals). This is true also for Yoon et al. [8], who provide a fully dynamic algorithm for network propagation but may lead to $O(mn)$ time under the settings considered here.

To tackle the dynamic computation challenge, we propose a novel algorithm that can handle general weights and several normalizations, including a symmetric normalization, a variant of which has been shown to be powerful in the biological domain [11, 12]. Our algorithm can handle n vertex removals in $O(mn^{3/4})$ total time. This yields a speedup of $\Omega(n^{1/4})$ over previous work. For real biological networks, this leads to a 50-fold to $100-fold$ speedup in the computations.

2 Preliminaries

We focus on undirected and weighted networks that represent protein-protein interactions. For a network G with n vertices and m edges, we denote by w the symmetric weighted adjacency matrix of the network. For a vertex u, we denote its set of neighbors by $N(u)$ and their number, i.e., the *degree* of u, by $d(u)$. The *weighted degree* $w(u)$ of u is the sum of weights of its adjacent edges. Two common normalizations of w to form a normalized matrix W are as follows: (i) normalizing each column of w to sum to 1, henceforth *weighted degree normalization*, and (ii) dividing each entry of w by the squared product of the weighted degrees of the corresponding nodes, i.e., $W_{ij} = w_{ij}/(\sqrt{w(i)}\sqrt{w(j)})$, henceforth *symmetric normalization* as used in [11].

Given a network G, a prior vector p of node relevance values (in [0,1]; typically 0 or 1), and a parameter $0 < \alpha < 1$, the *network propagation* transformation computes a score $s(v)$ for every node v that is a linear combination of its prior value and the average score of its network neighbors, reflecting its network proximity to the a-priori relevant nodes. Formally,

$$s(v) = \alpha p(v) + (1-\alpha) \sum_{u \in N(v)} s(u) W_{uv}$$

where $0 \leq \alpha \leq 1$ controls the tradeoff between prior information and network smoothing [7]. s, the propagation vector, can be computed analytically via matrix inversion or approximated using the "power iteration" algorithm which works as follows ([1, 9]):

1. Define $v_0 = p, i = 0$.
2. Compute $v_{i+1} = \alpha p + (1-\alpha) W v_i$.
3. While $i < \frac{\log(\epsilon)}{\log(1-\alpha)}$ increment i and goto (2).

The "power iteration" process is known to converge to the propagation vector ($\lim_{i\to\infty} v_i = s$), whenever the eigenvalues of W are at most 1 in absolute value, a condition satisfied

Algorithm 1 ForwardPush for weighted degree normalization.

```
1: procedure PUSH(v, R, P, α, W)
2:     P(v)+ = αR(v)
3:     for n ∈ N(v) do
4:         R(n)+ = (1 − α)R(v)W_vn
5:     R(v) = 0
6: procedure ForwardPush(W, α, ϵ, R, P)
7:     while ∃u|R(u)| > ϵd(u) do
8:         push(u, R, P, α, W)
9:     return P
```

by both normalizations presented above [7]. Here $\epsilon > 0$ is the required approximation bound on the sum of differences in absolute value (L1 error) between the computed and true propagation score. ϵ is typically chosen to be a constant smaller than 0.01. The following lemma characterizes the resulting propagation vector.

▶ **Lemma 2.1** ([2, 7]). *The propagation vector converges to* $\alpha(I - (1-\alpha)W)^{-1}p$.

Our algorithmic approach is motivated by the following lemma:

▶ **Lemma 2.2** ([6]). *Consider a random walk from prior distribution* p *using the weighted-degree normalized adjacency matrix* W*, where at each node* u *the walk stops with probability* α*. Then the total probability of the walk to stop at* u *is* $s(u)$*.*

3 The Forward-Push algorithm

In the following we describe our dynamic algorithms for the case of symmetric and weighted degree normalization. Our algorithm builds on the Forward-Push algorithm for the static case [3] which can be viewed as "simulating" random walks, “pushing” walks from one node to another and taking into account walks that stopped at any node, adding their probability to the node’s score. The algorithm is run until the residual walks have neglible effects (ϵ). For clarity, we fix the prior vector to p. The algorithm maintains two estimates per node u: the current estimate $P(u)$ of the probability to stop at u and the remaining probability $R(u)$ of walks that have reached u without stopping. The algorithm is given below and is called by initializing $P = 0^n$ (the zero vector) and $R = p$. For any nodes s, t we denote by $\pi(s,t)$ the score of t when propagating from s. Generalizing it, for any prior vector p and node u we denote by $\pi(p,u)$ the score of u when propagating from p. In the symmetric normalization case, $\pi(s,t) = \pi(t,s)$ for any two nodes s, t, while in the weighted degree normalization case it can be shown that $\pi(s,t)w(s) = \pi(t,s)w(t)$ (see, e.g., Lemma 1 in [13]). The following lemma is key in proving the correctness of the algorithm.

▶ **Lemma 3.1.** *The following equalities (algorithm’s invariants) are equivalent:*

1. $P(u) + \alpha R(u) = (1-\alpha) \sum_{x \in N(u)} P(x)W_{xu} + \alpha p_u$
2. $\pi(p,u) = P(u) + \sum_{x \in V} R(x)\pi(x,u)$

Proof. Denote $\pi = \alpha(I - (1-\alpha)W)^{-1}$. Note that $(\pi p)_u = \sum_{x \in V} \pi(x,u)p_x$, $(\pi R)_u = \sum_{x \in V} \pi(x,u)R(x)$. We can write (2) in vector form as: $\pi p = P + \pi R$. Multiplying both

sides by π^{-1} from the left we get: $p = \pi^{-1}P + R \leftrightarrow \alpha p = P - (1-\alpha)WP + \alpha R$. Rearranging terms we get the desired result. ◀

Using this lemma we can now justify the propagation approximation achieved by the ForwardPush algorithm.

▶ **Lemma 3.2.** *The ForwardPush algorithm maintains the invariants in 3.1.*

Proof. It suffices to prove that the first invariant holds. On initialization, $P = 0^n$ and $R = p$, hence the invariant holds. Suppose an arbitrary node u is pushed and denote by P', R' the updated values of P and R. Since $P'(u) = P(u) + \alpha R(u)$ and $R'(u) = 0$, we have $P'(u) + \alpha R'(u) = P(u) + \alpha R(u)$. Clearly, the right hand side of invariant (1) did not change for u as for any $x \in N(u)$ only $R(x)$ is changed while pushing u. For any other node x that is adjacent to u, $R'(x) = R(x) + (1-\alpha)R(u)W_{xu}$. Thus, the addition to the left hand side of the equation is $\alpha(1-\alpha)R(u)W_{xu}$ which is exactly the addition to the right hand side due to the update of $P(u)$. ◀

▶ **Lemma 3.3.** *For weighted-degree normalization, when ForwardPush() terminates:*

$$\sum_{t \in V} |P(t) - \pi(p,t)| \leq 2\epsilon m$$

Proof. Consider any node t and let e_t be the unit vector with 1 at coordinate t and 0 elsewhere. It follows from lemma 2.2 that for any node $u \in V$, $\sum_{t \in V} \pi(u,t) \leq 1$. By Lemma 3.1 and since $|R|_\infty \leq \epsilon$ upon termination,

$$\sum_{t \in V} |P(t) - \pi(p,t)| = \sum_{t \in V} \sum_{u \in V} |R(u)| \pi(u,t) \leq \sum_{t \in V} \sum_{u \in V} \epsilon d(u) \pi(u,t) = \sum_{u \in V} \epsilon d(u) \sum_{t \in V} \pi(u,t) \leq \sum_{u \in V} \epsilon d(u) = 2\epsilon m$$

◀

Note that the above lemma bounds the overall L1 approximation of the algorithm at $2\epsilon m$, hence ϵ is typically chosen to be smaller than $1/m$ to guarantee a constant overall error. After proving the correctness of the algorithm, we turn to bound its complexity:

▶ **Lemma 3.4.** *Every push of the algorithm, $|R|_1$, the sum of residuals in absolute value, decreases by at least $\alpha\epsilon d(v)$ where v is the node being pushed.*

Proof. Let R, R' denote the residuals before and after the push, respectively. Then

$$\sum_{u \in V} |R(u)| - \sum_{u \in V} |R'(u)| = |R(v)| + \sum_{u \in N(v)} |R(u)| - |R'(u)| \geq \quad (1)$$

$$|R(v)| - \sum_{u \in N(v)} |R'(u) - R(u)| = |R(v)| - (1-\alpha)|R(v)| \sum_{u \in N(v)} |W_{vu}| \geq \quad (2)$$

$$|R(v)| - (1-\alpha)|R(v)| = |R(v)|(1-(1-\alpha)) = \alpha|R(v)| \geq \alpha\epsilon d(v) \quad (3)$$

◀

▶ **Lemma 3.5.** *The ForwardPush algorithm with weighted degree normalization takes $O(\frac{|p|_1}{\alpha\epsilon})$ time.*

Proof. On initialization $\sum_{v \in V} |R(v)| = |p|_1$. Denote by u_i the vertex being pushed at step i of the algorithm. By Lemma 3.4, $\sum_{v \in V} |R(v)|$ decreases by at least $\alpha\epsilon d(u_i)$ at step i, thus the following holds: $\sum_i \alpha\epsilon d(u_i) \leq |p|_1$. Hence, $\sum_i d(u_i) \leq \frac{|p|_1}{\alpha\epsilon}$. As each push step of a node u_i can be implemented in $O(d(u_i))$ time, the total time is $O(\frac{|p|_1}{\alpha\epsilon})$. The residuals with absolute values greater than $\epsilon d(u_i)$ can be maintained in a linked list and an associated array at the same cost. ◀

We can generalize the above algorithm to any similar matrix $W = L^{-1}SL$ where L is an invertible diagonal matrix, S is a stochastic matrix, i.e: $S = wD^{-1}$, and D is the diagonal weighted degree matrix. For example, choosing $L = D^{1/2}$ yields the symmetric normalization. Note that such matrices satisfy the convergence condition as their eigenvalues are the same as those of S. The generalization is based on the following lemma:

▶ **Lemma 3.6.** *(Stochastic Lemma) Let $\psi(W, p, \alpha)$ be the result of network propagation with matrix $W = L^{-1}SL$ over prior p. Then $\psi(W, p, \alpha) = L^{-1}\psi(S, Lp, \alpha)$.*

Proof. Denote by v_n and v'_n the propagation vectors on W and S, respectively, after the nth step of the propagation process. We prove by induction that for every n, $v_n = L^{-1}v'_n$. For the base case $v_0 = p$ and $v'_0 = Lp$ so the claim trivially holds. Suppose the claim is true for n, then: $v_{n+1} = \alpha p + (1-\alpha)Wv_n = \alpha p + (1-\alpha)L^{-1}SL(L^{-1}v'_n) = L^{-1}(\alpha(Lp) + (1-\alpha)Sv'_n) = L^{-1}v'_{n+1}$. ◀

4 A dynamic decremental algorithm for vertex removals

Lemma 3.6 naturally suggests an adaptation for the *ForwardPush* algorithm for any normalization matrix W which is similar to a stochastic matrix. For concreteness and clarity, we will present a dynamic decremental algorithm for removing nodes under weighted symmetric normalization as used in [12]. The main idea is first to maintain a valid propagation result for weighted-degree normalization, and on deletion, using lemma 3.6, we fix the invariant. If this leads to high residuals, we will re-push them over the graph locally, to fix the total scores. In the following we set $L = D^{1/2}$ for symmetric normalization and $p' = LP$. For a removal of any node v, denote by S', D' the resulting matrices, by w' the updated network weights and by π' the updated propagation scores. Further denote by $\mathrm{Leaf}N(v)$ the set of neighbors of v of degree 1. W.l.o.g. v has at least one neighbor which is of degree greater than 1, otherwise the connected component of v will vanish after its removal. The algorithm is given below (Algorithm 2). *ForwardPushSym* computes the initial propagation. *FixRemoveEdge* handles edge removals and is used as a subroutine by *VertexRemoveProp* which handles the node removals.

The following lemmas, proved in the Appendix, establish the correctness and accuracy of the algorithm. Denote $\phi := \max_v\{\sqrt{w(v)}, 1/\sqrt{w(v)}\}$. In all our applications reported below, for both yeast and human, $\phi < 36$.

▶ **Lemma 4.1.** *When $ForwardPushSym(W, \alpha, \epsilon, p)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p, t)| < 2\epsilon m\phi$ where $\pi(p, t)$ is the propagation score for node t with prior p under symmetric normalization.*

▶ **Lemma 4.2.** *When $VertexRemoveProp(v, R, P, W, \epsilon)$ returns, $\sum_{t \in V} |\frac{P(t)}{\sqrt{w'(t)}} - \pi'(p, t)| < 2\epsilon m\phi$ where $\pi'(p, t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Algorithm 2 ForwardPush for symmetric normalization.

1: **procedure** $ForwardPushSym(W = D^{-1/2}SD^{1/2}, \alpha, \epsilon, p)$
2: $R \leftarrow D^{1/2}p, P \leftarrow 0^n$
3: $R, P \leftarrow ForwardPush(S, R, P, \alpha, \epsilon)$
4: **return** R, P
5: **procedure** $FixRemoveEdge(u, v, R, P, W = D^{-1/2}SD^{1/2})$
6: $w(v), w_u(v) := \sum_{x \in V} w_{xv}, \sum_{x \in V, x \neq u} w_{xv}$
7: $F_u(v) = \frac{w(v)}{w_u(v)}$
8: $R(v) += \frac{1}{\alpha}(1 - \frac{1}{F_u(v)})P(v) - \frac{(1-\alpha)}{\alpha}P(u)S_{uv} + (\sqrt{w_u(v)} - \sqrt{w(u)})p_v$
9: $P(v) /= F_u(v)$
10: Remove (u, v) from S and normalize v.
11: **procedure** $VertexRemoveProp(v, R, P, W = D^{-1/2}SD^{1/2}, \epsilon)$
12: **for** $u \in \text{Leaf}N(v)$ **do**
13: $R(u) = 0, P(u) = \alpha p_u$
14: **for** $u \in N(v) \backslash \text{Leaf}N(v)$ **do**
15: $FixRemoveEdge(v, u, R, P, W)$
16: $R(v) = 0, P(v) = \alpha p_v$
17: $ForwardPush(S', R, P, \alpha, \epsilon)$
18: **return** R, P

Algorithm 3 Multiple-vertex removal

1: **procedure** $RemoveNodes(node\_list, W = D^{-1/2}SD^{1/2}, \alpha, \epsilon_0, \epsilon_1, p)$
2: $R, P \leftarrow ForwardPushSym(W, \alpha, \epsilon_0, p)$
3: **for** $u \in node\_list$ **do**
4: Backup R,P
5: $R, P \leftarrow VertexRemoveProp(u, R, P, W, \epsilon_1)$
6: $res(u) \leftarrow D^{-1/2}P$
7: Restore R,P
8: **return** res

Given the vertex removal routine, we can perform n removals and corresponding propagations using Algorithm 3. For efficiency, we can choose different accuracies ($\epsilon_0, \epsilon_1 = \epsilon$) for different stages of the propagation with $\epsilon_0 \leq \epsilon_1$. This allows us to invest more time in the initial propagation in order to reduce the resulting residual sum, leading to time savings in subsequent computations. In order to bound the complexity of the algorithm we denote by R^u the vector of residuals after fixing invariants in $VertexRemoveProp$ (line (16) completion) and define the total sum in absolute value of residual changes following the removal of a vertex u:

$$\Delta R_u := \sum_{v \in V} |R^u(v) - R(v)|$$

▶ **Lemma 4.3.** *The total residual sum being pushed over n vertex removals is bounded by:*

$$T := \sum_{u \in V} (\sum_{v \in V} |R(v)| + \Delta R_u)$$

Proof. The total sum of residuals being pushed when removing a vertex u is bounded by

$\sum_{v\in V\setminus\{u\}} |R^u(v)|$. By the triangle inequality, $\sum\limits_{v\in V\setminus\{u\}} |R^u(v)| \leq \sum\limits_{v\in V} |R(v)| + \Delta R_u$. ◀

▶ **Corollary 4.4.** *The removal of n vertices and subsequent propagations take $O(\frac{T}{\alpha\epsilon_1})$ time.*

▶ **Lemma 4.5.** *Let u be a node being removed, and $F_u(v)$ the expression described in Algorithm (2). Then,*

$$\Delta R_u \leq R(u) + \sum_{v\in LeafN(u)} R(v) + \sum_{v\in N(u)\setminus LeafN(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v +$$
$$(\frac{1-\alpha}{\alpha})P(u) \sum_{v\in N(u)\setminus LeafN(u)} S_{uv} + \frac{1}{\alpha} \sum_{v\in N(u)\setminus LeafN(u)} P(v)(1 - \frac{1}{F_u(v)})$$

Proof. Note that $R(v), P(v)$ are positive for any v as they represent the initial ForwardPush result. Assume that u is removed and consider the changes to its neighbors in VertexRemove(). Clearly, for each leaf neighbor v, the change in residuals is $R(v)$. For non-leaf neighbors v, we can upper bound the residual change by:

$$\frac{1}{\alpha}\left(1 - \frac{1}{F_u(v)}\right) P(v) + |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v + \frac{1-\alpha}{\alpha}P(u)S_{uv}$$

Hence, summing over all of these neighbors and u itself leads to the required result. ◀

To bound the complexity of the algorithm we need the following notation and auxiliary lemmas.

▶ **Lemma 4.6.** $1 - \frac{1}{F_u(v)} = S_{uv}$

Proof. $1 - \frac{1}{F_u(v)} = \frac{\sum\limits_{x\in V} w_{xv}}{\sum\limits_{x\in V} w_{xv}} - \frac{\sum\limits_{x\neq u} w_{xv}}{\sum\limits_{x\in V} w_{xv}} = S_{uv}$ ◀

▶ **Lemma 4.7.** *After the initial ForwardPushSym, $\sum\limits_{v\in V} P(v) \leq \phi$*

Proof. Initially, $\sum_{v\in V} P(v) = 0$ and $\sum_{v\in V} |R(v)| = |p'|_1$. Anytime we increase $P(u)$ by $\alpha R(u)$ for some u, $\sum_{v\in V} |R(v)|$ is reduced by at least $\alpha|R(u)|$ by Lemma 3.4. Therefore, $\sum_{v\in V} |R(v)| \leq |p'|_1$ throughout, implying that $\sum\limits_{v\in V} P(v) \leq |p'|_1$.

By definition, $|D^{1/2}|_1 = \sup\limits_{v\neq 0} \frac{|D^{1/2}v|_1}{|v|_1}$. Hence, $|p'|_1 = |D^{1/2}p|_1 \leq \frac{|D^{1/2}p|_1}{|p|_1}|p|_1 \leq |D^{1/2}|_1|p|_1 \leq \phi$. ◀

▶ **Lemma 4.8.** $\sum\limits_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sqrt{w(v)}$

Proof. First, note that:

$$|\sqrt{w_u(v)} - \sqrt{w(v)}| = |\frac{(\sqrt{w_u(v)} - \sqrt{w(v)})(\sqrt{w_u(v)} + \sqrt{w(v)})}{\sqrt{w_u(v)} + \sqrt{w(v)}}|$$
$$= \frac{|w_u(v) - w(v)|}{|\sqrt{w_u(v)} + \sqrt{w(v)}|} \leq \frac{w_{vu}}{\sqrt{w(v)}}$$

Then, summing over all neighbors of v:

$$\sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| \leq \sum_{u\in N(v)} \frac{w_{vu}}{\sqrt{w(v)}} = \sqrt{w(v)}$$ ◀

▶ **Lemma 4.9.** $\sum_{u\in V} \Delta R_u \leq \frac{5\phi}{\alpha}$.

Proof. Let R, P be the updated residuals and the estimates after the initial application of ForwardPush() in line 2 of the RemoveNodes() algorithm. By Lemma 4.5, the total changes to the residuals are

$$\overbrace{\sum_{u\in V} R(u) + \sum_{u\in V}\sum_{v\in \text{Leaf}N(u)} R(v)}^{\text{part 1}} + \overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v}^{\text{part 2}} +$$
$$\overbrace{\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)\setminus \text{Leaf}N(u)} S_{uv}}^{\text{part 3}} + \frac{1}{\alpha}\overbrace{\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} P(v)(1-\frac{1}{F_u(v)})}^{\text{part 4}}$$

We will bound each part separately.

1. By definition, $\sum_{u\in V} R(u) \leq |D^{1/2}p|_1$. For a leaf $v \in V$, its residual will be summed once as its degree is 1. Overall, the sum is bounded by $2\sum_{u\in V} R(u) \leq 2|D^{1/2}p|_1 \leq 2\phi$.
2. By changing the summation order we get
$$\begin{aligned}\sum_{u\in V}\sum_{v\in N(u)\setminus \text{Leaf}N(u)} |\sqrt{w_u(v)} - \sqrt{w(v)}|p_v &= \\ \sum_{v\in V,\text{ not a leaf}} p_v \sum_{u\in N(v)} |\sqrt{w_u(v)} - \sqrt{w(v)}| &\overset{4.8}{\leq} \\ \sum_{v\in V,\text{ not a leaf}} p_v\sqrt{w(v)} &\leq |p|_1\phi = \phi\end{aligned}$$
3. we bound this part and get:
$$\sum_{u\in V}(\frac{1-\alpha}{\alpha})P(u)\sum_{v\in N(u)} S_{uv} \leq \frac{(1-\alpha)}{\alpha}\sum_{u\in V} P(u) \overset{4.7}{\leq} \frac{(1-\alpha)}{\alpha}\phi$$
4. Similar to part (1), by changing the order of summation we get that:
$$\begin{aligned}\frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)}(1-\frac{1}{F_u(v)}) &\overset{4.6}{=} \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v)\sum_{u\in N(v)} S_{uv} &\leq \\ \frac{1}{\alpha}\sum_{v\in V,\text{ not a leaf}} P(v) &\leq \phi/\alpha\end{aligned}$$

Overall we obtain a bound of $\frac{5\phi}{\alpha}$. ◀

We are now ready to state our main result:

▶ **Lemma 4.10.** *The total complexity of the propagation algorithm with n vertex removals is* $(\frac{\phi}{\alpha\epsilon_0} + \frac{mn\epsilon_0}{\alpha\epsilon_1} + \frac{\phi}{\epsilon_1}) = O(m\sqrt{n\phi})$

Proof. The initialization takes $O(\phi/(\alpha\epsilon_0))$ time. By Lemmas 4.3 and 4.9, the total time for n vertices removals and subsequent propagations is bounded by:

$$\frac{\sum_{u\in V}(\sum_{v\in V}|R(v)|) + \Delta R_u}{\alpha\epsilon_1} \leq \frac{\sum_{u\in V}(\sum_{v\in V}\epsilon_0 d(v)) + \Delta R_u}{\alpha\epsilon_1} = \frac{\sum_{u\in V}(2m\epsilon_0) + \Delta R_u}{\alpha\epsilon_1} = \frac{2mn\epsilon_0}{\alpha\epsilon_1} + \frac{\frac{5\phi}{\alpha}}{\alpha\epsilon_1}$$

Table 1 Tested Networks and their properties.

Graph	#Nodes	#Edges	Maximal Degree	Average Degree	ϕ
Human (HIPPIE) [4]	19,796	339,788	2173	17.1	35.67
Yeast (ANAT [15])	5138	76,467	2040	13.82	19.9
Human (ANAT [15])	15,517	259,161	2086	15.73	27.75

Table 2 Performance evaluation on a human network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	10.1s	9417.89s	886.54s
#node visits per update	302.24	55,175,737	2,618,256
L1-error	0.00121	0.00119	0.00101

Hence, The complexity is:

$$O(\frac{\phi}{\alpha\epsilon_0}+\frac{mn\epsilon_0}{\alpha\epsilon_1}+\frac{\phi}{\epsilon_1}) \overset{\epsilon_0=\sqrt{\frac{\phi\epsilon_1}{mn}}}{=} O(\sqrt{\frac{mn\phi}{\epsilon_1}}) \overset{\epsilon_1=\Theta(\frac{1}{m})}{=} O(m\sqrt{n\phi})$$

The first equality follows by finding the minimal solution for ϵ_0, ϵ_1, where ϵ_1 is chosen to be c/m to bound the overall L1 error by some additive constant. As the edge weights are at most 1 and at least some positive constant (otherwise, the edges are considered unreliable), $\phi \leq \sqrt{n}$. Hence, the total time is $O(mn^{3/4})$ and the amortized cost per node knockout is $O(\frac{m}{n^{1/4}})$. ◀

5 Results

We benchmark our algorithm, Dynamic ForwardPushSym (DynamicFP), against a static implementation of the power iteration method, as well as against the static ForwardPush algorithm (ForwardPushSym from scratch). We measure the performance of each algorithm in terms of time (seconds) and the number of nodes it visits throughout its execution. In order to evaluate the accuracy of the different algorithms we measured their L1 deviation from the true propagation vector (sum of absolute differences). All tests were done in Intel(R) Xeon(R) E5410 @ 2.33Ghz, 16GB RAM. We used the latest yeast and human protein-protein interaction (PPI) networks from ANAT [15] as well as the human PPI network of [4]. The tested networks and their properties are summarized in Table 1. To perform a comparative analysis of different propagation algorithms, we fixed the propagation parameters to $\alpha = 0.4$ and a prior set size of 100. In order to compute accurate propagation scores against which we could benchmark the different methods, we applied the power iteration method with $\epsilon' = \epsilon/n$. In the comparison itself, we applied the power iteration method with the maximal ϵ that leads to an overall L1 error of at most 10^{-2}. For Forward-Push we used the maximal ϵ_1 that leads to an overall L1 error of at most 10^{-2}. We also set $\epsilon_0 = \sqrt{\frac{\epsilon_1\phi}{mn}}$. The results upon making 1,000 vertex removals, where each vertex is removed separately from the original network, are summarized in Tables 2 and 3. Next, we compared our performance to the previous approach LazyFwdUpdate of [5]. To this end, we used a larger human PPI network from [4] and observed the performance of the two methods upon knockouts of all (non-prior) vertices, simulating a real application of these methods. Since LazyFwdUpdate handles only unweighted networks with degree-based normalization,

Table 3 Performance evaluation on a yeast network (ANAT) upon 1,000 vertex removals.

Algorithm	DynamicFP	ForwardPush	Power Iteration
Time	2.44s	2132.2s	102.91s
#node visits per update	1443.904	22,103,498	765,310
L1-error	0.0013	0.00121	0.0017

Table 4 Performance comparison with $\epsilon_0 = \epsilon_1$ (1) and $\epsilon_0 = \sqrt{\frac{\epsilon_1 \phi}{mn}}$ (2).

Algorithm	LazyFwdUpdate	DynamicFP (1)	DynamicFP (2)
Time	36.01s	23.60	9.71s
#node visits per-update	13,925	8024	2749
L1-error	0.0047	0.0056	0.0061

we used the corresponding variant (unweighted network; weighted-degree normalization) of the *DynamicFP* algorithm. The results, summarized in Table 4, show that our algorithm compares favorably to LazyFwdUpdate, especially when varying the ratio between ϵ_0 and ϵ_1.

6 Conclusions

We have devised a dynamic algorithm for network propagation that can handle a vertex deletion in $O(\frac{m}{n^{1/4}})$ time and provides a speedup of $\Omega(n^{1/4})$ over previous work. Importantly, the algorithm leads to huge speedups of up to a 50-100 fold on real data. Thus, it allows, for the first time, the application of costly methods (such as [12] and [14]) to examine multiple gene knockouts simultaneously. The proposed algorithm can substantially increase the number of different knockout effects that can be simulated in a reasonable time. While our work focused on vertex deletions that are very common in the biological domain, it would be interesting to extend our algorithm to a fully dynamic one. This may result in various applications that concern situations in which edges rather than nodes are perturbed.

References

1 Amy N. Langville and Carl D. Meyer. Deeper Inside PageRank. *Internet Mathematics*, 1(3), jan 2004. doi:10.1080/15427951.2004.10129091.

2 Dengyong Zhou, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Scholkopf. Learning with Local and Global Consistency. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, pages 321–328. MIT Press, 2004. URL: http://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf.

3 Glen Jeh and Jennifer Widom. Scaling Personalized Web Search. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, WWW '03, pages 271–279, New York, NY, USA, 2003. ACM. doi:10.1145/775152.775191.

4 Gregorio Alanis-Lobato, Miguel A. Andrade-Navarro, and Martin H. Schaefer. HIPPIE v2.0: enhancing meaningfulness and reliability of protein–protein interaction networks. *Nucleic Acids Research*, 45(D1):D408–D414, jan 2017. doi:10.1093/nar/gkw985.

5 Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate Personalized PageRank on Dynamic Graphs. *arXiv:1603.07796 [cs]*, 2016. arXiv: 1603.07796. URL: `http://arxiv.org/abs/1603.07796`.

6 Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web., nov 1999. URL: `http://ilpubs.stanford.edu:8090/422/`.

7 Lenore Cowen, Trey Ideker, Benjamin J. Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews. Genetics*, 18(9):551–562, 2017. `doi:10.1038/nrg.2017.38`.

8 Minji Yoon, WooJeong Jin, and U Kang. Fast and Accurate Random Walk with Restart on Dynamic Graphs with Guarantees. *arXiv:1712.00595 [cs]*, 2017. arXiv: 1712.00595. URL: `http://arxiv.org/abs/1712.00595`.

9 Monica Bianchini, Marco Gori, and Franco Scarselli. PageRank and Web communities. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 365–371, oct 2003. `doi:10.1109/WI.2003.1241217`.

10 Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient PageRank Tracking in Evolving Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, KDD '15, pages 875–884, New York, NY, USA, 2015. ACM. `doi:10.1145/2783258.2783297`.

11 Oron Vanunu, Oded Magger, Eytan Ruppin, Tomer Shlomi, and Roded Sharan. Associating Genes and Protein Complexes with Disease via Network Propagation. *PLOS Computational Biology*, 6(1):e1000641, 2010. `doi:10.1371/journal.pcbi.1000641`.

12 Ortal Shnaps, Eyal Perry, Dana Silverbush, and Roded Sharan. Inference of Personalized Drug Targets via Network Propagation. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 21:156–67, 2016. URL: `https://www.semanticscholar.org/paper/Inference-of-Personalized-Drug-Targets-via-Network-Shnaps-Perry/b57104bd662ffeb95bb150d00adb381caffce013`.

13 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Bidirectional PageRank Estimation: From Average-Case to Worst-Case. In *Algorithms and Models for the Web Graph - 12th International Workshop, WAW 2015, Eindhoven, The Netherlands, December 10-11, 2015, Proceedings*, WAW 2015, pages 164–176, New York, NY, USA, 2015. Springer-Verlag New York, Inc. `doi:10.1007/978-3-319-26784-5_13`.

14 Sushant Patkar, Assaf Magen, Roded Sharan, and Sridhar Hannenhalli. A network diffusion approach to inferring sample-specific function reveals functional changes associated with breast cancer. *PLoS Computational Biology 13(11): e1005793, in press*, 13, nov 2017. `doi:10.1371/journal.pcbi.1005793`.

15 Yomtov Almozlino, Nir Atias, Dana Silverbush, and Roded Sharan. ANAT 2.0: reconstructing functional protein subnetworks. *BMC bioinformatics*, 18(1):495, 2017. `doi:10.1186/s12859-017-1932-1`.

A Supplementary proofs

▶ **Lemma A.1.** *When* $ForwardPushSym(W, \alpha, \epsilon, p)$ *returns,* $\sum_{t \in V} |\frac{P(t)}{\sqrt{w(t)}} - \pi(p,t)| < 2\epsilon m\phi$ *where* $\pi(p,t)$ *is the propagation score for node* t *with prior* p *under symmetric normalization.*

Proof. By Lemma 3.3, $ForwardPush$ applied on S with prior p' returns a vector P of

estimated scores, where $\sum_{t\in V}|P(t)-\pi(p',t)| = |P-\pi p'|_1 \le 2\epsilon m$.

$$\begin{aligned}
|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m \rightarrow \\
|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1|D^{1/2}|_1 \rightarrow \\
|D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m|D^{-1/2}|_1 \rightarrow \\
|D^{-1/2}|_1|P-\psi(S,D^{1/2}p,\alpha)|_1 &\le 2\epsilon m\phi
\end{aligned}$$

Since $|D^{-1/2}|_1 = \sup_{v\neq 0}\frac{|D^{-1/2}v|_1}{|v|_1}$ it follows that $|D^{-1/2}v|_1 = \frac{|D^{-1/2}v|_1}{|v|_1}|v|_1 \le |D^{-1/2}|_1|v|_1$. Therefore: $|D^{-1/2}P - D^{-1/2}\psi(S,D^{1/2}p,\alpha)|_1 \le 2\epsilon m\phi$.
Note that $\psi(W,p,\alpha)$ is a vector of propagation scores $\pi(p,t)$ for all nodes t. By the stochastic Lemma 3.6, $\psi(W,p,\alpha) = D^{-1/2}\psi(S,D^{1/2}p,\alpha)$. Thus,

$$|D^{-1/2}P-\psi(W,p,\alpha)|_1 \le 2\epsilon m\phi \rightarrow \sum_{t\in V}|\frac{P(t)}{\sqrt{w(t)}}-\pi(p,t)| \le 2\epsilon m\phi \quad \blacktriangleleft$$

▶ **Lemma A.2.** *When $VertexRemoveProp(v,R,P,W,\epsilon)$ returns, $\sum_{t\in V}|\frac{P(t)}{\sqrt{w'(t)}}-\pi'(p,t)| < 2\epsilon m\phi$ where $\pi'(p,t)$ is the propagation score for node t with prior p under symmetric normalization after node v removal.*

Proof. Denote by R',P',S',D' the changed residuals and estimates, weights and sum of degrees (for an isolated vertex u we define $D^{-1}_{uu} = D_{uu} = 1$), respectively, after line (16) in $VertexRemoveProp$ and $p' := D^{1/2}p$, $p'' := D'^{1/2}p$. Note that $p'_u := \sqrt{w(u)}p_u$, $p''_u := \sqrt{w_v(u)}p_u$ by definition of D' (after node removal). We will first prove the following:

▶ **Lemma A.3.** *The following holds:*
1. *if $x \neq v$ then $P'(x)S'_{xu} = P(x)S_{xu}$*
2. *if $x = v$ and $u \in N(v)$ then $P'(x)S'_{xu} = 0$*
3. *if $x = v$ and $u \notin N(v)$ then $P'(x)S'_{xu} = P(x)S_{xu}$*

Proof.
1. Let $x \neq v$, if $P(x) \neq P'(x)$ then $x \in N(v)$ by algorithm def, hence $P'(x) = P(x)/F_u(v)$ and $S'_{xu} = S_{xu}F_u(v)$. Then for sure $P'(x)S'_{xu} = P(x)S_{xu}$. If $P(x) = P'(x)$ then x is not a neighbor of v, hence by algorithm def. $S'_{xu} = S_{xu}$.
2. Let $x = v$ and $u \in N(v)$, by algorithm def, $S'_{xu} = S'_{vu} = 0$ as the edge was removed, and $P'(x)S'_{xu} = 0$.
3. Let $x = v$ and $u \notin N(v)$, by algorithm def, $P'(x) = P(x)$, as no edge that connects to u was removed, also $S'_{xu} = S_{xu}$ hence $P(x)S_{xu} = P'(x)S'_{xu}$. ◀

We will show that after line (16) in $VertexRemoveProp$ (2), the invariants of $ForwardPush$ (3.1) are kept with respect to p'' as the prior, S' as the weights matrix and α.

- Let $u \in \text{Leaf}N(v)$, then after line (16) node u becomes isolated and $P'(u) + \alpha R'(u) = \alpha p_v = \alpha p''_v$ as required.
- Let $u \in N(v)$, we will show the invariant is kept for u. We know:
$$P(u) + \alpha R(u) = (1-\alpha)\sum_{x\in V}P(x)S_{xu} + \alpha p'_u \tag{4}$$
We want to show:
$$P'(u) + \alpha R'(u) = (1-\alpha)\sum_{x\in V}P'(x)S'_{xu} + \alpha p''_u \tag{5}$$

Hence, it is enough to show that RHS of (5) minus the RHS of (4) equals to the LHS of (5) minus the LHS of (4).
Using Lemma A.3, for any node $x \neq v$ it holds $P'(x)S'_{xu} = P(x)S_{xu}$ and for $x = v$ it holds $P'(x)S'_{xu} = 0$. Then:

$$\begin{aligned}((1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u) \quad &- \quad ((1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u) = \\ -(1-\alpha)P(v)S_{vu} \quad &- \quad \alpha(p'_u - p''_u)\end{aligned}$$

By Algorithm 2, line 8:

$$\begin{aligned}(P'(u) + \alpha R'(u)) - (P(u) + \alpha R(u)) \quad &= \quad (P'(u) - P(u)) + \alpha(R'(u) - R(u)) \\ &= \quad -(1 - \frac{1}{F_v(u)})P(u) + \alpha[\frac{1}{\alpha}(1 - \frac{1}{F_v(u)})P(u) \\ &- \quad \frac{(1-\alpha)}{\alpha}P(v)S_{vu} + (\sqrt{w_v(u)} - \sqrt{w(u)})p_u] \\ &= \quad -(1-\alpha)P(v)S_{vu} + \alpha(p''_u - p'_u)\end{aligned}$$

- Let $u \notin N(v)$, using Lemma A.3, $P'(x)S'_{xu} = P(x)S_{xu}$ for any x. By algorithm def. $P'(u) = P(u)$, $R'(u) = R(u)$ and $p'_u = p''_u$. Then:

$$\begin{aligned}P'(u) + \alpha R'(u) \quad &= \quad P(u) + \alpha R(u) = \\ (1-\alpha)\sum_{x\in V} P(x)S_{xu} + \alpha p'_u \quad &= \quad (1-\alpha)\sum_{x\in V} P'(x)S'_{xu} + \alpha p''_u\end{aligned}$$

Therefore, after applying $VertexRemoveProp$ we compute a valid propagation vector over the weights S'' with p'' as prior and α after node v removal. using A.1 we get the same approximation. ◀

Efficient Approximation Algorithms for Spanning Centrality

Shiqi Zhang
National University of Singapore
Southern University of Science and Technology
s-zhang@comp.nus.edu.sg

Renchi Yang*
Hong Kong Baptist University
renchi@hkbu.edu.hk

Jing Tang
The Hong Kong University of Science and Technology (Guangzhou)
The Hong Kong University of Science and Technology
jingtang@ust.hk

Xiaokui Xiao†
National University of Singapore
xkxiao@nus.edu.sg

Bo Tang†
Southern University of Science and Technology
tangb3@sustech.edu.cn

ABSTRACT

Given a graph $\mathcal{G}$, the *spanning centrality* (SC) of an edge e measures the importance of e for $\mathcal{G}$ to be connected. In practice, SC has seen extensive applications in computational biology, electrical networks, and combinatorial optimization. However, it is highly challenging to compute the SC of all edges (AESC) on large graphs. Existing techniques fail to deal with such graphs, as they either suffer from expensive matrix operations or require sampling numerous long random walks. To circumvent these issues, this paper proposes TGT and its enhanced version TGT+, two algorithms for AESC computation that offers rigorous theoretical approximation guarantees. In particular, TGT remedies the deficiencies of previous solutions by conducting deterministic graph traversals with carefully-crafted truncated lengths. TGT+ further advances TGT in terms of both empirical efficiency and asymptotic performance while retaining result quality, based on the combination of TGT with random walks and several additional heuristic optimizations. We experimentally evaluate TGT+ against recent competitors for AESC using a variety of real datasets. The experimental outcomes authenticate that TGT+ outperforms state of the arts often by over one order of magnitude speedup without degrading the accuracy.

CCS CONCEPTS

• **Mathematics of computing** → **Approximation algorithms**; *Markov-chain Monte Carlo methods*; **Graph algorithms**.

KEYWORDS

spanning centrality, graph traversal, random walk, eigenvector

ACM Reference Format:
Shiqi Zhang, Renchi Yang, Jing Tang, Xiaokui Xiao, and Bo Tang. 2023. Efficient Approximation Algorithms for Spanning Centrality. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3580305.3599323

*This work was partially done while at National University of Singapore.
†Corresponding authors: Xiaokui Xiao and Bo Tang.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599323

1 INTRODUCTION

Edge centrality is a graph-theoretic notion measuring the importance of each edge in the graph, which plays a vital role in analyzing social, sensor, and transportation networks [5, 11, 32, 37]. As pinpointed by Mavroforakis et al. [29], compared to the classic edge betweenness [9] based on shortest paths, *spanning centrality* (SC) [40] is a more ideal centrality for edges as it accommodates the information from longer paths. In particular, given a connected undirected graph $\mathcal{G}$, the SC $s(e)$ of an edge e is defined as the fraction of spanning trees of $\mathcal{G}$ (a tree-structure subgraph of $\mathcal{G}$ including all the nodes) that contains e. In simpler terms, the SC $s(e)$ measures how crucial the edge e is for $\mathcal{G}$ to remain connected, and hence, can be used to identify vulnerable edges in $\mathcal{G}$. Such a definition renders SC useful in infrastructure networks like electrical grids that require maintaining connectivity, i.e., stability and robustness against failures [3, 12]. In addition, SC also finds extensive applications in both practical and theoretical fields, including phylogenetics [40], graph sparsification [39], electric circuit analysis [10, 36], and combinatorial optimization [4, 18], to name a few.

Despite its usefulness, the problem of computing the SC values of all edges (AESC) in $\mathcal{G}$ remains challenging. To explain, let n and m be the number of nodes and edges in the graph $\mathcal{G}$, respectively. The graph $\mathcal{G}$ can have $O(n^n)$ spanning trees in the worst case. Hence, the exact AESC computation by enumerating all spanning trees is infeasible. The best-known algorithm [40] for the exact AESC computation is based on Kirchoff's matrix-tree theory [13, 42], which bears more than quadratic time $O(mn^{3/2})$, and thus, is prohibitive for massive graphs. To cope with this challenge, a series of approximation algorithms [14, 29, 34, 39] for AESC have been developed in recent years. Given an absolute error threshold ϵ, existing solutions focus on calculating an estimated SC $\hat{s}(e_{i,j})$ for each edge $e_{i,j}$ with at most ϵ absolute error in it. Although these methods allow us to trade result accuracy for execution time, they are still rather computationally expensive when $\mathcal{G}$ is sizable and ϵ is small. Spielman and Srivastava [39] propose to approximate AESC via its equivalent matrix-based definition, leading to $\tilde{O}\left(\frac{m}{\epsilon^2}\right)$ time in total. In the follow-up work [29], Mavroforakis et al. develop a fast implementation by incorporating a suite of heuristic optimizations that

considerably elevate its empirical efficiency without compromising its asymptotic performance. However, both methods become impractical when the matrices are high-dimensional and dense (i.e., n and m are large). To sidestep the shortcomings of matrices, Hayashi et al. [14] and Peng et al. [34] capitalize on the idea of using random walks for fast SC estimation, whereas these random walk-based techniques remain $\tilde{O}\left(\frac{m}{\epsilon^2}\right)$ time.

Motivated by the deficiencies of existing solutions, this paper presents two approximation algorithms for AESC: TGT and TGT+. At their hearts lie our improved bounds for random walk truncation, which are obtained through a rigorous theoretical analysis and novel exploitation of eigenvalues/eigenvectors pertaining to $\mathcal{G}$. Notably, compared to Peng et al.'s bound [34], our bound can achieve orders of magnitude of reduction in random walk length. Based thereon, TGT (Truncated Graph Traversal) conducts the graph traversal, i.e., deterministic version of random walks, from each node to probe nodes within the truncated length. In doing so, TGT outperforms the state of the arts in the case where the amount of random walks needed in them exceeds the graph size. To overcome the limitations of TGT on large graphs with high degrees, we further devise TGT+, whose idea is deriving rough estimations of AESC by graph traversals in TGT and refining the results using merely a handful of random walks. By including a greedy trade-off strategy and additional optimizations, we can orchestrate and optimize the entire TGT+ algorithm for enhanced practical efficiency. On the theoretical side, TGT+ propels the approximate AESC computation by improving the asymptotic performance to $\tilde{O}\left(\frac{n}{\epsilon^2}+m\right)$. Our extensive experiments on multiple benchmark graph datasets exhibit that TGT+ is often more than one order of magnitude faster compared to the state-of-the-art solutions while offering uncompromised or even superior result quality. Notably, on the Twitch dataset with 6.8 million edges, TGT+ can achieve 10^{-5} empirical error on average within 17 minutes for AESC, using a single CPU core, whereas the best competitor takes over 10 hours.

To summarize, we make the following contributions in this work:

- We derive an improved lower bound for the truncated random walk length and propose a first-cut solution TGT, which estimates AESC using the graph traversal operations. (Section 3)
- We develop an optimized solution TGT+, which integrates random walk sampling into TGT in an adaptive manner and improves over TGT in terms of practical efficiency. (Section 4)
- We compare our proposed solutions with 3 competitors on 5 real datasets and demonstrate the superiority of TGT+. (Section 5)

2 PRELIMINARIES

This section sets up the stage for our study by introducing basic notations, the formal problem definition of ϵ-approximate AESC computation, and the main competitors for AESC approximation.

2.1 Notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V}$ is a set of n nodes and $\mathcal{E}$ is a set of m edges. For each edge $e_{i,j} \in \mathcal{E}$, we say v_i and v_j are neighbors to each other, and we use $\mathcal{N}(v_i)$ to denote the set of neighbors of v_i, where the degree is $d(v_i) = |\mathcal{N}(v_i)|$. Throughout this paper, we use a boldface lower-case (resp. upper-case) letter $\vec{\mathbf{x}}$ (resp. $\mathbf{M}$) to represent a vector (resp. matrix), with its i-th element (resp. element at the i-th row and j-th column) denoted as $\vec{\mathbf{x}}[i]$ (resp. $\mathbf{M}[i,j]$). Given $\mathcal{G}$, we denote by $\mathbf{A}$ the adjacency matrix of $\mathcal{G}$, where $\mathbf{A}[i,j] = 1$ if $e_{i,j} \in \mathcal{E}$ and $\mathbf{A}[i,j] = 0$ otherwise. In addition, we let $\mathbf{D}$ be the degree diagonal matrix of $\mathcal{G}$ and the diagonal entry $\mathbf{D}[i,i] = d(v_i)$ for each node $v_i \in \mathcal{V}$. Let $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ be the random walk matrix (i.e., transition matrix) of $\mathcal{G}$, in which $\mathbf{P}[i,j] = \frac{1}{d(v_i)}$ if $e_{i,j} \in \mathcal{E}$ and $\mathbf{P}[i,j] = 0$ otherwise. Correspondingly, we denote $p_\ell(v_i, v_j) = \mathbf{P}^\ell[i,j]$, which can be interpreted as the probability of a random walk from node v_i visits node v_j at the ℓ-th hop, reflecting the proximity of nodes v_i, v_j. We refer to $p_\ell(v_i, v_j)$ as ℓ-hop TP (transition probability) of v_j w.r.t. v_i. In this paper, we assume $\mathcal{G}$ is connected and not bipartite. According to [31], the random walks over $\mathcal{G}$ are ergodic, i.e., $\lim_{\ell \to \infty} \mathbf{P}^\ell[i,j] = \frac{d(v_j)}{2m}\ \forall v_i, v_j \in \mathcal{V}$. Table 1 lists the notations that are frequently used in this paper.

Table 1: Frequently used notations.

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	An undirected graph $\mathcal{G}$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$.
n, m	The numbers of nodes and edges in $\mathcal{G}$.
$\mathcal{N}(v_i), d(v_i)$	The neighbor set and degree of node v_i.
$\mathbf{D}, \mathbf{P}$	The degree and transition matrices of $\mathcal{G}$, respectively.
$p_\ell(v_i, v_j)$	The ℓ-hop TP, i.e., $\mathbf{P}^\ell[i,j]$.
$s(e_{i,j}), \hat{s}(e_{i,j})$	The exact and estimated SC of edge $e_{i,j}$, respectively.
$\tau_{i,j}$	The truncated length for edge $e_{i,j}$ defined by Eq.(5).
ϵ, δ	The absolute error threshold and failure probability.
ω, γ	The number of eigenvectors and candidate nodes, respectively.

2.2 Problem Definition

*Definition 2.1 (*Spanning Centrality [40]*).* Given an undirected and connected graph $\mathcal{G}$, the SC $s(e_{i,j}) \in (0,1]$ of an edge $e_{i,j}$ is defined as the fraction of spanning trees of $\mathcal{G}$ that contains $e_{i,j}$.

Definition 2.1 presents the formal definition of SC. Recall that a *spanning tree* of graph $\mathcal{G}$ is a tree and spans over all nodes of $\mathcal{G}$. Intuitively, a high SC $s(e_{i,j})$ quantifies how crucial edge $e_{i,j}$ is for $\mathcal{G}$ to ensure connectedness. Since an edge $e_{i,j}$ with a high SC means that it appears in most spanning trees, all of them will fall apart once $e_{i,j}$ is removed from $\mathcal{G}$. In the extreme case where $s(e_{i,j}) = 1$, $\mathcal{G}$ will be disconnected when $e_{i,j}$ is excluded. To our knowledge, the state-of-the-art algorithm [40] for computing the exact AESC entails $O\left(mn^{\frac{3}{2}}\right)$ time, which is prohibitive for large graphs. Following previous works [14, 34], we focus on ϵ-approximate all-edge SC (AESC) computation, defined as follows. Particularly, we say an estimated SC $\hat{s}(e_{i,j})$ is ϵ-approximate if it satisfies Eq. (1).

Definition 2.2 (ϵ-Approximate AESC*).* Given an undirected and connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and an absolute error threshold $\epsilon \in (0,1)$, the ϵ-approximate AESC computation returns an estimated $\hat{s}(e_{i,j})$ for every edge $e_{i,j} \in \mathcal{E}$ such that

$$|\hat{s}(e_{i,j}) - s(e_{i,j})| \leq \epsilon. \tag{1}$$

2.3 State of the Arts

We briefly revisit four recent techniques for AESC computation: Fast-Tree [29], ST-Edge [14], MonteCarlo and MonteCarlo-C [34]. Other related works on SC will be reviewed later in Section 6.

Fast-Tree. Mavroforakis et al. [29] develop a fast implementation of [39] on the basis of the equivalence between SC and *effective resistance* (ER) [6] when node pairs are edges. To be more specific, as per [39], the ER of all edges are the diagonal elements of matrix $\mathbf{R} = \mathbf{B}\mathbf{L}^{\dagger}\mathbf{B}^{\top}$, where $\mathbf{B}$ and $\mathbf{L}^{\dagger}$ are the incidence matrix and the pseudoinverse of the Laplacian matrix of $\mathcal{G}$, respectively. Fast-Tree first employs random projections [1] to reduce high matrix dimensions and then deploys the SDD solver to solve the linear systems in the low-dimensional space, resulting in a linear time complexity of $O\left(\frac{m}{\epsilon^2}\log^2 n \log\left(\frac{1}{\epsilon}\right)\right)$. However, its practical efficiency is less than satisfactory on large graphs, as revealed by the experiments in [14].

ST-Edge. Based on Definition 2.1, Hayashi et al. [14] first sample a sufficient number of random spanning trees by Wilson's algorithm [48], and record the fraction of trees where edge $e_{i,j}$ appears as the estimated $\hat{s}(e_{i,j})$. As proved, the expected time to draw a spanning tree rooted at a random node v_r is $O\left(\sum_{v_i\in\mathcal{V}}\frac{d(v_i)}{m}\kappa(v_i,v_r)\right)$, where $\kappa(v_i,v_r)$ is the commute time between nodes v_i and v_r and is $O(m)$ [31]. Hence, to ensure the ϵ-approximate for estimated AESC values, ST-Edge runs in $O\left(\frac{m}{\epsilon^2}\log(\frac{m}{\delta})\right)$ time by sampling $O\left(\frac{1}{\epsilon^2}\log(\frac{m}{\delta})\right)$ spanning trees, rendering it costly when ϵ is small.

MonteCarlo and MonteCarlo-C. Very recently, Peng et al. [34] theoretically establish another equivalent definition of SC $s(e_{i,j})$:

$$s(e_{i,j}) = \sum_{\ell=0}^{\infty}\frac{p_\ell(v_i,v_i)}{d(v_i)} + \frac{p_\ell(v_j,v_j)}{d(v_j)} - \frac{p_\ell(v_i,v_j)}{d(v_j)} - \frac{p_\ell(v_j,v_i)}{d(v_i)}.$$

Thus, the problem is transformed into computing ℓ-hop TP values of every two nodes in $\{v_i, v_j\}$ for $0 \le \ell \le \infty$. The crux of MonteCarlo and MonteCarlo-C involves finding a truncated length τ for random walks, which ensures $|s_\tau(e_{i,j}) - s(e_{i,j})| \le \frac{\epsilon}{2}$, where

$$s_\tau(e_{i,j}) = \sum_{\ell=0}^{\tau}\frac{p_\ell(v_i,v_i)}{d(v_i)} + \frac{p_\ell(v_j,v_j)}{d(v_j)} - \frac{p_\ell(v_i,v_j)}{d(v_j)} - \frac{p_\ell(v_j,v_i)}{d(v_i)}. \quad (2)$$

Based thereon, MonteCarlo and MonteCarlo-C simulate random walks with length at most τ from v_i, v_j to approximate the ℓ-hop TP values such that $|\hat{s}(e_{i,j}) - s_\tau(e_{i,j})| \le \frac{\epsilon}{2}$ holds, connoting that $\hat{s}(e_{i,j})$ is ϵ-approximate. In particular, Peng et al. [34] provides the following bound for τ to ensure the ϵ-approximation

$$\tau \ge \left\lceil \log\left(\frac{4}{\epsilon-\epsilon\lambda}\right)\Big/\log\left(\frac{1}{\lambda}\right) - 1\right\rceil, \quad (3)$$

where λ is matrix $\mathbf{P}$'s second largest eigenvalue in absolute value.

The major distinction between MonteCarlo and MonteCarlo-C lies in the approach to computing ℓ-hop TP values. Specifically, MonteCarlo simply conducts random walks of length ℓ ($1 \le \ell \le \tau$) to approximate ℓ-hop TP values before aggregating them as the estimated SC. According to the Chernoff-Hoeffding bound, a total time complexity of $O\left(\frac{\tau^3\log(8\tau/\delta)}{\epsilon^2}\right)$ is needed to obtain an ϵ-approximate SC $\hat{s}(e_{i,j})$ with a success probability at least $1-\delta$. By contrast, MonteCarlo-C regards the ℓ-hop TP $p_\ell(v_i,v_j)$ ($1 \le \ell \le \tau$) as the collision probability of two random walks of length-$\frac{\ell}{2}$ from v_i and v_j, respectively, and then samples $40000 \times \left(\tau\sqrt{\tau\beta_\ell}/\epsilon + \tau^3\beta_\ell^{3/2}/\epsilon^2\right)$ length-$(\ell/2)$ random walks from respective nodes. The parameter β_ℓ is a constant depending on the graph structure, which is hard to compute in practice. Notice that both algorithms are originally designed for computing the ER of any node pair in $\mathcal{G}$, which overlook the unique property of edges and thus are not optimized for AESC computation. Moreover, they require an exorbitant amount of random walks due to the large τ (up to thousands when ϵ is small), significantly exacerbating the efficiency issues.

3 THE TGT ALGORITHM

In this section, we propose TGT, an iterative deterministic graph traversal approach to AESC processing based on the idea of computing the truncated SC (Eq. (2)) as in MonteCarlo. Particularly, TGT improves over MonteCarlo in two aspects. First and foremost, TGT offers significantly superior edge-wise lower bounds for truncated lengths by leveraging the well-celebrated theory of Markov chains [47] (Section 3.1). Further, TGT develops a deterministic graph traversal method to remedy the efficiency issue caused by substantial random walks needed in MonteCarlo (Section 3.2).

3.1 Improved Bounds for Truncated Lengths

Lemma 3.1 ([47]). *Given an undirected graph $\mathcal{G}$, let $1 = |\lambda_1| \ge |\lambda_2| \ge \cdots \ge |\lambda_n| \ge 0$ be the sorted absolute eigenvalues of $\mathbf{D}^{\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}$ and $\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \ldots, \boldsymbol{\varphi}_n$ be their corresponding normalized eigenvectors. Then, for any two nodes $v_i, v_j \in \mathcal{V}$ and any integer $\ell \ge 0$, we have*

$$\frac{p_\ell(v_i,v_j)}{d(v_j)} = \frac{p_\ell(v_j,v_i)}{d(v_i)} = \frac{1}{2m}\sum_{k=1}^{n}\vec{\mathbf{f}}_k[i]\cdot\vec{\mathbf{f}}_k[j]\cdot\lambda_k^\ell, \quad (4)$$

where $\vec{\mathbf{f}}_i = \sqrt{2m}\cdot\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\varphi}_i$ for $i = 1, 2, \ldots, n$, and $\vec{\mathbf{f}}_1$ is taken to be $\mathbf{1}$.

By Lemma 3.1, the ℓ-hop TP $p_\ell(v_i,v_j)$ can be computed based on the eigenvectors and eigenvalues of matrix $\mathbf{D}^{\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}$, and hence, the difference between $s_\tau(e_{i,j})$ and $s(e_{i,j})$ can be quantified via

$$|s_\tau(e_{i,j}) - s(e_{i,j})| = \left|\sum_{\ell=\tau+1}^{\infty}\frac{1}{2m}\sum_{k=1}^{n}(\vec{\mathbf{f}}_k[i] - \vec{\mathbf{f}}_k[j])^2\lambda_k^\ell\right|.$$

This suggests that we can utilize these eigenvectors and eigenvalues to determine a truncated length $\tau_{i,j}$ for edge $e_{i,j}$ so that $|s_\tau(e_{i,j}) - s(e_{i,j})| \le \frac{\epsilon}{2}$. Additionally, when $\ell = 1$ and $e_{i,j} \in \mathcal{E}$, we have $\frac{2m}{d(v_i)\cdot d(v_j)} = \sum_{k=1}^{n}\vec{\mathbf{f}}_k[i]\cdot\vec{\mathbf{f}}_k[j]\cdot\lambda_k$ as per Eq. (4). Given the above observations, we can establish an improved lower bound for the truncated length $\tau_{i,j}$ of each edge $e_{i,j}$, as shown in Theorem 3.2. For ease of exposition, we defer all proofs to [51].

Theorem 3.2. *Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $|s(e_{i,j}) - s_\tau(e_{i,j})| \le \epsilon$ holds for any edge $e_{i,j} \in \mathcal{E}$ when $\tau_{i,j}$ satisfies*

$$\tau_{i,j} \ge f(e_{i,j},\epsilon) \text{ and } \tau_{i,j} \equiv 1 \pmod 2 \quad (5)$$

$$f(e_{i,j},\epsilon) = \left\lceil \frac{\log\left(\frac{\frac{1}{d(v_i)}+\frac{1}{d(v_j)}-\frac{2}{d(v_i)\cdot d(v_j)}-\Upsilon}{(\epsilon-\Delta_t)\cdot(1-\lambda_\omega^2)}\right)}{\log\left(\frac{1}{|\lambda_\omega|}\right)} - 1\right\rceil, \quad (6)$$

Algorithm 1: CalTau

Data: Graph $\mathcal{G}$, $\{\lambda_1, \ldots, \lambda_\omega\}$, $\{\vec{\mathbf{f}}_1, \ldots, \vec{\mathbf{f}}_\omega\}$
Parameters: $e_{i,j}$, ϵ
Result: $\tau_{i,j}$

1 $\tau_{i,j} \leftarrow$ Eq. (6) with λ_2 and $\Delta_t = \Upsilon = 0$;
2 $\Upsilon \leftarrow$ Eq. (7); $t \leftarrow 1$;
3 **while** *true* **do**
4 $\Delta_t \leftarrow$ Eq. (8); $\tau' \leftarrow$ Eq. (6);
5 **if** $t \le \tau'$ **then** $\tau_{i,j} \leftarrow \tau'$; $t \leftarrow t + 2$;
6 **else break;**
7 **return** $\tau_{i,j} \leftarrow t$;

where

$$\Upsilon = \frac{1}{2m} \sum_{k=2}^{\omega-1} (\vec{\mathbf{f}}_k[i] - \vec{\mathbf{f}}_k[j])^2 \cdot (1 + \lambda_k), \tag{7}$$

$$\Delta_t = \frac{1}{2m} \sum_{k=2}^{\omega-1} (\vec{\mathbf{f}}_k[i] - \vec{\mathbf{f}}_k[j])^2 \cdot \frac{\lambda_k^{t+1}}{1 - \lambda_k}, \tag{8}$$

and t is an odd number ensuring $t \le \tau_{i,j}$.

Compared to Peng et al.'s τ in Eq. (3), our truncated length $\tau_{i,j}$ of edge $e_{i,j}$ in Theorem 3.2 is dependent to the degrees of nodes v_i, v_j, the ω-largest (typically $\omega = 128$) eigenvalues in absolute value and their corresponding eigenvectors of $\mathbf{D}^{\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}$, enabling up to orders of magnitude improvement in practice, as reported in Figure 1. Note that the eigenvalues and eigenvectors can be efficiently computed in the preprocessing stage (see Figure 4).

Algorithm 1 presents the pseudo-code of CalTau, an algorithm realizing the computation of $\tau_{i,j}$ on the basis of Theorem 3.2. Given graph $\mathcal{G}$, ω eigenvalues $\{\lambda_1, \ldots, \lambda_\omega\}$, eigenvectors $\{\vec{\mathbf{f}}_1, \ldots, \vec{\mathbf{f}}_\omega\}$, and parameters $e_{i,j}$, ϵ as inputs, CalTau initializes $\tau_{i,j}$ by Eq. (6) with λ_2 and $\Upsilon = \Delta_t = 0$ at Line 1, followed by setting $t = 1$ and calculating Υ according to Eq. (7) at Line 2. After that, CalTau increases t iteratively to search for the optimal t such that it is closest to but does not exceed Eq. (6), ensuring the validity of Theorem 3.2 (Lines 3–6). To be more precise, in each iteration, CalTau calculates a candidate truncated length τ' using Eq. (6), wherein Δ_t is obtained by Eq. (8) with current t. Next, if $t \le \tau'$, we update $\tau_{i,j}$ as τ' and increase t by 2 (Line 5). CalTau repeats the above procedure until the condition at Line 5 does not hold and returns t as $\tau_{i,j}$ at Line 7.

3.2 Complete Algorithm and Analysis

In light of Theorem 3.2, the problem of AESC computation in Definition 2.2 is reduced to computing the approximate SC $\hat{s}(e_{i,j}) = s_\tau(e_{i,j})$ as per Eq. (2) for each edge $e_{i,j} \in \mathcal{E}$. Unlike prior methods, TGT conducts a deterministic graph traversal from each node $v_i \in \mathcal{V}$ to compute ℓ-hop TP values $p_\ell(v_i, v_i)$ and $p_\ell(v_j, v_i)$ for $1 \le \ell \le \tau_{i,j}$ in an iterative manner, and aggregates them as

$$g_\tau(v_i, v_j) = \sum_{\ell=0}^{\tau} \frac{p_\ell(v_i, v_i)}{d(v_i)} - \sum_{\ell=0}^{\tau} \frac{p_\ell(v_j, v_i)}{d(v_i)} \tag{9}$$

to further derive $s_\tau(e_{i,j})$ by $g_\tau(v_i, v_j) + g_\tau(v_j, v_i)$. The pseudo-code of TGT is illustrated in Algorithm 2. In the course of graph traversal from each node $v_i \in \mathcal{V}$ (Lines 2–11), $p_0(v_i, v_j)$ and $g_\tau(v_i, v_j)$ $\forall v_j \in \mathcal{V}$

Algorithm 2: TGT

Data: Graph $\mathcal{G}$
Parameters: ϵ
Result: $s_\tau(e_{i,j})$ $\forall e_{i,j} \in \mathcal{E}$

1 **for** $v_i \in \mathcal{V}$ **do**
2 $p_0(v_j, v_i) \leftarrow 0$ $\forall v_j \in \mathcal{V} \setminus v_i$; $p_0(v_i, v_i) \leftarrow 1$;
3 $g_\tau(v_i, v_j) \leftarrow \frac{1}{d(v_i)}$ $\forall v_j \in \mathcal{N}(v_i)$;
4 $\tau_p \leftarrow \max_{v_j \in \mathcal{N}(v_i)}$ CalTau$(e_{i,j}, \epsilon)$;
5 **for** $\ell \leftarrow 1$ *to* τ_p **do**
6 $p_\ell(v_j, v_i) \leftarrow 0$ $\forall v_j \in \mathcal{V}$;
7 **for** $v_j \in \mathcal{V}$ *with* $p_{\ell-1}(v_j, v_i) > 0$ **do**
8 **for** $v_x \in \mathcal{N}(v_j)$ **do**
9 $p_\ell(v_x, v_i) \leftarrow p_\ell(v_x, v_i) + \frac{p_{\ell-1}(v_j, v_i)}{d(v_x)}$;
10 **for** $v_j \in \mathcal{N}(v_i)$ **do**
11 $g_\tau(v_i, v_j) \leftarrow g_\tau(v_i, v_j) + \frac{p_\ell(v_i, v_i)}{d(v_i)} - \frac{p_\ell(v_j, v_i)}{d(v_i)}$;
12 **for** $e_{i,j} \in \mathcal{E}$ **do** $s_\tau(e_{i,j}) \leftarrow g_\tau(v_i, v_j) + g_\tau(v_j, v_i)$;
13 **return** $s_\tau(e_{i,j})$ $\forall e_{i,j} \in \mathcal{E}$;

are initialized as Lines 2–3. Afterward, at Line 4, TGT invokes Algorithm 1 with absolute error ϵ and each edge $e_{i,j}$ that is adjacent to v_i. Let τ_p be the largest truncated length $\tau_{i,j}$ for all $v_j \in \mathcal{N}(v_i)$. Then, Algorithm 2 performs a τ_p-hop graph traversal originating from v_i (Lines 5–11). Specifically, at ℓ-th hop, TGT first sets $p_\ell(v_j, v_i) = 0$ $\forall v_j \in \mathcal{V}$. Subsequently, for each node v_j with non-zero $(\ell-1)$-hop TP $p_{\ell-1}(v_j, v_i)$, we scatter its value to its neighbors, i.e., visit each neighbor $v_x \in \mathcal{N}(v_j)$ by adding $\frac{p_{\ell-1}(v_j, v_i)}{d(v_x)}$ to $p_\ell(v_x, v_i)$. This operation essentially performs a sparse matrix-vector multiplication $p_\ell(\cdot, v_i) = \mathbf{P} \cdot p_{\ell-1}(\cdot, v_i)$. With $p_\ell(v_x, v_i)$ $\forall v_x \in \mathcal{V}$, TGT injects an increment of $\frac{p_\ell(v_i, v_i)}{d(v_i)} - \frac{p_\ell(v_j, v_i)}{d(v_i)}$ to $g_\tau(v_i, v_j)$ for each neighbor v_j of v_i. After the completion of all graph traversal operations, Algorithm 2 computes $s_\tau(e_{i,j})$ for each edge $e_{i,j}$ (Line 12) and returns them as the answers. The following theorem states the correctness and the worst-case time complexity of TGT.

Theorem 3.3. *Algorithm 2 returns ϵ-approximate* SC *values $s_\tau(e_{i,j})$ $\forall e_{i,j} \in \mathcal{E}$ using $O\left(nm \log\left(\frac{1}{\epsilon}\right)\right)$ time in the worst case.*

Notwithstanding its unsatisfying *worst-case* time complexity, by virtue of our improved lower bounds for truncated lengths in Section 3.1, the actual number of graph traversal operations from each node in Algorithm 2 (Lines 7–9) is far less than $O(m)$ when ϵ is non-diminutive, strengthening the superiority of TGT over MonteCarlo in empirical efficiency.

4 THE TGT+ ALGORITHM

Although TGT advances MonteCarlo in practical performance, we observe in our experiments that its cost is intolerable for massive graphs with high degrees. The reason is that the number of non-zero ℓ-hop TP values grows explosively at an astonishing rate till m (Lines 7–9 in Algorithm 2) on such graphs as ℓ increases, causing a quadratic computational complexity of $O(nm)$. The severity of the efficiency issue is accentuated in high-precision AESC computation,

i.e., ϵ is small. To alleviate this issue, we propose TGT+, an algorithm that significantly improves TGT in terms of both practical efficiency and asymptotic performance. The rest of this section proceeds as follows: Section 4.1 delineates the basic idea of TGT+, followed by several optimization techniques in Section 4.2. Finally, Section 4.3 describes the complete algorithm and analysis.

4.1 High-level Idea

Considering the sheer number of non-zero ℓ-hop TP values in TGT when ℓ is increased, we propose to calculate the TP values within $\tilde{\tau}$ (a small number) hops using TGT and harness random walks for the estimation of ℓ-hop TP with $\ell > \tilde{\tau}$. The rationale is that the amount of nodes in the vicinity of a given node v_i is often limited, and hence, can be efficiently covered by a graph traversal from v_i. On the contrary, far-reaching nodes from v_i can be multitudinous (up to millions in large graphs), where random walks suit the demand better by focusing on probing *important* nodes (i.e., with high TP values) in lieu of all of them. To fulfill the above-said idea, we first derive a truncated length $\tau_{i,j}$ such that $|s(e_{i,j}) - s_\tau(e_{i,j})| \leq \frac{\epsilon}{2}$ for each edge $e_{i,j} \in \mathcal{E}$. Next, the problem is computing an estimated SC $\hat{s}(e_{i,j})$ of each edge $e_{i,j}$ to ensure $|\hat{s}(e_{i,j}) - s_\tau(e_{i,j})| \leq \frac{\epsilon}{2}$ using graph traversals and random walks. To facilitate the seamless integration of random walks into TGT, we leverage the following crucial property of $g_\tau(v_i, v_j)$, a constituent part of SC $s_\tau(e_{i,j})$ as defined in Eq. (9).

Lemma 4.1. *For any integer τ and $\tilde{\tau}$ with $0 \leq \tilde{\tau} \leq \tau$,*

$$g_\tau(v_i, v_j) = g_{\tilde{\tau}}(v_i, v_j) + g_{\tilde{\tau}\to\tau}(v_i, v_j),$$

where

$$g_{\tilde{\tau}\to\tau}(v_i, v_j) = \sum_{v_x \in V} \frac{p_{\tilde{\tau}}(v_x, v_i)}{d(v_i)} \left(\sum_{\ell=1}^{\tau_{i,j}-\tilde{\tau}} p_\ell(v_i, v_x) - p_\ell(v_j, v_x) \right). \quad (10)$$

More concretely, given a cherry-picked length $\tilde{\tau}$ ($1 \leq \tilde{\tau} \leq \tau_{i,j}$), Lemma 4.1 implies that we can estimate $g_{\tilde{\tau}\to\tau}(v_i, v_j)$ by simulating random walks of lengths from 1 to $\tau_{i,j} - \tilde{\tau}$ after obtaining $g_{\tilde{\tau}}(v_i, v_j)$ and $p_{\tilde{\tau}}(\cdot, v_i)$ with TGT. Mathematically, if we conduct two length-$(\tau_{i,j} - \tilde{\tau})$ random walks W_i and W_j containing visited nodes from nodes v_i, v_j, respectively, we can define a random variable X as

$$X = \frac{1}{d(v_i)} \cdot \left(\sum_{v_x \in W_i} p_{\tilde{\tau}}(v_x, v_i) - \sum_{v_y \in W_j} p_{\tilde{\tau}}(v_y, v_i) \right). \quad (11)$$

By definition, the expectation $\mathbb{E}[X]$ of X is exactly $g_{\tilde{\tau}\to\tau}(v_i, v_j)$ in Eq. (10), indicating that X is an unbiased estimator of $g_{\tilde{\tau}\to\tau}(v_i, v_j)$. Suppose that the range of X is bounded by

$$|X| \leq \frac{\chi}{d(v_i)}. \quad (12)$$

Lemma 4.2 (Hoeffding's inequality [15]). *Let $Z_1, Z_2, \ldots, Z_{n_z}$ be independent random variables with Z_i ($\forall 1 \leq i \leq n_z$) is strictly bounded by the interval $[a_j, b_j]$. We define the empirical mean of these variables by $Z = \frac{1}{n_z} \sum_{i=1}^{n_z} Z_i$. Then,*

$$\mathbb{P}[|Z - \mathbb{E}[Z]| \geq \epsilon] \leq 2 \exp\left(-\frac{2n_z^2 \epsilon^2}{\sum_{j=1}^{n_z} (b_j - a_j)^2} \right).$$

Algorithm 3: CalChi

Data: Graph $\mathcal{G}$, edge (v_i, v_j), $p_{\tilde{\tau}}(\cdot, v_i)$
Parameters: γ
Result: χ

```
if γ > 0 then
    Identify C = {c_1, c_2, ..., c_γ} such that p_τ̃(c_1, v_i) ≥
      p_τ̃(c_2, v_i) ≥ ··· ≥ p_τ̃(c_γ, v_i) ≥ p_τ̃(v_l, v_i) ∀v_l ∈ V \ C;
    if ∃c_a, c_b ∈ C and (c_a, c_b) ∈ E then
        p̂_i ← max_{(c_a,c_b)∈E, ∀c_a,c_b∈C} p_τ̃(c_a, v_i) + p_τ̃(c_b, v_i);
    else p̂_i ← p_τ̃(c_1, v_i) + p_τ̃(c_γ, v_i) ;
    χ ← Eq. (17) with ρ_i = p̂_i;
else χ ← Eq. (15);
return χ;
```

It is straightforward to apply Hoeffding's inequality in Lemma 4.2 to derive the total number of random walks needed for the accurate estimation of $g_{\tilde{\tau}\to\tau}(v_i, v_j)$, i.e.,

$$R(e_{i,j}, \tau_{i,j} - \tilde{\tau}) = \frac{8\chi^2 \log(\frac{2m}{\delta})}{d(v_i)^2 \cdot \epsilon^2}. \quad (13)$$

In the subsequent section, we elucidate the determination of $\tilde{\tau}$ and χ so as to strike a good balance between graph traversal and random walks for optimized performance and meanwhile reduce the number $R(e_{i,j}, \tau_{i,j} - \tilde{\tau})$ of samples required.

4.2 Optimizations

4.2.1 **Adaptive determination of $\tilde{\tau}$.** Since the length of random walks for estimating $g_{\tilde{\tau}\to\tau}(v_i, v_j)$ $\forall v_j \in \mathcal{N}(v_i)$ is $\tau_{i,j} - \tilde{\tau}$, the computational overhead incurred by random walks from the given node v_i and its neighbors is hence bounded by

$$O\left(\sum_{v_j \in \mathcal{N}(v_i)} R(e_{i,j}, \tau_{i,j} - \tilde{\tau}) \cdot (\tau_{i,j} - \tilde{\tau}) \right),$$

which increases as $\tilde{\tau}$ decreases. Conversely, the graph traversal operations in TGT will reduce considerably when $\tilde{\tau}$ is lowered, as explained at the beginning of Section 4.1. In short, the length $\tilde{\tau}$ controls the trade-off between the deterministic graph traversal and random walks for each node $v_i \in \mathcal{V}$. Since it is hard to accurately quantify the graph traversal cost as a function regarding $\tilde{\tau}$ due to the complex graph structure, we make use of an adaptive strategy to determine $\tilde{\tau}$. More precisely, in the ℓ-th iteration of deterministic graph traversal (Lines 6-9 in Algorithm 2) originating from v_i, we set $\tilde{\tau} = \ell$ and switch the graph traversal to random walk simulations, if the following inequality holds:

$$\sum_{v_j \in \mathcal{V} \& p_\ell(v_j, v_i) != 0} d(v_j) > \sum_{v_j \in \mathcal{N}(v_i)} R(e_{i,j}, \tau_{i,j} - \ell), \quad (14)$$

where the l.h.s. and r.h.s. represent their respective costs for computing $(\ell + 1)$-hop TP values in the next iteration. The rationale of Eq. (14) is that we choose random walks rather than the graph traversal when the cost of the latter will outstrip the former.

Algorithm 4: TGT+

Data: Graph $\mathcal{G}$
Parameters: ϵ, δ, γ
Result: $\hat{s}(e_{i,j})\ \forall e_{i,j} \in \mathcal{E}$

1 **for** $v_i \in V$ **do**
2 $\quad \forall v_j \in \mathcal{N}(v_i)\ \tau_{i,j} \leftarrow$ CalTau$(e_{i,j}, \frac{\epsilon}{2})$; $\ell \leftarrow 0$;
$\quad$ Lines 3-4 are the same as Lines 2-3 in Algorithm 2;
5 $\quad$ **while** *Eq.* (14) *fails* **do**
$\quad\quad$ Lines 6-9 are the same as Lines 6-9 in Algorithm 2;
10 $\quad\quad \ell \leftarrow \ell + 1$;
11 $\quad \tilde{\tau} \leftarrow \ell$;
12 $\quad$ **for** $v_j \in \mathcal{N}(v_i)$ *such that* $\tau_{i,j} - \tilde{\tau} > 0$ **do**
13 $\quad\quad \chi \leftarrow$ CalChi$(\mathcal{G}, v_i, v_j, p_{\tilde{\tau}}(\cdot, v_i), \gamma)$;
14 $\quad\quad n_r \leftarrow R(e_{i,j}, \tau_{i,j} - \tilde{\tau})$ in Eq. (13);
15 $\quad\quad$ **for** $i \leftarrow 1$ *to* n_r **do**
16 $\quad\quad\quad$ Simulate two length-$(\tau_{i,j} - \tilde{\tau})$ random walks W_i, W_j from v_i, v_j, respectively;
17 $\quad\quad\quad \hat{g}_\tau(v_i, v_j) \leftarrow \hat{g}_\tau(v_i, v_j) + \frac{X}{n_r}$ with $X =$ Eq. (11);
18 **for** $e_{i,j} \in \mathcal{E}$ **do** $\hat{s}(e_{i,j}) \leftarrow \hat{g}_\tau(v_i, v_j) + \hat{g}_\tau(v_j, v_i)$;
19 **return** $\hat{s}(e_{i,j})\ \forall e_{i,j} \in \mathcal{E}$;

4.2.2 **Effective refinement of χ.** By the definition of X in Eq. (11), one may simply set χ as follows:

$$\chi = 2 \cdot (\tau - \tilde{\tau}) \cdot \left(\max_{v_x \in V} p_{\tilde{\tau}}(v_x, v_i) - \min_{v_y \in V} p_{\tilde{\tau}}(v_y, v_i) \right), \quad (15)$$

where $p_{\tilde{\tau}}(v_x, v_i)\ \forall v_x \in \mathcal{V}$ is known from TGT. Unfortunately, the empirical values of r.h.s. of Eq. (15) are usually innegligible on real graphs, resulting in a considerable number of random samples according to Eq. (13). Intuitively, given that W_i and W_j are the random walks from two adjacent nodes v_i, v_j (i.e., $e_{i,j} \in \mathcal{E}$), respectively, the nodes on W_i and W_j are highly overlapped. As a consequence, the difference between $\sum_{v_x \in W_i} p_{\tilde{\tau}}(v_x, v_i)$ and $\sum_{v_y \in W_j} p_{\tilde{\tau}}(v_y, v_i)$ in Eq. (11) (i.e., χ) would be insignificant in practice. Inspired by the aforementioned observation, we can establish the following lower and upper bounds for $\sum_{v_x \in W_i} p_{\tilde{\tau}}(v_x, v_i)$.

Lemma 4.3. *Let W_i be any length-ℓ ($\ell \geq 1$) random walk over $\mathcal{G}$ starting from node v_i and ρ_i be*

$$\rho_i = \max_{e_{x,y} \in \mathcal{E}} \{p_{\tilde{\tau}}(v_x, v_i) + p_{\tilde{\tau}}(v_y, v_i)\}. \quad (16)$$

Then, we have $LB(v_i, \ell) \leq \sum_{v_x \in W_i} p_{\tilde{\tau}}(v_x, v_i) \leq UB(v_i, \ell)$, where the lower and upper bounds $LB(v_i, \ell)$, $UB(v_i, \ell)$ are defined by

$$LB(v_i, \ell) = \min_{v_l \in \mathcal{N}(v_i)} \{p_{\tilde{\tau}}(v_l, v_i)\} + (\ell - 1) \cdot \min_{v_l \in \mathcal{V}} \{p_{\tilde{\tau}}(v_l, v_i)\}, \text{ and}$$

$$UB(v_i, \ell) = \max_{v_l \in \mathcal{N}(v_i)} \left\{ \frac{p_{\tilde{\tau}}(v_l, v_i)}{2} \right\} + \max_{v_l \in \mathcal{V}} \left\{ \frac{p_{\tilde{\tau}}(v_l, v_i)}{2} \right\} + \frac{(\ell - 1) \cdot \rho_i}{2}.$$

Using Lemma 4.3, a refined χ is at hand:

$$\chi = UB(v_i, \tau_{i,j} - \tilde{\tau}) + UB(v_j, \tau_{i,j} - \tilde{\tau}) - LB(v_i, \tau_{i,j} - \tilde{\tau}) - LB(v_j, \tau_{i,j} - \tilde{\tau}). \quad (17)$$

It is worth mentioning that $LB(v_i, \ell)$ and the first two terms in $UB(v_i, \ell)$ can be efficiently computed without sorting all the n nodes, since the actual number of non-zero entries in $p_{\tilde{\tau}}(\cdot, v_i)$ is limited due to our fine-tuned $\tilde{\tau}$, as remarked earlier. Therefore, the critical challenge to realize the derivation of the improved χ in Eq. (17) arises from the computation of ρ_i in Eq. (16), which incurs a high cost of $O(m \log m)$ if we search for the optimal edge $e_{x,y}$ ensuring Eq. (16) from $\mathcal{E}$ in a brute-force fashion. To tackle this problem, we propose a subroutine CalChi in Algorithm 3, which computes ρ_i for χ in a cost-effective manner, without jeopardizing its correctness. More specifically, instead of inspecting all the m edges in $\mathcal{G}$, CalChi first identifies a set $C = \{c_1, c_2, \cdots, c_\gamma\}$ of nodes from $\mathcal{V}$ with γ (γ is a small constant) largest $\tilde{\tau}$-hop TP values to v_i, in other words $p_{\tilde{\tau}}(c_1, v_i) \geq p_{\tilde{\tau}}(c_2, v_i) \geq \cdots \geq p_{\tilde{\tau}}(c_\gamma, v_i) \geq p_{\tilde{\tau}}(v_l, v_i)\ \forall v_l \in \mathcal{V} \setminus C$ (Line 2). After that, CalChi checks if any two nodes in C form an edge. If C does not contain such two nodes $(c_a, c_b) \in \mathcal{E}$, we set ρ_i's upper bound $\hat{\rho}_i$ to $p_{\tilde{\tau}}(c_1, v_i) + p_{\tilde{\tau}}(c_\gamma, v_i)$, otherwise we use the largest $p_{\tilde{\tau}}(c_a, v_i) + p_{\tilde{\tau}}(c_b, v_i)$ among all edges of $C \times C$, which is ρ_i itself (Lines 3-5). The rationale is that when no edges exists in $C \times C$, at least an endpoint v_y of the desired edge $e_{x,y}$ is outside C, meaning $p_{\tilde{\tau}}(v_y, v_i) \leq p_{\tilde{\tau}}(c_\gamma, v_i)$. In the meantime, another endpoint v_x of $e_{x,y}$ satisfies $p_{\tilde{\tau}}(v_x, v_i) \leq p_{\tilde{\tau}}(c_1, v_i)$. Accordingly, $p_{\tilde{\tau}}(c_1, v_i) + p_{\tilde{\tau}}(c_\gamma, v_i)$ can serve as an upper bound of ρ_i in this case. Eventually, CalChi calculates χ according to Eq. (17) by replacing ρ_i by its upper bound $\hat{\rho}_i$ (Line 6). Particularly, when $\gamma = 0$, CalChi degrades to computing χ by Eq. (15).

Table 2: Algorithms for ϵ-approximate AESC computation.

Algorithm	Time Complexity
Fast-Tree [29]	$O\left(\frac{m}{\epsilon^2} \log\left(\frac{1}{\epsilon}\right) \log\left(\frac{n}{\delta}\right)\right)$
ST-Edge [14]	$O\left(\frac{m}{\epsilon^2} \log\left(\frac{m}{\delta}\right)\right)$
MonteCarlo [34]	$O\left(\frac{m}{\epsilon^2} \log^4\left(\frac{1}{\epsilon}\right) \log\left(\frac{m}{\delta}\right)\right)$
MonteCarlo-C [34]	$O\left(\frac{m}{\epsilon^2} \log^4\left(\frac{1}{\epsilon}\right) \log\left(\frac{m}{\delta}\right)\right)$
Our TGT+	$O\left(\frac{1}{\epsilon^2} \log^3\left(\frac{1}{\epsilon}\right) \log\left(\frac{m}{\delta}\right) \cdot \sum_{v_i \in \mathcal{V}} \frac{1}{d(v_i)} + m\right)$

4.3 Complete Algorithm and Analysis

Algorithm 4 summarizes the procedure of TGT+, which begins with computing $g_\tau(v_i, v_j)$ for each node $v_i \in \mathcal{V}$ and each of its neighbors v_j as in TGT. Specifically, for each node $v_i \in \mathcal{V}$, TGT+ first computes $\tau_{i,j}$ for each neighbor v_j of v_i by taking $\epsilon/2$ as input (Line 2). Subsequently, TGT+ carries out graph traversals as illustrated in TGT for the computation of $g_{\tilde{\tau}}(v_i, v_j)\ \forall v_j \in \mathcal{N}(v_i)$ and $p_{\tilde{\tau}}(\cdot, v_i)$ (Lines 3-10). The iterative process of the graph traversal terminates when Eq. (14) holds and Algorithm 4 then proceeds to sampling random walks for each neighboring node v_j of v_i whose $g_{\tilde{\tau}}(v_i, v_j)$ is insufficiently accurate, i.e., $\tau_{i,j} > \tilde{\tau}$ (Line 12). In particular, TGT+ first invokes Algorithm 3 to obtain the refined χ (Line 13) before determining the number of random walks n_r at Line 14. Afterwards, TGT+ generates n_r length-$(\tau_{i,j} - \tilde{\tau})$ random walks W_i, W_j from nodes v_i, v_j, respectively (Lines 15-16). After each sampling, it increases $\hat{g}_\tau(v_i, v_j)$ by $\frac{X}{n_r}$, where X is a random variable based on Eq. (11) (Line 17). In the end, TGT+ computes $\hat{s}(e_{i,j}) = \hat{g}_\tau(v_i, v_j) + \hat{g}_\tau(v_j, v_i)$ for each edge $e_{i,j} \in \mathcal{E}$ and outputs them as the SC estimations. The following theorem expresses the correctness and complexity of it.

Theorem 4.4. *For any $\epsilon, \delta \in (0,1)$, Algorithm 4 returns the ϵ-approximate* SC *$\hat{s}(e_{i,j})$ $\forall e_{i,j} \in \mathcal{E}$ with the probability at least $1-\delta$, using $O\left(\frac{1}{\epsilon^2} \cdot \log^3(\frac{1}{\epsilon}) \cdot \log(\frac{m}{\delta}) \cdot \sum_{v_i \in \mathcal{V}} \frac{1}{d(v_i)} + m\right)$ expected time.*

The rationale of TGT+'s correctness has been explained in Section 4.1. For the time complexity, it comes from (i) the graph traversal in Lines 2-11, (ii) the random walk in Lines 12-17, and (iii) accessing each neighbor of each node in Line 18 with a total time of $O(m)$. With the adaptive switch condition in Eq.(14), TGT+ ensures that the cost of the first part does not exceed the second part, resulting in the cost of both is $O\left(\sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} 2 \cdot \tau_{i,j} \cdot R(e_{i,j}, \tau_{i,j})\right)$, where $R(e_{i,j}, \tau_{i,j}) = O\left(\frac{\tau_{i,j}^2 \cdot \log(\frac{m}{\delta})}{d(v_i)^2 \cdot \epsilon^2}\right)$ as Eq. (13) and $\tau_{i,j} = O\left(\log(\frac{1}{\epsilon})\right)$ as Line 1 of Algorithm 1. Hence, the time complexity of TGT+ turns to the formula in Theorem 4.4. Table 2 compares the expected time of the randomized algorithm for ϵ-approximate AESC computation. Notably, TGT+ eliminates an m term in its bound, where the term $\sum_{v_i \in \mathcal{V}} \frac{1}{d(v_i)}$ can be simplified as $O(n)$ or even $O(n/\log n)$ using Kantorovich inequality on *scale-free* graphs with $m/n = O(\log n)$, manifesting the superiority of TGT+ over existing solutions.

5 EXPERIMENTS

In this section, we introduce the experimental settings, followed by evaluating our truncation bound and showing the performance of the proposed TGT+. At last, we analyze the sensitivity of constants γ and ω in TGT+. All experiments are conducted on a Linux machine with Intel Xeon(R) Gold 6240@2.60GHz CPU and 377GB RAM in single-thread mode. None of the experiments need anywhere near all the memory. Due to space limitations, we refer interested readers to [51] for the scalability test.

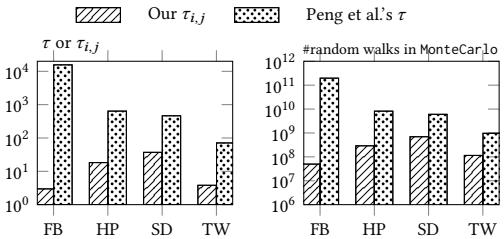
5.1 Experimental Setups

Datasets and groundtruths. We include 5 different types of real undirected graphs at different scales, whose statistics are shown in Table 3. All datasets are collected from SNAP [21] and used as datasets in previous works [14, 29, 34]. For each graph, we generate groundtruth AESC by first computing $\mathbf{P}^\tau$ with $0 \le \tau \le 1000$ in parallel and then assembling them into SC by Eq.(2).

Methods and parameters. We compare TGT and TGT+ with three recent algorithms for AESC: ST-Edge [14], MonteCarlo [34] and MonteCarlo-C [34], as introduced in Section 2.3. We exclude FastTree [29] from the competitors since it mainly offers relative approximation guarantees and is empirically shown significantly inferior to ST-Edge in [14]. Amid them, MonteCarlo and MonteCarlo-C are adapted for AESC computation, and the detailed modification is explained in Section 5.2. For the randomized algorithms TGT+, ST-Edge, MonteCarlo, and MonteCarlo-C, we follow [14] and set failure probability $\delta = 1/n$. Regarding MonteCarlo-C, we adopt the heuristic settings of β_i as suggested in [34], since they are unknown. For the proposed TGT and TGT+, we set the constants $\gamma = 10$ and $\omega = 128$, unless otherwise specified. For a fair comparison, all tested algorithms are implemented in C++ and compiled by g++ 7.5 with −O3 optimization. For reproducibility, the source code is available at: https://github.com/jeremyzhangsq/AESC.

Table 3: Datasets.

Name	#nodes	#edges
Facebook [30]	4,039	88,234
HepPh [20]	34,401	420,784
Slashdot [22]	77,360	469,180
Twitch [35]	168,114	6,797,557
Orkut [50]	3,072,441	117,185,082



(a) truncated lengths **(b) the number of walks**

Figure 1: Our $\tau_{i,j}$ vs. Peng et al.'s τ.

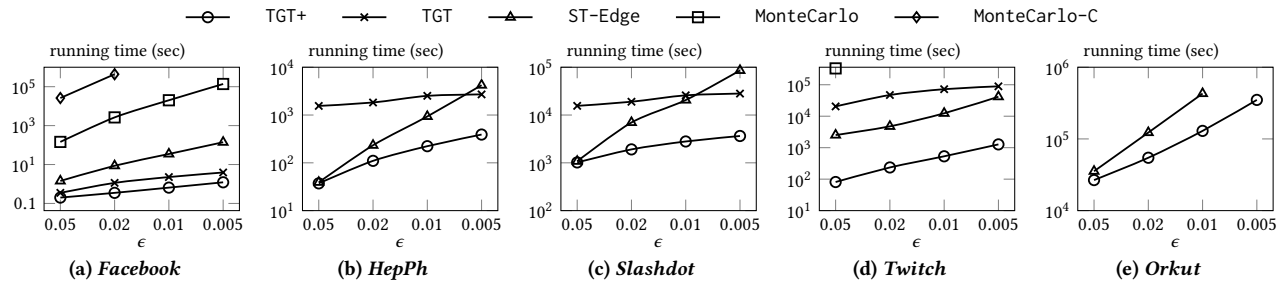
5.2 Empirical Study of $\tau_{i,j}$ and τ

In the first set of experiments, we evaluate the performance of the proposed truncated length in Section 3.1. Figure 1(a) reports the average of each edge $e_{i,j}$'s $\tau_{i,j}$ derived from Algorithm 1 and Peng et al.'s τ computed by Eq. (3) when $\epsilon = 0.01$ on Facebook (FB), HepPh (HP), Slashdot (SD) and Twitch (TW). We observe that our $\tau_{i,j}$ can significantly improve Peng et al.'s τ on all tested graphs. Notably, our $\tau_{i,j}$ is up to 3 orders of magnitude better than Peng et al.'s τ. Correspondingly, the computational overhead of MonteCarlo can be reduced by replacing Peng et al.'s τ with our $\tau_{i,j}$. We take MonteCarlo as an example to demonstrate its superiority. Figure 1(b) reports the average number of random walks, where the major overhead of MonteCarlo stems from, for estimating each SC. Akin to the observation from Figure 1(a), MonteCarlo with our truncated lengths $\tau_{i,j}$ requires at most 3 orders of magnitude fewer random walks than Peng et al.'s τ.

It is worth noting that MonteCarlo and MonteCarlo-C are designed for computing SC of a single node pair. Although our $\tau_{i,j}$ can remarkably cut down the number of random walks for an edge, there remain redundant random walks if invoking them for all edges individually. Hence, we further adapt MonteCarlo and MonteCarlo-C for efficient AESC computation by following the idea in TGT and TGT+ that iterate over each node. To summarize, for each node v_i, the adapted MonteCarlo and MonteCarlo-C first compute the largest τ_p among v_i's local neighborhood as Line 2 in Algorithm 2, and then compute the number of samplings based on τ_p. In the end, these extensions generate the corresponding random walks from v_i and estimate $s_\tau(e_{i,j})$ for each $v_j \in \mathcal{N}(v_i)$.

5.3 Performance Evaluation

In the second set of experiments, we evaluate the performance of each approach in terms of efficiency and accuracy. For efficiency, we report the average running times (measured in wall-clock time) after all input data are loaded into the memory. For accuracy, we measure the actual average absolute error of the estimated all edge SC returned by each algorithm on each dataset. We run each algorithm with various ϵ in $\{0.05, 0.02, 0.01, 0.005\}$, and report the



(a) ***Facebook*** (b) ***HepPh*** (c) ***Slashdot*** (d) ***Twitch*** (e) ***Orkut***

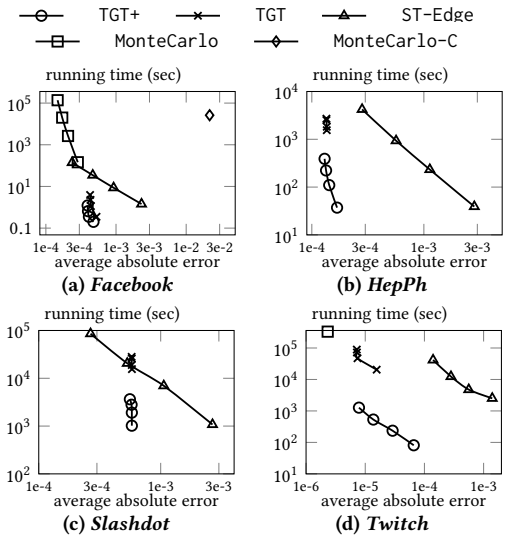
Figure 2: Running time of each algorithm by varying ϵ.

average evaluation score after repeating 3 trials. A method is excluded if it fails to report the result within 120 hours.

5.3.1 **Running time.** We first compare TGT+ with TGT and other competitors in terms of efficiency. Figure 2 reports each solution's running time for solving AESC with various ϵ settings. Benefiting from the truncation bound and the seamless combination of TGT and random walk samplings, the proposed TGT+ outperforms all competitors on all tested graphs and ϵ settings. Most notably, TGT+ improves the best competitor ST-Edge by at least one order of magnitude on Facebook and Twitch. We find that the improvement achieved by TGT+ becomes more remarkable as ϵ decreases. For example, TGT+ is 10.8× (resp. 23.5×) faster than ST-Edge on HepPh (resp. Slashdot) when $\epsilon = 0.005$. In addition, on the Orkut graph with 117 million edges, TGT+ is the only algorithm that can finish under all ϵ settings, demonstrating the scalability of our algorithm.

To evaluate the performance of the combination in TGT+, we next compare TGT+, TGT, MonteCarlo, and MonteCarlo-C, as all of them employ the edge-wise τ for the sake of fairness. As shown in Figure 2, MonteCarlo and MonteCarlo-C fail to return results within the allowed time limit in most cases. In particular, MonteCarlo can only terminate within 120 hours on Facebook and on Twitch when $\epsilon = 0.05$. The running time of MonteCarlo-C is even worse and is only feasible on Facebook with $\epsilon = 0.02, 0.05$. In contrast, TGT speeds up MonteCarlo and MonteCarlo-C by at least 2 orders of magnitude, demonstrating the superiority of the graph traversal in Section 3.2. However, TGT is still rather costly in comparison to TGT+. Specifically, TGT is only comparable to TGT+ on Facebook and is inferior to TGT+ on the rest graphs. For instance, TGT costs at least 1 and 2 orders of magnitude more time than TGT+ on Slashdot and Twitch, respectively, demonstrating the effectiveness of integrating deterministic traversal with randomized simulations in TGT+. To explain, the truncated length $\tau_{i,j}$ on the rest graphs (e.g., HepPh and Slashdot) is longer than that on Facebook, substantially increasing the overhead incurred by the graph traversal.

5.3.2 **Accuracy.** We next report the tradeoffs between average absolute error (in x-axis) and running time (in y-axis) in Figure 3. The results are sorted in the ascending order of ϵ, and the error-time curve closer to the lower left corner indicates a better performance. As shown, the overall observation is that TGT+ outperforms all competitors by achieving lower errors with less running time on all graphs. In particular, TGT+ achieves an average absolute error of 1.37E-05 with a time of 532 seconds on Twitch, while the closest solution TGT achieves an average absolute error of 1.56E-05 using



(a) ***Facebook*** (b) ***HepPh*** (c) ***Slashdot*** (d) ***Twitch***

Figure 3: Tradeoffs between running time and absolute error.

over 20,000 seconds (≈ 5.6 hours). Regarding TGT, we observe that, under the same ϵ setting, the actual absolute error of TGT is slightly smaller than TGT+. This is as expected since TGT leverages the largest $\tau_p = \max_{v_j \in \mathcal{N}(v_i)} \tau_{i,j}$ as the maximal iteration for v_i. In other words, the SC value for the edge $e_{i,j}$ is overestimated if $\tau_{i,j} < \tau_p$. Furthermore, we notice that the absolute error of MonteCarlo-C is an order of magnitude larger than the closest competitor ST-Edge on Facebook. This is due to that the heuristic settings [34] for input parameters β_i do not ensure the returned values are ϵ-approximate.

5.3.3 **Preprocessing time.** Recall that TGT, TGT+, MonteCarlo, and MonteCarlo-C rely on the eigen decomposition of the matrices pertaining to $\mathcal{G}$, e.g., $\mathbf{P}$ and $\mathbf{D}^{\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}$, in the preprocessing stage. In particular, MonteCarlo and MonteCarlo-C require the second largest eigenvalues, while TGT and TGT+ need the $\omega = 128$ largest eigenvalues and eigenvectors. Fortunately, by virtue of well-established techniques [33] and tools [19] for large-scale eigen decomposition, we can quickly obtain the desired eigenvalues and eigenvectors. Figure 4 reports the preprocessing time for TGT+ and vanilla MonteCarlo. As expected, the preprocessing time of TGT+ is comparable to MonteCarlo. In addition, compared to the running time for AESC displayed in Figure 2, the preprocessing costs are insignificant. For instance, the preprocessing time of TGT+ is about 45

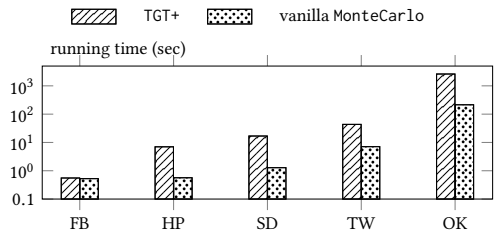
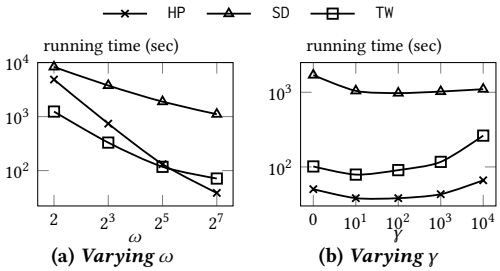


Figure 4: Preprocessing time of TGT+ and vanilla MonteCarlo.



(a) *Varying* ω **(b) *Varying* γ**

Figure 5: Varying constants in TGT+.

minutes for the Orkut (OK) graph with 117 million edges, whereas the running time is at least 7 hours. Notice that this preprocessing step only needs to be conducted once for a graph.

5.4 Parameter Analysis

In the last set of experiments, we study the effects of TGT+'s constant: (i) ω, the number of largest eigenvalues and eigenvectors of $\mathbf{D}^{\frac{1}{2}}\mathbf{P}\mathbf{D}^{-\frac{1}{2}}$ in Algorithm 1; (ii) γ, the number of candidates in Algorithm 3. In the sequel, we set $\epsilon = 0.05$ unless otherwise specified.

5.4.1 **Varying ω.** Figure 5(a) reports the running time of TGT+ by setting $\gamma = 10$ and varying $\omega \in \{2, 8, 32, 128\}$ on HepPh (HP), Slashdot (SD) and Twitch (TW). As expected, TGT+ costs less running times as more eigenvalues and eigenvectors are exploited. Specifically, the improvement of ω is more remarkable on HepPh, where the running time of $\omega = 128$ is about 126× faster than $\omega = 2$. Besides, the running time of TGT+ achieves about 8× and 17× improvements by varying ω from 2 to 128 on SD and TW, respectively.

5.4.2 **Varying γ.** Figure 5(b) reports the running time of TGT+ by fixing $\omega = 128$ and picking $\gamma \in \{0, 10^1, 10^2, 10^3, 10^4\}$ for the computation of χ on HP, SD and TW. We observe that the running time of TGT+ first decreases and then increases as more candidates are considered. To explain, when γ is too small, the upper bound χ for $|X|$ is too loose, rendering more random walks generated; when γ is too large, Algorithm 3 incurs more computational overhead. For example, TGT+ with $\gamma = 0$ costs about 2× more time than that with $\gamma = 10$ on HP and SD. Meanwhile, TGT+ with $\gamma = 10,000$ costs over 3× more time than that with $\gamma = 10$ on TW.

6 ADDITIONAL RELATED WORK

In the sequel, we review existing studies germane to our work.

Spanning centrality. Apart from the methods discussed in Section 2.3, there exist several techniques for estimating SC (i.e., effective resistance (ER)). Fouss et al. [8] propose to calculate the exact ER values for all pairs of nodes in the input graph G by first computing the Moore-Penrose pseudoinverse $\mathbf{L}^+$ of the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and then taking $\mathbf{L}^+[i,i] + \mathbf{L}^+[j,j] - \mathbf{L}^+[i,j] - \mathbf{L}^+[j,i]$ as the ER for any node pair $v_i, v_j \in V$. Teixeira et al. [40] and Mavroforakis et al. [29] utilize the random projection and symmetric diagonally dominant solvers to approximate the SC for all edges. After that, Jambulapati and Sidford [17] aime to compute the sketches of $\mathbf{L}$ and its pseudoinverse $\mathbf{L}^+$, and propose an algorithm for estimating ER values for all possible node pairs in $O(n^2/\epsilon)$ time. Besides MonteCarlo and MonteCarlo-C, Peng et al. [34] also propose two solutions by leveraging the connection between ER and the commute time [31] These works all focus on the ϵ-multiplicative approximation and are beyond the scope of this paper.

Personalized PageRank. Another line of related work is personalized PageRank (PPR). In past decades, the efficient computation of PPR has been extensively studied in a plethora of works [2, 7, 16, 23–28, 38, 43–46, 49]. Among them, some recent approaches [16, 23, 24, 27, 28, 38, 43–46] also leveraged the idea of combining the deterministic graph traversal [2, 7, 26] with random walk simulations. At first glance, it seems that we can simply adapt and extend these techniques for computing ϵ-approximate all edge SC. However, SC is much more sophisticated than PPR. This is due to that they are defined according to two inherently different types of random walks. More concretely, PPR leverages the one called *random walk with restart* (RWR) [41], which would stop at each visited node with a certain probability during the walk. In contrast, SC relies on simple random walks of various fixed lengths (from 1 to ∞), indicating that the walk in SC will not terminate as early as RWR does. Motivated by this, a linchpin of this work is a personalized truncation for the maximum random walk length. Correspondingly, the combination of graph traversal and random walks becomes more challenging.

7 CONCLUSION

In this paper, we propose two approximation algorithms for AESC computation. Our contributions consist of (i) enhanced lower bounds for truncating random walks, (ii) an algorithmic framework integrating the deterministic graph traversal with random walk sampling, and (iii) several carefully-designed optimization techniques for increasing efficiency. Our experiments on five real datasets demonstrate that our proposed algorithm significantly outperforms existing solutions in terms of practical efficiency without compromising theoretical and empirical accuracy. In the future, we plan to study AESC computation with relative error guarantees as well as under multithreading environments.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China (Grant No. U22B2060), by Shenzhen Fundamental Research Program (Grant No. 20220815112848002), by Guangzhou Municipal Science and Technology Bureau (Grant No. 2023A03J0667), and by HKUST(GZ) and HKBU under Start-up Grants. Bo Tang is also affiliated with the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China.

REFERENCES

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *PODS*. 274–281.

[2] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS*. 475–486.

[3] Andrey Bernstein, Daniel Bienstock, David Hay, Meric Uzunoglu, and Gil Zussman. 2014. Power grid vulnerability to geographically correlated failures—Analysis and control implications. In *INFOCOM*. 2634–2642.

[4] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. 2011. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC*. 273–282.

[5] Alfredo Cuzzocrea, Alexis Papadimitriou, Dimitrios Katsaros, and Yannis Manolopoulos. 2012. Edge betweenness centrality: A novel algorithm for QoS-based topology control over wireless sensor networks. *Journal of Network and Computer Applications* 35, 4 (2012), 1210–1217.

[6] Peter G Doyle and J Laurie Snell. 1984. *Random walks and electric networks*. Vol. 22. American Mathematical Soc.

[7] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.* 2, 3 (2005), 333–358.

[8] Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *TKDE* 19, 3 (2007), 355–369.

[9] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

[10] Arpita Ghosh, Stephen Boyd, and Amin Saberi. 2008. Minimizing effective resistance of a graph. *SIAM review* 50, 1 (2008), 37–66.

[11] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *PNAS* 99, 12 (2002), 7821–7826.

[12] Linqi Guo, Chen Liang, and Steven H Low. 2017. Monotonicity properties and spectral characterization of power redistribution in cascading failures. *SIGMETRICS* 45, 2 (2017), 103–106.

[13] John M Harris, Jeffry L Hirst, and Michael J Mossinghoff. 2000. *Combinatorics and graph theory*. Springer Science & Business Media.

[14] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2016. Efficient Algorithms for Spanning Tree Centrality. In *IJCAI*, Vol. 16. 3733–3739.

[15] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (1963), 13–30.

[16] Guanhao Hou, Xingguang Chen, Sibo Wang, and Zhewei Wei. 2021. Massively parallel algorithms for personalized PageRank. *PVLDB* 14, 9 (2021), 1668–1680.

[17] Arun Jambulapati and Aaron Sidford. 2018. Efficient $\tilde{O}(n/\epsilon)$ Spectral Sketches for the Laplacian and its Pseudoinverse. In *SODA*. 2487–2503.

[18] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. 2014. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *SODA*. 217–226.

[19] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. 1998. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM.

[20] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*. 177–187.

[21] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[22] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[23] Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. 2020. Index-free approach with theoretical guarantee for efficient random walk with restart query. In *ICDE*. 913–924.

[24] Wenqing Lin. 2019. Distributed algorithms for fully personalized pagerank on large graphs. In *WWW*. 1084–1094.

[25] Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. 2016. Personalized pagerank estimation and search: A bidirectional approach. In *WSDM*. 163–172.

[26] Peter Lofgren and Ashish Goel. 2013. Personalized pagerank to a target node. *arXiv* (2013).

[27] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. 2019. Baton: Batch one-hop personalized pageranks with efficiency and accuracy. *TKDE* 32, 10 (2019), 1897–1908.

[28] Siqiang Luo, Xiaokui Xiao, Wenqing Lin, and Ben Kao. 2019. Efficient batch one-hop personalized pageranks. In *ICDE*. 1562–1565.

[29] Charalampos Mavroforakis, Richard Garcia-Lebron, Ioannis Koutis, and Evimaria Terzi. 2015. Spanning edge centrality: Large-scale computation and applications. In *WWW*. 732–742.

[30] Julian J McAuley and Jure Leskovec. 2012. Learning to discover social circles in ego networks.. In *NeurIPS*. 548–56.

[31] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized algorithms*. Cambridge University Press.

[32] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[33] Beresford N Parlett. 1998. *The symmetric eigenvalue problem*. SIAM.

[34] Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. 2021. Local Algorithms for Estimating Effective Resistance. In *SIGKDD*. 1329–1338.

[35] Benedek Rozemberczki and Rik Sarkar. 2021. Twitch Gamers: a Dataset for Evaluating Proximity Preserving and Structural Role-based Node Embeddings. arXiv:2101.03091 [cs.SI]

[36] Gerta Rücker. 2012. Network meta-analysis, electrical networks and graph theory. *Research synthesis methods* 3, 4 (2012), 312–324.

[37] Jan Scheurer and Sergio Porta. 2006. Centrality and connectivity in public transport networks and their significance for transport sustainability in cities. In *World Planning Schools Congress, Global Planning Association Education Network*.

[38] Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. 2019. Real-time top-k personalized pagerank over large graphs on gpus. *PVLDB* 13, 1 (2019), 15–28.

[39] Daniel A Spielman and Nikhil Srivastava. 2008. Graph sparsification by effective resistances. In *STOC*. 563–568.

[40] Andreia Sofia Teixeira, Pedro T Monteiro, João A Carriço, Mário Ramirez, and Alexandre P Francisco. 2013. Spanning edge betweenness. In *MLG*, Vol. 24. 27–31.

[41] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*. 613–622.

[42] William Thomas Tutte and William Thomas Tutte. 2001. *Graph theory*. Vol. 21. Cambridge university press.

[43] Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibo Wang, and Zengfeng Huang. 2020. Personalized PageRank to a Target Node, Revisited. In *SIGKDD*. 657–667.

[44] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: effective indexing for approximate personalized pagerank. *PVLDB* 10, 3 (2016), 205–216.

[45] Sibo Wang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. 2019. Efficient algorithms for approximate single-source personalized pagerank queries. *TODS* 44, 4 (2019), 1–37.

[46] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: simple and effective approximate single-source personalized pagerank. In *SIGKDD*. 505–514.

[47] EL Wilmer, David A Levin, and Yuval Peres. 2009. Markov chains and mixing times. *American Mathematical Soc., Providence* (2009).

[48] David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *STOC*. 296–303.

[49] Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. 2021. Unifying the Global and Local Approaches: An Efficient Power Iteration with Forward Push. In *SIGMOD*. 1996–2008.

[50] Jaewon Yang and Jure Leskovec. 2012. Defining and Evaluating Network Communities Based on Ground-Truth. In *ICDM*. 745–754.

[51] Shiqi Zhang, Renchi Yang, Jing Tang, Xiaokui Xiao, and Bo Tang. 2023. Efficient Approximation Algorithms for Spanning Centrality. arXiv:2305.16086

Sampling Multiple Nodes in Large Networks: Beyond Random Walks

Omri Ben-Eliezer
Harvard University &
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
omrib@mit.edu

Talya Eden
Boston University &
Massachusetts Institute of Technology
Massachusetts, USA
teden@mit.edu

Joel Oren
Bosch Center for Artificial Intelligence
Haifa, Israel
joel.oren@il.bosch.com

Dimitris Fotakis
National Technical University of Athens
Athens, Greece
fotakis@cs.ntua.gr

ABSTRACT

Sampling random nodes is a fundamental algorithmic primitive in the analysis of massive networks, with many modern graph mining algorithms critically relying on it. We consider the task of generating a large collection of random nodes in the network assuming limited query access (where querying a node reveals its set of neighbors). In current approaches, based on long random walks, the number of queries per sample scales linearly with the mixing time of the network, which can be prohibitive for large real-world networks. We propose a new method for sampling multiple nodes that bypasses the dependence in the mixing time by explicitly searching for less accessible components in the network. We test our approach on a variety of real-world and synthetic networks with up to tens of millions of nodes, demonstrating a query complexity improvement of up to ×20 compared to the state of the art.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**; **Sketching and sampling**; **Theory and algorithms for application domains**; • **Information systems → Social networks**.

KEYWORDS

Graph and Network Sampling, Node Sampling

ACM Reference Format:
Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. 2022. Sampling Multiple Nodes in Large Networks: Beyond Random Walks. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3488560.3498383

1 INTRODUCTION

Random sampling of nodes according to a prescribed distribution has been extensively employed in the analysis of modern large-scale networks for more than two decades [22, 32]. Given the massive sizes of modern networks, and the fact that they are typically accessible through node (or edge) queries, random node sampling offers the most natural approach, and sometimes virtually the only approach, to fast and accurate solutions for network analysis tasks. These include, for example, estimation of the order [29], average degree and the degree distribution [11, 15, 59], number of triangles [2], clustering coefficient [51], and betweenness centrality [7], among many others. Moreover, node sampling is a fundamental primitive used by many standard network algorithms for quickly exploring networks, e.g., for detection of frequent subgraph patterns [42] or communities [44, 57], or for mitigating the effect of undesired situations, such as teleport in PageRank [21]. Hence, a significant volume of recent research is devoted to the efficiency of generating random nodes in large social and information networks; see, e.g., [8, 9, 26, 39, 41, 46, 60] and the references therein.

Problem formulation. We consider the task of implementing a *sampling oracle* that allows one to sample multiple nodes in a network according to a prescribed distribution (say, the uniform distribution). The collection of sampled nodes should be independent and identically distributed. Standard algorithms for random node sampling assume query access to the nodes of the network, where querying a node reveals its neighbors. As a starting point, we are given access to a single node from the network, and our goal is to generate a (possibly large) set of samples from the desired distribution using few queries. A typical efficiency measure is the amortized *query complexity* — the total number of node queries divided by the number of sampled nodes. This extends the framework proposed by Chierichetti et al. [8, 9], who studied the query complexity of sampling a *single* node. For simplicity, we focus on the important case of the *uniform distribution*, but our approach can be easily generalized to any natural distribution. We consider the regime where the desired number of node samples N is large.

Random walks and their limitations. Most previous work on node sampling has focused on random-walk-based approaches,

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02.
https://doi.org/10.1145/3488560.3498383

Talya Eden was supported by the NSF Grant CCF-1740751, Eric and Wendy Schmidt Fund for Strategic Innovation, Ben-Gurion University of the Negev and the computer science department of Boston University. Dimitris Fotakis was partially supported by NTUA Basic Research Grant (PEBE 2020) "Algorithm Design through Learning Theory: Learning-Augmented and Data-Driven Online Algorithms - LEADAlgo".
Source code: `https://github.com/omribene/sampling-nodes`.

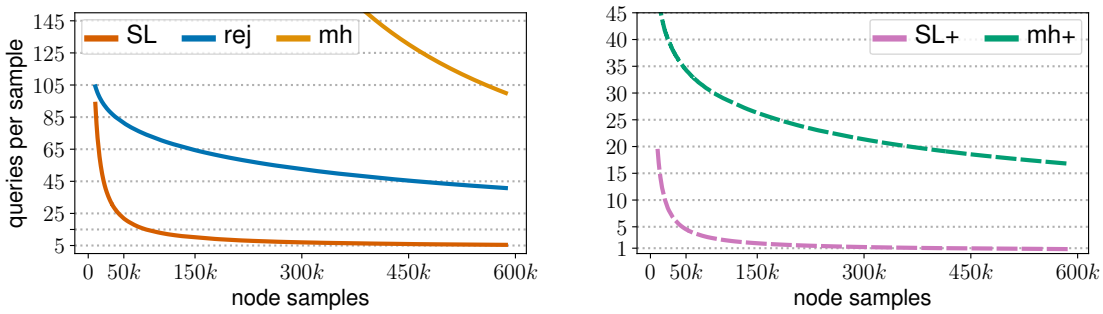


Figure 1: Amortized query complexity per sample in SinaWeibo, a network with 58.6M nodes and 261M edges. Our algorithm for the standard query model, SampLayer, is compared to rejection sampling and Metropolis-Hastings random walks (left). The variant for the stronger query model, SampLayer+, is compared to the stronger variant of MH (right).

which naturally exploit node query access to the network. They are versatile and achieve remarkable efficiency [8, 10, 26, 39]. The random walk starts from a seed node and proceeds from the current node to a random neighbor, until it (almost) converges to its stationary distribution. Then, a random node is selected according to the walk's stationary distribution, which can be appropriately modified if it differs from the desired one [8]. The number of steps before a random walk (almost) converges to the stationary distribution is called the *mixing time*, and usually denoted by t_{mix}.

RW-based approaches are very effective in highly connected networks with good expansion properties, where the mixing time is logarithmic [25]. In most real-world networks, however, the situation is more complicated. It is by now a well-known phenomenon that the mixing time in many real-world social networks is in the order of hundreds or even thousands, much higher than in idealized expander networks [13, 40, 43]. As part of this work, we prove lower bounds for sampling multiple nodes: sampling a collection of N (nearly) uniform and uncorrelated random nodes using a random walk may require $\Omega(N \cdot t_{\text{mix}})$ queries under standard assumptions on the network structure. Given the multiplicative dependence in t_{mix}, this bound becomes prohibitive as N grows larger.

1.1 Our Contribution

In light of the above discussion, we ask the following question:

Can we design a highly query-efficient method for sampling a large number of nodes that does not depend multiplicatively on the network's mixing time?

We answer this in the affirmative by presenting a novel algorithm for sampling nodes with a query complexity that is *up to a factor of 20 smaller* than that of state of the art random walk algorithms. To the best of our knowledge, this is the first node sampling method that is not based on long random walks.

Lower bound for random walks. We present an $\Omega(N \cdot t_{\text{mix}})$ lower bound for sampling N uniform and independent nodes from a network using naive random walks (that do not try to learn the network structure). Our lower bound construction is a graph consisting of a large expander-like portion and many small components connected to it by bridges, a structure that is very common among large social networks [38]. The main intuition is that sampling N nodes from the network requires the walk to visit many small communities, and thus cross $\Theta(N)$ bridges, which in expectation takes $\Theta(t_{\text{mix}})$ queries per bridge, and $\Theta(N \cdot t_{\text{mix}})$ in total.

Bypassing the multiplicative dependency. We present a new algorithm for sampling multiple nodes, SampLayer, whose query complexity does not depend multiplicatively on the mixing time. Our algorithm learns a structural decomposition of the network into one highly connected part and many small peripheral components. We also present a stronger variant of our algorithm, SampLayer+, that works in a more expressive query model, where querying a node also reveals the degrees (and not just the identifiers) of its neighbors.[1] Our approach is inspired by the core-periphery perspective on social networks [48]. We start by exploring the graph with a random walk that is biased towards higher degree nodes. With the high degree nodes in hand, we build a data structure that partitions all non-neighbors of these nodes into extremely small components. Then, we use the data structure to quickly reach these components (and subsequently, sample from them); the process involves running a BFS inside the reached component, which due to its tiny size does not require many queries.

We theoretically relate the query complexity of SampLayer to several network parameters and show that they are well-behaved in practice. The samples generated by our algorithm are provably independent and nearly uniform.

Improved empirical performance. We compare the amortized query complexity of SampLayer against those of the two most representative and standard random walk-based approaches for node sampling, rejection sampling (REJ) and the Metropolis-Hastings (MH) approach; SampLayer+ is compared against MH+, an analogue of MH in the degree-revealing model. For a complete description of REJ, MH and MH+ with a theoretical and empirical analysis of their query complexity, see the work of Chierichetti et al. [8].

We perform the comparisons on seven real world social and information networks with diverse characteristics, with the largest being SinaWeibo [58], which consists of more than 50M nodes and 250M edges. The results presented in Figures 1 and 8 and in

[1] Such strong queries are supported, e.g., by Twitter API (link1, link2).

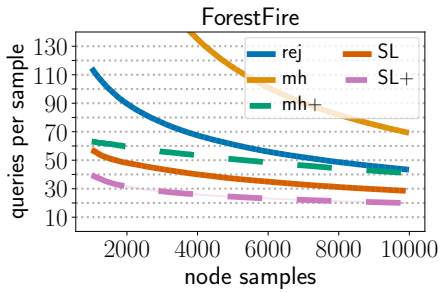


Figure 2: Amortized query complexity per sample in a Forest Fire graph with $1M$ nodes (parameters: $p_f = 0.37$, $p_b = 0.3$).

Section 4.1 show that when N is not extremely small, the query complexity of our algorithms significantly outperforms the random walk-based counterparts across the board. This holds both under the standard query model (i.e., SampLayer vs. REJ and MH) and in the stronger, degree-revealing query model (SampLayer+ vs MH+). In both models, and for all seven networks, we achieve at least 40% and up to 95% reduction in the query complexity. Remarkably, as shown in Figure 1, in some cases SampLayer may achieve a near-optimal query complexity of as little as 5 queries per sample, even when N is less than 1% of the network size. In SampLayer+ this is even more dramatic, essentially achieving one query per sample.

Generative models. One possible explanation to our findings might lie in the seminal work by Leskovec et al. [38]. In one of the most extensive analyses of the community structure in large real-world social and information networks, they examined more than 100 large networks in various domains. They discovered that most of the classical generative models at the time did not capture well the community structure and additional various properties of social networks (e.g., size of communities, their connectivity to the graph, scaling over time, etc). The *Forest Fire* generative model [36, 37] was developed to fill this gap, being more in-line with empirical findings. In this model, which is by now standard and well-investigated, edges are added in a way that creates small, barely connected pieces that are significantly larger and denser than random. We show that our algorithm performs very well on networks generated by this model (a consistent query complexity improvement of 30-50%) even for tiny core sizes; see Figure 2.

1.2 Related Work

Random-walk-based approaches for node sampling have been studied extensively in the last decade. The aforementioned work of Chierichetti et al. [8] is the closest to ours, studying the query complexity of such approaches. The analysis of Iwasaki and Shudo [26] also focuses on the average query complexity of random walks.

Using random node sampling to determine the properties of large-scale networks goes back to the seminal work of Leskovec and Faloutsos [34]. Since then, the performance of node sampling via random walks has been widely considered in the context of network parameter estimation. For example, Katzir et al. [28, 29] use random walks to estimate the network order and the clustering coefficient based on sampling and collision counting. Cooper et al. [10] present a general random-walk-based framework for estimating various network parameters; whereas Ribeiro et al. [46] extend these approaches to directed networks. Ribeiro and Towsley [45] use multidimensional random walks. Eden et al. [16–18] and Tětek and Thorup [55] study the query complexity of generating uniform edges given access to uniform nodes. Bera and Seshadhri [5] devise an accurate sublinear triangle counting algorithm that queries only a small fraction of the graph edges. For several other examples of network estimation works, see [20, 27, 30, 39, 41, 60].

In many of these works, improved query complexity is achieved by relaxing the requirement for independent samples. Understanding how dependencies between sampled nodes affect the outcome, however, inherently requires a complicated analysis tailored to the network-parameter at question. Such an analysis is not required for nodes generated by our approach, which are provably independent. From a technical viewpoint, our adaptive exploration of the network's isolated components bears some similarity to node sampling via deterministic exploration [50] and to *node probing* approaches (e.g., [6, 33, 52, 53]) for network completion [24, 31]. Given access to an incomplete copy of the network, Soundarajan et al. [52, 53] and LaRock et al. [33] discover unobserved parts via adaptive network exploration; see the survey by Eliassi-Rad et al. [19].

Utilizing core-periphery characteristics of networks for algorithmic purposes has received surprisingly little attention. The most relevant work is by Benson and Kleinberg [4], on link prediction. More weakly related is the study of k-cores of social network [1, 14], which aim to capture the subset of all "engaged" nodes by iteratively removing nodes of small degree.

2 LOWER BOUND FOR RANDOM WALKS

In this section, we quickly present the $\Omega(N \cdot t_{\text{mix}})$ lower bound on the number of queries required to sample N (nearly) independent and uniformly distributed nodes using random walks. See proof sketch in the full version [3]. For clarity, we focus our analysis on the most standard random walk, which at any given time proceeds from the current node to one of its neighbors, uniformly at random; such a random walk is used in rejection sampling (REJ). Similar lower bounds hold for other random walk variants that do not learn structural characteristics of the network, including MH and MH+.

Theorem 2.1. *For any n and $\log n \ll t \ll n^{\Theta(1)}$, there exists a graph G on n vertices with mixing time $t_{mix} = \Theta(t)$, that satisfies the following: for any $N \leq n^{\Theta(1)}$, any sampling algorithm based on uniform random walks that outputs a (nearly) uniform collection of N nodes must perform $\Omega(N \cdot t_{mix})$ queries.*

3 ALGORITHM

In order to bypass the multiplicative dependence in the mixing time, one needs to exploit structural characteristics of social networks in some way. One natural property is that the degree-distribution is top-heavy; furthermore, a large fraction of nodes in the network are well-connected to the high-degree nodes, whereas the remaining nodes decompose into small, weakly connected components [38, 48]. Figure 3 demonstrates this in a strong quantitative form. Suppose that we are able to access a collection of, say, the top 1% highest degree nodes in the network, and call these L_0. Denote their

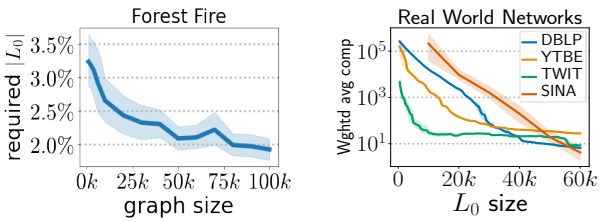


Figure 3: Very small L_0 suffices to shatter $L_{\geq 2}$-components in a Forest Fire graph (left); convergence of $L_{\geq 2}$ component sizes as $|L_0|$ grows, in four real-world networks (right).

neighbors by L_1 and the rest of the network by $L_{\geq 2}$. Does a small L_0 size suffice for $L_{\geq 2}$ to decompose into tiny components?

Our experiments indicate that the answer is positive, even if one instead generates L_0 greedily with our query access, starting from an arbitrary seed vertex. The details are given in the experimental section, but briefly, Figure 3 demonstrates that for both the Forest Fire model with standard parameters and for various real-world social networks with diverse characteristics, a very small L_0 size (ranging between 0.1% and 10% of the graph size, and in most cases about $1-2\%$) suffices for $L_{\geq 2}$ to decompose very effectively.

These results suggest a new approach to quickly reach nodes in the network: In a preprocessing phase, greedily capture L_0 as above, which decomposes the rest of the network into L_1 and $L_{\geq 2}$. L_1-nodes are easy to reach; $L_{\geq 2}$-nodes are reachable by attempting to visit a component from $L_{\geq 2}$ through a walk of length 2 from L_0, and then fully exploring the component via a BFS. Our algorithm, SampLayer, is based on this idea, also running size and reachability estimations to ensure the generated samples are close to uniform.

3.1 Algorithm Description

We next describe our algorithm, SampLayer, in detail. The algorithm runs in two phases: a structural learning phase and a sampling phase. In the first phase, the algorithm constructs a data structure providing fast access to nodes that are either very highly-connected (we call these nodes the L_0-layer) or neighbors thereof (the L_1-layer). This exploits the well-known fact that in large social and information networks, typically a large fraction of the nodes are connected to a highly influential core [48]. The node sampling itself takes place in the second phase, which uses the data structure to either sample from the core layers $L_0 \cup L_1$, or to explicitly cross bridges that lead to the small, less connected parts. These are edges from L_1 to nodes outside $L_0 \cup L_1$. We refer to these nodes as the $L_{\geq 2}$ layer. Once it reaches such a small component, the algorithm fully explores the component and uniformly samples a node from within it. Finally, our algorithm uses rejection sampling to ensure that (almost) all nodes are returned with equal probability.

Structural decomposition phase. Starting from an arbitrary node, we aim to capture the highest-degree nodes in the network. This is done by our procedure Generate-L_0 below. We add these nodes greedily, one by one, where intuitively, in every step the newly added node is the one we perceive (according to the information currently available) as the highest-degree one. We refer to this initial collection of high-degree nodes as the base layer, L_0.[2]

[2]We note that one can modify Generate-L_0 by first conducting a random walk using part of the L_0-construction budget, and only then continuing with the above process. While the added randomness could theoretically help escaping situations where the initial node is problematic in some way or there are multiple cores in the graph, in all networks that we tested adding such a random walk did not improve the quality of L_0; in fact, the existing algorithm captured essentially all nodes with very high degrees.

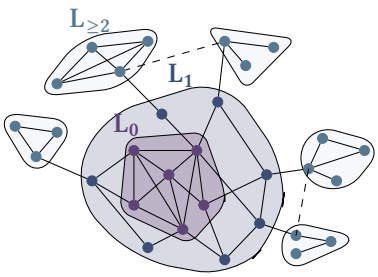


Figure 4: An illustration of a typical network and its layering by our algorithm, SampLayer. The $L_{\geq 2}$-layer components intuitively correspond to small communities that are weakly connected to the rest of the network.

Generate-L_0
Input: arbitrary vertex v_0, number ℓ_0.
Output: L_0 of size ℓ_0, L_1, $\mathcal{D}$.

(1) **Query** v_0 and let $L_0 = \{v_0\}$, $L_1 = N(v_0)$.
(2) Repeat $\ell_0 - 1$ times:
  (a) Pick $u \in L_1$ with maximum number of L_0 neighbors (break ties randomly).
  (b) **Query** u and remove it from L_1.
  (c) Add u to L_0 and add $N(u) \setminus L_0$ to L_1.
(3) Create a data structure $\mathcal{D}$ to sample edges between L_0 and L_1 uniformly at random.
(4) **Return** L_0, L_1 and $\mathcal{D}$.

The next layer, L_1, is the set of neighbors of L_0 that are not already in L_0, i.e. $L_1 = \bigcup_{v \in L_0} N(v) \setminus L_0$, where $N(v)$ denotes the set of neighbors of node v. Intuitively, the union of these two layers captures the well-connected or "expanding" part of the network. The neighbors of L_1 are denoted L_2 and the multi-layer consisting of all other nodes in the network is denoted by $L_{>2}$, where we also set $L_{\geq 2} = L_2 \cup L_{>2}$. See Figure 4 for a visualization of the layers and Figure 5 for a visualization of the structural decomposition phase of SampLayer and its variant SampLayer+.

Denote by $G_{\geq 2}$ the subgraph whose node set is $L_{\geq 2}$, and whose edge set includes all edges between L_2 and $L_{>2}$ and all edges between nodes in $L_{>2}$. Crucially, the size ℓ_0 of the generated L_0 should be sufficiently large so that the subgraph $G_{\geq 2}$ will "break" into many small connected components. (Note that we intentionally "ignore" edges between vertices that lie strictly in L_2, to make these components as small as possible.) In Section 4.2, we explore the typical size of $G_{\geq 2}$-components as a function of ℓ_0, and discuss how to determine the "correct" ℓ_0 value for the network at hand.

To complete this phase, we learn various parameters of the layering that are crucial for the sampling phase, including accurate approximations of the size of $L_{\geq 2}$ and the typical reachability of nodes in it. This is done using the procedures Estimate-Periphery-Size and Estimate-Baseline-Reachability, given in Section A.1. Specifically, estimating the size of $L_{\geq 2}$ is done by considering the

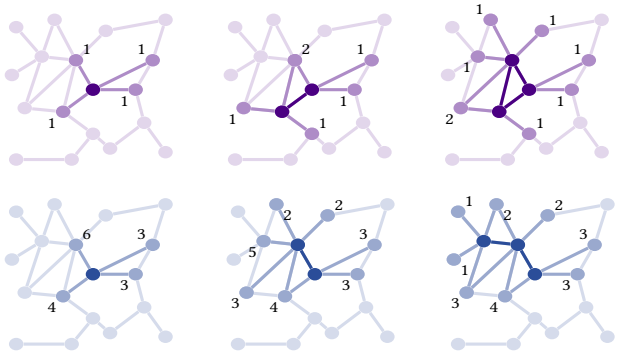


Figure 5: The structural decomposition phase, of generating (from left to right) the base layer L_0 in SampLayer (top, purple) and SampLayer+ (bottom, blue) from an arbitrary starting node. At any given step, the next layer L_1 consists of all neighbors of the L_0 nodes. The value next to each L_1-node indicates its number of neighbors in L_0 (in SampLayer) or its total degree (in SampLayer+).

bipartite graph with L_1 on one side and $L_{\geq 2}$ on the other. By sampling s_1 nodes from L_1 and $s_{\geq 2}$ nodes from $L_{\geq 2}$ (using the procedure Reach-$L_{\geq 2}$), we estimate the average degrees of the nodes of each side of the bipartite graph, from which we estimate $|L_{\geq 2}|$. The reachability distribution is approximated by calculating the reachabilities of the $s_{\geq 2}$ samples from $L_{\geq 2}$. This procedure receives as an input a parameter ε, and returns a "baseline reachability" which is approximately the ε-percentile of $L_{\geq 2}$-nodes in terms of reachability. In Section 4.1 we discuss how to practically choose s_1, $s_{\geq 2}$, and ε.

Sampling phase. Sampling from the core layers L_0 and L_1 is trivial; the challenge is to sample efficiently from $L_{\geq 2}$. Taking advantage of the layering, we sample random nodes in $L_{\geq 2}$ by combining walks of length 2 that start in L_0 and reach $L_{\geq 2}$, with a local BFS step that explores and returns a uniformly selected node in the reached $L_{\geq 2}$ component. The above process generates biased samples, as the vertices in different components have different probabilities to be reached in the initial 2-step walk. Hence, the final step in the sampling procedure is a rejection step, whose role is to unbias the distribution. Here, we compute a suitable *reachability score*, $rs(v)$ for every reached vertex. We then perform a rejection step, where the acceptance probability is inversely proportional to the reachability score of the chosen node. See Figure 6 for the pseudo-code and Figure 7 for an illustration of the sampling process.

Non-uniform distributions. For simplicity, our algorithm is presented for node generation according to the uniform distribution. We note that it can be adapted to generate other desirable distributions. For example, to conduct ℓ_p-sampling, the size estimation procedure should be replaced by a procedure that estimates the sum $\sum_{v \in L_{\geq 2}} (d(v))^p$ (and the corresponding sum for L_1), and the reachability distribution estimation should be adjusted accordingly.

3.2 Convergence to Uniformity

Our main theorem states that samples generated by our algorithm converge to (near-)uniformity. The proof builds in part on the fact

Sample
Input: $\bar{\ell}_{\geq 2}$ - size estimate for $L_{\geq 2}$. rs_0 - baseline reachability. (Both computed in the preprocessing step)
Output: An almost uniform node in the network.

(1) Choose a layer L_0, L_1 or $L_{\geq 2}$ with probability proportional to their sizes $|L_0|, |L_1|, \bar{\ell}_{\geq 2}$.
(2) If L_0 or L_1 are chosen then sample a uniform node in L_0 or L_1, respectively.
(3) If $L_{\geq 2}$ is chosen, then repeatedly do:
  (a) Invoke Reach-$L_{\geq 2}$ and let u and $rs(u)$ be the returned node and its reachability.
  (b) With probability $\min(\frac{rs_0}{rs(u)}, 1)$ **return** u. If not returned, **repeat** loop.

Reach-$L_{\geq 2}$
Input: The data structure $\mathcal{D}$.
Output: vertex $v \in L_{\geq 2}$, its component and reach. score.

(1) While no vertex w chosen:
  (a) Use $\mathcal{D}$ to sample a uniform edge between L_0 and L_1. Let u denote its L_1 endpoint.
  (b) **Query** u, and if it has neighbors in $L_{\geq 2}$, choose one of them, w, uniformly at random.
(2) Perform a local BFS of the component C of w in $G_{\geq 2}$.
(3) Choose a vertex v in C uniformly at random.
(4) Invoke Comp-Reachability to compute the reachability of C, $rs(C)$.
(5) **Return** v, and its reachability score, $rs(v) = rs(C)$.

Comp-Reachability
Input: An (already queried) component C of $G_{\geq 2}$.
Output: The reachability score of C.

(1) $\forall v \in C \cap L_2$:
  (a) **Query** all $u \in N(v) \cap L_1$. For each such u:
    (i) Let $d^-(u) = |N(u) \cap L_0|$, and $d^+(u) = |N(u) \cap L_2|$.
    (ii) $\forall u \in N(v) \cap L_1$ set $rs(u) = \frac{d^-(u)}{d^+(u)}$.
    (iii) Set $rs(v) = \sum_u rs(u)$.
(2) **Return** $rs(C) = \frac{1}{|C|} \sum_{v \in C} rs(v)$.

Figure 6: The sampling procedures.

that our algorithm can estimate the size of $L_{\geq 2}$ given sufficient effort in the preprocessing phase. Proofs are given in the full version [3]. For experiments on the size estimation procedures, see Section A.4.

Theorem 3.1. *If our size estimation for $L_{\geq 2}$ is in $(1 \pm o(1))|L_{\geq 2}|$, and if the baseline reachability rs_0 used in our algorithm is the $o(1)$-percentile in the reachability distribution, then the output node distribution of* Sample *is $o(1)$-close to uniform in total variation distance.*

We stress that even in the case that the L_0 generation process is unsuccessful (in a sense that it does not break the $L_{\geq 2}$ vertices into small components), it *always* holds that our algorithm returns a close to uniform vertex, provided that the size and reachability estimates are correct. That is, the correctness of our algorithm holds for *any* given L_0 (with high probability), and only the query complexity of subsequent sampling might be negatively affected, e.g., due to a high expected component size value.

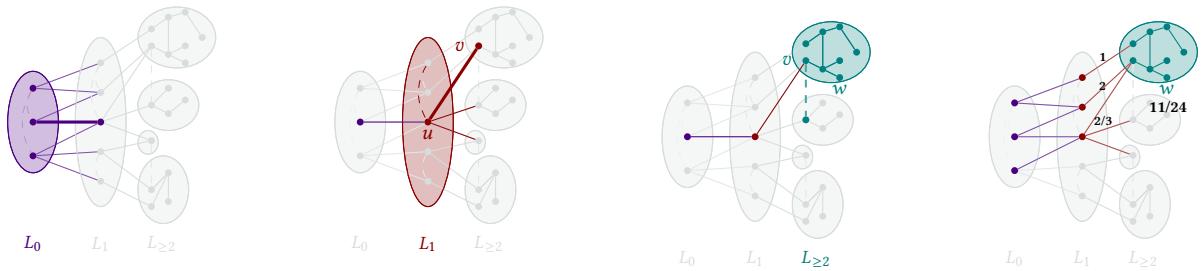


Figure 7: Sampling a node from $L_{\geq 2}$ in SampLayer. We start by picking a uniform edge L_0 and L_1, let u denote its L_1-endpoint. We next traverse a random edge from u to $v \in L_2$, if one exists. We then fully explore the $L_{\geq 2}$-component C containing v, choosing a uniformly random node $w \in C$. A final rejection step estimates how likely it is for the process to end at w.

3.3 Query Complexity

In this section, we analyze the query complexity of the sampling phase of our approach. We show here that the query complexity of sampling nodes using SampLayer is bounded as a function of several parameters related to the layered structure we maintain. The starting point of our analysis is immediately after the preprocessing phase is completed. In particular, L_0 and L_1 are already known, as well as a good estimate of the size of $L_{\geq 2}$. In addition, we have the ability to sample uniformly random edges between L_0 and L_1 without making any queries. We make the following assumptions.

- **Reachability distribution.** We assume that the reachabilities of nodes in L_2 are relatively balanced: the reachability score $rs(v)$ of every $v \in L_2$ satisfies $rs_0 \leq rs(v) \leq c \cdot rs_0$, where rs_0 is viewed as the "base reachability", and $c > 1$ is not large. We empirically verify this in Section 4.2.
- **Entry points.** Let α denote the fraction of edges e between L_0 and L_1, for which the L_1-endpoint of e has neighbors in L_2. Then α is precisely the probability that a single attempt at reaching $L_{\geq 2}$ succeeds (without taking the rejection step into account). In practice, α is known to be well-behaved [48], as most bridges to $L_{\geq 2}$ occur at higher-degree nodes of L_1.
- **Component sizes.** Set $w = \mathbb{E}[\,|CC(v)|\,]$, where $v \in L_{\geq 2}$ is (distributed as) the result of a single run of our procedure Reach-$L_{\geq 2}$, and $CC(v)$ is the $L_{\geq 2}$-component in which v resides. Intuitively, w measures the sizes of components that we reach, and we empirically validate that it is typically small on both synthetic and real-world networks, see Section 4.2.
- **Degrees of component nodes.** We assume that for all components C of $L_{\geq 2}$, the number of bridges from C to the rest of the network is at most $d \cdot |C|$, for a small integer d. This is in line with the well-observed fact [38, 48] that peripheral components are weakly connected to the network.

Our experiments verify that the parameters discussed here are indeed well-behaved when the size ℓ_0 of L_0 is chosen correctly – see Section 4.2 for more details. We bound the expected query complexity of our sampling algorithm as a function of the above parameters. Crucially, this implies that, once the preprocessing phase is complete, the query complexity does not directly depend on the network size or on the mixing time of long random walks. Due to space constraints, the proof appears in the full version [3].

Theorem 3.2. *The expected query complexity of sampling a single node using SampLayer is* $O\left(c \cdot \left(\frac{1}{\alpha} + wd\right)\right)$.

4 EMPIRICAL RESULTS

In this section, we describe several experiments we conducted, comparing our algorithms to previous approaches which are all based on random walks (Section 4.1), and explaining the query-efficiency of our methods (Section 4.2).

4.1 Evaluation of Query Complexity

The main experiment computes the amortized number of queries per sample of our algorithm, and compares it with the corresponding query complexity of existing RW-based approaches. In the standard query model, we compare our algorithm SampLayer with two random walk-based algorithms, Rejection sampling (REJ) and Metropolis-Hastings (MH). In the stronger query model, we compare SampLayer+ to Metropolis-Hastings "plus" (MH+). The methods REJ, MH, and MH+ were all described in detail by Chierichetti et al. [8]. In REJ, the algorithm performs a standard (unbiased) random walk, where nodes are subject to rejection sampling according to their degree; in MH the neighbor transition probabilities are controlled by the neighbors' degrees. MH+ is the same as MH, but assumes the stronger query model, where a node query also reveals the degrees of its neighbors. RW-based algorithms are most commonly used to sample multiple nodes by performing a long random walk, and sampling a new node once every fixed interval to allow for re-mixing. Indeed, as discussed in Section 2, to ensure that the

Dataset	n	m	d_{avg}	L_0 size	
				SL	SL+
Epinions [47]	76K	509K	13.4	3K	1K
Slashdot [38]	82K	948K	23.1	3K	2K
DBLP [56]	317K	1.05M	6.62	30K	20K
Twitter-Higgs [12]	457K	14.9M	65.1	25K	10K
Forest Fire [36, 37]	1M	6.75M	13.5	10K	10K
Youtube [56]	1.1M	2.99M	5.27	30K	10K
Pokec [54]	1.6M	30.6M	37.5	200K	100K
SinaWeibo [58]	58.7M	261M	8.91	500K	100K

Table 1: The list of networks we considered with numbers of nodes (n), edges (m), their average degrees (d_{avg}), and L_0 sizes we selected for SampLayer and SampLayer+.

node samples will be uniform and independent, the interval length must allow the walk to mix between subsequent samples.

Setting for our algorithm. We examine seven online social and information networks of varying sizes and characteristics, taken from widely used network repositories [35, 49]. We also examine our algorithm on a network generated by the Forest Fire model with parameters $p_f = 0.37$ and $p_b = 0.3$, which are standard for this model [36]. The networks, along with their basic properties, are described in Table 1. For each network, we performed a small grid search to obtain a reasonable value for the input parameter ℓ_0 (the target size of L_0) in our algorithm. The values we used for each network are given in Table 1. For the other two input parameters, s_1 and $s_{\geq 2}$, we observed that choices of 3,000 and 200 respectively are generally sufficient for SampLayer on the first seven networks (for Epinions and Slashdot, we picked $s_1 = 1{,}000$). In SampLayer+, values of $s_1 = 1{,}000$ and $s_{\geq 2} = 100$ are generally sufficient for the seven smaller networks. Separately, for SinaWeibo we picked larger values, of $s_1 = 30k$ and $s_{\geq 2} = 3k$ for both SampLayer and SampLayer+, since the network is substantially larger.

We ran each of our algorithms SampLayer and SampLayer+ for 5-10 times on each of the eight networks; the amortized query complexity we calculated is the average over these runs. As part of our pipeline, we verified the quality of our solution by configuring the algorithm's parameters so as to ensure that the samples generated by our algorithm are close to uniform. Specifically, we fixed a small empirical threshold t (0.01 in most cases) and parameters s_1 and $s_{\geq 2}$ as above, while varying the value of the error parameter ε in our algorithm. For each choice of ε, we ran the following for 10 times: we sampled n nodes using our algorithm (parameterized by ε), where n is the graph size. In each of the runs, we calculated the empirical distance to uniformity; if the average empirical distance over the 10 runs is more than t away from the expected value for a true uniform distribution, ε is discarded. Thus, our final choice of ε ensures near-uniformity of the output samples.

Setting for random walks. As mentioned, the standard approach to sampling multiple nodes using a random walk is by running a single long walk and extracting samples from it in fixed intervals. We examined this approach in two phases: Determining a good choice for the interval length, and evaluating the query complexity.

We judiciously set interval lengths that allow for proper mixing. This is explained in detail in Appendix A.3, but briefly, we generated a large number of short random walks from the same starting point and evaluated at what point in time these walks mix. To this end, we computed in each time step, for all walks simultaneously, the empirical distance to uniformity (using the same value of the empirical threshold t as in our algorithm) or the number of collisions, which is also an indicator of distance to uniformity [23]. To ensure the variance is controlled, we ran this procedure from 3-5 different starting points for each of the algorithms REJ, MH, MH+.

To compute the amortized query complexity, we ran the random walk algorithms on each of the networks, while keeping track of the cumulative number of queries. Then, we computed the mean number of queries per sample as the walk progressed.

Main results. Figure 8 depicts the comparison results for the six smaller real-world networks. Figure 1 and 2 show the results for SinaWeibo, and for a Forest Fire generated network, respectively. The number of node samples in each of the first six networks, as well as in the FF one, is between 0.1% and 1% of the total number of nodes. While the results at the lower end, 0.1%, show the relatively steep initial price of the structural learning phase of our algorithm, the higher end of our sample size clarifies the stark differences in performance between the methods. In SinaWeibo, due to its sheer size, we considered a wider interval, from $10k$ samples (less than 0.02% of the nodes) to about $600k$ samples (1%).

As is evident in the plots, SampLayer and SampLayer+ obtained significantly improved results compared to their RW-based counterparts. In all cases, and throughout the runs (as more samples are gathered), SampLayer demonstrated a query complexity that in all cases offers query complexity savings of at least 40%, and often much more, compared to both REJ and MH. Moreover, REJ consistently required fewer queries on average than its counterpart MH. This is in line with previous results from [8]. In the more powerful query model, SampLayer+ also gave at least 50% (and almost always better) improvement over its random walk analog MH+. The most dramatic improvement was for SinaWeibo, the largest network, where SampLayer and SampLayer+ yielded reductions reaching 90% and 95% in the query complexity, respectively, compared to their random walk counterparts. Curiously, as shown in Figure 1, the query complexity of SampLayer+ in SinaWeibo was in some cases less than one query per sample. While this may seem counter-intuitive at first, we note that node samples from L_1 are generated by our algorithm without any query cost. One feature of SinaWeibo that we observed is that a large majority of the nodes in the network are located in L_1, even for small L_0 sizes. Thus, many samples do not induce any query-cost.

Interestingly, our algorithm was challenged by the DBLP network, which required a costly L_0-construction stage, resulting in an initial disadvantage. However, as is clear in the figure, our algorithm recovers at samples of at least 1,000 nodes, to quickly reach consistent improvement of about 40% compared to REJ. We believe that these difficulties stem from the fact that in DBLP, a collaboration network, nodes have a weaker tendency to connect to very high degree nodes than in most social or information networks.

4.2 Other Experiments

Structural layering parameters. In Section 3.3 we have seen that two factors mostly control the query complexity of our sampling phase: the typical size of $L_{\geq 2}$-components, and the reachability distribution of nodes in $L_{\geq 2}$. We empirically demonstrate that for the "right" size of L_0, these two factors are well-behaved, explaining the improved query complexity of our method.

Define μ as the average component size of a node in $L_{\geq 2}$. This is a weighted (biased) average, giving more weight to the larger components. To see why this weighted average is of interest, recall our definition of w from Section 3.3: the expected size of a component reached by the procedure Reach-$L_{\geq 2}$. Under the assumption that all node reachabilities in $L_{\geq 2}$ are of the same order, it holds that $w = \Theta(\mu)$. To analyze the typical size of reached components in $L_{\geq 2}$, we computed the value of μ as a function of the L_0-size. This was done five times for each network, and is demonstrated for DBLP, Youtube, Twitter-Higgs, and SinaWeibo in Figure 3 (right).

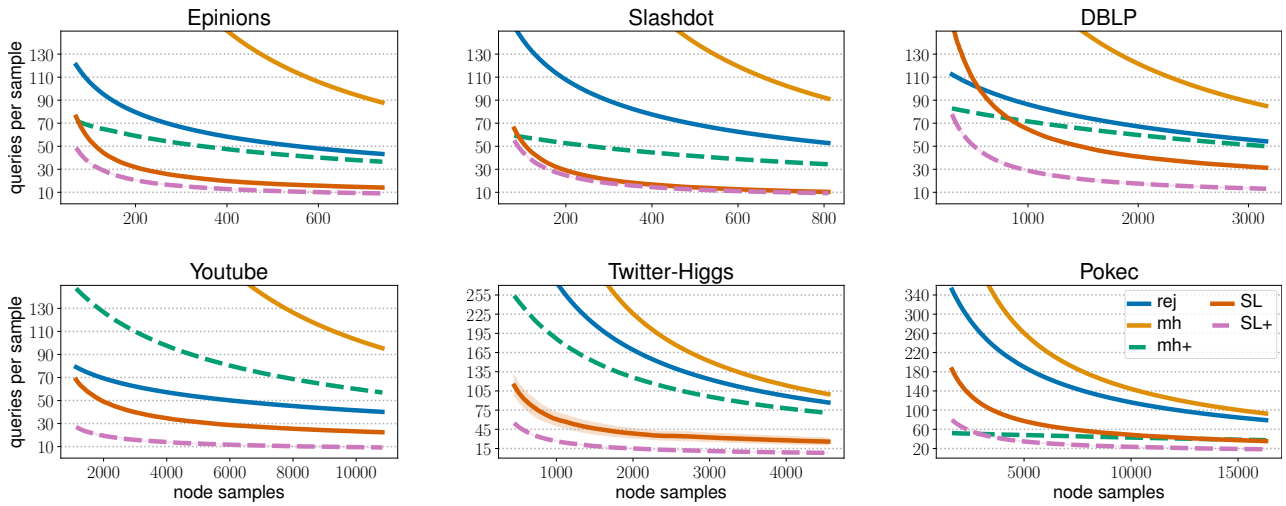


Figure 8: Amortized query complexity of our methods compared to random-walk-based node sampling methods. In the standard query model, SampLayer is compared against Rejection Sampling (REJ) and Metropolis Hastings (MH), and in the stronger query model, SampLayer+ is compared against Metropolis Hastings "plus" (MH+). For each of the networks, the number of node samples ranges between 0.1% and 1% of the network's order. See Section 4 and Table 1 for details.

Interestingly, the weighted average μ seems to decay exponentially as a function of the L_0-size (note that the y-axis here is log-scaled), until it converges to a constant. To the best of our knowledge, this exponential decay was not previously observed in the literature, and we believe it warrants further research; as the results show, the rate of decay differs majorly between different networks, and explains the choices of L_0-size that we made for the different networks: ideally, one would like L_0 to be small, while inducing a small enough μ-value (say, 10 or 20).

In Figure 3 (left), we investigated what minimal size of L_0 is required in a Forest Fire graph in order to satisfy $\mu \leq 10$. The experiment was run for different values of the graph size n, while fixing the FF parameters $p_f = 0.37$ and $p_b = 0.3$ as before. The results show that the required $|L_0|$ is clearly sublinear in n: they decay from about 3.3% for $n = 1$K to about 2% for $n = 100$K.

Other experiments. The experiments related to the reachability distribution and the size estimation are discussed in Section A.4.

5 CONCLUSIONS AND OPEN QUESTIONS

We have presented an algorithm that supports query-efficient sampling of multiple nodes in a large social network, exhibiting the efficiency of our algorithm compared to the state of the art on a variety of datasets, that include up to tens of millions of nodes. We gave theoretical bounds on our algorithm's complexity in terms of several graph parameters. We then empirically confirmed that, in all the social networks we have examined, these graph parameters are well-behaved. One major concern is the question of generality. That is, what is the scope of networks for which our methods are suitable. As stated in the experimental section, our algorithms gave better or comparable results on all social networks we have tested, and the good performance on the Forest Fire model, a realistic generative model, further hints to wide applicability.

Our algorithm has some disadvantages compared to random walks. First, it is substantially more complicated, and its analysis does not directly depend on one structural parameter of the network (such as the mixing time for random walks). Second, it inherently relies on a data structure which has a high memory consumption, and may not always be suitable in situations that require bounded-memory algorithms. Third, it depends on the size of L_0, which differs between different networks, so it requires fine tuning according to the given network. Finally, it exploits unique properties of social networks. These properties do not hold for bounded degree graphs, or for, say, real-world road networks. For such graphs, we expect our algorithm to perform worse than random walks.

Having said all that, our work demonstrates that in various real and synthetic social networks, significant query complexity savings can be obtained using structural decompositions and careful preprocessing. We do not believe that our algorithm should replace RW-based approaches. Indeed, these algorithms are the gold standard for node sampling and provide an exceptionally versatile primitive in network analysis. Rather, one can think of our algorithms as useful alternatives when multiple node samples are required and where the networks at question have suitable community structure. It would be very interesting to further explore approaches that combine the query-efficiency of our methods with the flexibility of random walks. For example, how can our core-periphery based data structure be used to accelerate the computation of important network parameters, or the detection of community structure?

ACKNOWLEDGMENTS

The authors wish to thank Suman K. Bera and C. Seshadhri for many helpful pointers and suggestions; Flavio Chierichetti and Ravi Kumar for valuable feedback; and the anonymous reviewers, for their insightful comments.

REFERENCES

[1] José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *Networks & Heterogeneous Media*, 3(2):371–393, 2008.

[2] L. Becchetti, P. Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010.

[3] Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. Sampling multiple nodes in large networks: Beyond random walks. *CoRR*, abs/2110.13324, 2021.

[4] A. R. Benson and J. M. Kleinberg. Link prediction in networks with core-fringe data. In *Proc. World Wide Web Conference (WWW 2019)*, pages 94–104, 2019.

[5] Suman K. Bera and C. Seshadhri. How to count triangles, without seeing the whole graph. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 306–316, 2020.

[6] C.A. Bliss, C.M. Danforth, and P.S. Dodds. Estimation of global network statistics from incomplete data. *PLoS ONE*, 9, 2014.

[7] M. Borassi and E. Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. *ACM J. Experimental Algorithmics*, 24(1):1.2:1–1.2:35, 2019.

[8] F. Chierichetti, A. Dasgupta, R. Kumar, S. Lattanzi, and T. Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International World Wide Web Conference (WWW 2016)*, pages 471–481, 2016.

[9] F. Chierichetti and S. Haddadan. On the complexity of sampling vertices uniformly from a graph. In *45th ICALP*, pages 149:1–149:13, 2018.

[10] C. Cooper, T. Radzik, and Y. Siantos. Estimating network parameters using random walks. *Social Netw. Analys. Mining*, 4(1):168, 2014.

[11] A. Dasgupta, R. Kumar, and T. Sarlós. On estimating the average degree. In *Proceedings of the 23rd International World Wide Web Conference (WWW 2014)*, pages 795–806. ACM, 2014.

[12] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific reports*, 3:2980, 10 2013.

[13] Matteo Dell'Amico and Yves Roudier. A measurement of mixing time in social networks. In *Proceedings of the 5th International Workshop on Security and Trust Management, Saint Malo, France*, page 72, 2009.

[14] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. k-core organization of complex networks. *Phys. Rev. Lett.*, 96:040601, 2006.

[15] T. Eden, S. Jain, A. Pinar, D. Ron, and C. Seshadhri. Provable and practical approximations for the degree distribution using sublinear graph samples. In *Proc. 2018 World Wide Web Conference WWW*, pages 449–458, 2018.

[16] T. Eden, D. Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 722–734. ACM, 2018.

[17] T. Eden and W. Rosenbaum. On sampling edges almost uniformly. In *1st Symposium on Simplicity in Algorithms, SOSA*, pages 7:1–7:9, 2018.

[18] Talya Eden, Saleet Mossel, and Ronitt Rubinfeld. Sampling multiple edges efficiently. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 51:1–51:15, 2021.

[19] T. Eliassi-Rad, R.S. Caceres, and T. LaRock. Incompleteness in networks: Biases, skewed results, and some solutions. In *SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 3217–3218. ACM, 2019.

[20] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A case study of unbiased sampling of OSNs. In *Proc. IEEE INFOCOM*, 2010.

[21] D. F. Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015.

[22] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[23] Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75. Springer, 2011.

[24] S. Hanneke and E.P. Xing. Network completion and survey sampling. In *Proc. 12th Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 209–215, 2009.

[25] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(4):439–561, 2006.

[26] K. Iwasaki and K. Shudo. Comparing graph sampling methods based on the number of queries. In *11th International Conference on Social Computing and Networking*, SocialCom '18, pages 1136–1143, 2018.

[27] L. Jin, Y. Chen, P. Hui, C. Ding, T. Wang, A. V. Vasilakos, B. Deng, and X. Li. Albatross sampling: Robust and effective hybrid vertex sampling for social graphs. In *Proc. 3rd ACM Intl. Workshop on MobiArch*, HotPlanet '11, pages 11–16, 2011.

[28] L. Katzir and S. J. Hardiman. Estimating clustering coefficients and size of social networks via random walk. *ACM Trans. Web*, 9(4):19:1–19:20, 2015.

[29] L. Katzir, E. Liberty, O. Somekh, and I. A. Cosma. Estimating sizes of social networks via biased sampling. *Internet Mathematics*, 10(3-4):335–359, 2014.

[30] Liran Katzir, Clara Shikhelman, and Eylon Yogev. Interactive proofs for social graphs. In *Advances in Cryptology – CRYPTO 2020*, pages 574–601, 2020.

[31] M. Kim and J. Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *Proc. of the 11th SIAM International Conference on Data Mining, (SDM 2011)*, pages 47–58. SIAM / Omnipress, 2011.

[32] J. M. Kleinberg. Detecting a network failure. *Internet Math.*, 1(1):37–55, 2003.

[33] T. LaRock, T. Sakharov, S. Bhadra, and T. Eliassi-Rad. Reducing network incompleteness through online learning: A feasibility study. In *Proc. of the 14th International Workshop on Mining and Learning with Graphs (MLG)*. ACM, 2018.

[34] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, 2006.

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

[36] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. 11th ACM SIGKDD conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[37] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

[38] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[39] R. Li, J. X. Yu, L. Qin, R. Mao, and T. Jin. On random walk based graph sampling. In *IEEE 31st Int. Conf. Data Engineering*, pages 927–938, 2015.

[40] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 383–389, 2010.

[41] A. Nazi, Z. Zhou, S. Thirumuruganathan, N. Zhang, and G. Das. Walk, not wait: Faster sampling over online social networks. *Proc. VLDB.*, 8(6):678–689, 2015.

[42] Giulia Preti, Gianmarco De Francisci Morales, and Matteo Riondato. Maniacs: Approximate mining of frequent subgraph patterns through sampling. In *Proc. 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1348–1358, 2021.

[43] Yi Qi, Wanyue Xu, Liwang Zhu, and Zhongzhi Zhang. Real-world networks are not always fast mixing. *The Computer Journal*, 2020.

[44] M. Rahmani, A. Beckus, A. Karimian, and G. K. Atia. Scalable and robust community detection with randomized sketching. *IEEE Transactions on Signal Processing*, 68:962–977, 2020.

[45] B. F. Ribeiro and D. F. Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010*, pages 390–403. ACM, 2010.

[46] B. F. Ribeiro, P. Wang, F. Murai, and D. Towsley. Sampling directed graphs with random walks. In *Proc. IEEE INFOCOM*, pages 1692–1700, 2012.

[47] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *ISWC*, pages 351–368, 2003.

[48] M. Rombach, M. Porter, J. Fowler, and P. Mucha. Core-periphery structure in networks. *SIAM Journal on Applied Mathematics*, 74(1):167–190, 2014.

[49] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proc. 29th AAAI Conf. Artificial Intelligence*, 2015.

[50] N. Salamanos, E. Voudigari, and E. J. Yannakoudakis. Deterministic graph exploration for efficient graph sampling. *Social Network Analysis and Mining*, 7(1):24, 2017.

[51] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining*, 7(4):294–307, 2014.

[52] S. Soundarajan, T. Eliassi-Rad, B. Gallagher, and A. Pinar. MaxOutProbe: An algorithm for increasing the size of partially observed networks. In *Workshop on Networks in the Social and Information Sciences, 29th Conference on Neural Information Processing Systems (NIPS 2015)*, 2015.

[53] S. Soundarajan, T. Eliassi-Rad, B. Gallagher, and A. Pinar. ϵ-WGX: adaptive edge probing for enhancing incomplete networks. In *Proc. of the 2017 ACM on Web Science Conference (WebSci 2017)*, pages 161–170. ACM, 2017.

[54] L. Takac and M. Zabovsky. Data analysis in public social networks. In *International Workshop Present Day Trends of Innovations*, 2012.

[55] Jakub Tětek and Mikkel Thorup. Sampling and counting edges via vertex accesses. *CoRR*, abs/2107.03821, 2021.

[56] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

[57] Se-Young Yun and Alexandre Proutiere. Community detection via random and adaptive sampling. In *Conference on learning theory*, pages 138–175. PMLR, 2014.

[58] Kai Zhang, Qian Yu, Kai Lei, and Kuai Xu. Characterizing tweeting behaviors of sina weibo users via public data streaming. In *Web-Age Information Management*, pages 294–297. Springer, 2014.

[59] Y. Zhang, E. D. Kolaczyk, and B. D. Spencer. Estimating network degree distributions under sampling: An inverse problem, with applications to monitoring social media networks. *Annals of Applied Statistics*, 9(1):166–199, 2015.

[60] Z. Zhou, N. Zhang, Z. Gong, and G. Das. Faster random walks by rewiring online social networks on-the-fly. *ACM Trans. Database Syst.*, 40(4):26:1–26:36, 2016.

A APPENDICES

A.1 Remaining procedures of SampLayer

We provide in Figure 9 the pseudocode of two structural analysis procedures in SampLayer: Estimate-Baseline-Reachability, for computing an estimate for the baseline reachability score; and Estimate-Periphery-Size, for estimating the size of $L_{\geq 2}$.

Estimate-Baseline-Reachability
Input: reachabilities $r_1 \leq \ldots \leq r_m$, param. $\varepsilon > 0$. Output: Unbiased ε-quantile for the reachabilities.

(1) $\forall i \in [m]$ set $w_i = \frac{r_1}{r_i}$. Let $cw_i = \frac{\sum_{j\in[i]} w_j}{\sum_{j\in[|S|]} w_j}$.
(2) **Return** minimum r_i for which $cw_i \geq \varepsilon$.

Estimate-Periphery-Size
Input: s_1 and $s_{\geq 2}$, numbers of samples to take from L_1 and $L_{\geq 2}$, respectively.
Output: $\overline{\ell}_{\geq 2}$ - the estimated size of $L_{\geq 2}$.

(1) **Query** s_1 nodes from L_1 uniformly at random. Denote by S_1 these queried nodes.
(2) $\forall v \in S_1$, compute $d^+(v) = N(v) \cap L_{\geq 2}$.
(3) Let $d^+_{avg}(S_1) = \frac{1}{|S_1|} \sum_{v\in S_1} d^+(v)$.
(4) Invoke Reach-$L_{\geq 2}$ for $s_{\geq 2}$ times and let S_2 denote the set of returned nodes.
(5) $\forall v \in S_2$, invoke Comp-Reachability(v) to compute the reachability score $rs(v)$.
(6) $\forall v \in S_2$, compute $d^-(v) = |N(v) \cap L_1|$.
(7) Let $trs(S_2) = \sum_{v\in S_2} 1/rs(v)$ and define $d^-_{avg}(S_2) = \frac{1}{trs(S_2)} \sum_{v\in S_2} d^-(v)/rs(v)$.
(8) **Return** $\overline{n}_{\geq 2} = |L_1| \cdot d^+_{avg}(S_1) \;/\; d^-_{avg}(S_2)$.

Figure 9: Missing procedures of SampLayer.

A.2 Description of SampLayer+

In Figure 10 we present the psuedocode of the procedures in SampLayer+ whose implementation differs from that in SampLayer. There are three such procedures: for generating L_0, reaching $L_{\geq 2}$, and computing the reachability of a node. All other procedures are as in SampLayer.

SampLayer+ takes advantage of the stronger query model (which also reveals the degrees of the neighbors) in two ways. First, in the L_0-generation phase, at each round we pick a neighbor with absolute maximum degree, rather than a "perceived" maximum degree as in SampLayer. This results in SampLayer+ generally adding higher-degree nodes to L_0 compared to SampLayer.

The second modification is during the sampling phase, and involves the reachability computation. We utilize the stronger query model of SampLayer+ to reach $L_{\geq 2}$ in ways that reduces the query-cost of rejection. Here, we have a way to guarantee that the random edge entering L_2 in our reaching attempt is uniform among all edges between L_1 and L_2; in SampLayer, we do not have such a guarantee. Thus, the reachability of a component is simply the number of edges entering it from L_1 divided by the component size. This makes the reachability both easier to compute (requires less queries) and more evenly distributed than in SampLayer.

A.3 Interval Length for Mixing

In the query complexity evaluation for random walks, Section 4.1, we mention that the walks are sampled every fixed interval, where the interval length should allow for mixing. Here we detail how the interval length is judiciously chosen.

Consider a collection of k random walks $W_1, \ldots, W_k$ with the same starting point, v. How can we determine how much steps of a random walk are required to mix? One natural way to do so is by considering the t'th nodes in each random walk, $W_1(t), \ldots, W_k(t)$, for different choices of t. We would like to set the interval length to the smallest t for which $W_1(t), \ldots, W_k(t)$ are sufficiently uniform. This is a standard statistical estimation task. The most standard approach to solving it is by calculating the *empirical distance* of the observed nodes $W_1(t), \ldots, W_k(t)$ to the uniform distribution over all nodes. For all networks except for SinaWeibo, we ran $k = n$ random walks, where n is the number of nodes in the network. One can show that for random variables $X_1, \ldots, X_n \in [n]$ that are generated uniformly at random, the empirical distance to uniformity is concentrated around $1/e$. We thus picked t to be the smallest for which the empirical distance to uniformity of $W_1(t), \ldots, W_n(t)$ is no more than some small parameter ζ (on real-valued networks we picked $\zeta = 0.01$, and for forest fire $\zeta = 0.03$) away from the ideal $1/e$.

To make for a fair comparison for our algorithm, we calculated what choice of the proximity parameter ε yields the same empirical

Generate-L_0
Input: arbitrary node v_0, number ℓ_0.
Output: L_0 of size ℓ_0, L_1, $\mathcal{D}^+$.

(1) **Query** v_0 and let $L_0 = \{v_0\}$, $L_1 = N(v_0)$.
(2) Repeat $\ell_0 - 1$ times: pick $u \in L_1$ with maximum degree (break ties randomly); **query** u; remove u from L_1; add u to L_0; and add $N(u) \setminus L_0$ to L_1.
(3) Create a data structure $\mathcal{D}^+$ that allows to sample a node in L_1 with probability proportional to its number of neighbors in $L_1 \cup L_2$.

Comp-Reachability
Input: An (already visited) component C of $G_{\geq 2}$.
Output: The reachability score of C.

(1) For every $v \in C$, set $rs'(v) = |N(v) \cap L_1|$.
(2) **Return** $rs(C) = \frac{1}{|C|} \sum_{v\in C} rs'(v)$.

Reach-$L_{\geq 2}$
Input: The data structure $\mathcal{D}^+$.
Output: A node in $L_{\geq 2}$, and its component and reachability score.

(1) While true:
  (a) Use $\mathcal{D}^+$ to sample a node $u \in L_1$ according to the distribution prescribed by $\mathcal{D}^+$.
  (b) Query u, compute $N(u) \cap L_0$, and pick a uniform neighbor w in $N(u) \setminus L_0 = N(u) \cap (L_1 \cup L_2)$.
  (c) If $w \in L_2$, exit loop. Otherwise, repeat loop.
(2) Perform BFS in $L_{\geq 2}$ to find the component C of w. Invoke Comp-Reachability to compute $rs(C)$.
(3) **Return** w,C, and the reachability $rs(C)$.

Figure 10: Pseudo-code for SampLayer+.

distance of ζ from $1/e$. To do so we ran a grid search over various small values of ε, where for each ε we averaged over 10 experiments of computing the empirical distance from uniformity of a set of n outputs of our algorithm (with values of the other parameters, $\ell_0, s_1, s_{\geq 2}$, as mentioned above).

For SinaWeibo, the empirical distance based evaluation is computationally infeasible, since it essentially requires $k = \tilde{\Omega}(n)$. Instead we used a more efficient estimate, based on the number of collisions in $W_1(t), \ldots, W_k(t)$. Distinguishing the uniform distribution from one that is ε-far in variation distance requires only $O(\sqrt{n}/\varepsilon^2)$ different walks [23]. Here we chose t to be the smallest for which the number of collisions among the different walks is less than three times the standard deviation of the number of collisions expected from the uniform distribution.

A.4 Size and Reachability Experiments

Reachability distribution. In our theoretical analysis, we claim that if the reachabilities of nodes in $L_{\geq 2}$ are all roughly of the same order, then the rejection step of our algorithm is not too costly. Here, we demonstrate that this assumption on the reachability distribution indeed approximately holds in practice. Specifically, Figure 11 presents the reachability distribution of the $L_{\geq 2}$-nodes in DBLP for $|L_0|$ of $30k$ (the distributions for other networks are similar). For clarity, we discarded the top 3% reachabilities, which form a thin upper tail, and only show the lower 97% here. As can be seen, indeed most reachabilities are roughly of the same order: almost all $L_{\geq 2}$-nodes in the experiment have reachability up to 1, where a majority of them are between roughly 0.05 and 0.2.

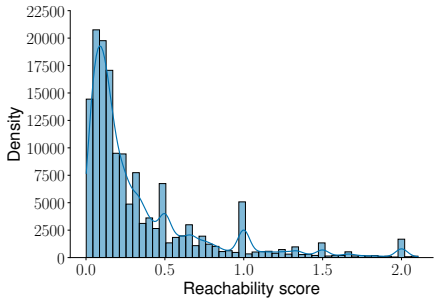


Figure 11: Reachability distribution in $L_{\geq 2}$ for DBLP.

Size estimation. As mentioned, our algorithm needs to compute an accurate size estimate for $L_{\geq 2}$ as part of its preprocessing. We next empirically demonstrate the quick convergence of our size estimate as a function of the numbers of nodes from L_1 and $L_{\geq 2}$ we visit during the preprocessing. Recall the size estimation procedure Estimate-Periphery-Size described in Section 3.1 (see also Section A.1). Here we demonstrate the quick rate of convergence of this procedure, validating our choices of the parameters s_1 and $s_{\geq 2}$.

The size of $L_{\geq 2}$ satisfies the following: $|L_{\geq 2}| = |L_1| \cdot d_1^+/d_2^-$, where d_1^+ is the average, over all nodes $v \in L_1$, of the number of neighbors of v in L_2; and d_2^- is the symmetric quantity, i.e. the (unweighted) average over all nodes in $L_{\geq 2}$ of their number of neighbors in L_1. Estimate-Periphery-Size estimates d_1^+ and d_2^- by taking s_1 samples from L_1 and $s_{\geq 2}$ sampled nodes from $L_{\geq 2}$ (using Reach-$L_{\geq 2}$, without rejection; as these are biased, we use weighted averaging). It then uses these estimates to approximate $|L_{\geq 2}|$. In the current experiment, we separately check how the chosen values of s_1 and $s_{\geq 2}$ affect the size estimate of $L_{\geq 2}$. We run two experiments, each of them five times for each of the networks; see the results for DBLP in Figure 12 as a representative example. First, we fix the value of d_2^-, and measure the error in the estimate $\tilde{\ell}_{\geq 2}$ when d_1^+ is computed as the average out-degree over s_1 samples. The second experiment is similar, except that d_1^+ and d_2^- switch roles: d_1^+ is fixed to its actual value, whereas we compute an estimate of d_2^- from $s_{\geq 2}$ nodes in $L_{\geq 2}$ obtained through our algorithm (without the rejection step), where each reached node is assigned a weight that is inversely proportional to its reachability. As Figure 12 shows, it suffices to take s_1 of order a few thousands (left) and $s_{\geq 2}$ of order a few hundreds (right) to obtain a small error in the size estimation.

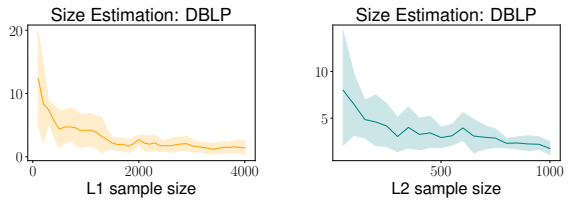


Figure 12: Error (%) of the size estimate obtained by SamPlayer for DBLP during our structural decomposition phase, as a function of the number s_1 of nodes queried from L_1 (left) and the number $s_{\geq 2}$ of reached nodes from L_2 (right).

A.5 Source Code

The source code for our algorithm can be found at:

`https://github.com/omribene/sampling-nodes`

For reference, we also provide the source code for the random walk algorithms to which we compared our method. See the `README.md` file for usage instructions.

A.6 System Specifications and Running Time

We ran all experiments on a 20-core CPU configuration: Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz; RAM: 128GB. The most time-consuming experiments were those involving the comparison with random walks, specifically the experiments determining the correct interval size for the random walks. In SinaWeibo, our largest graph, the running time for generating 1M random walks of proper length from one starting node was about three days. For all experiments involving our algorithm, the running time was at most a few hours (and usually up to a few minutes for the smaller networks).

FAST-PPR: Scaling Personalized PageRank Estimation for Large Graphs

Peter Lofgren
Department of Computer Science
Stanford University
plofgren@cs.stanford.edu

Siddhartha Banerjee
Department of Management Science & Engineering
Stanford University
sidb@stanford.edu

Ashish Goel
Department of Management Science & Engineering
Stanford University
ashishg@stanford.edu

C. Seshadhri
Sandia National Labs
Livermore, CA
scomand@sandia.gov

ABSTRACT

We propose a new algorithm, FAST-PPR, for computing personalized PageRank: given start node s and target node t in a directed graph, and given a threshold δ, FAST-PPR computes the Personalized PageRank $\pi_s(t)$ from s to t, guaranteeing a small relative error as long $\pi_s(t) > \delta$. Existing algorithms for this problem have a running-time of $\Omega(1/\delta)$; in comparison, FAST-PPR has a provable average running-time guarantee of $O(\sqrt{d/\delta})$ (where d is the average in-degree of the graph). This is a significant improvement, since δ is often $O(1/n)$ (where n is the number of nodes) for applications. We also complement the algorithm with an $\Omega(1/\sqrt{\delta})$ lower bound for PageRank estimation, showing that the dependence on δ cannot be improved.

We perform a detailed empirical study on numerous massive graphs, showing that FAST-PPR dramatically outperforms existing algorithms. For example, on the 2010 Twitter graph with 1.5 billion edges, for target nodes sampled by popularity, FAST-PPR has a 20 factor speedup over the state of the art. Furthermore, an enhanced version of FAST-PPR has a 160 factor speedup on the Twitter graph, and is at least 20 times faster on all our candidate graphs.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problems**]: Computations on Discrete Structures; G.2.2 [**Graph Theory**]: Graph Algorithms

General Terms

Algorithms,Theory

KDD '14, August 24 - 27 2014, New York, NY, USA.

ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623745.

Keywords

Personalized PageRank; Social Search

1. INTRODUCTION

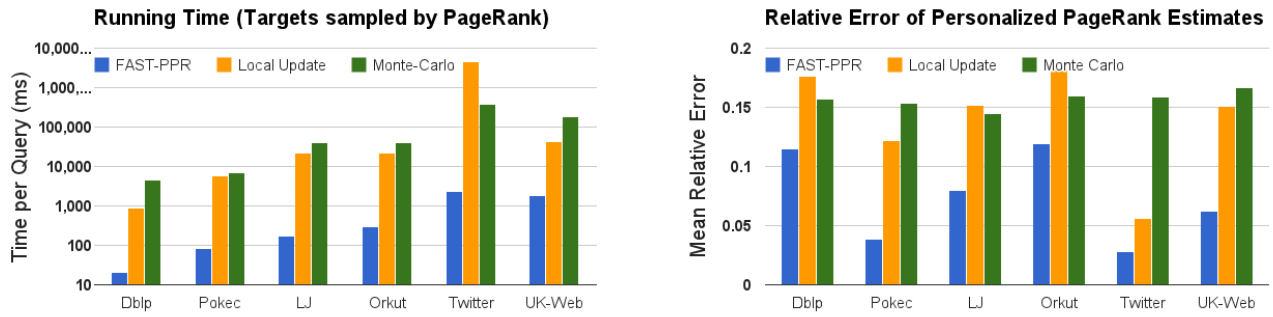
The success of modern networks is largely due to the ability to search effectively on them. A key primitive is PageRank [1], which is widely used as a measure of network importance. The popularity of PageRank is in large part due to its fast computation in large networks. As modern social network applications shift towards being more customized to individuals, there is a need for similar ego-centric measures of network structure.

Personalized PageRank (PPR) [1] has long been viewed as the appropriate ego-centric equivalent of PageRank. For a node u, the personalized PageRank vector π_u measures the frequency of visiting other nodes via short random-walks from u. This makes it an ideal metric for *social search*, giving higher weight to content generated by nearby users in the social graph. Social search protocols find widespread use – from personalization of general web searches [1, 2, 3], to more specific applications like collaborative tagging networks [4], ranking name search results on social networks [5], social Q&A sites [6], etc. In a typical personalized search application, given a set of candidate results for a query, we want to estimate the Personalized PageRank to each candidate result. This motivates the following problem:

Given source node s and target node t, compute the Personalized PageRank $\pi_s(t)$ up to a small relative error.

Since smaller values of $\pi_s(t)$ are more difficult to detect, we parameterize the problem by threshold δ, requiring small relative errors only if $\pi_s(t) > \delta$. Current techniques used for PPR estimation (see Section 2.1) have $\Omega(1/\delta)$ running-time – this makes them infeasible for large networks when the desired $\delta = O(1/n)$ or $O((\log n)/n)$.

In addition to social-search, PPR is also used for a variety of other tasks across different domains: friend recommendation on Facebook [7], who to follow on Twitter [8], graph partitioning [9], community detection [10], and other applications [11]. Other measures of personalization, such as personalized SALSA and SimRank [12], can be reduced to PPR. However, in spite of a rich body of existing work [2,



(a) Running time (in log scale) of different algorithms

(b) Relative accuracy of different algorithms

Figure 1: Comparison of Balanced FAST-PPR, Monte-Carlo and Local-Update algorithms in different networks – 1000 source-target pairs, threshold $\delta = \frac{4}{n}$, teleport probability $\alpha = 0.2$. Notice that Balanced FAST-PPR is 20 times faster in all graphs, without sacrificing accuracy. For details, see Section 6.

13, 9, 14, 15, 16, 17], estimating PPR is often a bottleneck in large networks.

1.1 Our Contributions

We develop a new algorithm, *Frontier-Aided Significance Thresholding for Personalized PageRank* (FAST-PPR), based on a new bi-directional search technique for PPR estimation:

• **Practical Contributions:** We present a simple implementation of FAST-PPR which requires no pre-processing and has an average running-time of $O(\sqrt{d/\delta})$[1]. We also propose a simple heuristic, Balanced FAST-PPR, that achieves a significant speedup in practice.

In experiments, FAST-PPR outperforms existing algorithms across a variety of real-life networks. For example, in Figure 1, we compare the running-times and accuracies of FAST-PPR with existing methods. Over a variety of data sets, FAST-PPR is significantly faster than the state of the art, with the same or better accuracy.

To give a concrete example: in experiments on the Twitter-2010 graph [19], Balanced FAST-PPR takes less than 3 seconds for random source-target pairs. In contrast, Monte Carlo takes more than 6 minutes and Local Update takes more than an hour. More generally in all graphs, FAST-PPR is at least 20 times faster than the state-of-the-art, without sacrificing accuracy.

• **Theoretical Novelty:** FAST-PPR is the first algorithm for PPR estimation with $O(\sqrt{d/\delta})$ average running-time, where $d = m/n$ is the average in-degree[2]. Further, we modify FAST-PPR to get $O(1/\sqrt{\delta})$ worst-case running-time, by pre-computing and storing some additional information, with a required storage of $O(m/\sqrt{\delta})$.

We also give a new running-time *lower bound* of $\Omega(1/\sqrt{\delta})$ for PPR estimation, which essentially shows that the dependence of FAST-PPR running-time on δ cannot be improved.

Finally, we note that FAST-PPR has the same performance gains for computing PageRank with arbitrary *preference vectors* [2], where the source is picked from a distribution over nodes. Different preference vectors are used for various applications [2, 1]. However, for simplicity of presentation, we focus on Personalized PageRank in this work.

2. PRELIMINARIES

Given $G(V, E)$, a directed graph, with $|V| = n, |E| = m$, and adjacency matrix A. For any node $u \in V$, we denote $\mathcal{N}^{out}(u), d^{out}(u)$ as the out-neighborhood and out-degree respectively; similarly $\mathcal{N}^{in}(u), d^{in}(u)$ are the in-neighborhood and in-degree. We define $d = \frac{m}{n}$ to be the average in-degree (equivalently, average out-degree).

The personalized PageRank vector π_u for a node $u \in V$ is the stationary distribution of the following random walk starting from u: at each step, return to u with probability α, and otherwise move to a random out-neighbor of the current node. Defining $D = \text{diag}(d^{out}(u))$, and $W = D^{-1}A$, the *personalized PageRank* (PPR) vector of u is given by:

$$\pi_u^T = \alpha \mathbf{e}_u^T + (1-\alpha)\pi_u^T.W, \tag{1}$$

where $\mathbf{e}_u$ is the identity vector of u. Also, for a target node t, we define the *inverse-PPR* of a node w with respect to t as $\pi_t^{-1}(w) = \pi_w(t)$. The inverse-PPR vector $\{\pi_t^{-1}(w)\}_{w \in V}$ of t sums to $n\pi(t)$, where $\pi(t)$ is the global PageRank of t.

Note that the PPR for a uniform random pair of nodes is $1/n$ – thus for practical applications, we need to consider δ of the form $O(1/n)$ or $O(\log n/n)$. We reinforce this choice of δ using empirical PPR data from the Twitter-2010 graph in Section 6.2 – in particular, we observe that only 1% of (s, t) pairs have PPR greater than $4/n$. In order to process real-time queries, one option would be precompute all answers, and for each node u, store a list of all targets with PPR at least δ. This results in $O(1)$ running-time at query, but requires $\Omega(1/\delta)$ storage per node, which is not feasible in large networks. The other extreme is to eschew storage and compute the PPR at query time – however, as we discuss

[1]We assume here that the desired relative error and teleport probability α are constants – complete scaling details are provided in our technical report [18].

[2]Formally, for (s, t) with $\pi_s(t) > \delta$, FAST-PPR returns an estimate $\widehat{\pi}_s(t)$ with relative error c, incurring $O\left(\frac{1}{c^2}\sqrt{\frac{d}{\delta}}\sqrt{\frac{\log(1/p_{fail})\log(1/\delta)}{\alpha^2\log(1/(1-\alpha))}}\right)$ average running-time; refer to our technical report [18] for complete details.

below, existing algorithms for computing PPR have a worst-case running-time of $\Omega(1/\delta)$. For the relevant values of δ, this is infeasible.

2.1 Existing Approaches for PPR Estimation

There are two main techniques used to compute PageRank/PPR vectors. One set of algorithms use the power iteration. Since performing a direct power iteration may be infeasible in large networks, a more common approach is to use *local-update* versions of the power method, similar to the Jacobi iteration. This technique was first proposed by Jeh and Widom [2], and subsequently improved by other researchers [20, 9]. The algorithms are primarily based on the following recurrence relation for π_u:

$$\pi_u^T = \alpha e_u^T + \frac{(1-\alpha)}{|\mathcal{N}^{out}(u)|} \cdot \sum_{v \in \mathcal{N}^{out}(u)} \pi_v^T \tag{2}$$

Another use of such local update algorithms is for estimating the inverse-PPR vector for a target node. Local-update algorithms for inverse-PageRank are given in [14] (where inverse-PPR is referred to as the 'contribution PageRank vector'), and [21] (where it is called 'susceptibility'). However, one can exhibit graphs where these algorithms need a running-time of $O(1/\delta)$ to get additive guarantees on the order of δ.

Eqn. 2 can be derived from the following probabilistic re-interpretations of PPR, which also lead to an alternate set of randomized or *Monte-Carlo* algorithms. Given any random variable L taking values in $\mathbb{N}_0$, let $RW(u, L) \triangleq \{u, V_1, V_2, \ldots, V_L\}$ be a random-walk of random length $L \sim Geom(\alpha)$[3], starting from u. Then we can write:

$$\pi_u(v) = \mathbb{P}[V_L = v] \tag{3}$$

In other words, $\pi_u(v)$ is the probability that v is the last node in $RW(u, L)$. Another alternative characterization is:

$$\pi_u(v) = \alpha \mathbb{E}\left[\sum_{i=0}^{L} \mathbb{1}_{\{V_i = v\}}\right],$$

i.e., $\pi_u(v)$ is proportional to the number of times $RW(u, L)$ visits node v. Both characterizations can be used to estimate $\pi_u(\cdot)$ via Monte Carlo algorithms, by generating and storing random walks at each node [13, 15, 16, 17]. Such estimates are easy to update in dynamic settings [15]. However, for estimating PPR values close to the desired threshold δ, these algorithms need $\Omega(1/\delta)$ random-walk samples.

2.2 Intuition for our approach

The problem with the basic Monte Carlo procedure – generating random walks from s and estimating the distribution of terminal nodes – is that to estimate a PPR which is $O(\delta)$, we need $\Omega(1/\delta)$ walks. To circumvent this, we introduce a new *bi-directional estimator* for PPR: given a PPR-estimation query with parameters (s, t, δ), we first work backward from t to find a suitably large set of 'targets', and then do random walks from s to test for hitting this set.

Our algorithm can be best understood through an analogy with the shortest path problem. In the bidirectional shortest path algorithm, to find a path of length l from node s to node t, we find all nodes within distance $\frac{l}{2}$ of t, find all nodes within distance $\frac{l}{2}$ of s, and check if these sets intersect. Similarly, to test if $\pi_s(t) > \delta$, we find all w with $\pi_w(t) > \sqrt{\delta}$ (we call this the *target set*), take $O(1/\sqrt{\delta})$ walks from the start node, and see if these two sets intersect. It turns out that these sets might not intersect even if $\pi_s(t) > \delta$, so we go one step further and consider the *frontier set* – nodes outside the target set which have an edge into the target set. We can prove that if $\pi_s(t) > \delta$ then random walks are likely to hit the frontier set.

Our method is most easily understood using the characterization of $\pi_s(t)$ as the probability that a single walk from s ends at t (Eqn. (3)). Consider a random walk from s to t – at some point, it must enter the frontier. We can then decompose the probability of the walk reaching t into the product of two probabilities: the probability that it reaches some node w in the frontier, and the probability that it reaches t starting from w. The two probabilities in this estimate are typically much larger than the overall probability of a random walk reaching t from s, so they can be estimated more efficiently. Figure 2 illustrates this bi-directional scheme.

2.3 Additional Definitions

To formalize our algorithm, we first need some additional definitions. We define a set B to be a blanket set for t with respect to s if all paths from s to t pass through B. Given blanket set B and a random walk $RW(s, L)$, let H_B be the first node in B hit by the walk (defining $H_B = \perp$ if the walk does not hit B before terminating). Since each walk corresponds to a unique H_B, we can write $\mathbb{P}[V_L = v]$ as a sum of contributions from each node in B. Further, from the memoryless property of the geometric random variable, the probability a walk ends at t conditioned on reaching $w \in B$ before stopping is exactly $\pi_s(t)$. Combining these, we have:

$$\pi_s(t) = \sum_{w \in B} \mathbb{P}[H_B = w] \cdot \pi_w(t). \tag{4}$$

In other words, the PPR from s to t is the sum, over all nodes w in blanket set B, of the probability of a random walk hitting B first at node w, times the PPR from w to t.

Recall we define the *inverse-PPR vector* of t as $\pi_t^{-1} = (\pi_w(t))_{w \in V}$. Now we introduce two crucial definitions:

DEFINITION 1 (TARGET SET). *The target set $T_t(\epsilon_r)$ for a target node t is given by:*

$$T_t(\epsilon_r) := \{w \in V : \pi_t^{-1}(w) > \epsilon_r\}.$$

DEFINITION 2 (FRONTIER SET). *The frontier set $F_t(\epsilon_r)$ for a target node t is defined as:*

$$F_t(\epsilon_r) := \left(\bigcup_{v \in T_t(\epsilon_r)} \mathcal{N}_v^{in}\right) \setminus T_t(\epsilon_r).$$

The target set $T_t(\epsilon_r)$ thus contains all nodes with inverse-PPR greater than ϵ_r, while the frontier set $F_t(\epsilon_r)$ contains all nodes which are in-neighbors of $T_t(\epsilon_r)$, but not in $T_t(\epsilon_r)$.

2.4 A Bidirectional Estimator

The next proposition illustrates the core of our approach.

PROPOSITION 1. *Set $\epsilon_r < \alpha$. Fix vertices s, t such that $s \notin T_t(\epsilon_r)$.*

1. *Frontier set $F_t(\epsilon_r)$ is a blanket set of t with respect to s.*

[3] i.e., $\mathbb{P}[L = i] = \alpha(1-\alpha)^i \; \forall \, i \in \mathbb{N}_0$

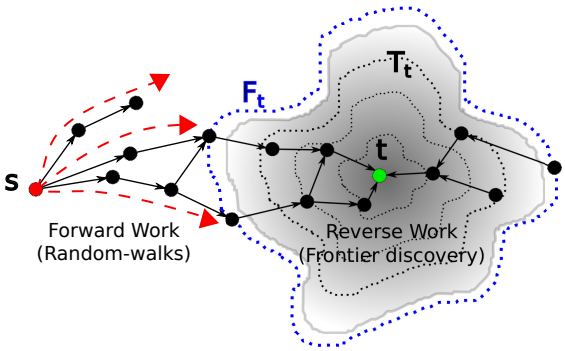


Figure 2: The FAST-PPR Algorithm: We first work backward from the target t to find the frontier F_t (with inverse-PPR estimates). Next we work forward from the source s, generating random-walks and testing for hitting the frontier.

2. For random-walk $RW(s, L)$ with length $L \sim Geom(\alpha)$:

$$\mathbb{P}[RW(s, L) \text{ hits } F_t(\epsilon_r)] \geq \frac{\pi_s(t)}{\epsilon_r}$$

PROOF. For brevity, let $T_t = T_t(\epsilon_r), F_t = F_t(\epsilon_r)$. By definition, we know $\pi_t(t) \geq \alpha$ – thus $t \in T_t$, since $\epsilon_r < \alpha$. The frontier set F_t contains all neighbors of nodes in T_t which are not themselves in T_t – hence, for any source node $u \notin T_t$, a path to t must first hit a node in F_t.

For the second part, since F_t is a blanket set for s with respect to t, Eqn. 4 implies $\pi_s(t) = \sum_{w \in F_t} \mathbb{P}[H_{F_t} = w]\, \pi_w(t)$, where H_{F_t} is the first node where the walk hits F_t. Note that by definition, $\forall w \in F_t$ we have $w \notin T_t$ – thus $\pi_w(t) \leq \epsilon_r$. Applying this bound, we get:

$$\pi_s(t) \leq \epsilon_r \sum_{w \in F_t} \mathbb{P}[H_{F_t} = w] = \epsilon_r \mathbb{P}[RW(s, L) \text{ hits } F_t(\epsilon_r)].$$

Rearranging, we get the result. □

The aim is to estimate $\pi_s(t)$ through Eqn. 4. By the previous proposition, $F_t(\epsilon_r)$ is a blanket set. We will determine the set $F_t(\epsilon_r)$ and estimate all quantities in the right side of Eqn. 4, thereby estimating $\pi_s(t)$.

We perform a simple heuristic calculation to argue that setting $\epsilon_r \approx \sqrt{\delta}$ suffices to estimate $\pi_s(t)$. Previous work shows that $T_t(\epsilon_r)$ can be found in $O(d/\epsilon_r)$ time [14, 21] – using this we can find $F_t(\epsilon_r)$. Now suppose that we know all values of $\pi_w(t)$ $(\forall w \in F_t(\epsilon_r))$. By Eqn. 4, we need to estimate the probability of random walks from s hitting vertices in $F_t(\epsilon_r)$. By the previous proposition, the probability of hitting $F_t(\epsilon_r)$ is at least δ/ϵ_r – hence, we need $O(\epsilon_r/\delta)$ walks from s to ensure we hit $F_t(\epsilon_r)$. All in all, we require $O(d/\epsilon_r + \epsilon_r/\delta)$ – setting $\epsilon_r = \sqrt{d\delta}$ we get a running-time bound of $O(\sqrt{d/\delta})$. In reality, however, we only have (coarse) PPR estimates for nodes in the frontier – we show how these estimates can be boosted to get the desired guarantees, and also empirically show that, in practice, using the frontier estimates gives good results. Finally, we show that $1/\sqrt{\delta}$ is a fundamental lower bound for this problem.

We note that bi-directional techniques have been used for estimating fixed-length random walk probabilities in *regular undirected graphs* [22, 23]. These techniques do not extend to estimating PPR – in particular, we need to consider *directed graphs, arbitrary node degrees and walks of random length.* Also, Jeh and Widom [2] proposed a scheme for PageRank estimation using intermediate estimates from a *fixed skeleton* of target nodes. However there are no running-time guarantees for such schemes; also, the target nodes and partial estimates need to be pre-computed and stored. Our algorithm is fundamentally different as it constructs *separate target sets for each target node* at query-time.

3. THE FAST-PPR ALGORITHM

We now develop the *Frontier-Aided Significance Thresholding* algorithm, or FAST-PPR, specified in Algorithm 1. The input-output behavior of FAST-PPR is as follows:

- **Inputs:** The primary inputs are graph G, teleport probability α, start node s, target node t, and threshold δ – for brevity, we suppress the dependence on G and α. We also need a *reverse threshold* ϵ_r – in subsequent sections, we discuss how this parameter is chosen.
- **Output:** An estimate $\widehat{\pi}_s(t)$ for $\pi_s(t)$.

The algorithm also requires two parameters, c and β – the former controls the number of random walks, while the latter controls the quality of our inverse-PPR estimates in the target set. In our pseudocode (Algorithms 1 and2), we specify the values we use in our experiments – the theoretical basis for these choices is provided in Section 3.2.

Algorithm 1 FAST-PPR(s, t, δ)

Inputs: graph G, teleport probability α, start node s, target node t, threshold δ

Set accuracy parameters c, β (in our experiments we use $c = 350$, $\beta = 1/6$).
Call FRONTIER(t, ϵ_r, β) to obtain target set $T_t(\epsilon_r)$, frontier set $F_t(\epsilon_r)$, and inverse-PPR values $(\pi_t^{-1}(w))_{w \in F_t(\epsilon_r) \cup T_t(\epsilon_r)}$.
if $s \in T_t(\epsilon_r)$ **then**
  return $\pi_t^{-1}(s)$
else
  Set number of walks $k = c\epsilon_r/\delta$ (See Theorem 3 for details)
  for index $i \in [k]$ **do**
    Generate $L_i \sim Geom(\alpha)$
    Generate random-walk $RW(s, L_i)$
    Determine H_i, the first node in $F_t(\epsilon_r)$ hit by RW_i; if RW_i never hits $F_t(\epsilon_r)$, set $H_i = \perp$
  end for
  return $\widehat{\pi}_s(t) = (1/k) \sum_{i \in [k]} \pi_t^{-1}(H_i)$
end if

FAST-PPR needs to know sets $T_t(\epsilon_r), F_t(\epsilon_r)$ and inverse-PPR values $(\pi_t^{-1}(w))_{w \in F_t(\epsilon_r) \cup T_t(\epsilon_r)}$. These can be obtained (approximately) from existing algorithms of [14, 21]. For the sake of completeness, we provide pseudocode for the procedure FRONTIER (Algorithm 2) that obtains this information. The following combines Theorems 1 and 2 in [21].

THEOREM 1. *FRONTIER(t, ϵ_r, β) algorithm computes estimates $\widehat{\pi}_t^{-1}(w)$ for every vertex w, with a guarantee that $\forall w$, $|\widehat{\pi}_t^{-1}(w) - \pi_t^{-1}(w)| < \beta\epsilon_r$. The average running time (over all choices of t) is $O(d/(\alpha\epsilon_r))$, where $d = m/n$ is the average degree of the graph.*

Observe that the estimates $\widehat{\pi}_t^{-1}(w)$ are used to find approximate target and frontier sets. Note that the running

Algorithm 2 FRONTIER$(G, \alpha, t, \epsilon_r, \beta)$ [14, 21]

Inputs: graph G, teleport probability α, target node t, reverse threshold ϵ_r, accuracy factor β

Define additive error $\epsilon_{inv} = \beta\epsilon_r$

Initialize (sparse) estimate-vector $\widehat{\pi}_t^{-1}$ and (sparse) residual-vector r_t as: $\begin{cases}\widehat{\pi}_t^{-1}(u) = r_t(u) = 0 \text{ if } u \neq t \\ \widehat{\pi}_t^{-1}(t) = r_t(t) = \alpha\end{cases}$

Initialize target-set $\widehat{T}_t = \{t\}$, frontier-set $\widehat{F}_t = \{\}$

while $\exists w \in V \; s.t. \; r_t(w) > \alpha\epsilon_{inv}$ **do**

  for $u \in \mathcal{N}^{in}(w)$ **do**

    $\Delta = (1-\alpha).\frac{r_t(w)}{d^{out}(u)}$

    $\widehat{\pi}_t^{-1}(u) = \widehat{\pi}_t^{-1}(u) + \Delta, r_t(u) = r_t(u) + \Delta$

    if $\widehat{\pi}_t^{-1}(u) > \epsilon_r$ **then**

      $\widehat{T}_t = \widehat{T}_t \cup \{u\}$, $\widehat{F}_t = \widehat{F}_t \cup \mathcal{N}^{in}(u)$

    end if

  end for

  $r_t(w) = 0$

end while

$\widehat{F}_t = \widehat{F}_t \setminus \widehat{T}_t$

return $\widehat{T}_t, \widehat{F}_t, (\widehat{\pi}_t^{-1}(w))_{w \in \widehat{F}_t \cup \widehat{T}_t}$

time for a given t is proportional to the frontier size $|\widehat{F}_t|$. It is relatively straightforward to argue (as in [21]) that $\sum_{t \in V} |\widehat{F}_t| = \Theta(nd/(\alpha\epsilon_r))$.

In the subsequent subsections, we present theoretical analyses of the running times and correctness of FAST-PPR. The correctness proof makes an excessively strong assumption of perfect outputs for FRONTIER, which is not true. To handle this problem, we have a more complex variant of FAST-PPR that can be proven theoretically – refer to our technical report [18] for details. Nonetheless, our empirical results show that FAST-PPR does an excellent job of estimating $\pi_s(t)$.

3.1 Running-time of FAST-PPR

THEOREM 2. *Given parameters δ, ϵ_r, the running-time of the FAST-PPR algorithm, averaged over uniform-random pairs s, t, is $O(\alpha^{-1}(d/\epsilon_r + \epsilon_r/\delta))$.*

PROOF. Each random walk $RW(u, Geom(\alpha))$ takes $1/\alpha$ steps on average, and there are $O(\epsilon_r/\delta)$ such walks performed – this is the *forward time*. On the other hand, from Theorem 1, we have that for a random (s,t) pair, the average running time of FRONTIER is $O(d/(\alpha\epsilon_r))$ – this is the *reverse time*. Combining the two, we get the result. □

Note that the reverse time bound above is averaged across choice of target node; for some target nodes (those with high global PageRank) the reverse time may be much larger than average, while for others it may be smaller. However, the forward time is similar for all source nodes, and is predictable – we exploit this in Section 5.2 to design a balanced version of FAST-PPR which is much faster in practice. In terms of theoretical bounds, the above result suggests an obvious choice of ϵ_r to optimize the running time:

COROLLARY 1. *Set $\epsilon_r = \sqrt{d\delta}$. Then FAST-PPR has an average per-query running-time of $O(\alpha^{-1}\sqrt{d/\delta})$.*

3.2 FAST-PPR with Perfect FRONTIER

We now analyze FAST-PPR in an idealized setting, where we assume that FRONTIER returns *exact inverse-PPR estimates* – i.e., the sets $T_t(\epsilon_r)$, $F_t(\epsilon_r)$, and the values $\{\pi_t^{-1}(w)\}$ are known exactly. This is an unrealistic assumption, but it gives much intuition into *why* FAST-PPR works. In particular, we show that if $\pi(s,t) > \delta$, then with probability at least 99%, FAST-PPR returns an estimate with relative error at most 1/4; furthermore, if $\pi(s,t) < \delta$, then FAST-PPR returns an estimate with additive error at most $\delta/4$. Note that the above guarantees are chosen for ease of exposition – in our main theoretical result (provided in our technical report [18]), we show that FAST-PPR can achieve any desired relative error target and success probability.

THEOREM 3. *For any s, t, δ, ϵ_r, FAST-PPR outputs an estimate $\widehat{\pi}_s(t)$ such that with probability > 0.99:*

$$|\pi_s(t) - \widehat{\pi}_s(t)| \leq \max(\delta, \pi_s(t))/4.$$

PROOF. We choose $c = \max(48 \cdot 8e\ln(100), 4\log_2(100))$; this choice of parameter c is for ease of exposition in the computations below, and has not been optimized.

To prove the result, note that FAST-PPR performs $k = c\epsilon_r/\delta$ i.i.d. random walks $RW(s, L)$. We use H_i to denote the first node in $F_t(\epsilon_r)$ hit by the ith random walk. Let $X_i = \pi_t^{-1}(H_i)$ and $X = \sum_{i=1}^{k} X_i$. By Eqn. 4, $\mathbb{E}[X_i] = \pi_s(t)$, so $\mathbb{E}[X] = k\pi_s(t)$. Note that $\widehat{\pi}_s(t) = X/k$. As result, $|\pi_s(t) - \widehat{\pi}_s(t)|$ is exactly $|X - \mathbb{E}[X]|/k$.

It is convenient to define scaled random variables $Y_i = X_i/\epsilon_r$ and $Y = X/\epsilon_r$ before we apply standard Chernoff bounds. We have $|\pi_s(t) - \widehat{\pi}_s(t)| = (\epsilon_r/k)|Y - \mathbb{E}[Y]| = (\delta/c)|Y - \mathbb{E}[Y]|$. Also, $\pi_s(t) = (\delta/c)\mathbb{E}[Y]$. Crucially, because $H_i \in F_t(\epsilon_r)$, $X_i = \pi_{H_i}(t) < \epsilon_r$, so $Y_i \leq 1$. Hence, we can apply the following two Chernoff bounds (refer to Theorem 1.1 in [24]):

1. $\mathbb{P}[|Y - \mathbb{E}[Y]| > \mathbb{E}[Y]/4] < \exp(-\mathbb{E}[Y]/48)$
2. For any $b > 2e\mathbb{E}[Y], \mathbb{P}[Y > b] \leq 2^{-b}$

Now, we perform a case analysis. Suppose $\pi_s(t) > \delta/(4e)$. Then $\mathbb{E}[Y] > c/(4e)$, and

$$\begin{aligned}\mathbb{P}[|\pi_s(t) - \widehat{\pi}_s(t)| > \pi_s(t)/4] &= \mathbb{P}[|Y - \mathbb{E}[Y]| > \mathbb{E}[Y]/4] \\ &< \exp(-c/48 \cdot 4e) < 0.01\end{aligned}$$

Suppose $\pi_s(t) \leq \delta/(4e)$. Then, $\delta/4 > 2e\pi_s(t)$ implying $c/4 > 2e\mathbb{E}[Y]$. By the upper tail:

$$\mathbb{P}[\widehat{\pi}_s(t) > \delta/4] = \mathbb{P}[Y > c/4] \leq 2^{-c/4} < 0.01$$

The proof is completed by trivially combining both cases. □

4. LOWER BOUND FOR PPR ESTIMATION

In this section, we prove that any algorithm that accurately estimates PPR queries up to a threshold δ must look at $\Omega(1/\sqrt{\delta})$ edges of the graph. Thus, our algorithms have the optimal dependence on δ. The numerical constants below are chosen for easier calculations, and are not optimized.

We assume $\alpha = 1/100\log(1/\delta)$, and consider randomized algorithms for the following variant of Significant-PPR, which we denote as *Detect-High*(δ) – for all pairs (s,t):

- If $\pi_s(t) > \delta$, output ACCEPT with probability $> 9/10$.
- If $\pi_s(t) < \frac{\delta}{2}$, output REJECT with probability $> 9/10$.

We stress that the probability is over the random choices of the algorithm, *not* over s, t. We now have the following lower bound:

THEOREM 4. *Any algorithm for* Detect-High(δ) *must access* $\Omega(1/\sqrt{\delta})$ *edges of the graph.*

PROOF OUTLINE. The proof uses a lower bound of Goldreich and Ron for *expansion testing* [25]. The technical content of this result is the following – consider two distributions $\mathcal{G}_1$ and $\mathcal{G}_2$ of undirected 3-regular graphs on N nodes. A graph in $\mathcal{G}_1$ is generated by choosing three uniform-random perfect matchings of the nodes. A graph in $\mathcal{G}_2$ is generated by randomly partitioning the nodes into 4 equally sized sets, $V_i, i \in \{1, 2, 3, 4\}$, and then, within each V_i, choosing three uniform-random matchings.

Consider the problem of distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$. An adversary arbitrarily picks one of these distributions, and generates a graph G from it. A *distinguisher* must report whether G came from $\mathcal{G}_1$ or $\mathcal{G}_2$, and it must be correct with probability $> 2/3$ regardless of the distribution chosen. Theorem 7.5 of Goldreich and Ron [25] asserts the following:

THEOREM 5 (THEOREM 7.5 OF [25]). *Any distinguisher must look at* $\sqrt{N}/5$ *edges of the graph.*

We perform a direct reduction, relating the *Detect-High*(δ) problem to the Goldreich and Ron setting – in particular, we show that an algorithm for *Detect-High*(δ) which requires less than $1/\sqrt{\delta}$ queries can be used to construct a distinguisher which violates Theorem 5. The complete proof is provided in our technical report [18]. □

5. FURTHER VARIANTS OF FAST-PPR

The previous section describes vanilla FAST-PPR, with a proof of correctness assuming a perfect FRONTIER. We now present some variants of FAST-PPR. We give a theoretical variant that is a truly provable algorithm, with no assumptions required – however, vanilla FAST-PPR is a much better practical candidate and it is what we implement. We also discuss how we can use pre-computation and storage to obtain worst-case guarantees for FAST-PPR. Finally, from the practical side, we discuss a workload-balancing heuristic that provides significant improvements in running-time by dynamically adjusting ϵ_r.

5.1 Using Approximate Frontier-Estimates

The assumption that FRONTIER returns perfect estimates is theoretically untenable – Theorem 1 only ensures that each inverse-PPR estimate is correct up to an additive factor of $\epsilon_{inv} = \beta\epsilon_r$. It is plausible that for every w in the frontier $\widehat{F}_t(\epsilon_r)$, $\pi_t^{-1}(w) < \epsilon_{inv}$, and FRONTIER may return a zero estimate for these PPR values. It is not clear how to use these noisy inverse-PPR estimates to get the desired accuracy guarantees.

To circumvent this problem, we observe that estimates $\pi_t^{-1}(w)$ for any node $w \in \widehat{T}_t(\epsilon_r)$ *are in fact accurate up to a multiplicative factor.* We design a procedure to bootstrap these 'good' estimates, by using a special 'target-avoiding' random walk – this modified algorithm gives the desired accuracy guarantee with only an additional $\log(1/\delta)$ factor in running-time. The final algorithm and proof are quite intricate – due to lack of space, the details are deferred to our technical report [18].

5.2 Balanced FAST-PPR

In FAST-PPR, the parameter ϵ_r can be chosen freely while preserving accuracy. Choosing a larger value leads to a smaller frontier and less forward work at the cost of more reverse work; a smaller value requires more reverse work and fewer random walks. To improve performance in practice, we can optimize the choice of ϵ_r based on the target t, to balance the reverse and forward time. Note that for any value of ϵ_r, the forward time is proportional to $k = c\epsilon_r/\delta$ (the number of random walks performed) – for any choice of ϵ_r, it is easy to estimate the forward time required. Thus, instead of committing to a single value of ϵ_r, we propose a heuristic wherein we *dynamically decrease* ϵ_r *until the estimated remaining forward time equals the reverse time already spent.*

We now describe this Balanced FAST-PPR algorithm in brief. Instead of pushing from any node w with residual $r_t(w)$ above a fixed threshold $\alpha\epsilon_{inv}$, we now push from the node w with the largest residual value – this follows a similar algorithm proposed in [21]. From [14, 21], we know that a current maximum residual value of r_{max} implies an additive error guarantee of $\frac{r_{max}}{\alpha}$ – this corresponds to a dynamic ϵ_r value of $\frac{r_{max}}{\alpha\beta}$. At this value of ϵ_r, the number of forward walks required is $k = c\epsilon_r/\delta$. By multiplying k by the average time needed to generate a walk, we get a good prediction the amount of forward work still needed – we can then compare it to the time spent on revere-work and adjust ϵ_r till they are equal. Thus Balanced Fast-PPR is able to dynamically choose ϵ_r to balance the forward and reverse running-time. In Section 6.5, we experimentally show how this change balances forward and reverse running-time, and significantly reduces the average running-time.

5.3 FAST-PPR using Stored Oracles

All our results for FAST-PPR have involved average-case running-time bounds. To convert these to worst-case running-time bounds, we can precompute and store the frontier for all nodes, and only perform random walks at query time. To obtain the corresponding storage requirement for these *Frontier oracles*, observe that for any node $w \in V$, it can belong to the target set of at most $\frac{1}{\epsilon_r}$ nodes, as $\sum_{t \in V} \pi_t^{-1}(w) = 1$. Summing over all nodes, we have:

$$\begin{aligned}\text{Total Storage} &\leq \sum_{t \in V} \sum_{w \in T_t} \sum_{u \in \mathcal{N}^{in}(w)} \mathbb{1}_{\{u \in F_t\}} \\ &\leq \sum_{w \in V} \sum_{t \in V : w \in T_t} d^{in}(w) \leq \frac{m}{\epsilon_r}\end{aligned}$$

To further cut down on running-time, we can also precompute and store the random-walks from all nodes, and perform appropriate joins at query time to get the FAST-PPR estimate. This allows us to implement FAST-PPR on any distributed system that can do fast intersections/joins. More generally, it demonstrates how the modularity of FAST-PPR can be used to get variants that tradeoff between different resources in practical implementations.

6. EXPERIMENTS

We conduct experiments to explore three main questions:

1. How fast is FAST-PPR relative to previous algorithms?
2. How accurate are FAST-PPR's estimates?
3. How is FAST-PPR's performance affected by our design choices: use of frontier and balancing forward/reverse running-time?

Table 1: Datasets used in experiments

Dataset	Type	# Nodes	# Edges
DBLP-2011	undirected	1.0M	6.7M
Pokec	directed	1.6M	30.6M
LiveJournal	undirected	4.8M	69M
Orkut	undirected	3.1M	117M
Twitter-2010	directed	42M	1.5B
UK-2007-05	directed	106M	3.7B

6.1 Experimental Setup

• **Data-Sets:** To measure the robustness of FAST-PPR, we run our experiments on several types and sizes of graph, as described in Table 1. Pokec and Twitter are both social networks in which edges are directed. The LiveJournal, Orkut (social networks) and DBLP (collaborations on papers) networks are all undirected – for each, we have the largest connected component of the overall graph. Finally, our largest dataset with 3.7 billion edges is from a 2007 crawl of the UK domain [26, 27]. Each vertex is a web page and each edge is a hyperlink between pages.

For detailed studies of FAST-PPR, we use the Twitter-2010 graph, with 41 million users and 1.5 billion edges. This presents a further algorithmic challenge because of the skew of its degree distribution: the average degree is 35, but one node has more than 700,000 in-neighbors.

The Pokec [28], Live Journal [29], and Orkut [29] datasets were downloaded from the Stanford SNAP project [30]. The DBLP-2011 [26], Twitter-2010 [26] and UK 2007-05 Web Graph [26, 27] were downloaded from the Laboratory for Web Algorithmics [19].

• **Implementation Details:** We ran our experiments on a machine with a 3.33 GHz 12-core Intel Xeon X5680 processor, 12MB cache, and 192 GB of 1066 MHz Registered ECC DDR3 RAM. Each experiment ran on a single core and loaded the graph used into memory before beginning any timings. The RAM used by the experiments was dominated by the RAM needed to store the largest graph using the SNAP library format [30], which was about 21GB.

For reproducibility, our C++ source code is available at: http://cs.stanford.edu/~plofgren/fast_ppr/

• **Benchmarks:** We compare FAST-PPR to two benchmark algorithms: Monte-Carlo and Local-Update.

Monte-Carlo refers to the standard random-walk algorithm [13, 15, 16, 17] – we perform $\frac{c_{MC}}{\delta}$ walks and estimate $\pi_u(v)$ by the fraction of walks terminating at v. For our experiments, we choose $c_{MC} = 35$, to ensure that the relative errors for Monte-Carlo are the same as the relative error bounds chosen for Local-Update and FAST-PPR (see below). However, even in experiments with $c_{MC} = 1$, we find that FAST-PPR is still 3 times faster on all graphs and 25 times faster on the largest two graphs (refer to our technical report [18] for additional plots).

Our other benchmark, Local-Update, is the state-of-the-art local power iteration algorithm [14, 21]. It follows the same procedure as the FRONTIER algorithm (Algorithm 2), but with the additive accuracy ϵ_{inv} set to $\delta/2$. Note that a backward local-update is more suited to computing PPR forward schemes [9, 2] as the latter lack natural performance guarantees on graphs with high-degree nodes.

• **Parameters:** For FAST-PPR, we set the constants $c = 350$ and $\beta = 1/6$ – these are guided by the Chernoff bounds we use in the proof of Theorem 3. For vanilla FAST-PPR, we simply choose $\epsilon_r = \sqrt{\delta}$.

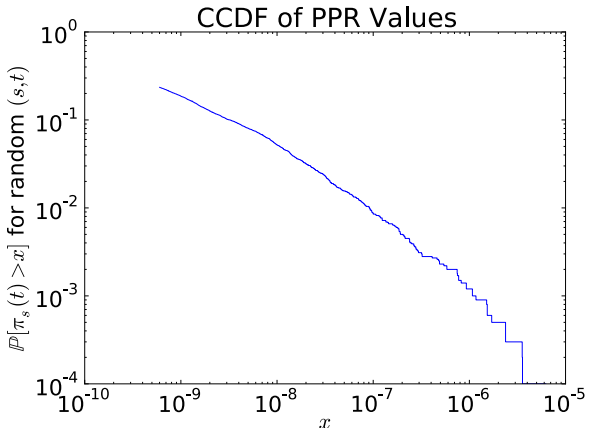


Figure 3: Complementary cumulative distribution for 10,000 (s,t) pairs sampled uniformly at random on the Twitter graph.

6.2 Distribution of PPR values

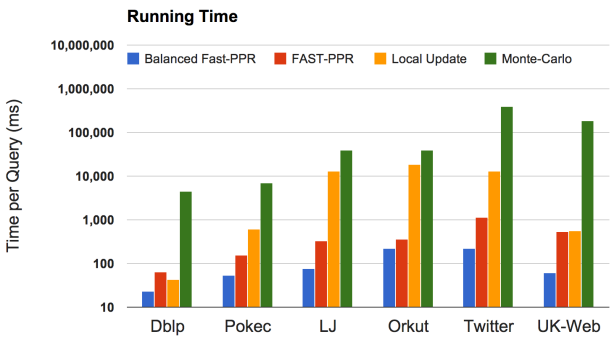
For all our experiments, we use $\delta = \frac{4}{n}$. To understand the importance of this threshold, we study the distribution of PPR values in real networks. Using the Twitter graph as an example, we choose 10,000 random (s,t) pairs and compute $\pi_s(t)$ using FAST-PPR to accuracy $\delta = \frac{n}{10}$. The complementary cumulative distribution function is shown on a log-log plot in Figure 3. Notice that the plot is roughly linear, suggesting a power-law. Because of this skewed distribution, only 2.8% of pairs have PPR above $\frac{1}{n}$ = 2.4e-8, and less than 1% have PPR over $\frac{4}{n}$ = 9.6e-8.

6.3 Running Time Comparisons

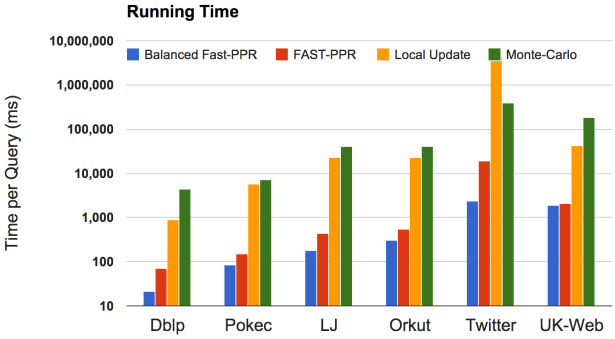
After loading the graph into memory, we sample 1000 source/target pairs (s,t) uniformly at random. For each, we measure the time required for answering PPR-estimation queries with threshold $\delta = 4/n$, which, as we discuss above, is fairly significant because of the skew of the PPR distribution. To keep the experiment length less than 24 hours, for the Local-Update algorithm and Monte-Carlo algorithms we only use 20 and 5 pairs respectively.

The running-time comparisons are shown in Figure 4 – we compare Monte-Carlo, Local-Update, vanilla FAST-PPR, and Balanced FAST-PPR. We perform an analogous experiment where target nodes are sampled according to their global PageRank value. This is a more realistic model for queries in personalized search applications, with searches biased towards more popular targets. The results, plotted in Figure 4(b), show even better speedups for FAST-PPR. All in all, FAST-PPR is many orders of magnitude faster than the state of the art.

The effect of target global PageRank: To quantify the speedup further, we sort the targets in the Twitter-2010 graph by their global PageRank, and choose the first target in each percentile. We measure the running-time of the four algorithms (averaging over random source nodes), as shown in Figure 5. Note that FAST-PPR is much faster than previous methods for the targets with high PageRank.



(a) Sampling targets uniformly



(b) Sampling targets from PageRank distribution

Figure 4: Average running-time (on log-scale) for different networks. We measure the time required for Significant-PPR queries (s, t, δ) with threshold $\delta = \frac{4}{n}$ for 1000 (s, t) pairs. For each pair, the start node is sampled uniformly, while the target node is sampled uniformly in Figure 4(a), or from the global PageRank distribution in Figure 4(b). In this plot we use teleport probability $\alpha = 0.2$.

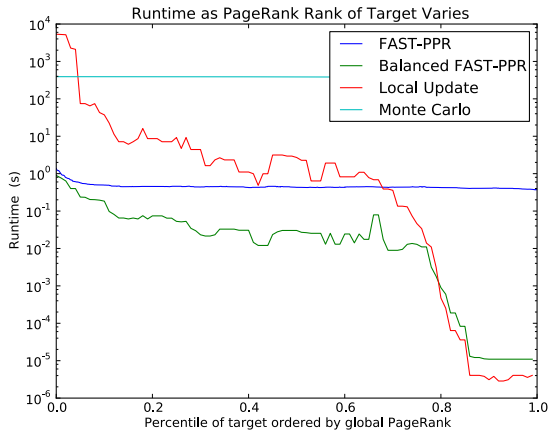


Figure 5: Execution time vs. global PageRank of the target. Target nodes were sorted by global PageRank, then one was chosen from each percentile. We use $\alpha = 0.2$, and 5-point median smoothing.

Note also that large PageRank targets account for most of the average running-time – thus improving performance in these cases causes significant speedups in the average running time. We also see that Balanced FAST-PPR has significant improvements over vanilla FAST-PPR, especially for lower PageRank targets.

To give a sense of the speedup achieved by FAST-PPR, consider the Twitter-2010 graph. Balanced FAST-PPR takes less than 3 seconds for Significant-PPR queries with targets sampled from global PageRank – in contrast, Monte Carlo takes more than 6 minutes and Local Update takes more than an hour. In the worst-case, Monte Carlo takes 6 minutes and Local Update takes 6 hours, while Balanced FAST-PPR takes 40 seconds. Finally, the estimates from FAST-PPR are twice as accurate as those from Local Update, and 6 times more accurate than those from Monte Carlo.

6.4 Measuring the Accuracy of FAST-PPR

We measure the empirical accuracy of FAST-PPR. For each graph, we sample 25 targets uniformly at random, and compute their ground truth inverse-PPR vectors by running a power iteration up to an additive error of $\delta/100$ (as before, we use $\delta = 4/n$). Since larger PPR values are easier to compute than smaller PPR values, we sample start nodes such that $\pi_s(t)$ is near the significance threshold δ. In particular, for each of the 25 targets t, we sample 50 random nodes from the set $\{s : \delta/4 \leq \pi_s(t) \leq \delta\}$ and 50 random nodes from the set $\{s : \delta \leq \pi_s(t) \leq 4\delta\}$.

We execute FAST-PPR for each of the 2500 (s, t) pairs, and measure the empirical error – the results are compiled in Table 2. Notice that FAST-PPR has mean relative error less than 15% and max relative error less than 65% on all graphs – this is sufficiently accurate to make it useful for personalized search.

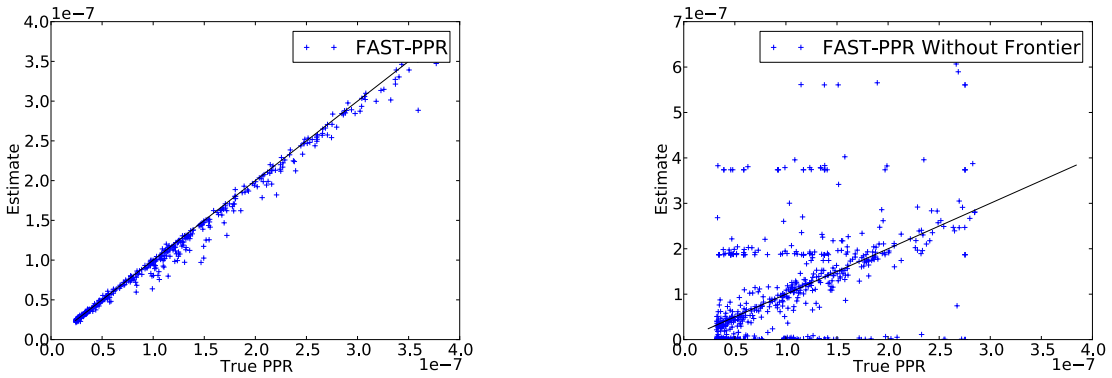
To make sure that FAST-PPR is not sacrificing accuracy for improved running-time, we also compute the relative error of Local-Update and Monte-Carlo, using the same parameters as for our running-time experiments. For each of the 2500 (s, t) pairs, we run Local-Update, and measure its relative error. For testing Monte-Carlo, we use our knowledge of the ground truth PPR, and the fact that each random-walk from s terminates at t with probability $p_s = \pi_s(t)$. This allows us to simulate Monte-Carlo by directly sampling from a Bernoulli variable with mean $\pi_s(t)$ – this statistically identical to generating random-walks and testing over all pairs. Note that actually simulating the walks would take more than 50 days of computation for 2500 pairs. The relative errors are shown in Figure 1(b). Notice that FAST-PPR is more accurate than the state-of-the-art competition on all graphs. This shows that our running time comparisons are using parameters settings that are fair.

6.5 Some Other Empirical Observations

• **Necessity of the Frontier:** Another question we study experimentally is whether we can modify FAST-PPR to compute the target set $T_t(\epsilon_r)$ and then run Monte-Carlo walks until they hit the target set (rather than the frontier set $F_t(\epsilon_r)$). This may appear natural, as the target set is also a blanket set, and we have good approximations for inverse-PPR values in the target set. Further, using only the

Table 2: Accuracy of FAST-PPR (with parameters as specified in Section 6.1)

	Dblp	Pokec	LJ	Orkut	Twitter	UK-Web
Threshold δ	4.06e-06	2.45e-06	8.25e-07	1.30e-06	9.60e-08	3.78e-08
Average Additive Error	5.8e-07	1.1e-07	7.8e-08	1.9e-07	2.7e-09	2.2e-09
Max Additive Error	4.5e-06	1.3e-06	6.9e-07	1.9e-06	2.1e-08	1.8e-08
Average Relative Error:	0.11	0.039	0.08	0.12	0.028	0.062
Max Relative Error	0.41	0.22	0.47	0.65	0.23	0.26



(a) FAST-PPR (b) FAST-PPR using target set instead of frontier

Figure 6: The importance of using the frontier. In each of these plots, a perfect algorithm would place all data points on the line $y = x$. Notice how using inverse-PPR estimates from the target set rather than the frontier results in significantly worse accuracy.

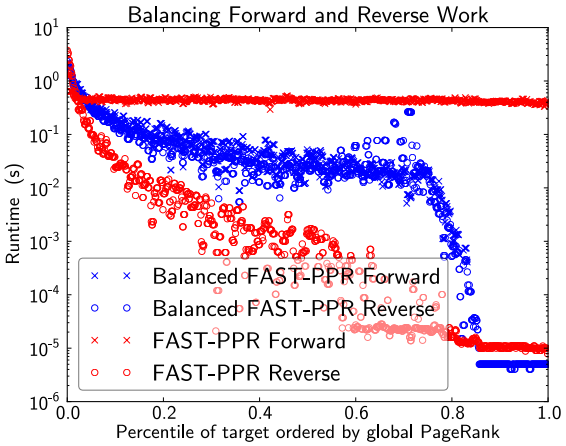


Figure 7: Forward and reverse running-times (on log-scale) for FAST-PPR (in red) and Balanced FAST-PPR (in blue) as we vary the global PageRank of the target node on the x-axis. Data is for the Twitter-2010 graph and is smoothed using median-of-five smoothing. Notice how there is a significant gap between the forward and backward work in Fast-PPR, and that this gap is corrected by Balanced Fast-PPR.

target set would reduce the dependence of the running-time on d, and also reduce storage requirements for an oracle-based implementation.

It turns out however that *using the frontier is critical* to get good accuracy. Intuitively, this is because nodes in the target set may have high inverse-PPR, which then increases the variance of our estimate. This increase in variance can be visually seen in a scatterplot of the true vs estimated value, as shown in Figure 6(b) – note that the estimates generated using the frontier set are much more tightly clustered around the true PPR values, as compared to the estimates generated using the target set.

• **Balancing Forward and Reverse Work:** Balanced FAST-PPR, as described in Section 5.2, chooses reverse threshold ϵ_r dynamically for each target node. Figure 4 shows that Balanced FAST-PPR improves the average running-time across all graphs.

In Figure 7, we plot the forward and reverse running-times for FAST-PPR and Balanced FAST-PPR as a function of the target PageRank. Note that for high global-PageRank targets, FAST-PPR does too much reverse work, while for low global-PageRank targets, FAST-PPR does too much forward work – this is corrected by Balanced FAST-PPR.

Acknowledgements

Peter Lofgren is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

Ashish Goel and Siddhartha Banerjee were supported in part by the DARPA XDATA program, by the DARPA GRAPHS program via grant FA9550- 12-1-0411 from the U.S. Air Force Office of Scientific Research (AFOSR) and the De-

fense Advanced Research Projects Agency (DARPA), and by NSF Award 0915040.

C. Seshadhri is at Sandia National Laboratories, which is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

7. REFERENCES

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web.," 1999.

[2] G. Jeh and J. Widom, "Scaling personalized web search," in *Proceedings of the 12th international conference on World Wide Web*, ACM, 2003.

[3] P. Yin, W.-C. Lee, and K. C. Lee, "On top-k social web search," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ACM, 2010.

[4] S. A. Yahia, M. Benedikt, L. V. Lakshmanan, and J. Stoyanovich, "Efficient network aware search in collaborative tagging sites," *Proceedings of the VLDB Endowment*, 2008.

[5] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto, "Efficient search ranking in social networks," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, ACM, 2007.

[6] D. Horowitz and S. D. Kamvar, "The anatomy of a large-scale social search engine," in *Proceedings of the 19th international conference on World wide web*, ACM, 2010.

[7] L. Backstrom and J. Leskovec, "Supervised random walks: predicting and recommending links in socialnetworks," in *Proceedings of the fourth ACM international conference on Web searchand data mining*, ACM, 2011.

[8] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh, "Wtf: The who to follow service at twitter," in *Proceedings of the 22nd international conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2013.

[9] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, 2006.

[10] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, p. 3, ACM, 2012.

[11] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," 2006.

[12] T. Sarlós, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz, "To randomize or not to randomize: space optimal summaries for hyperlink analysis," in *Proceedings of the 15th international conference on World Wide Web*, ACM, 2006.

[13] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova, "Monte carlo methods in pagerank computation: When one iteration is sufficient," *SIAM Journal on Numerical Analysis*, 2007.

[14] R. Andersen, C. Borgs, J. Chayes, J. Hopcraft, V. S. Mirrokni, and S.-H. Teng, "Local computation of pagerank contributions," in *Algorithms and Models for the Web-Graph*, Springer, 2007.

[15] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank," *Proceedings of the VLDB Endowment*, 2010.

[16] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng, "Multi-scale matrix sampling and sublinear-time pagerank computation," *Internet Mathematics*, 2013.

[17] A. D. Sarma, A. R. Molla, G. Pandurangan, and E. Upfal, "Fast distributed pagerank computation," *Distributed Computing and Networking*, 2013.

[18] P. Lofgren, S. Banerjee, A. Goel, and C. Seshadhri, "Fast-ppr: Scaling personalized pagerank estimation for large graphs," tech. rep., Stanford University, 2014. arXiv preprint arXiv:1404.3181.

[19] "Laboratory for web algorithmics." http://law.di.unimi.it/datasets.php. Accessed: 2014-02-11.

[20] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós, "Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments," *Internet Mathematics*, 2005.

[21] P. Lofgren and A. Goel, "Personalized pagerank to a target node," *arXiv preprint arXiv:1304.4658*, 2013.

[22] O. Goldreich and D. Ron, "On testing expansion in bounded-degree graphs," in *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, Springer, 2011.

[23] S. Kale, Y. Peres, and C. Seshadhri, "Noise tolerance of expanders and sublinear expander reconstruction," in *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, IEEE, 2008.

[24] D. Dubhashi and A. Panconesi, *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[25] O. Goldreich and D. Ron, "Property testing in bounded degreegraphs," *Algorithmica*, 2002. Conference version in STOC 1997.

[26] 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*, ACM Press, 2011.

[27] P. Boldi, M. Santini, and S. Vigna, "A large time-aware graph," *SIGIR Forum*, vol. 42, no. 2, 2008.

[28] L. Takac and M. Zabovsky, "Data analysis in public social networks," in *International. Scientific Conf. & Workshop Present Day Trends of Innovations*, 2012.

[29] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, (San Diego, CA), October 2007.

[30] "Stanford network analysis platform (snap)." http://http://snap.stanford.edu/. Accessed: 2014-02-11.

SILVAN: Estimating Betweenness Centralities with Progressive Sampling and Non-uniform Rademacher Bounds

LEONARDO PELLEGRINA, Dept. of Information Engineering, University of Padova, Italy
FABIO VANDIN, Dept. of Information Engineering, University of Padova, Italy

"Sim Sala Bim!" – Silvan
https://en.wikipedia.org/wiki/Silvan_(illusionist)

Betweenness centrality is a popular centrality measure with applications in several domains, and whose exact computation is impractical for modern-sized networks. We present SILVAN, a novel, efficient algorithm to compute, with high probability, accurate estimates of the betweenness centrality of all nodes of a graph and a high-quality approximation of the top-k betweenness centralities. SILVAN follows a progressive sampling approach, and builds on novel bounds based on Monte-Carlo Empirical Rademacher Averages, a powerful and flexible tool from statistical learning theory. SILVAN relies on a novel estimation scheme providing *non-uniform* bounds on the deviation of the estimates of the betweenness centrality of all the nodes from their true values, and a refined characterisation of the number of samples required to obtain a high-quality approximation. Our extensive experimental evaluation shows that SILVAN extracts high-quality approximations while outperforming, in terms of number of samples and accuracy, the state-of-the-art approximation algorithm with comparable quality guarantees.

CCS Concepts: • **Information systems → Data mining**; • **Mathematics of computing → Probabilistic algorithms**; • **Theory of computation → Sketching and sampling**.

Additional Key Words and Phrases: Betweenness Centrality, Rademacher Averages, Random Sampling

ACM Reference Format:
Leonardo Pellegrina and Fabio Vandin. 2022. SILVAN: Estimating Betweenness Centralities with Progressive Sampling and Non-uniform Rademacher Bounds. *ACM Trans. Knowl. Discov. Data.* 1, 1 (June 2022), 44 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The computation of node centrality measures, which are scores quantifying the importance of nodes, is a fundamental task in graph analytics [41]. *Betweenness centrality* is a popular centrality measure, defined first in sociology [1, 28], that quantifies the importance of a node as the fraction of shortest paths in the graph that go through the node.

The computation of the *exact* betweenness centrality for all nodes in a graph $G = (V, E)$ can be obtained with Brandes' algorithm [16] in time $O(|V||E|)$ for unweighted graphs and in time $O(|V||E| + |V|^2 \log |V|)$ for graphs with positive weights, which is impractical for modern networks with up to hundreds of millions of nodes and edges. Several works (e.g., [26, 57]) proposed heuristics to improve Brandes' algorithm, but they do not improve on its worst-case complexity. In fact, for unweighted graphs a corresponding lower bound (based on the Strong Exponential Time Hypothesis) was proved in [11]. The impracticality of the exact computation for modern networks,

Authors' addresses: Leonardo Pellegrina, Dept. of Information Engineering, University of Padova, Via Gradenigo 6b, Padova, Italy, 35131, pellegri@dei.unipd.it; Fabio Vandin, Dept. of Information Engineering, University of Padova, Via Gradenigo 6b, Padova, Italy, 35131, fabio.vandin@unipd.it.

and the use of betweenness centrality mostly in exploratory analyses of the data, have motivated the study of efficient algorithms to compute approximations of the betweenness centrality, trading precision for efficiency.

Several works [12, 21, 49, 52] have recently proposed sampling approaches to approximate the betweenness centrality of all nodes in a graph. The main idea is to sample shortest paths uniformly at random, and use such paths to estimate the betweenness centrality of the nodes. As for all sampling approaches, the main difficulty is then to relate the estimates obtained from the samples with the corresponding exact quantities, providing tight trade-offs between guarantees on the quality of the estimates and the required computational work. [12, 21, 49, 52] all provide rigorous approximations of the betweenness centrality, and [21, 49, 52] rely on tools from statistical learning theory, such as the *VC-dimension* [62], the *pseudodimension* [47], or *Rademacher Averages* [33], which have been successfully used to obtain rigorous approximations for other data mining tasks (e.g., pattern mining [50, 51, 53]). For pattern mining, recent work [45] has shown that a more advanced tool from statistical learning theory, namely *Monte-Carlo Empirical Rademacher Averages (MCERA)* [3] (see Sect. 3.2), leads to improved results, mostly thanks to it *data-dependent* nature (in contrast to *distribution-free* tools such as the VC-dimension and the pseudodimension). Indeed, the MCERA was recently used in BAVARIAN [21] to obtain an unifying framework compatible with different estimators of the betweenness centrality.

Our contributions. In this work we study the problem of approximating the betweenness centralities of nodes in a graph. We propose SILVAN (*eStimatIng betweenness centraLities with progressiVe sAmpling and Non-uniform rademacher bounds*), a novel, efficient, progressive sampling algorithm to approximate betweenness centralities while providing rigorous guarantees on the quality of various approximations.

- Our first contribution is *empirical peeling*, a novel technique that we introduce to obtain sharp *non-uniform data-dependent bounds* on the maximum deviation of families of functions (Section 4.1). Empirical peeling is based on the MCERA and relies on an effective *data-dependent* approach to *partition* a family of functions according to their empirically estimated variance; this allows to fully exploit *variance-dependent* bounds at the core of the technique. Our algorithm SILVAN (Section 4.2) relies on such novel bounds to provide guarantees on the approximation of the betweenness centrality that are much sharper than the ones obtained by previous works; these new contributions make SILVAN a practical algorithm for obtaining different approximations of the betweenness centrality. In fact, we show that combining the MCERA with empirical peeling allows us to design flexible algorithms with different guarantees (e.g., additive or relative) and for different tasks (e.g., estimating all betweenness centralities or, in Section 4.4, the top-k ones). This is the first work that obtains different types of approximation guarantees based on the MCERA. Most importantly, our approach is general and of independent interest, as it may apply to other problems, even outside of data mining applications.
- We derive a new bound on the sufficient number of samples to approximate the betweenness centrality for all nodes (Section 4.3), that naturally combines with the progressive sampling strategy of SILVAN by introducing an upper limit to the number of samples required to converge. Our new bound is governed by key quantities of the underlying graph, not considered by previous works, such as the *average shortest path length*, and the *maximum variance* of betweenness centrality estimators, significantly improving the state-of-the-art bounds for the task. Our proof combines techniques from combinatorial optimization and key results from theory of concentration inequalities. While previous results were tailored to analyse a specific estimator of the betweenness centrality, our result is general, since it applies to all available estimators of the betweenness

centrality. Furthermore, we extend this result to obtain sharper *relative* deviation bounds from a random sample.
- We perform an extensive experimental evaluation (Section 5), showing that SILVAN improves the state-of-the-art by requiring a fraction of the sample sizes and running times to achieve a given approximation quality or, equivalently, sharper guarantees for the same amount of work. Our experimental evaluations shows that SILVAN's guarantees, provided by our theoretical analysis, hold with a true approximation error close to its probabilistic upper bound, confirming the sharpness of our analysis. For the extraction of the top-k betweenness centralities, our algorithm provides faster approximations, using less samples, and with fewer false positives.

2 RELATED WORK

We now review the works on approximating the betweenness centralities that are most relevant to our contributions. In particular, we focus on approaches that provide guarantees on the quality of the approximation, an often necessary requirement.

The first practical sampling algorithm to approximate the betweenness centrality of all nodes with guarantees on the quality of the approximation is presented in [49]. Studying the VC-dimension of shortest paths, Riondato and Kornaropoulos [49] proved that when $O(\log(D/\delta)/\varepsilon^2)$ shortest paths are sampled uniformly at random, the approximations are within an additive error ε of the exact centralities with probability $\geq 1-\delta$, where D is (an upper bound to) the vertex diameter of the graph. While interesting, this result is characterized by the *worst-case* and *distribution-free* nature of the VC-dimension, and thus provides an overly conservative bound and cannot be used to design a *progressive* sampling approach.

The first rigorous progressive sampling algorithm is ABRA [52], which builds on the theory of Rademacher averages and pseudodimension and does not require an estimate of the vertex diameter. However, ABRA leverages a *deterministic* and *worst-case* upper bound to the Rademacher complexity (based on Massart's Lemma [38, 40, 59]); in a different scenario, Pellegrina et al. [45] show it provides conservative results in most cases compared to its Monte Carlo approximation given by the MCERA. In addition, similarly to [49], ABRA obtains *uniform and variance-agnostic bounds* that hold for all nodes in the graph G. The most recent approach is BAVARIAN [21], which addresses some of the limitations of ABRA using the MCERA and variance-aware tail bounds. Leveraging the flexibility and generality of the MCERA, BAVARIAN is compatible with different estimators of the betweenness centrality, but still obtains *uniform* approximation bounds not sensible to the heterogeneity of the centrality of different nodes of the graph. In contrast, our algorithm SILVAN uses *efficiently computable, non-uniform, and variance-dependent bounds* for different subsets of the nodes, which lead to a significant reduction of the number of samples and running times required to obtain rigorous guarantees w.r.t. all methods mentioned above.

A different approach has been proposed by KADABRA [12], a progressive sampling algorithm based on adaptive sampling. KADABRA is not based on tools from statistical learning theory, and our experimental evaluation shows that KADABRA is the state-of-the-art solution for the task. KADABRA is based on a weighted union bound, using a data-dependent scheme to assign different probabilistic confidences on the estimates of the betweenness centrality of each individual node, achieving improved approximations compared to the algorithm of [49] and to ABRA [52]. Our algorithm SILVAN uses a similar intuition and data-dependent approach, but with crucial differences. In particular, KADABRA assigns confidence parameters δ_v for each $v \in V$, such that the probability that the approximation of the betweenness centrality for node v not being accurate is at most δ_v *for each node* $v \in V$, with $\sum_{v \in V} \delta_v \leq \delta$. In contrast, our approach uses Rademacher averages and empirical peeling to obtain variance-dependent approximations that are valid for *sets of nodes*, exploiting correlations among nodes instead of considering each node individually. As we show in

our experimental evaluation, this leads to significant improvements on the approximation quality compared to KADABRA.

Most of existing methods [12, 49, 52] propose variants of their algorithms for the approximation of the top-k betweenness centralities. Our algorithm SILVAN achieves better results for this task as well, thanks to the non-uniform bounding scheme from the use of empirical peeling.

Other papers study different, but related, problems and notions of centralities. de Lima et al. [24] use an approach similar to [49], and based on pseudodimension, to estimate percolation centrality. Chechik et al. [18] propose an algorithm based on probability proportional to size sampling to estimate closeness centralities, which is the inverse of the average distance of a node to all other nodes in a graph, a popular importance measure in the study of social networks. Boldi and Vigna [10] consider closeness and harmonic centralities though HyperLogLog counters [9, 27]. Bergamini et al. [4] propose a new algorithm for selecting the k nodes with the highest closeness centralities in a graph. Mahmoody et al. [36] study the problem of finding a set of at most k nodes of maximum betweenness centrality, where the betweenness centrality of a set is defined analogously to the betweenness centrality of nodes [32, 64], and present efficient randomized algorithms to obtain rigorous approximations with high probability. Bergamini et al. [5] study the problem of increasing the betweenness centrality of a node by adding new links. Other recent works considered extending the computation of the betweenness centrality to dynamic graphs [6, 7, 31], uncertain graphs [54], and temporal networks [55]. SILVAN uses estimates of the vertex diameter of graphs, for which several approximation approaches have been proposed (e.g., [22, 23, 35]), including some for distributed frameworks [17].

3 PRELIMINARIES

In this section we introduce the basic notions used in the remaining of the paper.

3.1 Graphs and Betweenness Centrality

Let $G = (V, E)$ be a graph. For ease of exposition, we focus on unweighted graphs, however our algorithms can be easily adapted to weighted graphs. For any pair (s, t) of different nodes ($s \neq t$), let σ_{st} be the number of shortest paths between s and t, and let $\sigma_{st}(v)$ be the number of shortest paths between s and t that *pass through* (i.e., contain) v, with $s \neq v \neq t$. The (normalized) *betweenness centrality* $b(v)$ of a node $v \in V$ is defined as

$$b(v) = \frac{1}{|V|(|V|-1)} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$

Some of our algorithms rely on the knowledge of the *vertex diameter* D of a graph G, defined as the maximum number of nodes in any shortest path of G. (If G is unweighted, the vertex diameter of G is equal to the diameter of G plus 1.)

3.2 Rademacher Averages

Rademacher averages are a core concept in statistical learning theory [33] and in the study of empirical processes [14]. We now present the main notions and results used in our work and defer additional details to [14, 40, 59]. Let $\mathcal{X}$ be a finite domain and consider a probability distribution γ over the elements of $\mathcal{X}$. Let $\mathcal{F}$ be a family of functions from $\mathcal{X}$ to $[0, 1]$, and let $\mathcal{S} = \{\tau_1, \ldots, \tau_m\}$ be a collection of m independent and identically distributed samples from $\mathcal{X}$ taken according to γ. For each function $f \in \mathcal{F}$, define its average value over the sample $\mathcal{S}$ as $\mu_{\mathcal{S}}(f) = \frac{1}{m}\sum_{i=1}^{m} f(s_i)$ and its expectation, taken w.r.t. $\mathcal{S}$, as $\mu_{\gamma}(f) = \mathbb{E}_{\mathcal{S}}[\mu_{\mathcal{S}}(f)]$. Note that, by definition, $\mu_{\mathcal{S}}(f)$ is an *unbiased* estimator of $\mu_{\gamma}(f)$. Given $\mathcal{S}$, we are interested in bounding the *supremum deviation* $\mathrm{D}(\mathcal{F}, \mathcal{S})$ of

$\mu_{\mathcal{S}}(f)$ from $\mu_{\gamma}(f)$ among all $f \in \mathcal{F}$, that is

$$\mathsf{D}(\mathcal{F}, \mathcal{S}) = \sup_{f \in \mathcal{F}} |\mu_{\mathcal{S}}(f) - \mu_{\gamma}(f)|.$$

The *Empirical Rademacher Average* (ERA) $\hat{\mathsf{R}}(\mathcal{F}, \mathcal{S})$ of $\mathcal{F}$ on $\mathcal{S}$ is a key quantity to obtain a data-dependent upper bound to the supremum deviation $\mathsf{D}(\mathcal{F}, \mathcal{S})$. Let $\boldsymbol{\sigma} = \langle \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_m \rangle$ be a collection of m i.i.d. Rademacher random variables (r.v.'s), each taking value in $\{-1, 1\}$ with equal probability. The ERA $\hat{\mathsf{R}}(\mathcal{F}, \mathcal{S})$ of $\mathcal{F}$ on $\mathcal{S}$ is

$$\hat{\mathsf{R}}(\mathcal{F}, \mathcal{S}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\sigma}_i f(s_i) \right].$$

Computing the ERA $\hat{\mathsf{R}}(\mathcal{F}, \mathcal{S})$ is usually intractable, since there are 2^m possible assignments for $\boldsymbol{\sigma}$ and for each such assignment a supremum over the functions in $\mathcal{F}$ must be computed. A useful approach to obtain sharp probabilistic bounds on the ERA is given by Monte-Carlo estimation [3]. For $c \geq 1$, let $\boldsymbol{\sigma} \in \{-1, 1\}^{c \times m}$ be a $c \times m$ matrix of i.i.d. Rademacher r.v.'s. The *c-samples Monte-Carlo Empirical Rademacher average (c-MCERA)* $\hat{\mathsf{R}}^c_m(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma})$ of $\mathcal{F}$ on $\mathcal{S}$ using $\boldsymbol{\sigma}$ is:

$$\hat{\mathsf{R}}^c_m(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma}) = \frac{1}{c} \sum_{j=1}^{c} \sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{\sigma}_{j,i} f(s_i).$$

The c-MCERA allows to obtain sharp *data-dependent* probabilistic upper bounds to the supremum deviation, as they directly estimate the expected supremum deviation of sets of functions by taking into account their correlation. For this reason, they are often significantly more accurate than other methods [45], such as the ones based on often loose *deterministic upper bounds* to Rademacher averages (e.g., Massart's Lemma [38]), or other *distribution-free* notions of complexity, such as the VC-dimension. In general, the c-MCERA may be hard to compute, due to the supremums over $\mathcal{F}$ [3]. However, for the case of betweenness centralities, we show in Section 4.2 that all quantities relevant to the c-MCERA can be efficiently and incrementally updated as shortest paths are randomly sampled.

4 SILVAN: EFFICIENT PROGRESSIVE ESTIMATION OF BETWEENNESS CENTRALITIES

In this section we introduce SILVAN (*eStimatIng betweenness centraLities with progressiVe sAmpling and Non-uniform rademacher bounds*) and the techniques at its core.

We start, in Section 4.1, by presenting the empirical peeling technique and the related main technical results, which provide sharp data-dependent non-uniform approximation bounds supporting our algorithms. We then describe, in Section 4.2, our algorithm SILVAN that builds on such improved bounds to obtain an approximation within *additive* error ε of the betweenness centrality for all nodes via progressive sampling. We then present, in Section 4.3, improved bounds on the number of sufficient samples to achieve absolute approximations with high probability. These bounds are naturally combined with the progressive sampling scheme of SILVAN. Finally, in Section 4.4 we introduce SILVAN-TopK, an extension of SILVAN to obtain a relative approximation of the k nodes with highest betweenness centrality.

4.1 Non-uniform Bounds via Empirical Peeling

In this section we introduce *empirical peeling*, a new data-dependent scheme based on the c-MCERA to obtain sharp non-uniform bounds to the supremum deviation. The main idea behind empirical peeling is to *partition* the set of functions $\mathcal{F}$ in order to obtain the best possible bounds for different

subsets of $\mathcal{F}$. Classical concentration inequalities, such as Bernstein's and Bennet's [14], are well suited to control the deviation $\mathsf{D}(\{f\}, \mathcal{S})$ of a single function f, and to derive an approximation whose accuracy depends on its variance $Var(f)$. Instead, when *simultaneously* bounding the deviation of multiple functions belonging to a set of functions $\mathcal{F}$, the accuracy of the probabilistic bound on the supremum deviation $\mathsf{D}(\mathcal{F}, \mathcal{S})$ has a strong but natural dependence on the *maximum* variance $\sup_{f\in\mathcal{F}} Var(f)$. However, when the variances of the members of $\mathcal{F}$ are highly heterogenous, this leads to a significant loss of accuracy in the approximation of functions with variance much smaller than the maximum (i.e., we obtain a "blurred" approximation of functions f' with $Var(f') \ll \sup_{f\in\mathcal{F}} Var(f)$). We propose an intuitive solution to achieve a higher granularity in the approximation: we partition $\mathcal{F}$ into $t \geq 1$ subsets $\{\mathcal{F}_j, j \in [1, t]\}$ with $\bigcup_j \mathcal{F}_j = \mathcal{F}$, such that functions with similar variance belong to the same subset $\mathcal{F}_j$; this allows to control the supremum deviations $\mathsf{D}(\mathcal{F}_j, \mathcal{S})$ for each $\mathcal{F}_j$ separately, exploiting the fact that the maximum variance is now computed on each subset $\mathcal{F}_j$ instead on the entire set $\mathcal{F}$. This idea leads to sharp *non-uniform bounds* that are locally valid for each subset $\mathcal{F}_j$ of $\mathcal{F}$, and it is the main motivation and intuition behind empirical peeling.

The idea of computing a stratification of the set of functions under consideration is at the core of peeling, an important technique in the study of fine properties of empirical processes, extensively studied in statistical learning theory [2, 14, 29, 61]. However, the issue with existing peeling techniques is that the partition $\{\mathcal{F}_j\}$ either relies on strong assumptions about $\mathcal{F}$, or depends on information computed from the sample $\mathcal{S}$; this latter approach incurs in non-trivial issues due to the dependency between the bounds to $\mathsf{D}(\mathcal{F}_j, \mathcal{S})$ and $\{\mathcal{F}_j\}$, since both are estimated on the same data $\mathcal{S}$. For this reason, available methods are often loose (e.g., with bounds featuring very large constants) and thus received scant application in practical scenarios. Instead, the main idea behind *empirical peeling* is to use an *independent* sample $\mathcal{S}'$ to partition the set $\mathcal{F}$. This simple but effective idea significantly simplifies the analysis as it resolves the above mentioned dependency issues, with minimal additional work (as $\mathcal{S}'$ can be taken to be much smaller than $\mathcal{S}$).

Before discussing efficient and effective procedures to partition $\mathcal{F}$, we present a result providing the probabilistic guarantees at the core of empirical peeling, in which we assume the partitioning is fixed *before* drawing the sample $\mathcal{S}$. This improved bound (Theorem 4.2 below) on the supremum deviations holds for families $\mathcal{F}$ of functions f with value in $[0, 1]$, building on Monte Carlo Rademacher Averages introduced in Section 3.2. We use this result in SILVAN to obtain sharp data-dependent non-uniform bounds on the approximation of betweenness centrality.

Before stating Theorem 4.2, we remark that it is based on a novel tight probabilistic bound for the concentration of the c-MCERA for general families of functions (Theorem 4.1) that scales with the *empirical wimpy variance* of $\mathcal{F}$ (defined below). Our novel bound proves that the c-MCERA is a *sub-gaussian* random variable [14], therefore it satisfies concentration bounds that are *uniformly sharper* than state-of-the-art sub-gamma results (i.e., Theorem 11 of [43] and Theorem 5 of [20]).

The *empirical wimpy variance* $w_{\mathcal{F}}(\mathcal{S})$ of $\mathcal{F}$ on $\mathcal{S}$ is defined [14] as

$$w_{\mathcal{F}}(\mathcal{S}) = \sup_{f\in\mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} (f(s_i))^2 .$$

Theorem 4.1. *For $c, m \geq 1$, let $\boldsymbol{\sigma} \in \{-1, 1\}^{c\times m}$ be an $c \times m$ matrix of Rademacher random variables, such that $\boldsymbol{\sigma}_{j,i} \in \{-1, 1\}$ independently and with equal probability. Then, with probability $\geq 1 - \delta$ over $\boldsymbol{\sigma}$, it holds*

$$\hat{\mathsf{R}}(\mathcal{F}, \mathcal{S}) \leq \hat{\mathsf{R}}^c_m(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma}) + \sqrt{\frac{4 w_{\mathcal{F}}(\mathcal{S}) \ln\left(\frac{1}{\delta}\right)}{cm}}.$$

The proof of Theorem 4.1, deferred to the Appendix, leverages a concentration inequality for functions uniformly distributed on the binary hypercube (see Section 5.2 of [14]).

We are now ready to state the main technical result of this Section. (The proof is in Section A.1.)

Theorem 4.2. *Let $\mathcal{F} = \bigcup_{j=1}^{t} \mathcal{F}_j$ be a family of functions with codomain in $[0,1]$. Let $\mathcal{S}$ be a sample of size m taken i.i.d. from a distribution γ. Denote $v_{\mathcal{F}_j}$ such that $\sup_{f\in\mathcal{F}_j} Var(f) \leq v_{\mathcal{F}_j}$. For any $\delta \in (0,1)$, define*

$$\tilde{\mathsf{R}}_j \doteq \hat{\mathsf{R}}^c_m(\mathcal{F}_j, \mathcal{S}, \boldsymbol{\sigma}) + \sqrt{\frac{4w_{\mathcal{F}_j}(\mathcal{S}) \ln\left(\frac{4t}{\delta}\right)}{cm}},$$

$$\mathsf{R}_j \doteq \tilde{\mathsf{R}}_j + \frac{\ln\left(\frac{4t}{\delta}\right)}{m} + \sqrt{\left(\frac{\ln\left(\frac{4t}{\delta}\right)}{m}\right)^2 + \frac{2\ln\left(\frac{4t}{\delta}\right)\tilde{\mathsf{R}}_j}{m}},$$

$$\varepsilon_{\mathcal{F}_j} \doteq 2\mathsf{R}_j + \sqrt{\frac{2\ln\left(\frac{4t}{\delta}\right)\left(v_{\mathcal{F}_j} + 4\mathsf{R}_j\right)}{m}} + \frac{\ln\left(\frac{4t}{\delta}\right)}{3m}. \tag{1}$$

With probability at least $1-\delta$ over the choice of $\mathcal{S}$ and $\boldsymbol{\sigma}$, it holds $\mathsf{D}(\mathcal{F}_j, \mathcal{S}) \leq \varepsilon_{\mathcal{F}_j}$ for all $j \in [1,t]$.

From Theorem 4.2 we observe that, since each $v_{\mathcal{F}_j}$ strongly affects $\varepsilon_{\mathcal{F}_j}$, as it typically dominates (1), partitioning $\mathcal{F}$ according to different stratifications of $v_{\mathcal{F}_j}$ is very beneficial to obtain sharp *non-uniform* bounds. We remark that recent works based on Monte Carlo Rademacher Averages [21, 45] used bounds that apply to the particular case $t = 1$ (without any partitioning of $\mathcal{F}$) to obtain a *uniform* variance-dependent bound that can be very loose for most functions as it ignores any heterogeneity of variances within $\mathcal{F}$ (see Thereom 3.2 of [45] and Theorem 3.1 of [21]).

We note that in many cases, appropriate values for variance upper bounds $v_{\mathcal{F}_j}$ are not known. The following result upper bounds every supremum variance $\sup_{f\in\mathcal{F}_j} Var(f)$ of all sets of functions $\{\mathcal{F}_j\}$ using the empirical wimpy variances $w_{\mathcal{F}_j}(\mathcal{S})$. This bound conveniently defines sharp data-dependent values of $v_{\mathcal{F}_j}$ that we plug in (1). Our proof is based on the self-bounding properties of the function $w_{\mathcal{F}_j}(\mathcal{S})$, proved by [44].

Proposition 4.3. *With probability at least $1-\delta$ it holds, for all $j \in [1,t]$,*

$$\sup_{f\in\mathcal{F}_j} Var(f) \leq v_{\mathcal{F}_j} \doteq w_{\mathcal{F}_j}(\mathcal{S}) + \frac{\ln\left(\frac{t}{\delta}\right)}{m} + \sqrt{\left(\frac{\ln\left(\frac{t}{\delta}\right)}{m}\right)^2 + \frac{2w_{\mathcal{F}_j}(\mathcal{S})\ln\left(\frac{t}{\delta}\right)}{m}}. \tag{2}$$

Theorem 4.2 and Proposition 4.3 are easily combined by replacing $4/\delta$ by $5/\delta$ in Theorem 4.2, and $1/\delta$ by $5/\delta$ in (2); with this adjustment we obtain that both statements hold simultaneously with probability at least $1-\delta$.

4.2 SILVAN

In this Section we introduce SILVAN, our algorithm, based on the contributions of Section 4.1, to compute rigorous approximations of the betweenness centrality of all nodes in a graph.

We first describe, in Section 4.2.1, the algorithm to efficiently sample shortest paths, that is at the core of SILVAN to approximate the betweenness centrality. We then present SILVAN in Section 4.2.2.

4.2.1 Sampling Shortest Paths. SILVAN works by sampling shortest paths in G uniformly at random and using the fraction of shortest paths containing v as an unbiased estimator of its betweenness centrality $b(v)$. The first estimator following this idea was introduced by Riondato and

Kornaropoulos [49] (the rk estimator). The idea is to first samples two uniformly random nodes s, t, and then a uniformly distributed shortest path π between s and t. With this procedure the probability $\Pr[v \in \pi]$ that a node v is internal to π is $\Pr[v \in \pi] = b(v)$. A more refined approach was proposed by [50] (the ab estimator), which considers *all* shortest paths between s and t instead of only one, approximating the betweenness centrality $b(v)$ as the *fraction* of such shortest paths passing through v. The ab estimator has been shown to provide higher quality approximations than the rk in practice [21]; this is because, intuitively, it updates estimations among all nodes involved in shortest paths between s and t, and thus, informally, provides "more information per sample". Computationally, the set Π_{st} of shortest paths between s and t, required by both the rk and ab estimators, can be obtained in time $O(|E|)$ using a (truncated) BFS, initialized from s and expanded until t is found. For the rk estimator, a faster approach based on a *balanced bidirection BFS* was proposed and analysed by Borassi and Natale [12]: they show that all information required to sample one shortest path between two vertices s and t can be obtained in time $O(|E|^{\frac{1}{2}})$ with high probability on several random graph models, and experimentally on real-world instances. While this approach drastically speeds-up betweenness centrality approximations via the rk estimator [12], an analogous extension of this technique to the ab estimator is currently lacking.

Our sampling algorithm extends the balanced bidirection BFS to the ab estimator; this allows to combine superior statistical properties of ab with the much faster balanced bidirection BFS enjoyed by rk. Our main idea is that, once the set of all shortest paths Π_{st} between s and t is *implicitly* computed by the two BFSs, then it is very efficient to sample multiple shortest paths uniformly at random from Π_{st} (while [12] only sampled one shortest path).

SILVAN samples shortest paths with the following procedure:

(1) sample two uniformly random nodes s, t;
(2) performs a balanced bidirection BFS starting from s and t, until the two BFSs "meet";
(3) sample uniformly at random $\lceil \alpha\sigma_{st} \rceil$ shortest paths from the set Π_{st} of shortest paths between s and t, where $\sigma_{st} = |\Pi_{st}|$ is the number of shortest paths between s and t and $\alpha \geq 1$ a positive constant.

It is easy to see that the expected fraction of shortest paths sampled using this procedure containing v is equal to the betweenness centrality $b(v)$ of v. In particular, for each node $v \in V$ and a bag of shortest paths τ obtained from this sampling procedure, define the function $f_v(\tau)$, with $f_v(\tau) = |\tau|^{-1} \sum_{\pi \in \tau} \mathbb{1}[v \in \pi]$ where $\mathbb{1}[v \in \pi] = 1$ if v is internal to the shortest path $\pi \in \tau$, 0 otherwise. Consequently, the set of functions we use for betweenness centrality approximation contains all f_v with $v \in V$, so that $\mathcal{F} = \{f_v, v \in V\}$. By considering a sample $\mathcal{S}$ of size m taken as described above, we define the estimate $\tilde{b}(v)$ of the betweenness centrality $b(v)$ of v as

$$\tilde{b}(v) = \mu_{\mathcal{S}}(f_v) = \frac{1}{m} \sum_{\tau \in \mathcal{S}} f_v(\tau).$$

We have that $\tilde{b}(v)$ is an unbiased estimator of $\mu_\gamma(f_v) = b(v)$, so that $\mathbb{E}_{\mathcal{S}}[\tilde{b}(v)] = b(v)$:

$$\mathbb{E}_{\mathcal{S}}[\tilde{b}(v)] = \mathbb{E}_\tau[f_v(\tau)] = \mathbb{E}\left[\frac{1}{|\tau|} \sum_{\pi \in \tau} \mathbb{1}[v \in \pi]\right] = \mathbb{E}[\mathbb{1}[v \in \pi]] = \Pr(v \in \pi) = b(v).$$

Regarding α, from standard Poisson approximation to the balls and bins model [40], we obtain that the expected fraction of shortest paths that are not sampled from the set Π_{st} in step (3) is $\sigma_{st}(1 - 1/\sigma_{st})^{\alpha\sigma_{st}} \approx e^{-\alpha}$. Consequently, to ensure that the set of sampled shortest paths well represents Π_{st}, we set α to $\ln \frac{1}{\lambda}$ where λ is a small value (e.g., in practice we use $\lambda = 0.1$).

4.2.2 SILVAN algorithm. SILVAN is based on a *progressive sampling* approach. At a high level, the algorithm works in iterations, and in iteration i SILVAN extracts an approximation of the values $b(v)$ for all $v \in V$ from a sample $\mathcal{S}_i$, which is a collection of $m_i = |\mathcal{S}_i|$ randomly sampled bags of shortest paths. Then SILVAN checks whether a suitable *stopping condition* is satisfied. If the *stopping condition* is satisfied, the algorithm reports the achieved approximation. It is important that the stopping condition is satisfied as soon as possible, as each sample is expensive to compute, in particular for large graphs.

In this section we show how to achieve an *ε-approximation* of the set $BC(G) = \{b(v) : v \in V\}$, defined as follows.

Definition 4.4. A set $\tilde{BC}(G) = \{\tilde{b}(v) : v \in V\}$ is an *ε-approximation* of $BC(G) = \{b(v) : v \in V\}$ if it holds, for all $v \in V$, that $|b(v) - \tilde{b}(v)| \leq \varepsilon$.

Algorithm 1 describes the algorithm SILVAN to compute an ε-approximation of $BC(G)$ by employing the techniques introduced in Section 4.1. SILVAN can be logically divided into two

Algorithm 1: SILVAN

Input: Graph $G = (V, E)$; $c, m' \geq 1$; $\varepsilon, \delta \in (0, 1)$.
Output: ε-approximation of BC(G) with probability $\geq 1 - \delta$

1 $\mathcal{S}' \leftarrow$ `sampleSPs`(m');
2 $\{\mathcal{F}_j, j \in [1, t]\} \leftarrow$ `empiricalPeeling`$(\mathcal{F}, \mathcal{S}')$;
3 $\hat{m} \leftarrow$ `sufficientSamples`$(\mathcal{F}, \mathcal{S}', \delta/2)$;
4 $\{m_i\} \leftarrow$ `samplingSchedule`$(\mathcal{F}, \mathcal{S}')$;
5 **forall** $j \in [1, t]$ **do** $\varepsilon_{\mathcal{F}_j} \leftarrow 1$;
6 $i \leftarrow 0$; $\mathcal{S}_0 \leftarrow \emptyset$; $\boldsymbol{\sigma} \leftarrow$ empty matrix;
7 **while** *not* `stoppingCond`$(\varepsilon, \{\varepsilon_{\mathcal{F}_j}\}, \hat{m}, m_i)$ **do**
8 $\quad i \leftarrow i + 1$; $d_m \leftarrow m_i - m_{i-1}$;
9 $\quad \mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup$ `sampleSPs`(d_m);
10 $\quad \boldsymbol{\sigma} \leftarrow$ add columns {`sampleRrvs`(d_m, c)} to $\boldsymbol{\sigma}$;
11 $\quad \tilde{b}, \tilde{r}, \{\nu_{\mathcal{F}_j}\} \leftarrow$ `updateEstimates`$(\mathcal{S}_i, \boldsymbol{\sigma}, \{\mathcal{F}_j\})$;
12 $\quad$ **forall** $j \in [1, t]$ **do**
13 $\quad\quad \hat{\mathrm{R}}^c_{m_i}(\mathcal{F}_j, \mathcal{S}_i, \boldsymbol{\sigma}) \leftarrow \frac{1}{c} \sum_{x=1}^{c} \max_{v \in V, f_v \in \mathcal{F}_j} \{\tilde{r}(v, x)\}$;
14 $\quad\quad \varepsilon_{\mathcal{F}_j} \leftarrow$ `epsBound`$(\hat{\mathrm{R}}^c_{m_i}(\mathcal{F}_j, \mathcal{S}_i, \boldsymbol{\sigma}), \nu_{\mathcal{F}_j}, \delta/2^{i+1})$;
15 **return** $\tilde{b}$;

phases: in the first phase (lines 1-4), SILVAN generates a sample $\mathcal{S}'$ that is used for empirical peeling (Section 4.1) to partition $\mathcal{F}$ into t subsets $\{\mathcal{F}_j, j \in [1, t]\}$. The second phase (lines 5-15) describes the main operations of the algorithm to approximate the betweenness centrality.

We start by describing the first phase. In line 1, the sample $\mathcal{S}'$ is generated using the procedure `sampleSPs`(m'), which samples uniformly at random m' bags of shortest paths (following the procedure described in Section 4.2.1), where $m' \geq 1$ is given in input. After obtaining $\mathcal{S}'$, SILVAN uses the procedure `empiricalPeeling` (line 2) to partition $\mathcal{F}$ into t subsets. `empiricalPeeling` splits $\mathcal{F}$ according to an estimate of the variance of its members (we describe a simple but very effective partitioning scheme in more details at the end of this section). Using $\mathcal{S}'$, SILVAN computes an upper bound $\hat{m}$ to the number of samples required to obtain an ε approximation with the procedure `sufficientSamples` (line 3), which is described in Section 4.3. This bound $\hat{m}$ guarantees

that any sample $\mathcal{S}$ with size $\geq \hat{m}$ provides an ε approximation with probability $\geq 1-\delta/2$. Then, the algorithm fixes a sampling schedule given by the values of m_i using the function `samplingSchedule` (line 4). We note that arbitrary schedules can be used, but we discuss in Section 5 a data-dependent scheme that leverages $\mathcal{S}'$.

We now describe the second phase of the algorithm. First, in line 5 it initializes all values of $\varepsilon_{\mathcal{F}_j}$ to 1. Then, in line 6, the other variables used by SILVAN are initialized: i is the index of the iteration, while m_i is the size of the sample $\mathcal{S}_i$ considered at the i-th iteration. The algorithm initializes $\boldsymbol{\sigma}$ as an empty matrix, needed at every iteration to compute the c-MCERA (Section 3.2). In line 7, the iterations of SILVAN begin, which terminate according to the stopping condition `stoppingCond` (defined below). In every iteration of the while loop, SILVAN performs the following operations: first, it increments i; at the i-th iteration, it generates $d_m = m_i - m_{i-1}$ new samples (line 9) using `sampleSPs`(d_m), adding them to the set $\mathcal{S}_{i-1}$ to obtain $\mathcal{S}_i$. Then, it extends $\boldsymbol{\sigma}$ (line 10) adding d_m columns (each composed by c rows), so that $\boldsymbol{\sigma} \in \{0,1\}^{c \times m_i}$ in order to consider a sample $\mathcal{S}_i$ of size m_i. Such columns are generated with the procedure `sampleRrvs`(c, d_m) that samples a $c \times d_m$ matrix in which each entry is a Rademacher r.v. (Section 3.2). SILVAN updates all estimates needed for the approximation (line 11) using the procedure `updateEstimates`. This procedure uses the sample $\mathcal{S}_i$, the matrix $\boldsymbol{\sigma}$, and the partition $\{\mathcal{F}_j\}$ to compute three quantities: $\tilde{b}$ is a vector of $|V|$ components containing the estimates $\tilde{b}(v)$ of $b(v)$ for all $v \in V$; $\tilde{r}$ is a matrix of $|V| \times c$ components, in which each entry $\tilde{r}(v,x)$ is defined as the estimated c-MCERA for the function f_v using the x-th row $\boldsymbol{\sigma}_{x,\cdot}$ of $\boldsymbol{\sigma}$, such that $\tilde{r}(v,x) = \hat{\mathrm{R}}^c_{m_i}(\{f_v\}, \mathcal{S}_i, \boldsymbol{\sigma}_{x,\cdot})$; these values are required to compute the c-MCERA of each set $\mathcal{F}_i$. Then, the set $\{\nu_{\mathcal{F}_j}\}$ contains probabilistic upper bounds to supremum variances $\sup_{f \in \mathcal{F}_j} Var(f)$, such that $\sup_{f \in \mathcal{F}_j} Var(f) \leq \nu_{\mathcal{F}_j}$; we take each $\nu_{\mathcal{F}_j}$ as in (2) (replacing $1/\delta$ by $2^{i+1}5/\delta$ for reasons discussed below). While we describe `updateEstimates` as a separate procedure executed after the creation of $\mathcal{S}_i$ for ease the presentation, in practice all of these quantities can be updated *incrementally* as every new sample is added to $\mathcal{S}_i$. More precisely, the algorithm increases $\tilde{b}(v)$ for all nodes $v \in \pi$ and for all π in each sample τ, and similarly $\tilde{r}(v,j)$ all $j \in [1,c]$; analogously, it updates $w_{\mathcal{F}_j}(\mathcal{S}_i)$ for all j as each new sample is obtained. All these operations are done in $O(c|\tau|D)$ time per sample, where D is the vertex diameter of the graph, since $|\pi| \leq D, \forall \pi \in \tau$. Furthermore, every sample generated within `sampleSPs` can be sampled and processed in parallel with minimal synchronization. After $\mathcal{S}_i$ is created and processed, the algorithm computes (line 13), for all partitions $\mathcal{F}_j$ of $\mathcal{F}$, the c-MCERA $\hat{\mathrm{R}}^c_{m_i}(\mathcal{F}_j, \mathcal{S}_i, \boldsymbol{\sigma})$ using the values stored in $\tilde{r}$. Then, it computes (line 14) a probabilistic upper bound $\varepsilon_{\mathcal{F}_j}$ to the supremum deviation $\mathrm{D}(\mathcal{F}_j, \mathcal{S})$ for each partition $\mathcal{F}_j$ using the function `epsBound`. This function returns $\varepsilon_{\mathcal{F}_j}$ from (1) replacing $4/\delta$ by $2^{i+1}5/\delta$. This value of δ takes into account the fact that we want simultaneous guarantees for all iterations of the algorithm and for all the probabilistic estimates of $\nu_{\mathcal{F}_j}$, as we formally prove with Proposition 4.5. SILVAN continues to iterate until the stopping condition `stoppingCond` is true: since we are interested in an ε-approximation, `stoppingCond` checks that $\varepsilon \geq \varepsilon_{\mathcal{F}_j}, \forall j \in [1,t]$, or that $m_i \geq \hat{m}$. When `stoppingCond` is true, SILVAN returns the approximation $\tilde{b}$ (line 15).

The following result establishes the guarantees provided by SILVAN. The proof (Section A.2) follows from contributions of Section 4.1 and the samples bound we formally describe in Section 4.3.

Proposition 4.5. *With probability $\geq 1-\delta$, the output $\tilde{b}$ of SILVAN is a ε-approximation of $BC(G)$.*

We now describe a simple but effective criteria to partition $\mathcal{F}$, implementing the `empiricalPeeling` method. First, we denote with $\tilde{w}_v$ the estimated wimpy variance of the function f_v on sample $\mathcal{S}'$ as

$$\tilde{w}_v = \frac{1}{|\mathcal{S}'|} \sum_{\tau \in \mathcal{S}'} \left(f_v(\tau)\right)^2 .$$

We assign each function f_v for each node $v \in V$ to the set $\mathcal{F}_j$ with index $j = \lceil \log_a(\min\{\tilde{w}_v^{-1}, |\mathcal{S}'|\}) \rceil$ for a constant $a > 1$. Intuitively, this allows to split $\mathcal{F}$ into (at most) $t = \lceil \log_a(|\mathcal{S}'|) \rceil$ partitions, such that each set $\mathcal{F}_j$ groups functions with variances in $[1/a^{j+1}, 1/a^j]$, therefore within a multiplicative factor a. Our main intuition is that the empirical wimpy variances $w_{\mathcal{F}_j}(\mathcal{S})$ control the accuracy of the bounds on the supremum deviations $\mathrm{D}(\mathcal{F}_j, \mathcal{S})$ (through $\hat{v}_j$ in Theorem 4.2 and Proposition 4.3); this partitioning scheme fully exploits the non-uniform variance-dependent bounds at the core of SILVAN since the empirical wimpy variances $w_{\mathcal{F}_j}(\mathcal{S})$ are approximated by $w_{\mathcal{F}_j}(\mathcal{S}')$ and are $w_{\mathcal{F}_j}(\mathcal{S}') \leq 1/a^j$, which decrease exponentially with j.

4.3 Upper Bound to the Number of Samples

In this section we prove new upper bounds to the sufficient number of samples to obtain accurate approximations of betweenness centrality. In Section 4.3.1 we prove a new bound on the number of samples required to obtian an absolute ε approximation. Our proof is based on a novel connection between key results from combinatorial optimization [37] and fundamental concentration inequalities [14]. Then, in Section 4.3.2 we show that the same technique can be applied to guarantee that, from a sample of a given size, the estimates of the betweenness centrality satisfy tight relative deviation bounds. In Section 4.3.3 we provide empirical bounds on the average shortest path length, a key parameter governing our novel bounds. For ease of presentation, we defer the proofs to the Appendix (Section A.3).

4.3.1 Absolute Approximation. The following result shows an improved bound to the number of samples to obtain an absolute ε approximation. We obtain a *distribution-dependent* bound, since it takes into account the *maximum* variance of the betweenness centrality estimators. In addition, our bound scales with the *average shortest path length* ρ, a key graph characteristic not considered by previous results. A first observation is that the average shortest path length is equal to the sum of betweenness centrality $b(v)$ over all $v \in V$. If we denote Π as the set of shortest paths of the graph G, and $|\pi|$ the number of internal nodes to the path $\pi \in \Pi$, then it holds

$$\rho = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} |\pi| = \sum_{v \in V} b(v).$$

The key intuition behind our new bounds (Theorems 4.6 and 4.7, formally stated below) is that the betweenness centrality measure satisfies a form of negative correlation among vertices: the existance of a node v with high betweenness centrality $b(v)$ constraints the sum of the betweenness centrality of all other nodes to be at most $\rho - b(v)$; intuitively, this means that the number of vertices of G with high betweenness centrality cannot be arbitrarily large. Moreover, we assume that the maximum variance $\max_{v \in V} Var_\gamma(f_v)$ of the betweenness centrality estimators is at most $\hat{v}$, rather than using the worst-case bound of $\max_{v \in V} Var_\gamma(f_v) \leq 1/4$. Consequently, the estimates $\tilde{b}(v)$ which can incur in large deviations w.r.t. to their expected values $b(v)$ may not only be (naïvely) bounded by the number $n = |V|$ of vertices of G, but are tightly constrained by the parameters ρ and $\hat{v}$. Building on this idea, we are able to characterize an upper bound to the probability of not obtaining an absolute ε approximation from a sample of size m, where this probability is taken over the space of graphs with average shortest path length at most ρ, and such that the maximum estimator variance is at most $\hat{v}$. The key technical tool we use to achieve this is to express this probability as an instance of a Bounded Knapsack Problem [37]; we explicitly optimize this combinatorial problem through different relaxations, leading to sharp upper bounds. We remark that taking advantage of these additional constraints on the space of possible graphs is in strong contrast with the best available results, based on worse-case analyses leading to more conservative guarantees.

Theorem 4.6. *Let $\mathcal{F} = \{f_v, v \in V\}$ be a set of functions from a domain $\mathcal{X}$ to $[0,1]$. Let a distribution γ such that $\mathbb{E}_{\tau\sim\gamma}[f_v(\tau)] = b(v)$. Define $\hat{v} \in (0, 1/4]$, $\rho \geq 0$ such that*

$$\max_{v\in V} Var_\gamma(f_v) \leq \hat{v}\ , \text{ and } \sum_{v\in V} b(v) \leq \rho.$$

Fix $\delta \in (0,1)$, $\varepsilon \in (0,1)$, and define the functions $g(x) = x(1-x)$ and $h(x) = (1+x)\ln(1+x) - x$ for $x \geq 0$. Let $\hat{x}_1$, $\hat{x}_2$, and $\hat{x}$ be

$$\hat{x}_1 = \inf\left\{x : \frac{1}{2} - \sqrt{\frac{\varepsilon}{3} - \frac{\varepsilon^2}{9}} \leq x \leq \frac{1}{2}, g(x)h\left(\frac{\varepsilon}{g(x)}\right) \leq 2\varepsilon^2\right\},\ \ \hat{x}_2 = \frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}},\ \ \hat{x} = \min\{\hat{x}_1, \hat{x}_2\}.$$

Let $\mathcal{S}$ be an i.i.d. sample of size $m \geq 1$ taken from $\mathcal{X}$ according to γ such that

$$m \geq \sup_{x\in(0,\hat{x}]} \left\{\frac{\ln\left(\frac{2\rho}{x\delta}\right)}{g(x)h\left(\frac{\varepsilon}{g(x)}\right)}\right\}. \tag{3}$$

With probability $\geq 1-\delta$ over $\mathcal{S}$, it holds $\mathsf{D}(\mathcal{F}, \mathcal{S}) \leq \varepsilon$.

To make the bound (3) more interpretable, we make the following observations. First, while the r.h.s. of (3) is implicit, and difficult to express in closed form, it can be easily computed with a numerical procedure (e.g., $\hat{x}_1$ can be easily obtained with a binary search in the interval $[1/2 - \sqrt{\varepsilon/3 - \varepsilon^2/9}, 1/2]$, exploiting the convexity of $g(x)h(\varepsilon/g(x))$ in $(0,1)$). Then, we remark that for typical values of the parameters (e.g., $\delta, \varepsilon \leq 0.25 \leq \rho$), the maximum of (3) is attained at $x^\star \approx \hat{x} \leq \hat{x}_2$; furthermore, we note that $h(x) \geq x^2/2(1 + x/3)$ for $x \geq 0$, and $\hat{x}_2 \geq \hat{v} = g(\hat{x}_2)$. Combining all these facts, a very accurate approximation of m in (3) is

$$m \approx \frac{2\hat{v} + \frac{2}{3}\varepsilon}{\varepsilon^2}\left(\ln\left(\frac{2\rho}{\hat{v}}\right) + \ln\left(\frac{1}{\delta}\right)\right) \in O\left(\frac{\hat{v}+\varepsilon}{\varepsilon^2}\ln\left(\frac{\rho}{\delta\hat{v}}\right)\right).$$

Since ρ corresponds to (an upper bound to) the *average* number of internals nodes in shortest paths of G, it is immediate to conclude that ρ cannot exceed the vertex diameter D (the maximum shortest path length). From these observations, it is natural to compare our new bound with the state-of-the-art result based on the VC-dimension presented by Riondato and Kornaropoulos [49]; they show that $O(\ln(D/\delta)/\varepsilon^2)$ samples are enough for obtaining an ε-approximation with probability at least $1-\delta$. Similarly to their result, our new bound is independent of the size of the graph (e.g., $|V|$ or $|E|$), and essentially recovers it since $\hat{v} \leq 1/4$ and $\rho \leq D$, but provides much tighter results when smaller bounds to $\hat{v}$ and ρ are available: it is *distribution-dependent*, rather than *distribution-free*. Interestingly, in many real-world graphs the average shortest path length is typically very small (a phenomena observed in small-world networks [63] for which $\rho \in O(\log|V|)$), and often much smaller than the diameter.

Since ρ is usually not known in advance, in Section 4.3.3 we prove that, given a (not necessarily tight) upper bound to D, ρ can be sharply estimated as the *average* number of internal nodes of the shortest paths in a sample $\mathcal{S}$, resulting in a very efficient data-dependent bound. In addition, $\hat{v}$ can be sharply upper bounded by Proposition 4.3 (defining $\hat{v} \doteq \nu_{\mathcal{F}_j}$ with $t = 1$).

As anticipated in Section 4.2, Theorem 4.6 is not only of theoretical interest, but is used in SILVAN to design its sampling schedule (e.g., by upper bounding the number of samples $\hat{m}$ it needs to process). SILVAN estimates both $\hat{v}$ and ρ using $\mathcal{S}'$, and then plug them in Theorem 4.6 to obtain $\hat{m}$; these operations are part of the procedure `sufficientSamples` (see Algorithm 1). These key results contributes to make SILVAN processing a significantly lower number of samples to obtain an approximation of desired quality in many practical cases, as we will show in our experimental

evaluation. Moreover, we note that Thereom 4.6 applies to all available (unbiased) estimators of the betweenness centrality, such as the ones employed by BAVARIAN [21], providing a tighter upper bound to the sufficient number of random samples required by different estimators as well.

4.3.2 Relative Bounds. In this section we extend Theorem 4.6 to obtain new sharp relative deviation bounds; these bounds are very useful to derive sharp confidence intervals on the values of betweenness centrality $b(v)$ from a random sample, and are particularly very accurate for smaller values of $b(v)$.

Theorem 4.7. *Let $\mathcal{F}$, γ, g, $\hat{v}$, and ρ as in Theorem 4.6, and define $n = |V|$. Denote $\mathcal{S}$ as an i.i.d. sample of size $m \geq 1$ from γ. It holds, with probability $\geq 1-\delta$ and for all $v \in V$,*

$$|b(v) - \tilde{b}(v)| \leq \sqrt{\frac{2\min\{g(b(v)), \hat{v}\} \ln\left(\frac{4}{\delta}\min\left\{\frac{\rho}{b(v)}, n\right\}\right)}{m}} + \frac{\ln\left(\frac{4}{\delta}\min\left\{\frac{\rho}{b(v)}, n\right\}\right)}{3m} \doteq d_r(b(v)).$$

Theorem 4.7 yields sharp upper and lower bounds to $b(v)$ (for all $v \in V$), which are easily obtained (e.g., with a binary search) from their empirical estimates $\tilde{b}(v)$, as evident from the following Corollary.

Corollary 4.8. *Assume the setting of Theorem 4.7. It holds with probability $\geq 1-\delta$*

$$\min\left\{x \in [0, \tilde{b}(v)] : \tilde{b}(v) \leq x + d_r(x)\right\} \leq b(v) \leq \max\left\{x \in [\tilde{b}(v), 1] : x \leq \tilde{b}(v) + d_r(x)\right\}, \quad \forall v \in V.$$

We remark that the above bound is significantly more accurate than bounds obtained using standard tools (i.e., combining Bernstein's inequality and a union bound over n events) for most interesting values of $b(v)$ (more precisely, when $b(v) \geq 2\rho/n$ since $\ln(4\rho/(b(v)\delta)) \ll \ln(2n/\delta)$).

4.3.3 Empirical Bounds to the Average Shortest Path Length. In this section we present sharp empirical bounds on the average shortest path length, a key quantity involved in the sample bounds introduced in Section 4.3. The first result (Proposition 4.9) is based on the application of Bernstein's inequality [14], while the second (Proposition 4.10) uses the Empirical Bernstein Bound introduced by Maurer and Pontil [39].

Proposition 4.9. *Let D be the vertex diameter of the graph G. Let an i.i.d. sample $\mathcal{S}$ of size m, and denote $\tilde{\rho} = \sum_{v \in V} \tilde{b}(v)$. Then, for a fixed $\delta \in (0,1)$, it holds with probability $\geq 1-\delta$*

$$\sum_{v \in V} b(v) \leq \rho \doteq \tilde{\rho} + \sqrt{\frac{5}{3}\left(\frac{D\ln\left(\frac{1}{\delta}\right)}{m}\right)^2 + \frac{2D\tilde{\rho}\ln\left(\frac{1}{\delta}\right)}{m}} + \frac{4D\ln\left(\frac{1}{\delta}\right)}{3m}.$$

The following gives typically slightly sharper bounds than Proposition 4.9 since it involves an empirical estimator $\Lambda(\mathcal{S})$ of the variance of $\sum_{v \in V} b(v)$.

Proposition 4.10. *Assume the setting of Proposition 4.9 with $\mathcal{S} = \{\tau_1, \ldots, \tau_m\}$. Define $\Lambda(\mathcal{S})$ as*

$$\Lambda(\mathcal{S}) = \frac{1}{m(m-1)} \sum_{1 \leq i < j \leq m} \left(\sum_{v \in V} f_v(\tau_i) - \sum_{v \in V} f_v(\tau_j)\right)^2.$$

Then, for a fixed $\delta \in (0,1)$, it holds with probability $\geq 1-\delta$

$$\rho \leq \tilde{\rho} + \sqrt{\frac{2\Lambda(\mathcal{S})\ln\left(\frac{2}{\delta}\right)}{m}} + \frac{7D\ln\left(\frac{2}{\delta}\right)}{3m}.$$

4.4 Top-k Approximation

In this Section we present SILVAN-TopK, an extension of SILVAN to compute high-quality *relative* approximations of the k most central vertices.

While in some cases additive approximations, for which we guarantee that $|\tilde{b}(v) - b(v)| \leq \varepsilon$ for all $v \in V$, are sufficient, in several practical cases *relative* approximations, for which the desired bound to $|\tilde{b}(v) - b(v)|$ depends on the value $b(v)$, may be more informative. Such approximations are particularly relevant for the problem of estimating the k *most central nodes*, as the value of their betweenness centrality is typically highly skewed. In such cases, one may prefer to have relative approximations of the type "$\tilde{b}(v)$ is within 10% of the value of $b(v)$" than an additive approximation of the type "$|\tilde{b}(v) - b(v)| \leq 0.01$" that may be either unnecessarily precise for high values of $b(v)$, or uninformative for low values of $b(v)$. Furthermore, the user needs to only fix k and the relative accuracy, a much more natural choice for exploratory analyses, in which the centrality scores of the top-k nodes are unknown. We note that, from a "statistical" point of view, obtaining relative approximations is a challenging problem; in statistical learning theory, it is well known that it is not possible to obtain them efficiently using uniform additive approximations as a proxy [13]; this motivates the development of specialized techniques for the task (e.g., [19, 30, 34]).

In this section we show that empirical peeling, introduced in Section 4.1, is naturally suited to the problem of computing relative approximations of the set of top-k central nodes, and can do so progressively and adaptively as samples are processed. First, let $v_1, \ldots, v_n$ be the nodes sorted according to their betweenness centrality, such that $b(v_i) \geq b(v_{i+1})$. The set $TOP(k)$ of top-k nodes is defined as $TOP(k) = \{(v_i, b(v_i)) : i \leq k\}$. We now define the relative approximation we are interested in obtaining.

Definition 4.11. For $\eta \in (0, 1)$ and $k \geq 1$, a set $\tilde{TOP}(k) = \{(v, \tilde{b}(v))\}$ with $v \in V, \tilde{b}(v) \in [0, 1]$, is a *$\eta$-relative approximation* of $TOP(k)$ if all the following hold:

$$\{v : (v, \tilde{b}(v)) \in \tilde{TOP}(k)\} \supseteq \{v : (v, b(v)) \in TOP(k)\}, \tag{4}$$

$$|b(v) - \tilde{b}(v)| \leq \eta b(v), \forall (v, \tilde{b}(v)) \in \tilde{TOP}(k), \tag{5}$$

$$b(v) \geq b(v_k)\left(\frac{1-\eta}{1+\eta}\right)^2, \forall v : (v, b(v)) \notin TOP(k), (v, \tilde{b}(v)) \in \tilde{TOP}(k). \tag{6}$$

Informally, (4) ensures that all nodes in $TOP(k)$ are in the approximation; (5) ensures that all the estimates in the approximation are close to the true values of the betweenness centrality, within relative accuracy given by η; (6) guarantees that nodes v not in the set $TOP(k)$ of the top-k nodes are in the approximation only if their betweenness centrality $b(v)$ is not too far from $b(v_k)$, the centrality of the k-th node.

We now discuss how to modify SILVAN to obtain an approximation $\tilde{TOP}(k)$ of $TOP(k)$ with the aforementioned guarantees. Assume, at the end of some iteration of SILVAN, that we have confidence intervals $CI_v = [\ell(v), u(v)]$ for each $v \in V$, such that $b(v) \in CI_v, \forall v$. Such confidence intervals are derived from bounds on supremum deviations: for a node v such that $f_v \in \mathcal{F}_j$ for some $j \in [1, t]$, and assuming that $\varepsilon_{\mathcal{F}_j} \geq \mathrm{D}(\mathcal{F}_j, \mathcal{S})$, we define

$$\ell(v) \doteq \tilde{b}(v) - \varepsilon_{\mathcal{F}_j} \text{ and}$$
$$u(v) \doteq \tilde{b}(v) + \varepsilon_{\mathcal{F}_j}.$$

Naturally, the validity of such confidence intervals is *probabilistic*, and thus we aim to obtain a η-relative approximation with high probability. In order to verify that a given set $\tilde{TOP}(k)$ is a

η-relative approximation of $TOP(k)$, we inspect the confidence intervals CI_v for each candidate v to be included in $\tilde{TOP}(k)$.

Proposition 4.12. *For each $v \in V$, denote the intervals $CI_v = [\ell(v), u(v)]$ with $\ell(v) \leq \tilde{b}(v) \leq u(v)$. Let $v_1^\ell, \ldots, v_n^\ell$ be the sequence of nodes ordered according to $\ell(\cdot)$, such that $\ell(v_i^\ell) \geq \ell(v_{i+1}^\ell)$. Define the set $\tilde{TOP}(k)$ as $\tilde{TOP}(k) = \{(v, \tilde{b}(v)) : u(v) \geq \ell(v_k^\ell)\}$, and assume that, for all $(v, \tilde{b}(v)) \in \tilde{TOP}(k)$,*

$$\frac{\tilde{b}(v)}{1+\eta} \leq \ell(v) \leq b(v) \leq u(v) \leq \frac{\tilde{b}(v)}{1-\eta}. \tag{7}$$

Then, $\tilde{TOP}(k)$ is a η-relative approximation of $TOP(k)$.

Building on this result, Algorithm 2 describes our algorithm SILVAN-TopK to compute η-relative approximations of $TOP(k)$.

Algorithm 2: SILVAN-TopK

Input: Graph $G = (V, E)$; $c, k \geq 1$; $\eta, \delta \in (0, 1)$.
Output: Relative η-approximation of top-k central nodes with probability $\geq 1 - \delta$

1 **while** *not* `stoppingCondFirst()` **do** $\mathcal{S}' \leftarrow \mathcal{S}' \cup$ `sampleSPs(1)` ;
2 $\{\mathcal{F}_j, j \in [1, t]\} \leftarrow$ `empiricalPeeling`$(\mathcal{F}, \mathcal{S}')$;
3 $\{m_i\} \leftarrow$ `samplingSchedule`$(\mathcal{F}, \mathcal{S}')$;
4 $i \leftarrow 0$; $\mathcal{S}_0 \leftarrow \emptyset$; $\boldsymbol{\sigma} \leftarrow$ empty matrix;
5 **while** *not* `stoppingCondTopk()` **do**
6 $\quad i \leftarrow i + 1$; $d_m \leftarrow m_i - m_{i-1}$;
7 $\quad \mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup$ `sampleSPs`(d_m);
8 $\quad \boldsymbol{\sigma} \leftarrow$ add columns $\{$`sampleRrvs`$(d_m, c)\}$ to $\boldsymbol{\sigma}$;
9 $\quad \tilde{b}, \tilde{r}, \{\nu_{\mathcal{F}_j}\} \leftarrow$ `updateEstimates`$(\mathcal{S}_i, \boldsymbol{\sigma}, \{\mathcal{F}_j\})$;
10 $\quad$ **forall** $j \in [1, t]$ **do**
11 $\quad\quad \hat{\mathrm{R}}^c_{m_i}(\mathcal{F}_j, \mathcal{S}_i, \boldsymbol{\sigma}) \leftarrow \frac{1}{c} \sum_{x=1}^{c} \max_{v \in V, f_v \in \mathcal{F}_j} \{\tilde{r}(v, x)\}$;
12 $\quad\quad \varepsilon_{\mathcal{F}_j} \leftarrow$ `epsBound`$(\hat{\mathrm{R}}^c_{m_i}(\mathcal{F}_j, \mathcal{S}_i, \boldsymbol{\sigma}), \nu_{\mathcal{F}_j}, \delta/2^i)$;
13 $\quad\quad$ **forall** $v : f_v \in \mathcal{F}_j$ **do** $[\ell(v), u(v)] \leftarrow [\tilde{b}(v) - \varepsilon_{\mathcal{F}_j}, \tilde{b}(v) + \varepsilon_{\mathcal{F}_j}]$;
14 $\quad \tilde{TOP}(k) \leftarrow \{(v, \tilde{b}(v)) : u(v) \geq \ell(v_k^\ell)\}$;
15 **return** $\tilde{TOP}(k)$;

As SILVAN, SILVAN-TopK is divided in two phases: in the first phase (lines 1-3) it samples $\mathcal{S}'$, uses it for empirical peeling (line 2), and defines the sampling schedule (line 3). Then, it obtains the η-relative approximations using progressive sampling in the second phase (lines 4-15).

The first phase of SILVAN-TopK, instead of considering a fixed number of samples m' for $\mathcal{S}'$ (as in Algorithm 1), continues to draw shortest paths taken at random until at least k distinct nodes have been observed at least a constant number of times (therefore after $\approx 1/b(v_k)$ samples); when this is verified, the function `stoppingCondFirst` returns true (line 1) and the generation of $\mathcal{S}'$ stops. Following this scheme, the first phase *adapts* to the (unknown) value of $b(v_k)$.

The second phase of SILVAN-TopK is similar to SILVAN. At iteration i, after obtaining bounds $\varepsilon_{\mathcal{F}_j}$ on supremum deviations $\mathrm{D}(\mathcal{F}_j, \mathcal{S}_i)$ from the sample $\mathcal{S}_i$, the algorithm defines the confidence intervals $[\ell(v), u(v)]$ w.r.t. $b(v)$ (line 13). Then, it creates the set $\tilde{TOP}(k)$ including all vertices with upper bound $u(v)$ at least $\ell(v_k^\ell)$, where $\ell(v_k^\ell)$ is the k-th lower bound (line 14), as defined

in Proposition 4.12. To obtain the approximation described by Definition 4.11, SILVAN-TopK outputs $\tilde{TOP}(k)$ when its stopping condition `stoppingCondTopk` verifies that (7) holds for all $(v, \tilde{b}(v)) \in \tilde{TOP}(k)$. Note that the algorithm does not need to know $b(v_k)$ (or $b(v)$ for any v), as the left and rightmost inequalities in (7) only depend on empirical quantities. From the probabilistic guarantees implied by Theorem 4.2 and from Proposition 4.12, the following result easily follows.

Proposition 4.13. *The output of SILVAN-TopK is a η-relative approximation of $TOP(k)$ with probability $\geq 1 - \delta$.*

We remark that this general approach can be adapted easily to other definitions of relative approximations (e.g., [19, 34]). As we will show in our experimental evaluation, empirical peeling is essential to achieve η-relative approximations efficiently.

5 EXPERIMENTS

We implemented SILVAN and tested it on several real-world graphs. In our experimental evaluations we assess the effectiveness of the progressive sampling approach of SILVAN to approximate the betweenness centrality of all nodes, and evaluate the performance of SILVAN-TopK in approximating the top-k most central nodes.

Experimental Setup. We implemented SILVAN by extending the `C++` implementation of `KADABRA` made available from its authors[1]. All the code was compiled with `GCC 8` and run on a machine with 2.30 GHz Intel Xeon CPU, 512 GB of RAM, on Ubuntu 20.04, with a total of 64 cores. All experiments were performed using multithreading on all threads. Our implementation of SILVAN, with automated scripts to reproduce all experiments, is available online[2]. We compare SILVAN with `KADABRA`, that has been shown [12] to uniformly and significantly outperform previous methods, and with `BAVARIAN` [21], the most recent method for betweenness centrality approximation. When referring to `BAVARIAN`, we consider its variant based on progressive sampling (denoted `BAVARIAN-P`, see Alg. 2 and Sect. 4.2 of [21]) which addresses the same problem solved by SILVAN and `KADABRA`, and we tested it using all different estimators for the betweenness centrality presented in [21] (called `rk`, `ab`, and `bp`).

Graphs. We tested SILVAN on 7 undirected and 11 directed real-world graphs from SNAP[3] and KONECT[4], most of them previously analysed by `KADABRA` [12] and other previous methods [21, 49, 52]. The characteristics of the graphs are described in detail in Table 1.

5.1 Absolute Approximation

We first consider the task of computing an ε absolute approximation to the betweenness centrality of all nodes.

For every graph, we ran all algorithms with parameter $\varepsilon \in \{0.01, 0.005, 0.0025, 0.001, 0.0005\}$, chosen to have comparable magnitude to the betweenness centrality of the most central nodes (i.e., see col. $\hat{\xi}$ of Table 1); this is required to compute meaningful approximations (i.e., an ε absolute approximation is useless when the centralities of the most central nodes are much smaller than ε). We fix $\delta = 0.05$, and use c Monte Carlo Rademacher vectors with $c = 25$ for SILVAN and `BAVARIAN` (note that $c = k$ in [21]). We do not show results for other values of δ, as this parameter has minimal impact on the results, due to the use of exponential tail bounds (see δ in (1) and (3)). Regarding c, we follow [45] and [21], that have shown that sharp bounds are obtained even with a low number of

[1] https://github.com/natema/kadabra
[2] https://github.com/VandinLab/SILVAN
[3] http://snap.stanford.edu/data/index.html
[4] http://konect.cc/networks/

Table 1. Statistics of undirected (top section) and directed (bottom section) graphs. D is the vertex diameter, ρ is an upper bound of the average shortest path lenth, and $\hat{\xi}$ is an upper bound of $\max_v\{b(v)\}$.

G	$\lvert V\rvert$	$\lvert E\rvert$	D	ρ	$\hat{\xi}$
actor-collaboration	3.82e5	3.31e7	13	2.87	0.0090
ca-AstroPh	1.87e4	1.98e5	14	3.20	0.0285
ca-GrQc	5.24e3	1.44e4	17	3.51	0.0450
com-amazon	3.34e5	9.25e5	44	11.97	0.0450
com-dblp	3.17e5	1.04e6	21	6.27	0.0162
com-youtube	1.13e6	2.98e6	20	4.68	0.2573
email-Enron	3.66e4	1.83e5	11	2.78	0.0749
cit-HepPh	3.45e4	4.21e5	12	5.35	0.1817
cit-HepTh	2.77e4	3.52e5	13	2.10	0.1237
email-EuAll	2.65e5	4.20e5	14	0.56	0.0121
p2p-Gnutella31	6.25e4	1.47e5	11	2.16	0.0071
soc-Epinions1	7.58e4	5.08e5	14	2.11	0.0210
soc-LiveJournal1	4.84e6	6.90e7	16	4.58	0.0270
soc-pokec	1.63e6	3.06e7	16	3.94	0.0802
wiki-Talk	2.39e6	5.02e6	9	0.26	0.0037
wiki-topcats	1.79e6	2.85e7	9	5.87	0.0985
wiki-Vote	7.11e3	1.03e5	7	0.66	0.0240
wikipedia-link-en	1.35e7	4.37e8	10	3.21	0.0300

Monte Carlo trials, and that there are minimal improvements using $c > 30$. We ran all algorithms 10 times and report averages ± stds. We limit the execution time of each run to 6 hours; we terminate the algorithm when exceeding this threshold.

For the empirical peeling scheme of SILVAN, we sample $m' = \log(1/\delta)/\varepsilon$ shortest paths to generate $\mathcal{S}'$; we note that m' always results in a very small fraction of the overall samples analysed by SILVAN. Regarding the sampling schedule followed in the second phase, we use $\mathcal{S}'$ to identify a minimum number m_1 of samples before starting to evaluate the stopping condition. To do so, we perform a binary search to identify the minimum m_1 such that (1) (with $R_j = 0$) is not larger than ε; this gives an optimistic first guess of the number of samples to process for obtaining an ε-aproximation. We then increase each m_i with a geometric progression, such that $m_i = 1.2 \cdot m_{i-1}$. While a geometric progression is considered to be optimal [48], we note that the procedure `samplingSchedule` can be implemented with general schedules.

As described in Section 4.3, SILVAN uses the procedure `sufficientSamples` to obtain an upper bound $\hat{m}$ to the number of samples to process to obtain an ε absolute approximation: it does so using $\mathcal{S}'$, computing an upper bound ρ to the average shortest path length (Theorem 4.10) and an upper bound $\hat{v}$ to the suprem variance of the estimators $\sup_{f_v \in \mathcal{F}} Var(f_v)$ ($\hat{v} = v_{\mathcal{F}_j}$ with $t = 1$ in Proposition 4.3). Then, `sufficientSamples` plugs these estimates in Theorem 4.6 to compute $\hat{m}$. The `empiricalPeeling` procedure of SILVAN follows the scheme described at the end of Section 4.2 using $a = 2$.

For the progressive sampling schedule of `BAVARIAN`, we use the same geometric progression parameter of SILVAN (equal to 1.2, analogous to the parameter θ in [21]).

Figure 1 shows the results for this set of experiments comparing SILVAN to KADABRA, while Figure 2 shows the results comparing SILVAN to BAVARIAN for the estimator ab (more results in Figure 8, and analogous plots for rk and bp in Figures 9 and 10, all in Appendix).

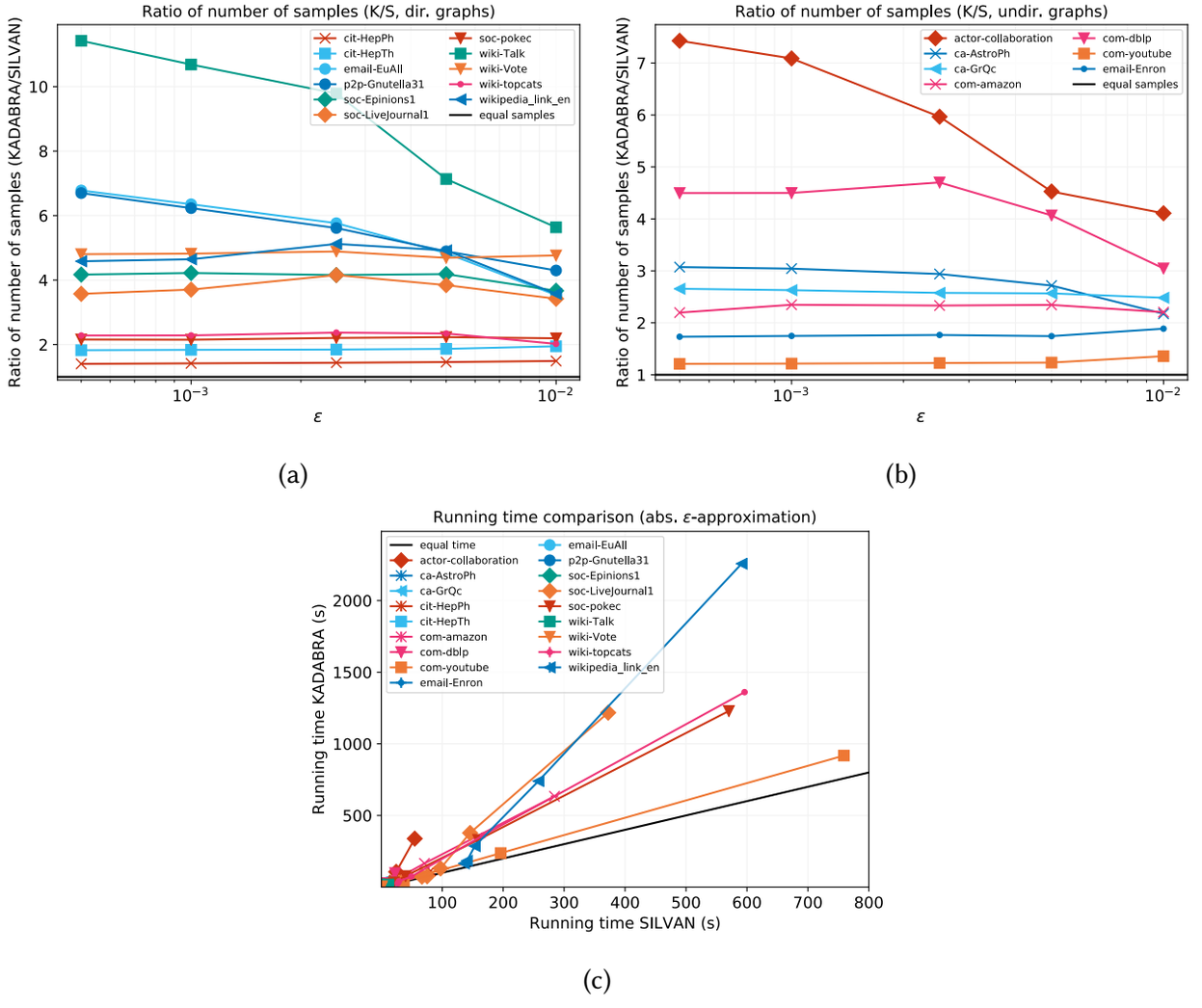


Fig. 1. Comparison between the performance of KADABRA and SILVAN for obtaining absolute approximations. (a): ratios of the number of samples required by KADABRA and the number of samples required by SILVAN for directed graphs. (b): as (a) for undirected graphs. (c): comparison of the running times of KADABRA (y axis) and SILVAN (x axis) for all graphs. Additional plots in Figure 7.

5.1.1 Sample sizes. In Figures 1 (a) and (b) we show the ratios between the number of samples required by KADABRA and SILVAN to converge (we sum the number of samples of both phases, for both algorithms) for directed and undirected graphs. We can see that the number of samples needed by SILVAN is always smaller than KADABRA, by at least 20%; for 14 out of 18 graphs, SILVAN finished after processing *less than half* of the samples considered by KADABRA, and may require up to an order of magnitude less samples. By inspecting the graphs' statistics (Table 1), the largest improvements are obtained for graphs with smallest $\sup_{v \in V} b(v) \leq \hat{\xi}$. In fact, the number of samples required by SILVAN (Figure 4 (a)) varies significantly among graphs, with strong dependence on $\hat{\xi}$. Notice that $\sup_{v \in V} b(v)$ upper bounds the maximum variance $\sup_{v \in V} Var(f_v)$. A potential cause of the gap between SILVAN and KADABRA may depend on the use of the VC-dimension based bound

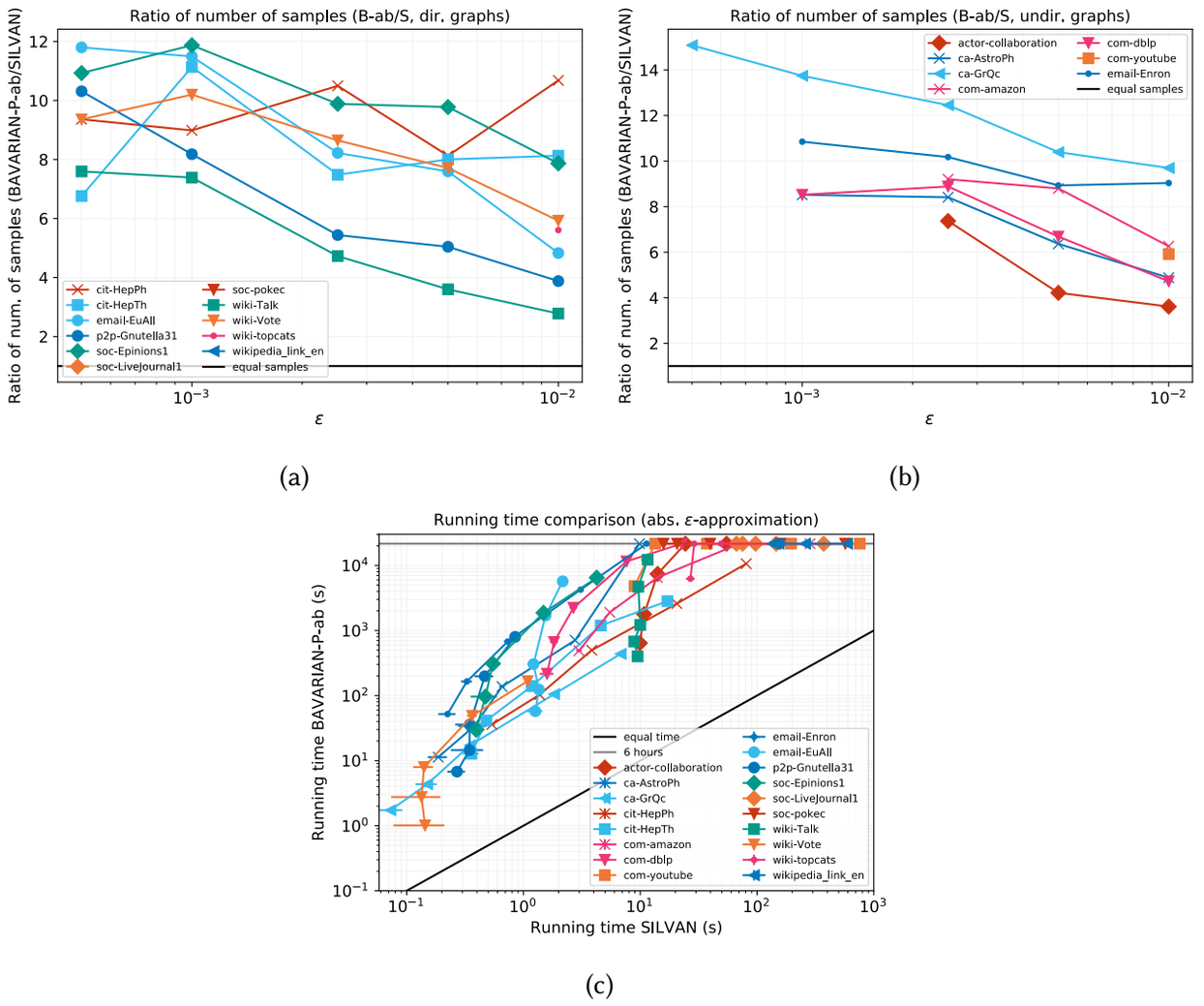


Fig. 2. Comparison between the performance of BAVARIAN (ab estimator) and SILVAN for obtaining absolute approximations. (a): ratios of the number of samples required by BAVARIAN and the number of samples required by SILVAN for directed graphs. (b): as (a) for undirected graphs. (c): comparison of the running times of BAVARIAN (y axis) and SILVAN (x axis) for all graphs (axes in logarithmic scale). Additional plots in Figures 8-10.

in the adaptive sampling analysis of KADABRA; such bound is indeed required for its correctness, but it is agnostic to any property of the underlying graph (apart from the vertex diameter) and thus results in overly conservative guarantees in such cases. This confirms the significance of SILVAN's sharp *variance-adaptive bounds*. In addition, the fact that SILVAN obtains simultaneous and non-uniform data-dependent approximations for *sets of nodes*, exploting correlations among nodes through the use of the c-MCERA, leads to refined guarantees.

We now compare SILVAN with BAVARIAN in terms of sample sizes. We remark that the plots for sample sizes only show the results for cases in which BAVARIAN terminates in reasonable time (i.e, in less than 6 hours), while figures for running times show a lower bound for such cases. From Figures 2 (a) and (b), we can see that SILVAN always requires a fraction of the samples needed by BAVARIAN: at most *half* of the samples for all graphs, and at most 1/4 of the samples for 17 out of 18 graphs, with an improvement of up to one order of magnitude. We observed analogous results for the rk estimator (Figures 9 (c) and (d)). The bp estimator resulted to be the most efficient version of BAVARIAN in terms of number of samples; this is not suprising, since one bp's sample considers

all shortest paths starting from a single node, rather than shortest paths between two nodes (on the other hand, each sample is potentially much more expensive to compute). In any case, the number of samples needed by SILVAN is always smaller than BAVARIAN-bp by at least 10%, up to a factor 5.

Overall, SILVAN obtains high-quality approximations at a fraction of the samples required by state-of-the-art methods; this highlights the significance of SILVAN's non-uniform approximation approach via empirical peeling and its novel improved bounds on the number of sufficient samples presented in Section 4.3.

5.1.2 Running times. We now discuss how the reduction in the number of samples impacts the overall running times. We observed that, generally, the running time roughly increases linearly with the sample size (Figure 4 (b) shows that the relationship between the sample sizes and the running times of SILVAN is essentialy linear). In fact, the time spent on sampling shortest paths is usually the dominating cost of the algorithms.

In Figure 1 (c) we compare the running times of SILVAN (x axis) and KADABRA (y axis) (we show ratios and axis in log. scale in Figure 7). While for smaller graphs both SILVAN and KADABRA terminate very quickly (e.g., in < 10 seconds), for the largest and most demanding graphs the reduction on the number of samples achieved by SILVAN has a sensible and significant impact on the running times, as clearly shown in Figure 1 (c). For instance, SILVAN analyses the most demanding graph (wikipedia-link-en) in less than 1/3 of the time required by KADABRA when $\varepsilon \leq 10^{-3}$ (see Figure 7 (f)). This is a consequence of significantly reducing the required samples, and also reflects the capability of SILVAN to compute the c-MCERA *incrementally* as shortest paths are sampled, incurring in a negligible computational overhead.

In Figure 2 (c) we compare the running times of SILVAN (x axis) and BAVARIAN using the ab estimator (y axis) (additional plots and other estimators in Figures 8-10). Note that we report a lower bound to the running time of BAVARIAN when exceeding 6 hours ($= 2.16 \cdot 10^4$ seconds); BAVARIAN exceeded this threshold on most large graphs and for smaller values of ε, while SILVAN never required more than 17 minutes ($= 10^3$ seconds). Overall, we observed SILVAN to be at least one order of magnitude faster than BAVARIAN, up to 3 orders of magnitude. We observed very similar results for the rk estimator (Figure 9). SILVAN is also at least one order of magnitude faster than BAVARIAN using the bp estimator for all but for the wiki-Vote graph, for which it is > 3 times faster (Figure 10). SILVAN's improvements are due to both the significant reduction in the number of samples (as discussed previously) thanks to its non-uniform approximation scheme, and from the fact that SILVAN leverages a more efficient algorithm for sampling shortest paths, based on the balanced bidirectional BFS, drastically reducing the computational requirement for the task.

We conclude that SILVAN requires much fewer resources to obtain rigorous approximations of the betweenness centrality of all nodes of the same quality, or, equivalently, sharper guarantees for the same amount of work.

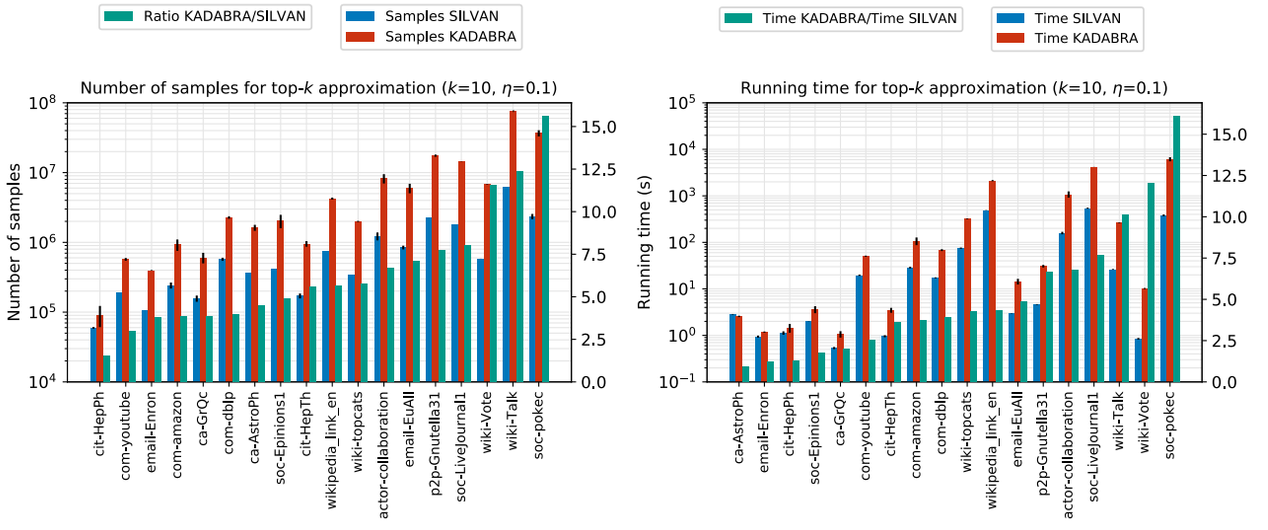
5.1.3 Quality of SILVAN's approximations. Finally, we investigated the accuracy of the approximations reported by SILVAN by computing the exact betweenness centrality of all the nodes of 6 graphs (3 undirected and 3 directed, representative of other instances) and measuring $\mathsf{D}(\mathcal{F}, \mathcal{S})$ over all runs. We show these results in Figure 5 (see Appendix). As expected from our theoretical analysis, we always observed $\mathsf{D}(\mathcal{F}, \mathcal{S}) \leq \varepsilon$, thus SILVAN is more accurate than guaranteed. However, the gap between $\mathsf{D}(\mathcal{F}, \mathcal{S})$ and ε is not large, confirming the sharpness of the guarantees provided by SILVAN. We remark that the exact approach requires several hours on the larger graphs we considered for this set of experiments (e.g., for the com-dblp graph, the exact approach implemented in Networkit [60] requires > 1 hour to terminate using all 64 cores), and does not complete in reasonable time for the largest instances (e.g., [12] reports that ≈ 1 week is necessary for graphs of size similar to the largest of our test set). Instead, SILVAN finishes in at most few

minutes for the lowest value of ε (e.g., always less than 20 seconds for com-dblp), and it is much faster for other cases.

5.2 Top-k Approximation

We now present experiments on the task of computing relative approximation of the set of top-k most central nodes. SILVAN is the first method that allows to approximate the top-k most central nodes with *relative* and *non-uniform* bounds via empirical peeling, differently from previous methods that focus on additive approximations (or rely on uniform additive approximations as a proxy) [12, 52]; therefore, we compare SILVAN [12] with KADABRA, the best performing approach that allows to obtain approximations of comparable quality. We recall that the top-k approximation proposed in [12], for a given k and ε, guarantees an ε additive approximation of the top-k nodes; however, the confidence intervals for some of the nodes can be relaxed (i.e., be wider than 2ε) if they can be ranked correctly with looser accuracy. Instead, SILVAN guarantees that all nodes are well estimated within the relative accuracy η. For given k and η, we first run SILVAN on all graphs; when finished, we store the maximum absolute deviation ε_k required to guarantee all properties of Definition 4.11, and we run KADABRA with k and ε_k.

We considered $k \in \{5, 10, 25\}$ and $\eta \in \{0.25, 0.1, 0.05\}$. As in previous experiments, SILVAN's empirical peeling follows the procedure described in Section 4.2 with $a = 2$. We report avgs. ± stds over 10 runs.



(a) (b)

Fig. 3. Comparison of number of samples (a) and running times (b) of SILVAN-TopK with KADABRA for obtaining top-k approximations for $k = 10$ and $\eta = 0.1$ (all other combinations shown in Figure 11).

Figure 3 shows the number of samples and running time of both algorithms to obtain their respective top-k approximations for $k = 10$ and $\eta = 0.1$, representative of other cases (we show all combinations of k and η in Figure 11 in Appendix).

From Figure 3 (a) we see that SILVAN-TopK requires a fraction of the samples of KADABRA, even if offering stronger guarantees (no confidence intervals are relaxed according to the ranking): SILVAN-TopK requires at most 2/3 of KADABRA's samples, and finishes after processing less than 1/3 of the samples for 17 out of 18 graphs; for 10 graphs, it needs less than 1/5 of the samples, and less than 1/10 for 3 graphs.

The reduction of the number of samples, similarly to the previous setting, significantly impacts the running times. From Figure 3 (b) we conclude that, while in cases in which both algorithms conclude very quickly (e.g., in less than 3 seconds on small graphs) they obtain comparable performances, on larger graphs SILVAN significantly outperforms KADABRA. In fact, for 14 graphs SILVAN-TopK finished in less than half of the time of KADABRA, and it is at least 5 times faster in 6 cases. For three of the most demanding graphs, SILVAN is more than 10 times faster. Overall, SILVAN analyses all graphs in < 0.5 hours, while KADABRA needs > 4 hours. We conclude that SILVAN significantly reduces the computational requirements for the task, potentially allowing much more interactive exploratory analyses.

Additionally, we compared the quality of the output of KADABRA w.r.t. SILVAN-TopK when using the same number of samples: we stop KADABRA at the number of samples required by SILVAN-TopK. Figure 6 in Appendix shows runs taken at random for 3 graphs, using $k = 10$ and $\eta = 0.1$. From Figure 6 we can see that SILVAN-TopK (in blue) provides much tighter upper and lower bounds than KADABRA (in red); obtaining sharper confidence intervals on the top-k nodes has a drastic effect on the capability of the algorithms to rank nodes correctly. Consequently, for the same work, SILVAN-TopK reports much less false positives (e.g., 16 vs 45 results for com-dblp) and can clearly identify the rank of the most central nodes.

6 CONCLUSIONS

We introduced SILVAN, a novel progressive sampling algorithm to estimate the betweenness centrality of all nodes in a graph. SILVAN relies on new bounds on supremum deviation of functions, based on the c-MCERA and non-uniform approximation scheme via empirical peeling. We present variants of SILVAN to obtain additive approximations, and relative approximations for the top-k betweenness centrality. Our experimental results show that SILVAN significantly outperforms state-of-the-art approaches for approximating betweenness centrality with the same guarantees.

There are multiple interesting directions for future work. While in this work we considered various approximations of the betweenness centrality in a static setting, recent works considered extending the problem to dynamic [6, 7, 31], temporal [55], and uncertain graphs [54], or different types of centralities [24], all settings in which we believe the ideas behind our algorithm SILVAN could lead to improved approximations.

Furthermore, the empirical peeling scheme we introduced in this work is general: it can be applied to sets of functions with arbitrary domains, so it can potentially benefit randomized approximation algorithms in other settings, such as interesting [53, 56, 58] and significant pattern mining [46], and sequential hypothesis testing [25].

Acknowledgments. This work was supported, in part, by MIUR of Italy, under PRIN Project n. 20174LF3T8 AHeAD and grant L. 232 (Dipartimenti di Eccellenza), and by the University of Padova under project "SID 2020: RATED-X".

REFERENCES

[1] Jac M Anthonisse. 1971. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde* BN 9/71 (1971).

[2] Peter L Bartlett, Olivier Bousquet, Shahar Mendelson, et al. 2005. Local rademacher complexities. *The Annals of Statistics* 33, 4 (2005), 1497–1537.

[3] Peter L. Bartlett and Shahar Mendelson. 2002. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research* 3, Nov (2002), 463–482.

[4] Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. 2019. Computing top-k closeness centrality faster in unweighted graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13, 5 (2019), 1–40.

[5] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. 2018. Improving the betweenness centrality of a node by adding links. *Journal of Experimental Algorithmics (JEA)* 23 (2018), 1–32.
[6] Elisabetta Bergamini and Henning Meyerhenke. 2015. Fully-dynamic approximation of betweenness centrality. In *Algorithms-ESA 2015*. Springer, 155–166.
[7] Elisabetta Bergamini, Henning Meyerhenke, and Christian L Staudt. 2014. Approximating betweenness centrality in large evolving networks. In *2015 Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 133–146.
[8] Rajendra Bhatia and Chandler Davis. 2000. A better bound on the variance. *The American Mathematical Monthly* 107, 4 (2000), 353–357.
[9] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. 2011. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th international conference on World Wide Web*. 625–634.
[10] Paolo Boldi and Sebastiano Vigna. 2013. In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond. In *2013 IEEE 13th International Conference on Data Mining Workshops*. IEEE, 621–628.
[11] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. 2016. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science* 322 (2016), 51–67.
[12] Michele Borassi and Emanuele Natale. 2019. KADABRA is an adaptive algorithm for betweenness via random approximation. *Journal of Experimental Algorithmics (JEA)* 24 (2019), 1–35.
[13] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. 2005. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics* 9 (2005), 323–375.
[14] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. 2013. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press.
[15] Olivier Bousquet. 2002. A Bennett concentration inequality and its application to suprema of empirical processes. *Comptes Rendus Mathematique* 334, 6 (2002), 495–500.
[16] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of mathematical sociology* 25, 2 (2001), 163–177.
[17] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. 2020. Distributed Graph Diameter Approximation. *Algorithms* 13, 9 (2020), 216.
[18] Shiri Chechik, Edith Cohen, and Haim Kaplan. 2015. Average distance queries through weighted samples in graphs and metric spaces: high scalability with tight statistical guarantees.. In *APPROX/RANDOM*.
[19] Corinna Cortes, Spencer Greenberg, and Mehryar Mohri. 2019. Relative deviation learning bounds and generalization with unbounded loss functions. *Annals of Mathematics and Artificial Intelligence* 85, 1 (2019), 45–70.
[20] Cyrus Cousins and Matteo Riondato. 2020. Sharp uniform convergence bounds through empirical centralization. *Advances in Neural Information Processing Systems* 33 (2020), 15123–15132.
[21] Cyrus Cousins, Chloe Wohlgemuth, and Matteo Riondato. 2021. Bavarian: Betweenness Centrality Approximation with Variance-Aware Rademacher Averages. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 196–206.
[22] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. 2013. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science* 514 (2013), 84–95.
[23] Pierluigi Crescenzi, Roberto Grossi, Leonardo Lanzi, and Andrea Marino. 2012. On computing the diameter of real-world directed (weighted) graphs. In *International Symposium on Experimental Algorithms*. Springer, 99–110.
[24] Alane M de Lima, Murilo VG da Silva, and André L Vignatti. 2020. Estimating the Percolation Centrality of Large Networks through Pseudo-dimension Theory. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1839–1847.
[25] Lorenzo De Stefani and Eli Upfal. 2019. A Rademacher Complexity Based Method for Controlling Power and Confidence Level in Adaptive Statistical Analysis. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 71–80.
[26] Dóra Erdős, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. 2015. A divide-and-conquer algorithm for betweenness centrality. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 433–441.
[27] Philippe Flajolet and G Nigel Martin. 1983. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 76–82.
[28] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.
[29] Sara A Geer and Sara van de Geer. 2000. *Empirical Processes in M-estimation*. Vol. 6. Cambridge university press.
[30] Sariel Har-Peled and Micha Sharir. 2011. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry* 45, 3 (2011), 462–496.
[31] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully dynamic betweenness centrality maintenance on massive networks. *Proceedings of the VLDB Endowment* 9, 2 (2015), 48–59.

[32] Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. 2012. A framework for the evaluation and management of network centrality. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 427–438.
[33] Vladimir Koltchinskii and Dmitriy Panchenko. 2000. Rademacher processes and bounding the risk of function learning. In *High dimensional probability II*. Springer, 443–457.
[34] Yi Li, Philip M Long, and Aravind Srinivasan. 2001. Improved bounds on the sample complexity of learning. *J. Comput. System Sci.* 62, 3 (2001), 516–527.
[35] Clémence Magnien, Matthieu Latapy, and Michel Habib. 2009. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics (JEA)* 13 (2009), 1–10.
[36] Ahmad Mahmoody, Charalampos E Tsourakakis, and Eli Upfal. 2016. Scalable betweenness centrality maximization via sampling. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1765–1773.
[37] Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., USA.
[38] Pascal Massart. 2000. Some applications of concentration inequalities to statistics. *Annales de la Faculté des sciences de Toulouse: Mathématiques* 9, 2 (2000), 245–303.
[39] Andreas Maurer and Massimiliano Pontil. 2009. Empirical Bernstein bounds and sample variance penalization. *arXiv preprint arXiv:0907.3740* (2009).
[40] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.
[41] Mark Newman. 2018. *Networks*. Oxford university press.
[42] Luca Oneto, Alessandro Ghio, Davide Anguita, and Sandro Ridella. 2013. An improved analysis of the Rademacher data-dependent bound using its self bounding property. *Neural Networks* 44 (2013), 107–111.
[43] Leonardo Pellegrina. 2020. Sharper convergence bounds of Monte Carlo Rademacher Averages through Self-Bounding functions. *arXiv preprint arXiv:2010.12103* (2020).
[44] Leonardo Pellegrina. 2021. *Rigorous and Efficient Algorithms for Significant and Approximate Pattern Mining*. Ph.D. Thesis. http://www.dei.unipd.it/~pellegri/thesis/leonardo_pellegrina_tesi.pdf.
[45] Leonardo Pellegrina, Cyrus Cousins, Fabio Vandin, and Matteo Riondato. 2020. MCRapper: Monte-Carlo Rademacher Averages for Poset Families and Approximate Pattern Mining. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2165–2174.
[46] Leonardo Pellegrina, Matteo Riondato, and Fabio Vandin. 2019. SPuManTE: Significant Pattern Mining with Unconditional Testing. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. ACM, New York, NY, USA, 1528–1538. https://doi.org/10.1145/3292500.3330978
[47] David Pollard. 2012. *Convergence of stochastic processes*. Springer Science & Business Media.
[48] Foster Provost, David Jensen, and Tim Oates. 1999. Efficient progressive sampling. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. 23–32.
[49] Matteo Riondato and Evgenios M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475.
[50] Matteo Riondato and Eli Upfal. 2014. Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. *ACM Trans. Knowl. Disc. from Data* 8, 4 (2014), 20. https://doi.org/10.1145/2629586
[51] Matteo Riondato and Eli Upfal. 2015. Mining Frequent Itemsets through Progressive Sampling with Rademacher Averages. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, 1005–1014.
[52] Matteo Riondato and Eli Upfal. 2018. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *ACM Trans. Knowl. Disc. from Data* 12, 5 (2018), 61.
[53] Matteo Riondato and Fabio Vandin. 2018. MiSoSouP: Mining Interesting Subgroups with Sampling and Pseudodimension. In *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Disc. and Data Mining (KDD '18)*. ACM, 2130–2139.
[54] Arkaprava Saha, Ruben Brokkelkamp, Yllka Velaj, Arijit Khan, and Francesco Bonchi. 2021. Shortest paths and centrality in uncertain networks. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1188–1201.
[55] Diego Santoro and Ilie Sarpe. 2022. ONBRA: Rigorous Estimation of the Temporal Betweenness Centrality in Temporal Networks. *arXiv preprint arXiv:2203.00653* (2022).
[56] Diego Santoro, Andrea Tonon, and Fabio Vandin. 2020. Mining Sequential Patterns with VC-Dimension and Rademacher Complexity. *Algorithms* 13, 5 (2020), 123.
[57] Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. 2013. Shattering and compressing networks for betweenness centrality. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 686–694.
[58] Ilie Sarpe and Fabio Vandin. 2021. odeN: Simultaneous Approximation of Multiple Motif Counts in Large Temporal Networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1568–1577.

[59] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press.
[60] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Network Science* 4, 4 (2016), 508–530. https://doi.org/10.1017/nws.2016.20
[61] Aad W Van Der Vaart, Aad van der Vaart, Adrianus Willem van der Vaart, and Jon Wellner. 1996. *Weak convergence and empirical processes: with applications to statistics.* Springer Science & Business Media.
[62] V. N. Vapnik and A. Ya. Chervonenkis. 1971. On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability & Its Applications* 16, 2 (1971), 264. https://doi.org/10.1137/1116025
[63] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.
[64] Yuichi Yoshida. 2014. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* 1416–1425.

A APPENDIX

A.1 Proofs of Section 4.1

We prove Theorem 4.1, our new concentration bound of the c-MCERA towards the ERA, one of our main technical contributions.

Theorem 4.1. *For $c, m \geq 1$, let $\boldsymbol{\sigma} \in \{-1,1\}^{c\times m}$ be an $c \times m$ matrix of Rademacher random variables, such that $\boldsymbol{\sigma}_{j,i} \in \{-1,1\}$ independently and with equal probability. Then, with probability $\geq 1-\delta$ over $\boldsymbol{\sigma}$, it holds*

$$\hat{\mathsf{R}}\left(\mathcal{F},\mathcal{S}\right) \leq \hat{\mathsf{R}}^{c}_{m}(\mathcal{F},\mathcal{S},\boldsymbol{\sigma}) + \sqrt{\frac{4w_{\mathcal{F}}(\mathcal{S})\ln\left(\frac{1}{\delta}\right)}{cm}}.$$

Proof. We first invoke an important result on the concentration of functions uniformly distributed on the binary hypercube.

Theorem A.1 (Thm. 5.3, [14]). *For $k > 0$, let a function $g : \{-1,1\}^k \to \mathbb{R}$ and assume that X is uniformly distributed on $\{-1,1\}^k$. Let $v > 0$ be such that*

$$\sum_{i=1}^{k}\left(g\left(x\right) - g\left(\overline{x}^i\right)\right)_+^2 \leq v$$

for all $x = (x_1,\ldots,x_k) \in \{-1,1\}^k$, where

$$\overline{x}^i = (x_1,\ldots,x_{i-1},-x_i,x_{i+1},\ldots,x_k)$$

is a copy of x with the i-th component multiplied by -1, and $(b)_+ = \max\{b,0\}$ is the positive part of $b \in \mathbb{R}$. Then, the random variable $Z \doteq g(X)$ satisfies, for all $t > 0$,

$$\Pr\left(Z > \mathbb{E}\left[Z\right] + t\right), \Pr\left(Z < \mathbb{E}\left[Z\right] - t\right) \leq \exp\left(-t^2/v\right).$$

We first observe the equivalence

$$\left(g\left(x\right) - g\left(\overline{x}^i\right)\right)_+ = \left(g\left(x\right) - g\left(\overline{x}^i\right)\right)\mathbb{1}\left[g\left(x\right) > g\left(\overline{x}^i\right)\right] \tag{8}$$

for the function g as stated in Thm. A.1; consequently, to apply the result we only have to consider the cases in which $g\left(x\right) > g(\overline{x}^i)$, as otherwise (8) is equal to 0. We define the function $g : \{-1,1\}^{cm} \to \mathbb{R}$ as $g(\boldsymbol{\sigma}) \doteq nm\hat{\mathsf{R}}^{c}_{m}(\mathcal{F},\mathcal{S},\boldsymbol{\sigma})$. For a given $\boldsymbol{\sigma}$, denote with $f_j^\star$ one of the functions attaining the supremum of $\sup_{f\in\mathcal{F}}\{\sum_{i=1}^{m}\boldsymbol{\sigma}_{j,i}f(s_i)\}$. Note that we can safely assume the supremum to be always achieved as we assume functions $\in \mathcal{F}$ to be bounded (or, at least, we assume $\sup_{f\in\mathcal{F}}\max_{s\in\mathcal{S}}|f(s)|$ is bounded). We define $\overline{\boldsymbol{\sigma}}_{j,i}$ as a copy of $\boldsymbol{\sigma}$ with the (j,i) component $\boldsymbol{\sigma}_{j,i}$ of $\boldsymbol{\sigma}$ multiplied by -1. Thus, $g(\overline{\boldsymbol{\sigma}}_{j,i})$ is evaluated as $g(\boldsymbol{\sigma})$ but with $\boldsymbol{\sigma}_{j,i}$ having flipped sign. We first prove that $g(\overline{\boldsymbol{\sigma}}_{j,i}) \geq$

$g(\boldsymbol{\sigma}) - 2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i)$ for all $i \in [1, m]$ and for all $j \in [1, c]$, following ideas developed in [43, 44]. From the definition of g given above, we have

$$
\begin{aligned}
g(\overline{\boldsymbol{\sigma}}_{j,i}) = cm\hat{\mathrm{R}}_m^c(\mathcal{F}, \mathcal{S}, \overline{\boldsymbol{\sigma}}_{j,i}) &= \sum_{\substack{h=1 \\ h\neq j}}^{c} \sup_{f\in\mathcal{F}} \left\{ \sum_{t=1}^{m} \boldsymbol{\sigma}_{h,t} f(s_t) \right\} + \sup_{f\in\mathcal{F}} \left\{ \sum_{\substack{t=1 \\ t\neq i}}^{m} \boldsymbol{\sigma}_{j,t} f(s_t) - \boldsymbol{\sigma}_{j,i} f(s_i) \right\} \\
&= \sum_{\substack{h=1 \\ h\neq j}}^{c} \sup_{f\in\mathcal{F}} \left\{ \sum_{t=1}^{m} \boldsymbol{\sigma}_{h,t} f(s_t) \right\} + \sup_{f\in\mathcal{F}} \left\{ \sum_{t=1}^{m} \boldsymbol{\sigma}_{j,t} f(s_t) - 2\boldsymbol{\sigma}_{j,i} f(s_i) \right\} \\
&\geq \sum_{\substack{h=1 \\ h\neq j}}^{c} \sup_{f\in\mathcal{F}} \left\{ \sum_{t=1}^{m} \boldsymbol{\sigma}_{h,t} f(s_t) \right\} + \sum_{t=1}^{m} \boldsymbol{\sigma}_{j,t} f_j^\star(s_t) - 2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i) \\
&= \sum_{h=1}^{c} \sup_{f\in\mathcal{F}} \left\{ \sum_{t=1}^{m} \boldsymbol{\sigma}_{h,t} f(s_t) \right\} - 2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i) = g(\boldsymbol{\sigma}) - 2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i).
\end{aligned}
$$

Therefore, we have that

$$
\begin{aligned}
\sum_{i=1}^{m}\sum_{j=1}^{c} \left(g(\boldsymbol{\sigma}) - g(\overline{\boldsymbol{\sigma}}_{j,i})\right)_+^2 &\leq \sum_{i=1}^{m}\sum_{j=1}^{c} \left(g(\boldsymbol{\sigma}) - \left(g(\boldsymbol{\sigma}) - 2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i)\right)\right)_+^2 \\
&= \sum_{i=1}^{m}\sum_{j=1}^{c} \left(2\boldsymbol{\sigma}_{j,i} f_j^\star(s_i)\right)_+^2 \leq \sum_{i=1}^{m}\sum_{j=1}^{c} 4 f_j^\star(s_i)^2 \leq 4cmw_{\mathcal{F}}.
\end{aligned}
$$

We apply Theorem A.1 to $Z = g(\boldsymbol{\sigma})$ with $v = 4cmw_{\mathcal{F}}$, obtaining

$$
\begin{aligned}
\Pr\left(cm\hat{\mathrm{R}}_m^c(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma}) > cm\hat{\mathrm{R}}(\mathcal{F}, \mathcal{S}) + t\right) &\leq \exp\left(\frac{-t^2}{4cmw_{\mathcal{F}}}\right), \qquad (9) \\
\Pr\left(cm\hat{\mathrm{R}}_m^c(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma}) < cm\hat{\mathrm{R}}(\mathcal{F}, \mathcal{S}) - t\right) &\leq \exp\left(\frac{-t^2}{4cmw_{\mathcal{F}}}\right).
\end{aligned}
$$

The substitution $\varepsilon = t/(cm)$ in (9) yields the inequality

$$
\Pr\left(\hat{\mathrm{R}}(\mathcal{F}, \mathcal{S}) > \hat{\mathrm{R}}_m^c(\mathcal{F}, \mathcal{S}, \boldsymbol{\sigma}) + \varepsilon\right) \leq \exp\left(-\frac{cm\varepsilon^2}{4w_{\mathcal{F}}}\right). \qquad (10)
$$

The statement follows from imposing the r.h.s. of (10) to be $\leq \delta$ and solving for ε. □

The proof of our main result (Theorem 4.2) builds on the combination of the most refined concentration inequalities relating Rademacher averages to the supremum deviation [14], that we now introduce. Let the *Rademacher complexity* $\mathrm{R}(\mathcal{F}, m)$ of a set of functions $\mathcal{F}$ be defined as $\mathrm{R}(\mathcal{F}, m) = \mathbb{E}_{\mathcal{S}}\left[\hat{\mathrm{R}}(\mathcal{F}, \mathcal{S})\right]$. The following central result relates $\mathrm{R}(\mathcal{F}, m)$ to the *expected* supremum deviation.

Lemma A.2. *Symmetrization Lemma [40, 59] Let Z be either*

$$
\sup_{f\in\mathcal{F}} \left\{\mu_{\mathcal{S}}(f) - \mu_{\gamma}(f)\right\} \text{ or } \sup_{f\in\mathcal{F}} \left\{\mu_{\gamma}(f) - \mu_{\mathcal{S}}(f)\right\}.
$$

It holds $\mathbb{E}_{\mathcal{S}}[Z] \leq 2\mathrm{R}(\mathcal{F}, m)$.

The following gives a variance-dependent bound to the supremum deviation above its expectation, and it is due to Bousquet [15].

Theorem A.3. *(Thm. 2.3 [15]) Let $\hat{v}_{\mathcal{F}} \geq \sup_{f\in\mathcal{F}}\{Var(f)\}$, and Z be either $\sup_{f\in\mathcal{F}}\left\{\mu_{\mathcal{S}}(f)-\mu_{Y}(f)\right\}$ or $\sup_{f\in\mathcal{F}}\left\{\mu_{Y}(f)-\mu_{\mathcal{S}}(f)\right\}$. Then, with probability at least $1-\lambda$ over $\mathcal{S}$, it holds*

$$Z \leq \mathbb{E}_{\mathcal{S}}[Z] + \sqrt{\frac{2\ln\left(\frac{1}{\lambda}\right)\left(\hat{v}_{\mathcal{F}} + 2\mathbb{E}[Z]\right)}{m}} + \frac{\ln\left(\frac{1}{\lambda}\right)}{3m}.$$

The next result bounds $\mathsf{R}(\mathcal{F},m)$ above its estimate $\hat{\mathsf{R}}(\mathcal{F},\mathcal{S})$.

Theorem A.4. *[14, 42] With probability $\geq 1-\lambda$ over $\mathcal{S}$, it holds*

$$\mathsf{R}(\mathcal{F},m) \leq \hat{\mathsf{R}}(\mathcal{F},\mathcal{S}) + \sqrt{\left(\frac{\ln\left(\frac{1}{\lambda}\right)}{m}\right)^2 + \frac{2\ln\left(\frac{1}{\lambda}\right)\hat{\mathsf{R}}(\mathcal{F},\mathcal{S})}{m}} + \frac{\ln\left(\frac{1}{\lambda}\right)}{m}.$$

We are now ready to prove Theorem 4.2, the main technical contribution of Section 4.2.

Theorem 4.2. *Let $\mathcal{F} = \bigcup_{j=1}^{t}\mathcal{F}_j$ be a family of functions with codomain in $[0,1]$. Let $\mathcal{S}$ be a sample of size m taken i.i.d. from a distribution γ. Denote $v_{\mathcal{F}_j}$ such that $\sup_{f\in\mathcal{F}_j} Var(f) \leq v_{\mathcal{F}_j}$. For any $\delta\in(0,1)$, define*

$$\tilde{\mathsf{R}}_j \doteq \hat{\mathsf{R}}^{c}_{m}(\mathcal{F}_j,\mathcal{S},\boldsymbol{\sigma}) + \sqrt{\frac{4w_{\mathcal{F}_j}(\mathcal{S})\ln\left(\frac{4t}{\delta}\right)}{cm}},$$

$$\mathsf{R}_j \doteq \tilde{\mathsf{R}}_j + \frac{\ln\left(\frac{4t}{\delta}\right)}{m} + \sqrt{\left(\frac{\ln\left(\frac{4t}{\delta}\right)}{m}\right)^2 + \frac{2\ln\left(\frac{4t}{\delta}\right)\tilde{\mathsf{R}}_j}{m}},$$

$$\varepsilon_{\mathcal{F}_j} \doteq 2\mathsf{R}_j + \sqrt{\frac{2\ln\left(\frac{4t}{\delta}\right)\left(v_{\mathcal{F}_j}+4\mathsf{R}_j\right)}{m}} + \frac{\ln\left(\frac{4t}{\delta}\right)}{3m}. \tag{1}$$

With probability at least $1-\delta$ over the choice of $\mathcal{S}$ and $\boldsymbol{\sigma}$, it holds $\mathsf{D}(\mathcal{F}_j,\mathcal{S}) \leq \varepsilon_{\mathcal{F}_j}$ for all $j\in[1,t]$.

Proof. Define the events $E_{i,j}$, E_j, and E as

$$E_{1,j} = \text{“}\hat{\mathsf{R}}\left(\mathcal{F}_j,\mathcal{S}\right) > \tilde{\mathsf{R}}_j\text{”},\quad E_{2,j} = \text{“}\mathsf{R}(\mathcal{F}_j,m) > \mathsf{R}_j\text{”},$$
$$E_{3,j} = \text{“}\sup_{f\in\mathcal{F}_j}\left\{\mu_{\mathcal{S}}(f)-\mu_{Y}(f)\right\} > \varepsilon_{\mathcal{F}_j}\text{”},\quad E_4 = \text{“}\sup_{f\in\mathcal{F}_j}\left\{\mu_{Y}(f)-\mu_{\mathcal{S}}(f)\right\} > \varepsilon_{\mathcal{F}_j}\text{”},$$
$$E_j = \text{“}\mathsf{D}(\mathcal{F}_j,\mathcal{S}) > \varepsilon_{\mathcal{F}_j}\text{”},\quad E = \text{“}\exists j : \mathsf{D}(\mathcal{F}_j,\mathcal{S}) > \varepsilon_{\mathcal{F}_j}\text{”}.$$

Note that the statement is proved if $\Pr(E)\leq\delta$, and that the event E is contained in the event $\bigcup_j E_j$, meaning that $\Pr(E)\leq\Pr(\bigcup_j E_j)\leq\sum_j\Pr(E_j)$. Moreover, each event E_j is contained in the event $\bigcup_i E_{i,j}$, therefore $\Pr(E_j)\leq\sum_i\Pr(E_{i,j})$. To upper bound the probabilities of $E_{1,j}$, $E_{2,j}$, and $E_{3,j}$, we apply Lemma A.2, and Theorems A.3-A.4, replacing $\mathcal{F}$ by $\mathcal{F}_j$ and λ by $\delta/(4t)$:

(1) $\Pr(E_{1,j})\leq\delta/(4t)$ follows from Theorem 4.1 (replacing δ by $\delta/(4t)$);
(2) $\Pr(E_{2,j})\leq\delta/(4t)$ follows from Theorem A.4;
(3) From Lemma A.2 and twice the application of Theorem A.3, $\Pr(E_{3,j})$ and $\Pr(E_{4,j})$ are bounded below $\lambda=\delta/(4t)$.

The event E is true with probability at most δ, since $\Pr(E)\leq\sum_j\Pr(E_j)\leq\sum_j\sum_i\Pr(E_{i,j})\leq\delta$. □

We now prove Proposition 4.3, which shows that $\sup_{f\in\mathcal{F}} Var(f)$ can be tightly estimated using the empirical wimpy variance $w_{\mathcal{F}}(\mathcal{S})$. We prove it using the self-bounding properties of the function $w_{\mathcal{F}}(\mathcal{S})$, proved in [44].

Proposition 4.3. *With probability at least* $1-\delta$ *it holds, for all* $j \in [1,t]$,

$$\sup_{f\in\mathcal{F}_j} Var(f) \le v_{\mathcal{F}_j} \doteq w_{\mathcal{F}_j}(\mathcal{S}) + \frac{\ln\left(\frac{t}{\delta}\right)}{m} + \sqrt{\left(\frac{\ln\left(\frac{t}{\delta}\right)}{m}\right)^2 + \frac{2w_{\mathcal{F}_j}(\mathcal{S})\ln\left(\frac{t}{\delta}\right)}{m}}. \tag{2}$$

Proof. We use the fact that

$$\sup_{f\in\mathcal{F}_j} Var(f) = \sup_{f\in\mathcal{F}_j} \left\{\mathbb{E}[f^2] - \mathbb{E}[f]^2\right\} \le \sup_{f\in\mathcal{F}_j} \mathbb{E}[f^2],$$

so we focus on bounding the wimpy variance $\sup_{f\in\mathcal{F}_j}\mathbb{E}[f^2]$ of $\mathcal{F}_j$. We use following result.

Theorem A.5. *(Thm. 7.5.8 [44]) With probability* $\ge 1-\lambda$ *over* $\mathcal{S}$ *it holds*

$$\sup_{f\in\mathcal{F}} \mathbb{E}\left[f^2\right] \le w_{\mathcal{F}}(\mathcal{S}) + \frac{\ln\left(\frac{1}{\lambda}\right)}{m} + \sqrt{\left(\frac{\ln\left(\frac{1}{\lambda}\right)}{m}\right)^2 + \frac{2w_{\mathcal{F}}(\mathcal{S})\ln\left(\frac{1}{\lambda}\right)}{m}}.$$

We apply Theorem A.5 to each set $\mathcal{F}_j$ (using $\lambda = \delta/t$), obtaining the statement. □

A.2 Proofs of Section 4.2

We now prove Proposition 4.5 which provides the probabilistic guarantees of SILVAN.

Proposition 4.5. *With probability* $\ge 1-\delta$, *the output* $\tilde{b}$ *of SILVAN is a* ε*-approximation of* $BC(G)$.

Proof. We prove that the statement is a consequence of the following facts:

(1) the c-MCERA for all $\mathcal{F}_j$ is computed correctly at line 13;
(2) $\hat{v}_j$, at the end of iteration i, are computed, for all j, using the bound of Proposition 4.3 with probability $\delta/(2^{i+1}5)$ (i.e., in Proposition 4.3, δ is replaced by $\delta/(2^{i+1}5)$);
(3) $\varepsilon_{\mathcal{F}_j}$, at the end of iteration i, are computed, for all j, using the bound of Theorem 4.2 with probability $\delta/(2^{i+1}5)$ (i.e., in Theorem 4.2 $\delta/4$ is replaced by $\delta/(2^{i+1}5)$);
(4) $\hat{m}$ is computed s.t. $\mathrm{D}(\mathcal{F},\mathcal{S}) \le \varepsilon$ (with $|\mathcal{S}| \ge \hat{m}$) with probability $\ge 1-\delta/2$;
(5) SILVAN stops at an iteration i when $\max_j \varepsilon_j \le \varepsilon$ or at the first i s.t. $m_i \ge \hat{m}$.

We now prove the statement. Let X be a random variable equal to the index of the iteration in which the algorithm stops. Let the events A_i and $\hat{A}$ be defined as

$$A_i = \text{“at } i\text{-th iteration, } \exists v : f_v \in \mathcal{F}_j, \varepsilon_{\mathcal{F}_j} < |b(v) - \tilde{b}(v)|\text{”}\,,$$
$$\hat{A} = \text{“at the first } i \text{ s.t. } m_i \ge \hat{m}, \exists v : f_v \in \mathcal{F}_j, \varepsilon_{\mathcal{F}_j} < |b(v) - \tilde{b}(v)|\text{”}\,.$$

The algorithm is correct if both $\hat{A}$ and A_X are false, thus we want to prove that $\Pr(A_X \cup \hat{A}) \le \delta$. (Following analogous derivations discussed in [12], we remark that A_X depends on the random variable X, and it should not be confused with A_i for a fixed i.) It is enough to prove that $\Pr(A_X) + \Pr(\hat{A}) \le \delta$; since we assume that the fact (4) is true, we already have $\Pr(\hat{A}) \le \delta/2$. We proceed to show that $\Pr(A_X) \le \delta/2$ to prove the statement. Assuming facts (1), (2), and (3) are all true, at the end of each iteration i, for $i = 1, 2, \ldots$, it holds that $\mathrm{D}(\mathcal{F}_j,\mathcal{S}) \le \varepsilon_j$ for all j with probability $\ge 1-\delta/(2^{i+1})$, combining Proposition 4.3 and Theorem 4.2. Consequently, $\Pr(A_i) \le \delta/(2^{i+1}), \forall i \ge 1$. We have that

$$\Pr(A_X) = \sum_i \Pr\left(A_i \cap \text{“}X = i\text{”}\right) \le \sum_i \Pr(A_i) \le \sum_i \frac{\delta}{2^{i+1}} \le \delta/2.$$

□

A.3 Proofs of Section 4.3

We first prove our novel refined upper limit on the number of samples required to achieve a ε absolute approximation of the betweenness centrality of all nodes in a graph.

THEOREM 4.6. *Let $\mathcal{F} = \{f_v, v \in V\}$ be a set of functions from a domain $\mathcal{X}$ to $[0,1]$. Let a distribution γ such that $\mathbb{E}_{\tau\sim\gamma}[f_v(\tau)] = b(v)$. Define $\hat{v} \in (0, 1/4]$, $\rho \geq 0$ such that*

$$\max_{v\in V} Var_\gamma(f_v) \leq \hat{v} \text{ , and } \sum_{v\in V} b(v) \leq \rho.$$

Fix $\delta \in (0,1)$, $\varepsilon \in (0,1)$, and define the functions $g(x) = x(1-x)$ and $h(x) = (1+x)\ln(1+x) - x$ for $x \geq 0$. Let $\hat{x}_1$, $\hat{x}_2$, and $\hat{x}$ be

$$\hat{x}_1 = \inf\left\{x : \frac{1}{2} - \sqrt{\frac{\varepsilon}{3} - \frac{\varepsilon^2}{9}} \leq x \leq \frac{1}{2}, g(x)h\left(\frac{\varepsilon}{g(x)}\right) \leq 2\varepsilon^2\right\}, \quad \hat{x}_2 = \frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}}, \quad \hat{x} = \min\{\hat{x}_1, \hat{x}_2\}.$$

Let $\mathcal{S}$ be an i.i.d. sample of size $m \geq 1$ taken from $\mathcal{X}$ according to γ such that

$$m \geq \sup_{x\in(0,\hat{x}]} \left\{ \frac{\ln\left(\frac{2\rho}{x\delta}\right)}{g(x)h\left(\frac{\varepsilon}{g(x)}\right)} \right\}. \tag{3}$$

With probability $\geq 1-\delta$ over $\mathcal{S}$, it holds $\mathrm{D}(\mathcal{F}, \mathcal{S}) \leq \varepsilon$.

PROOF OF THM. 4.6. For a sample $\mathcal{S}$ of size m, define the events A and A_v as

$$A = \text{“}\exists v \in V : |b(v) - \tilde{b}(v)| > \varepsilon\text{”},$$
$$A_v = \text{“}|b(v) - \tilde{b}(v)| > \varepsilon\text{”}.$$

From a union bound, we have that

$$\Pr(A) = \Pr\left(\bigcup_{v\in V} A_v\right) \leq \sum_{v\in V} \Pr(A_v).$$

Then, through the application of Hoeffding's and Bennet's inequalities [14], Bathia and Davis inequality on variance [8] (which implies $Var(f_v) \leq g(b(v))$), and from the fact that $sup_{f_v\in\mathcal{F}} Var(f_v) \leq \hat{v}$, it holds, for all $v \in V$,

$$\Pr(A_v) \leq 2\min\left\{\exp\left(-2m\varepsilon^2\right), \exp\left(-m\min\{\hat{v}, g(b(v))\}h\left(\frac{\varepsilon}{\min\{\hat{v}, g(b(v))\}}\right)\right)\right\}.$$

By defining the functions $H(m,\varepsilon) = \exp\left(-2m\varepsilon^2\right)$, $B(x,m,\varepsilon) = \exp\left(-mxh\left(\frac{\varepsilon}{x}\right)\right)$, and $\psi(x)$ (see below), we rewrite

$$\Pr(A) \leq \sum_{v\in V} 2\min\left\{H(m,\varepsilon), B(\min\{\hat{v}, g(b(v))\}, m, \varepsilon)\right\} \doteq \sum_{v\in V} \psi(b(v)). \tag{11}$$

Since the values of $b(v)$ are not known a priori, it is not possible to directly compute the r.h.s. of (11). However, we show how to obtain a sharp upper bound by leveraging constraints on the possible values of $b(v)$ imposed by $\hat{v}$ and ρ. To do so, we define an appropriate optimization problem w.r.t. the (unknown) values of $b(v)$. Denote with m_x the number of nodes of V that we assume have $b(v) = x$, for $x \in \mathbb{Q} \cap (0,1)$ (we can safely ignore nodes v with $b(v) = 0$ or $b(v) = 1$, since f_v is

constant, and $\Pr(A_v) = 0$); then, we define the following constrained optimization problem over the variables m_x:

$$\max \sum_{x\in(0,1),m_x>0} m_x\psi(x),$$
$$\text{subject to} \sum_{x\in(0,1),m_x>0} xm_x \leq \rho,$$
$$0 \leq m_x \leq \frac{\rho}{x}, m_x \in \mathbb{N}.$$

The first contraint follows from $\sum_{v\in V} b(v) \leq \rho$, while the second set of constraints imposes that m_x are positive integers and that there cannot be more than ρ/x nodes with $b(v) = x$ by definition of ρ. Therefore, from (11), the value of the objective function of the optimal solution of this problem upper bounds $\Pr(A)$, as we consider a worst-case configuration of the admissible values of $b(v)$ (i.e., the graph G belongs to the space of all possible graphs with the above mentioned constraints). We recognize this formulation as a specific instance of the Bounded Knapsack Problem (BKP) [37] over the variables m_x, where items with label x are selected m_x times, with unitary profit $\psi(x)$ and weight x; each item can be selected at most ρ/x times, while the total knapsack capacity is ρ. We are not interested in the optimal solution of the integer problem, but rather in its upper bound given by the optimal solution of the continous relaxation, in which we let $m_x \in \mathbb{R}$. Informally, such solution is obtained by choosing at maximum possible capacity every item in decreasing order of profit-weight ratio $\psi(x)/x$ until the total capacity is filled (Chapter 3 of [37]). In our case, from the particular definition of the constraints, it is enough to fully select the item with higher profit-weight ratio to fill the entire knapsack. More formally, let $x^\star$ be

$$x^\star = \arg\max_{x\in(0,1)} \left\{ \frac{\psi(x)}{x} \right\};$$

the optimal solution to the continous relaxation is $m_{x^\star} = \rho/x^\star$, $m_x = 0, \forall x \neq x^\star$, while the optimal objective is equal to

$$\frac{\rho\psi(x^\star)}{x^\star} \geq \Pr(A).$$

We note that $x^\star$ always exists, as $\psi(x)/x$ is a positive, bounded and continous function in $(0, 1)$. We now simplify the search of $x^\star$ limiting the range of x. First, we prove that $x^\star \in (0, \hat{x}_2]$. Observe that the function $B(\min\{\hat{v}, g(x)\}, m, \varepsilon)$ (thus $\psi(x)$) is symmetric w.r.t. $1/2$ (since $g(x) = g(1-x)$); also, we note that

$$\min\{\hat{v}, g(x)\} = \begin{cases} \hat{v}, \ \frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}} \leq x \leq \frac{1}{2} + \sqrt{\frac{1}{4} - \hat{v}}, \\ g(x), \ \text{otherwise}, \end{cases}$$

which means that $\psi(x)$ is constant for $x \in [\frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}}, \frac{1}{2} + \sqrt{\frac{1}{4} - \hat{v}}]$. From these observation, we prove by contradiction that it holds that $x^\star \leq \frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}} = \hat{x}_2$. Assume that $x^\star > \frac{1}{2} + \sqrt{\frac{1}{4} - \hat{v}}$; then defining $x' = 1 - x^\star$ we have

$$\frac{\psi(x^\star)}{x^\star} = \frac{\psi(x')}{1 - x'} < \frac{\psi(x')}{x'},$$

which contradicts the definition of $x^\star$. Now, assume that $x^\star \in (\frac{1}{2} - \sqrt{\frac{1}{4} - \hat{v}}, \frac{1}{2} + \sqrt{\frac{1}{4} - \hat{v}}]$; we have

$$\frac{\psi(x^\star)}{x^\star} = \frac{\psi(\hat{x}_2)}{x^\star} < \frac{\psi(\hat{x}_2)}{\hat{x}_2},$$

which is another contradiction; therefore, we conclude that $x^\star \in (0, \hat{x}_2]$. We can write

$$x^\star = \arg\max_{x \in (0,1)} \left\{ \frac{\psi(x)}{x} \right\} = \arg\max_{x \in (0,\hat{x}_2]} \left\{ \frac{2\min\{H(m,\varepsilon), B(g(x),m,\varepsilon)\}}{x} \right\} \doteq \arg\max_{x \in (0,\hat{x}_2]} \left\{ \frac{\psi_1(x)}{x} \right\}.$$

We now prove that $x^\star \in (0, \hat{x}_1]$. Note that $\psi_1(x)$ is symmetric around $1/2$; furthermore,

$$\min\{H(m,\varepsilon), B(g(x),m,\varepsilon)\} = \begin{cases} B(g(x),m,\varepsilon), & g(x)h\left(\frac{\varepsilon}{g(x)}\right) \geq 2\varepsilon^2, \\ H(m,\varepsilon), & \text{otherwise.} \end{cases}$$

The function $g(x)h(\varepsilon/g(x))$ is monotonically increasing for $0 < x < 1/2$ and decreasing for $1/2 < x < 1$. Denote $h_1(x) = 1 + x - \sqrt{1+2x}$ for $x \geq 0$. We use the facts that $9h_1(x/3) \leq h(x) \leq x^2/2$ for all $x \geq 0$, so we have that $g(x)h\left(\varepsilon/g(x)\right) = 2\varepsilon^2$ holds for a pair x_1 and x_2 such that $1/2 - \sqrt{\varepsilon/3 - \varepsilon^2/9} \leq x_1 \leq 1/2$, $1/2 \leq x_2 \leq 1/2 + \sqrt{\varepsilon/3 - \varepsilon^2/9}$, and $1 = x_1 + x_2$ (this follows easily from knowing that the inverse of h_1 over $[0, +\infty)$ is $h_1^{-1}(x) = x + \sqrt{2x}$). We can write

$$\min\{H(m,\varepsilon), B(g(x),m,\varepsilon)\} = \begin{cases} B(g(x),m,\varepsilon), & x \in (0,x_1] \cup [x_2,1), \\ H(m,\varepsilon), & \text{otherwise,} \end{cases}$$

Exploting the symmetry of $\psi_1(x)$, it is easy to prove that $x^\star \in (0, x_1]$ by following a similar argument used above. Then, note that $x_1 \leq \hat{x}_1$, $\psi_1(x) \leq 2B(g(x),m,\varepsilon)$, and $\hat{x} = \min\{\hat{x}_1, \hat{x}_2\}$; we obtain

$$\Pr(A) \leq \sup_{x \in (0, \min\{x_1, \hat{x}_2\}]} \left\{ \frac{\rho\psi_1(x)}{x} \right\} \leq \sup_{x \in (0, \min\{\hat{x}_1, \hat{x}_2\}]} \left\{ \frac{\rho\psi_1(x)}{x} \right\} \leq \sup_{x \in (0,\hat{x}]} \left\{ \frac{\rho 2B(g(x),m,\varepsilon)}{x} \right\}.$$

We now show that if m is chosen as assumed in the statement, it holds $\Pr(A) \leq \delta$, thus proving the Theorem. We have, for $x \in (0, \hat{x}]$,

$$\frac{\rho 2B(g(x),m,\varepsilon)}{x} \leq \delta \text{ if } m \geq \frac{\ln\left(\frac{2\rho}{x\delta}\right)}{g(x)h\left(\frac{\varepsilon}{g(x)}\right)},$$

which follows from (3). □

Theorem 4.7. *Let $\mathcal{F}$, γ, g, $\hat{v}$, and ρ as in Theorem 4.6, and define $n = |V|$. Denote $\mathcal{S}$ as an i.i.d. sample of size $m \geq 1$ from γ. It holds, with probability $\geq 1 - \delta$ and for all $v \in V$,*

$$|b(v) - \tilde{b}(v)| \leq \sqrt{\frac{2\min\{g(b(v)), \hat{v}\} \ln\left(\frac{4}{\delta} \min\left\{\frac{\rho}{b(v)}, n\right\}\right)}{m}} + \frac{\ln\left(\frac{4}{\delta} \min\left\{\frac{\rho}{b(v)}, n\right\}\right)}{3m} \doteq d_r(b(v)).$$

Proof. Denote the event $A =$ "$\exists v : |b(v) - \tilde{b}(v)| > d_r(b(v))$", and the events $A_v =$ "$|b(v) - \tilde{b}(v)| > d_r(b(v))$". Our goal is to show that $\Pr(A) \leq \delta$, which proves the statement. First, through a union bound and Bennet's inequality we obtain that

$$\Pr(A) = \Pr\left(\bigcup_{v \in V} A_v\right) \leq \sum_{v \in V} \Pr(A_v) \leq \sum_{v \in V} 2B(\min\{\hat{v}, g(x)\}, m, d_r(b(v))), \tag{12}$$

with $B(x, m, y) = \exp(-mxh(y/x))$ and $h(x) = (1+x)\ln(1+x) - x$ for $x \geq 0$. We define an optimization problem to upper bound (12):

$$\begin{aligned} \max \quad & \sum_{x \in (0,1)} 2m_x B(\min\{\hat{v}, g(x)\}, m, d_r(x)), \\ \text{subject to} \quad & \sum_{x \in (0,1)} x m_x \leq \rho, \\ & \sum_{x \in (0,1)} m_x \leq n \doteq |V|, \\ & 0 \leq m_x \leq \frac{\rho}{x}, m_x \in \mathbb{R}. \end{aligned}$$

This problem is similar to the one introduced in the proof of Theorem 4.6, but with an additional constraint which imposes that the number of vertices $\sum_{x \in (0,1)} m_x$ with $b(v) \in (0,1)$ is at most n, where n is the number of vertices $|V|$ of the graph G (note that we ignore vertices with $b(v) = 0$ or $b(v) = 1$ since the corresponding estimator f_v is constant, therefore $\Pr(A_v) = 0$). We obtain an upper bound to the optimal solution by upper bounding the objective function and through a Lagrangean relaxation. Using the fact that $h(x) \geq 9h_1(x/3), \forall x \geq 0$, with $h_1(x) = 1 + x - \sqrt{1+2x}$, we define the function $\psi(x)$

$$\psi(x) \doteq 2\exp\left(-m\min\{g(x), \hat{v}\}\left(\frac{d_r(x)}{3\min\{g(x), \hat{v}\}}\right)\right) \geq 2B(\min\{g(x), \hat{v}\}, m, d_r(x)).$$

After straightforward simplifications, we observe from the definitions of $\psi(x)$ and $d_r(x)$ that

$$\psi(x) = \frac{\delta}{2\min\left\{\frac{\rho}{x}, n\right\}} = \begin{cases} \frac{x\delta}{2\rho}, x \geq \frac{\rho}{n}, \\ \frac{\delta}{2n}, x < \frac{\rho}{n}. \end{cases}$$

Fix any $\lambda \geq 0$; the optimal solution of the following problem upper bounds the optimal solution of the problem above:

$$\begin{aligned} \max \quad & \sum_{x \in (0,1)} m_x \psi(x) + \lambda\left(n - \sum_{x \in (0,1)} m_x\right), \\ \text{subject to} \quad & \sum_{x \in (0,1)} x m_x \leq \rho, \\ & 0 \leq m_x \leq \frac{\rho}{x}, m_x \in \mathbb{R}. \end{aligned}$$

Rewriting the objective function, we have

$$\sum_{x \in (0,1)} m_x \psi(x) + \lambda\left(n - \sum_{x \in (0,1)} m_x\right) = \sum_{x \in (0,1)} m_x\left(\psi(x) - \lambda\right) + \lambda n.$$

Ignoring the constant λn, we expressed the problem as another Bounded Knapsack Problem formulation with profit $(\psi(x) - \lambda)$ for items with label x; we compute its optimal solution as done in the proof of Theorem 4.6. We fix $\lambda = \psi(\rho/n) = \delta/(2n)$, and remark that $\psi(x)$ is increasing with x: it holds $\psi(x) \leq \psi(\rho/n), \forall x \in (0, \rho/n]$, and $\psi(x) \geq \psi(\rho/n), \forall x \in [\rho/n, 1)$; therefore, define $x^\star$ as

$$x^\star = \arg\max_{x \in (0,1)} \left\{\frac{\psi(x) - \psi(\rho/n)}{x}\right\};$$

it follows that $x^\star \geq \rho/n$ and $\psi(x^\star) \geq \psi(\rho/n)$ from observations made above. The optimal solution is given by $m_{x^\star} = \rho/x^\star, m_x = 0, \forall x \neq x^\star$, with objective

$$\frac{\rho}{x^\star}\psi(x^\star) + n\psi(\rho/n) - \frac{\rho}{x^\star}\psi(\rho/n) \leq \delta,$$

proving the statement. □

Before proving data-dependent upper bounds to ρ, we show the following straightforward fact.

Lemma A.6. *Let D be the vertex diameter of a graph G. Then $\rho \leq D$.*

Proof. From the definition of ρ and from linearity of expectation, we have

$$\begin{aligned}\rho &= \sum_{v\in V} b(v) = \sum_{v\in V} \mathbb{E}_{\mathcal{S}}[\tilde{b}(v)] = \sum_{v\in V} \mathbb{E}_{\mathcal{S}}\left[\frac{1}{m}\sum_{\pi\in\mathcal{S}} \tilde{b}_v(\pi)\right] \\ &= \frac{1}{m}\mathbb{E}_{\mathcal{S}}\left[\sum_{\pi\in\mathcal{S}}\sum_{v\in V} \tilde{b}_v(\pi)\right] \leq \frac{1}{m}\mathbb{E}_{\mathcal{S}}\left[\sum_{\pi\in\mathcal{S}} D\right] \\ &= \frac{1}{m}\mathbb{E}_{\mathcal{S}}\left[mD\right] = D.\end{aligned}$$

□

The result below shows that given a (not necessarily tight) upper bound to D, the average shortest path length ρ can be sharply estimated as the *average* number of internal nodes of the shortest paths in a sample $\mathcal{S}$, resulting in a very efficient data-dependent bound.

Proposition 4.9. *Let D be the vertex diameter of the graph G. Let an i.i.d. sample $\mathcal{S}$ of size m, and denote $\tilde{\rho} = \sum_{v\in V} \tilde{b}(v)$. Then, for a fixed $\delta \in (0,1)$, it holds with probability $\geq 1-\delta$*

$$\sum_{v\in V} b(v) \leq \rho \doteq \tilde{\rho} + \sqrt{\frac{5}{3}\left(\frac{D\ln\left(\frac{1}{\delta}\right)}{m}\right)^2 + \frac{2D\tilde{\rho}\ln\left(\frac{1}{\delta}\right)}{m}} + \frac{4D\ln\left(\frac{1}{\delta}\right)}{3m}.$$

Proof. From the definition of $\tilde{\rho}$, we have

$$\tilde{\rho} = \sum_{v\in V}\frac{1}{m}\sum_{\pi\in\mathcal{S}} \tilde{b}_v(\pi) = \frac{1}{m}\sum_{\pi\in\mathcal{S}}\sum_{v\in V} \tilde{b}_v(\pi),$$

with $\mathbb{E}_{\mathcal{S}}[\tilde{\rho}] = \rho$. We recognize $\tilde{\rho}$ as an average of m independent (bounded) random variables; it holds, for all π, that $0 \leq \sum_{v\in V} \tilde{b}_v(\pi) \leq D$, and, from [8], $Var\left(\sum_{v\in V} \tilde{b}_v(\pi)\right) \leq (D-\rho)\rho \leq D\rho$. We define the random variables X_i, for $i \in [1,m]$, as

$$X_i = \rho - \sum_{v\in V} \tilde{b}_v(\pi_i),$$

noting that $\mathbb{E}[X_i] = 0$, $X_i \leq \rho \leq D$, and $\mathbb{E}[X_i^2] = Var(X_i) \leq D\rho$. Therefore, we apply Bernstein's inequality (Thm. 2.10 [14]) to the sum $\sum_{i=1}^m X_i = m(\rho - \tilde{\rho})$, obtaining

$$\Pr\left(\rho \geq \tilde{\rho} + \sqrt{\frac{2\ln\left(\frac{1}{\delta}\right)\rho D}{m}} + \frac{\ln\left(\frac{1}{\delta}\right)D}{3m}\right) \leq \delta.$$

To conclude the proof, we need the following straightforward Lemma.

Lemma A.7. *Let $u, v, y \geq 0$. The fixed point of*

$$r(x) = u + \sqrt{v + yx}$$

is at

$$x = u + \frac{y}{2} + \sqrt{\frac{y^2}{4} + uy + v}.$$

The statement follows by applying Lemma A.7 to the inequality

$$\rho \leq \tilde{\rho} + \sqrt{\frac{2 \ln\left(\frac{1}{\delta}\right) \rho D}{m}} + \frac{\ln\left(\frac{1}{\delta}\right) D}{3m}.$$

□

Proposition 4.10. *Assume the setting of Proposition 4.9 with $\mathcal{S} = \{\tau_1, \ldots, \tau_m\}$. Define $\Lambda(\mathcal{S})$ as*

$$\Lambda(\mathcal{S}) = \frac{1}{m(m-1)} \sum_{1 \leq i < j \leq m} \left(\sum_{v \in V} f_v(\tau_i) - \sum_{v \in V} f_v(\tau_j) \right)^2.$$

Then, for a fixed $\delta \in (0, 1)$, it holds with probability $\geq 1 - \delta$

$$\rho \leq \tilde{\rho} + \sqrt{\frac{2\Lambda(\mathcal{S}) \ln\left(\frac{2}{\delta}\right)}{m}} + \frac{7D \ln\left(\frac{2}{\delta}\right)}{3m}.$$

Proof. As discussed in the proof of Proposition 4.9, $\tilde{\rho}$ is an average of m i.i.d. (bounded below by D) random variables. The result follows from Corollary 5 of [39] after scaling $\tilde{\rho}$ by $1/D$. □

A.4 Proofs of Section 4.4

Proof of Proposition 4.12. We first note that, as the k-th most central node v_k and its centrality $b(v_k)$ are both unknown, we need a principled way to identify *bounds* to $b(v_k)$. Let $v_1^\ell, \ldots, v_n^\ell$ be the sequence of nodes ordered according to $\ell(\cdot)$, such that $\ell(v_i^\ell) \geq \ell(v_{i+1}^\ell)$. Then, let $v_1^u, \ldots, v_n^u$ be the sequence of nodes ordered according to $u(\cdot)$, such that $u(v_i^u) \geq u(v_{i+1}^u)$. We have the following relations between $b(v_k)$, $u(v_k^u)$, and $\ell(v_k^\ell)$.

Lemma A.8. *It holds $\ell(v_k^\ell) \leq b(v_k) \leq u(v_k^u)$.*

Proof. We prove the statement by contradiction. Assume that it holds $b(v_k) < \ell(v_k^\ell)$. From the definition of $\ell(v_k^\ell)$, that there are k nodes $\{v_i^\ell, i \leq k\}$ such that $l(v_i^\ell) \geq \ell(v_k^\ell) > b(v_k)$, and it holds, for all of them, that $b(v_i^\ell) \geq \ell(v_i^\ell)$. This implies that there are k nodes such that $b(v_i^\ell) > b(v_k)$, that is in contradiction with the definition of v_k. □

We now continue proving the Proposition. The first guarantee is immediate from Lemma A.8: since it holds for all $v \in V$, that $b(v) \in CI_v$, we have that $u(v) \geq b(v) \geq \ell(v)$. Therefore, if $v \in TOP(k)$, then $b(v) \geq b(v_k)$, thus we have $u(v) \geq b(v) \geq b(v_k) \geq \ell(v_k^\ell)$; this means that v is in $\tilde{TOP}(k)$ as $u(v) \geq \ell(v_k^\ell)$. The second follows directly from (7). We now focus on the third. Let $\tilde{v}_1, \ldots, \tilde{v}_k$ be the sequence of nodes in order of $\tilde{b}(\cdot)$, such that $\tilde{b}(\tilde{v}_i) \geq \tilde{b}(\tilde{v}_{i+1})$. From (7) we have that, for all $\tilde{v}_i$,

$$\frac{\tilde{b}(\tilde{v}_i)}{1+\eta} \leq \ell(\tilde{v}_i) \leq u(\tilde{v}_i) \leq \frac{\tilde{b}(\tilde{v}_i)}{1-\eta},$$

but also that

$$\frac{\tilde{b}(\tilde{v}_{i+1})}{1+\eta} \leq \frac{\tilde{b}(\tilde{v}_i)}{1+\eta}, \text{ and } \frac{\tilde{b}(\tilde{v}_{i+1})}{1-\eta} \leq \frac{\tilde{b}(\tilde{v}_i)}{1-\eta}.$$

Notice that $v_k^\ell \neq v_k^u$ in general; nevertheless, we show it is possible to bound $u(v_k^u) - \ell(v_k^\ell)$. Considering $\tilde{v}_k$, we have that

$$\frac{\tilde{b}(\tilde{v}_k)}{1+\eta} \leq \ell(v_k^\ell) \leq u(v_k^u) \leq \frac{\tilde{b}(\tilde{v}_k)}{1-\eta};$$

the lower bound to $\ell(v_k^\ell)$ follows from the definition of v_k^ℓ, since there are k values $\ell(\tilde{v}_i) \geq \tilde{b}(\tilde{v}_i)/(1+\eta), i \geq k$; the upper bound to $u(v_k^u)$ follows from the fact that there are k values $u(\tilde{v}_i) \leq \tilde{b}(\tilde{v}_i)/(1-\eta), i \geq k$. We combine previous inequalities to obtain

$$b(v) \geq l(v) \geq \frac{\tilde{b}(v)}{1+\eta} \geq u(v)\frac{1-\eta}{1+\eta} \geq \ell(v_k^\ell)\frac{1-\eta}{1+\eta} \geq b(v_k)\left(\frac{1-\eta}{1+\eta}\right)^2.$$

□

A.5 Additional Experimental Results

This Section presents additional experimental results.

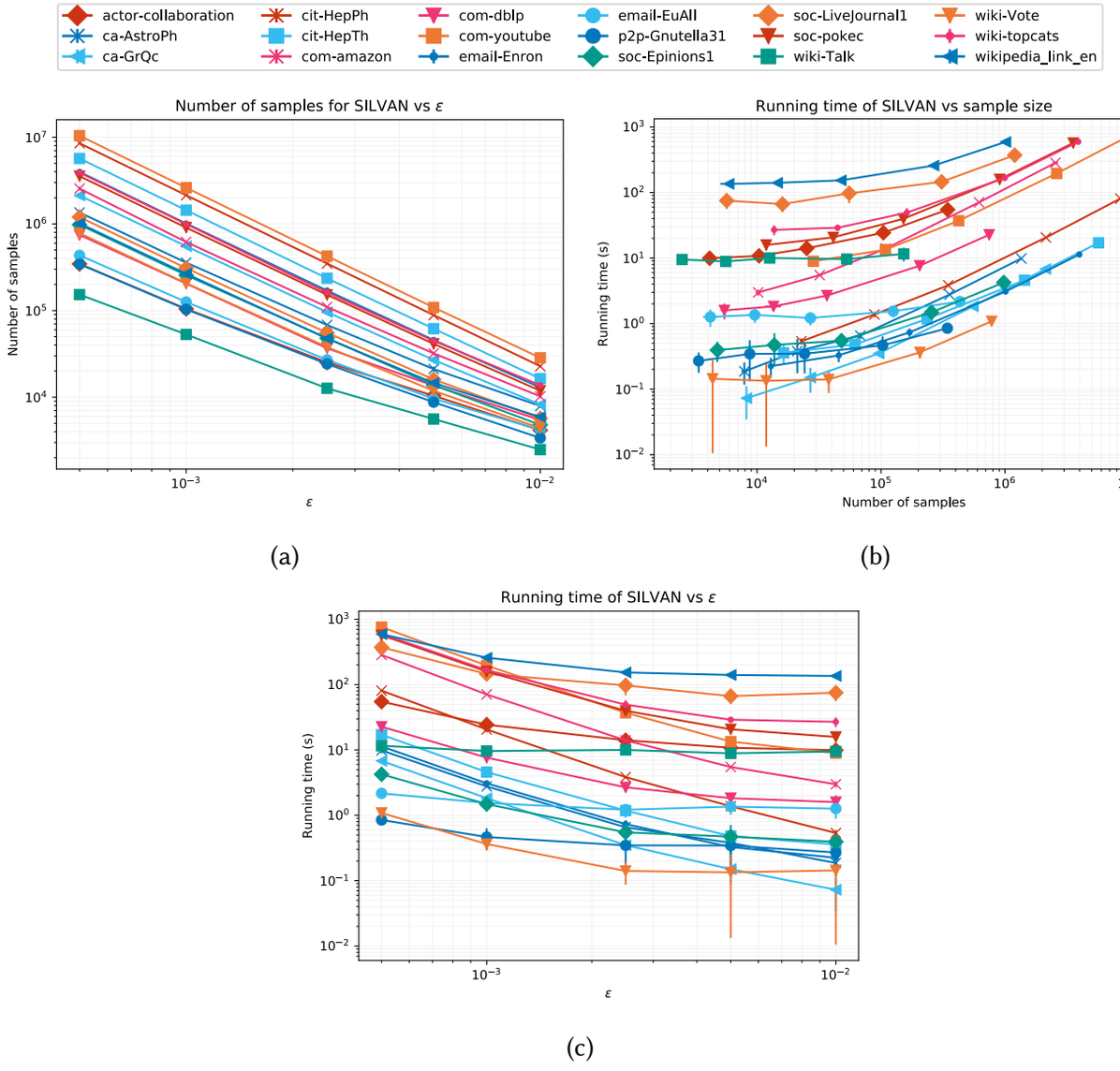


Fig. 4. Resources required by SILVAN for obtaining absolute ε approximations. (a): Number of samples for SILVAN vs. ε. (b): Running times of SILVAN vs. Number of samples. (c): Running times of SILVAN vs. ε.

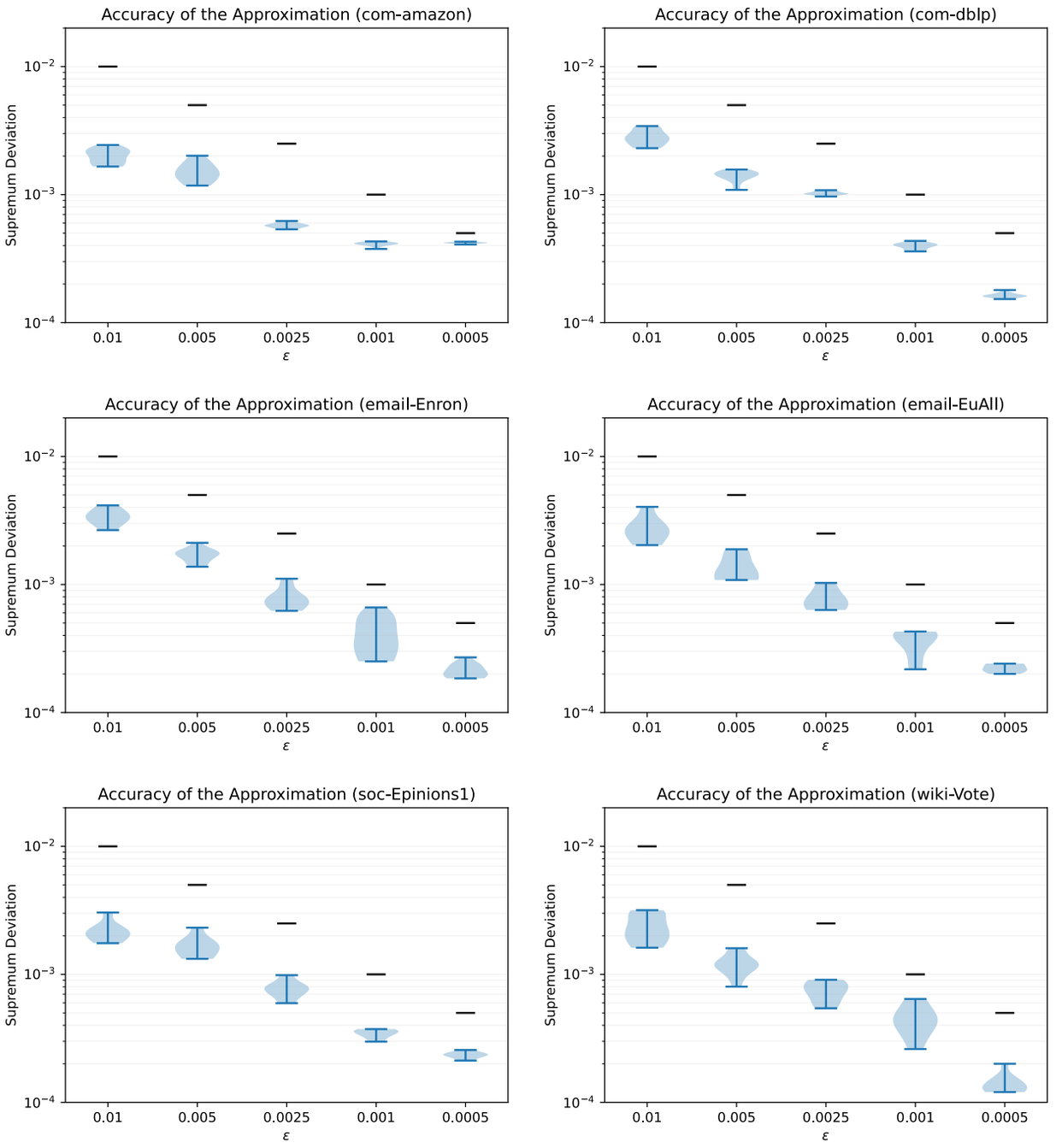


Fig. 5. Violin plot showing the empirical distribution of the supremum deviation $\mathrm{D}(\mathcal{F}, \mathcal{S})$ (blue violins) as function of ε (x axis, and black horizontal bars) for 3 undirected and 3 directed graphs over 10 runs.

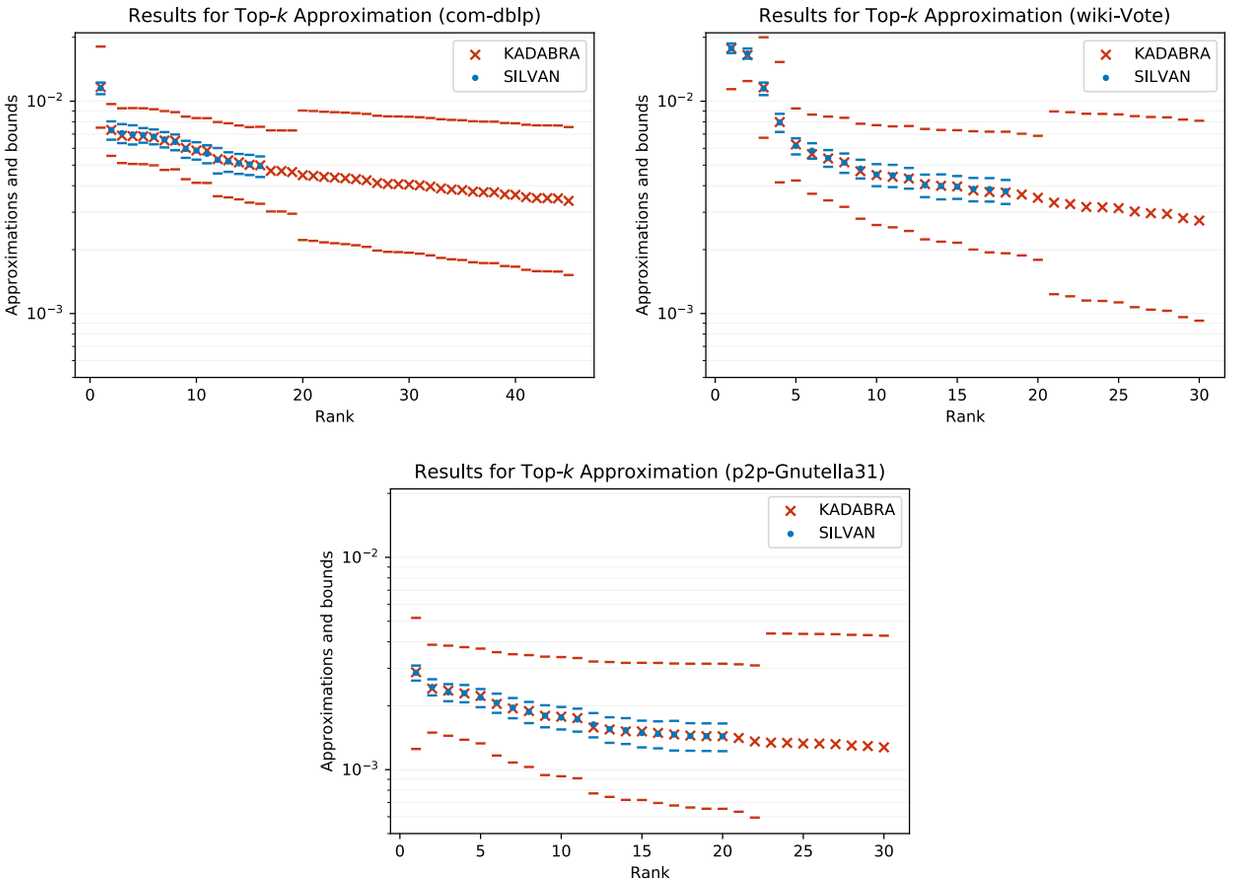


Fig. 6. Comparison of the quality of the output of SILVAN-TopK and KADABRA for top-k approximation for 3 graphs (with $k = 10$ and $\eta = 0.1$) when using the same resources: KADABRA is stopped after processing the same number of samples required by SILVAN-TopK to stop. The x axis is the rank of the node reported in output by both algorithms, the y axis shows the estimated values $\tilde{b}$ of the betweenness centrality (dots and crosses) and upper and lower bounds w.r.t. the exact values b (horizontal bars).

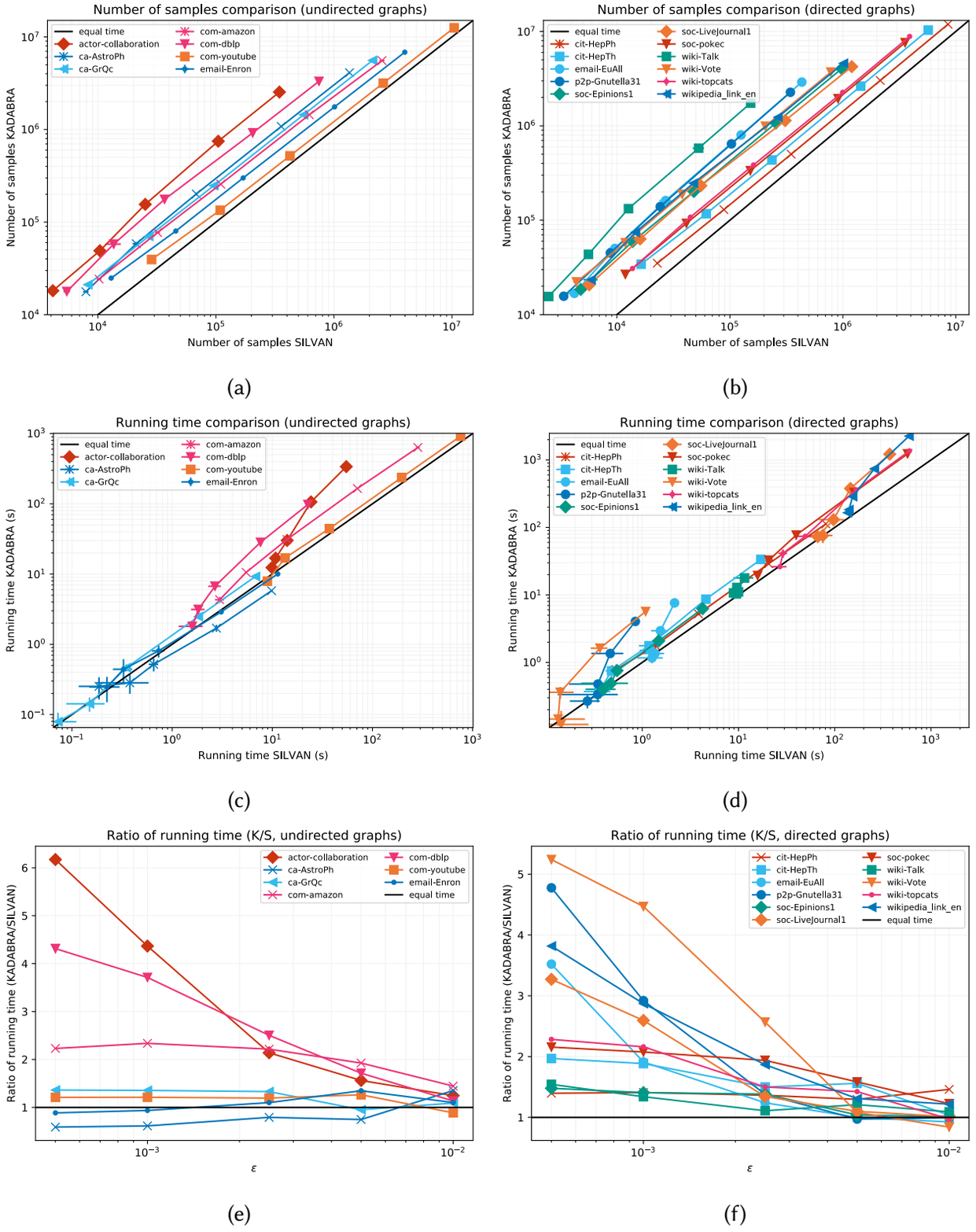


Fig. 7. Additional figures for comparing the performance of KADABRA and SILVAN for obtaining an absolute ε approximation. (a): comparison of the number of samples for KADABRA (y axis) and SILVAN (x axis) for undirected graphs. (b): analogous of (a) for directed graphs. (c): comparison of the running times of KADABRA (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (e): analogous of (d) for directed graphs. (c): ratios of the running times of KADABRA and SILVAN for undirected graphs. (d): analogous of (c) for directed graphs.

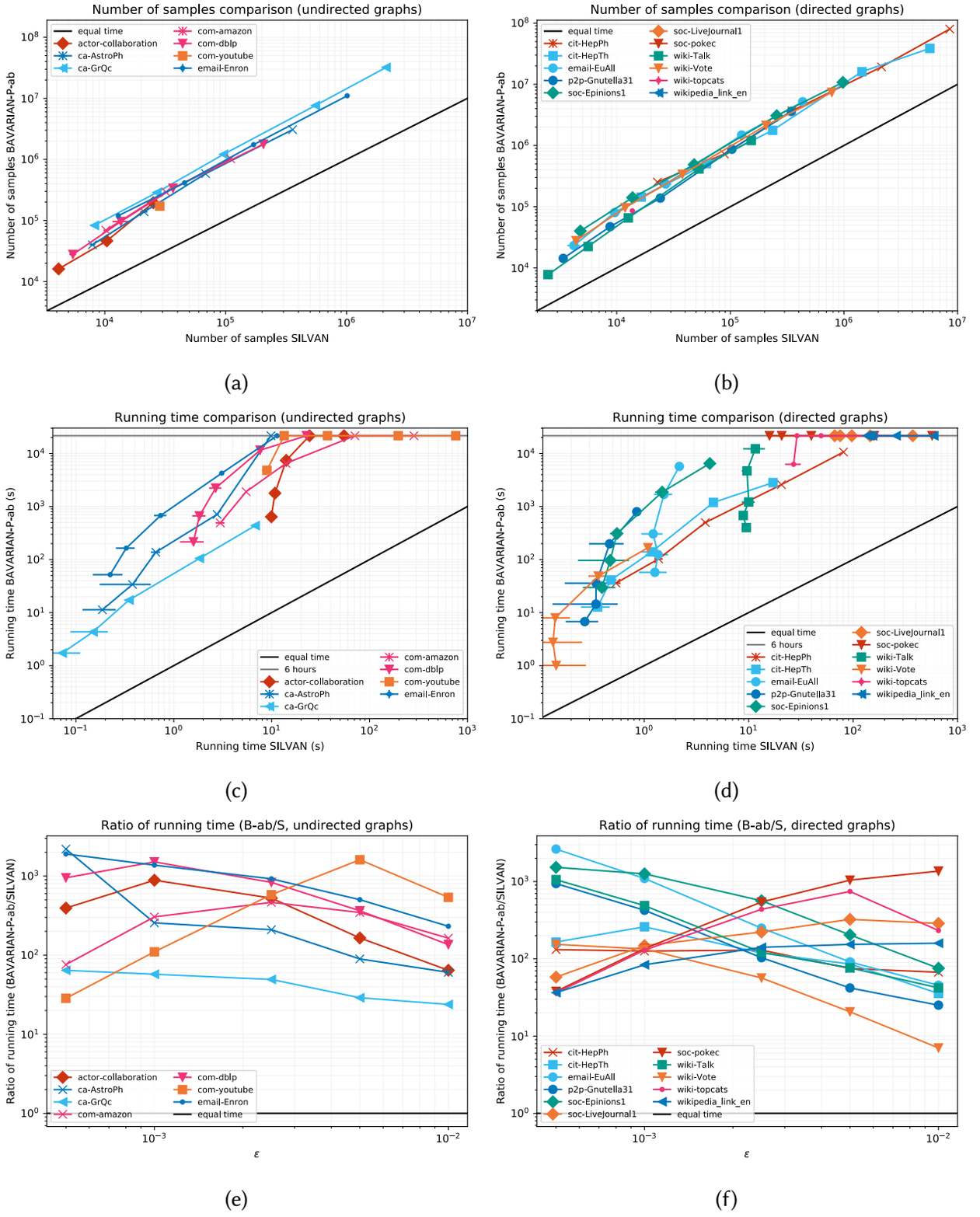


Fig. 8. Additional figures for comparing the performance of KADABRA and BAVARIAN-P (ab estimator) for obtaining an absolute ε approximation. (a): comparison of the number of samples for BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (b): analogous of (a) for directed graphs. (c): comparison of the running times of BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (d): analogous of (c) for directed graphs. (e): ratios of the running times of BAVARIAN and SILVAN for undirected graphs. (f): analogous of (e) for directed graphs.

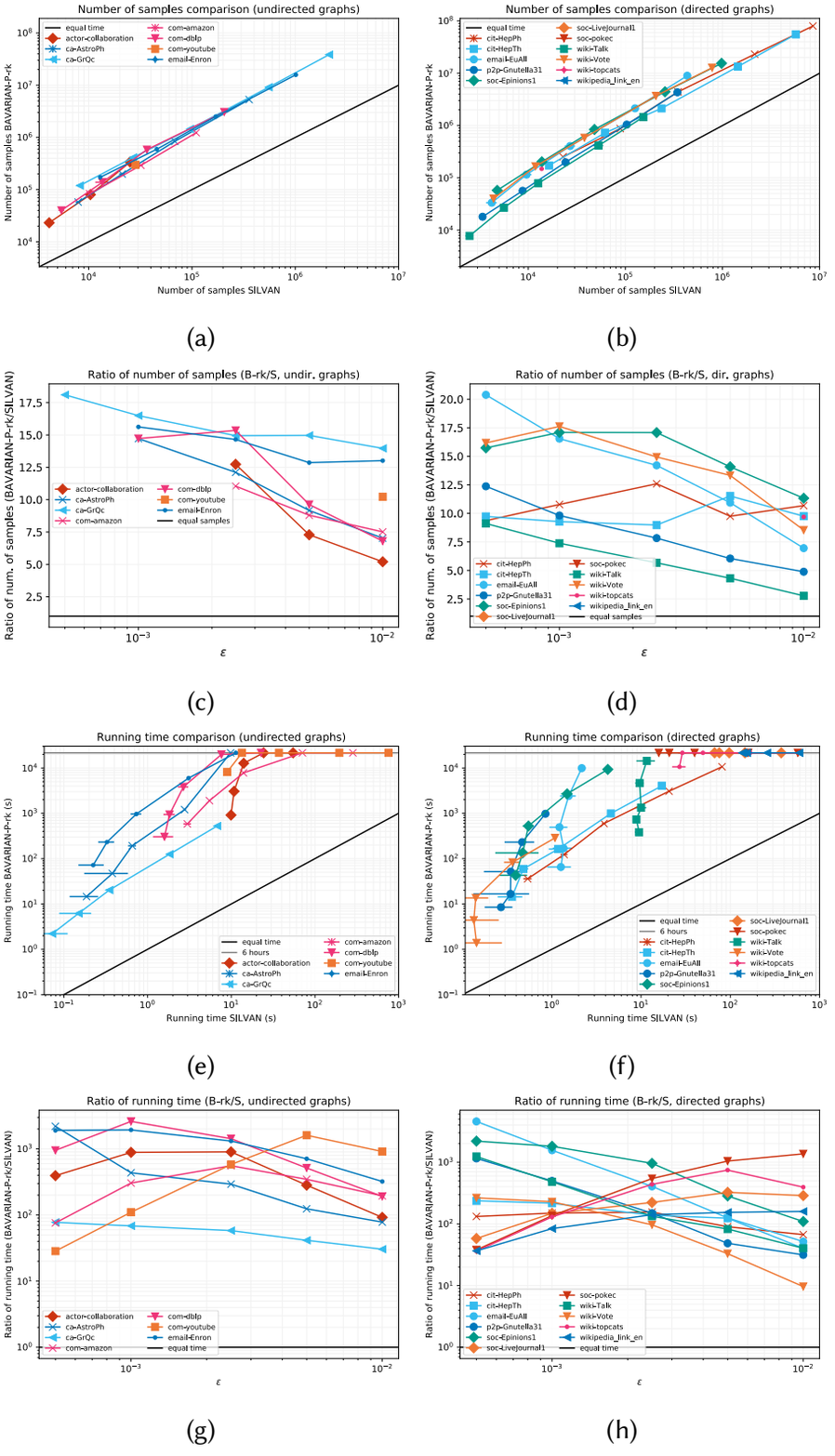


Fig. 9. Figures comparing the performance of KADABRA and BAVARIAN-P (rk estimator) for obtaining an absolute ε approximation. (a): comparison of the number of samples for BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (b): analogous of (a) for directed graphs. (c): ratios of the number of samples for BAVARIAN and SILVAN for undirected graphs. (d): analogous of (c) for directed graphs. (e): comparison of the running times of BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (f): analogous of (e) for directed graphs. (g): ratios of the running times of BAVARIAN and SILVAN for undirected graphs. (h): analogous of (g) for directed graphs.

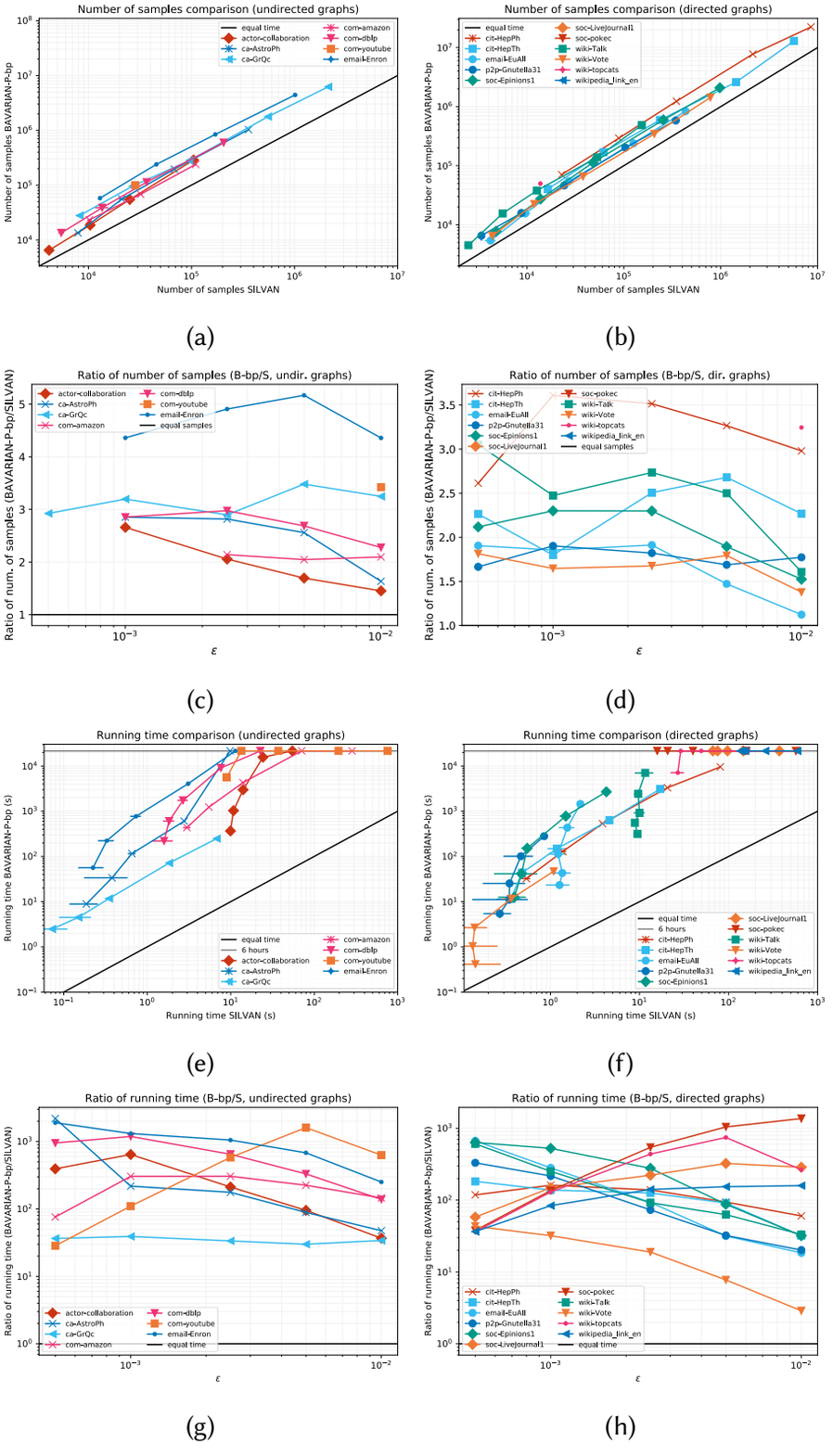


Fig. 10. Figures comparing the performance of KADABRA and BAVARIAN-P (bp estimator) for obtaining an absolute ε approximation. (a): comparison of the number of samples for BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (b): analogous of (a) for directed graphs. (c): ratios of the number of samples for BAVARIAN and SILVAN for undirected graphs. (d): analogous of (c) for directed graphs. (e): comparison of the running times of BAVARIAN (y axis) and SILVAN (x axis) for undirected graphs (axes in logarithmic scales). (f): analogous of (e) for directed graphs. (g): ratios of the running times of BAVARIAN and SILVAN for undirected graphs. (h): analogous of (g) for directed graphs.

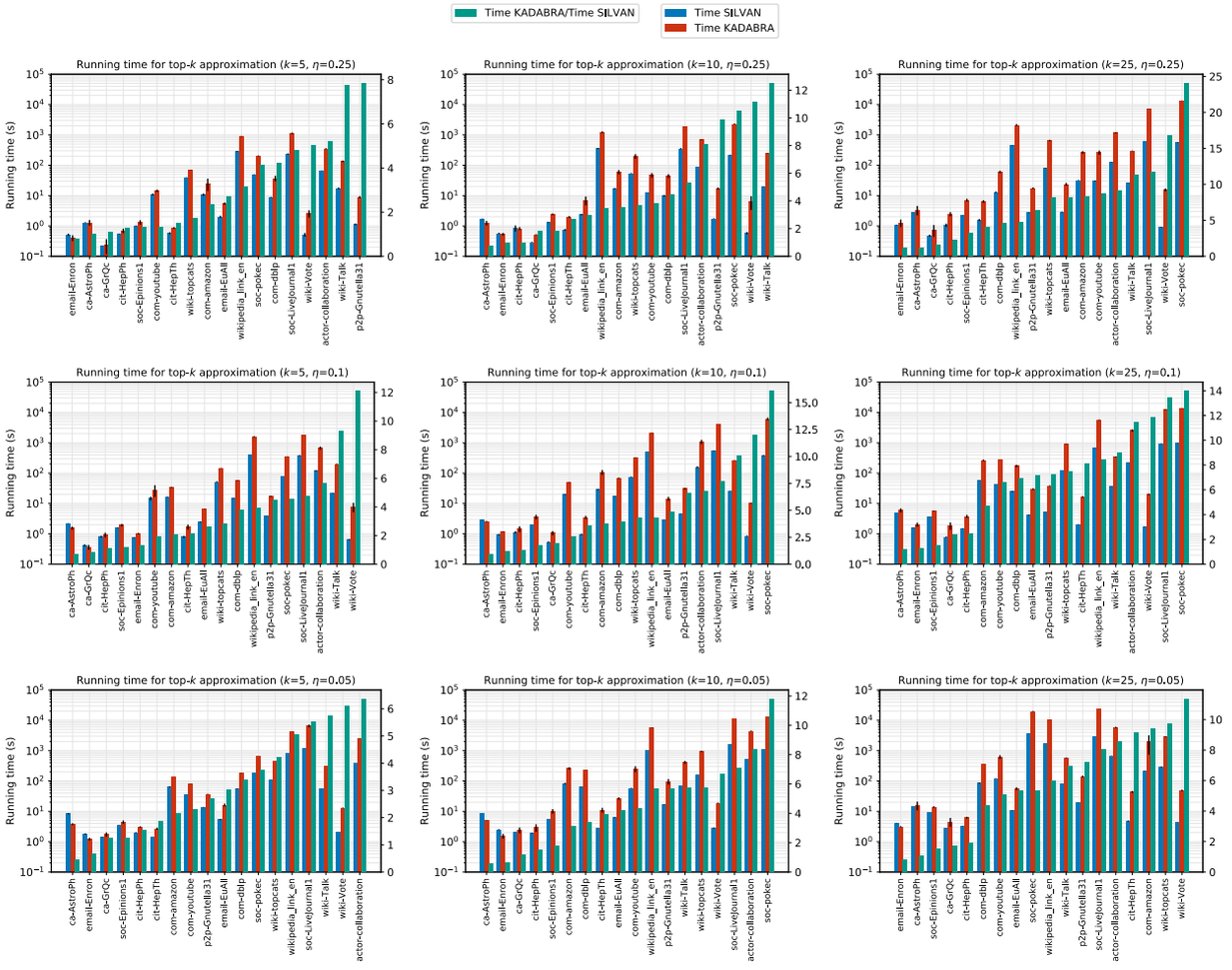


Fig. 11. Comparison of running times of SILVAN-TopK with KADABRA for obtaining top-k approximations for all combinations of k and η.

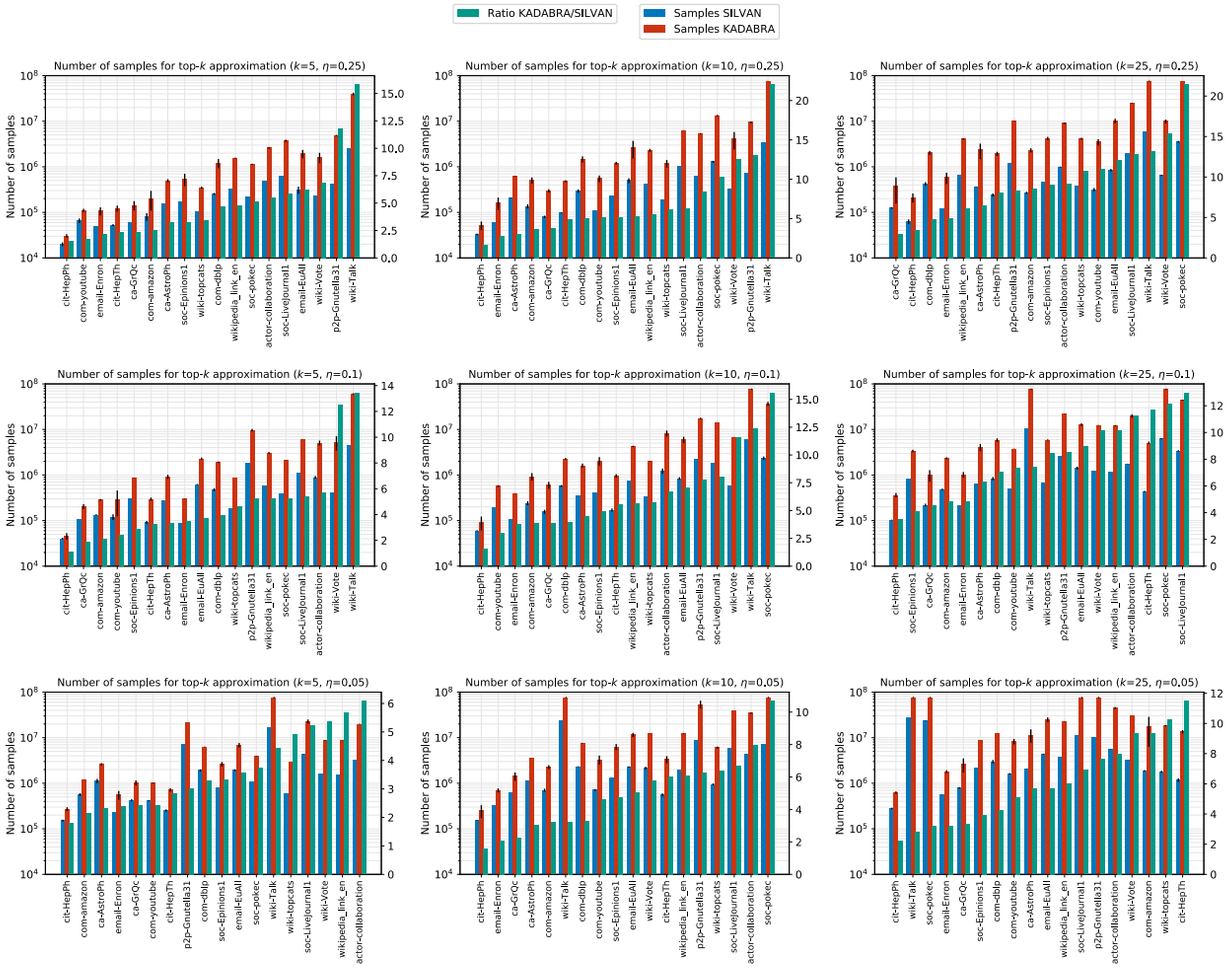


Fig. 12. Comparison of number of samples of SILVAN-TopK with KADABRA for obtaining top-k approximations for all combinations of k and η.

The VLDB Journal (2020) 29:1501–1525
https://doi.org/10.1007/s00778-020-00624-7

REGULAR PAPER

Temporal locality-aware sampling for accurate triangle counting in real graph streams

Dongjin Lee[1] · **Kijung Shin**[2] · **Christos Faloutsos**[3]

Received: 11 November 2019 / Revised: 25 April 2020 / Accepted: 28 July 2020 / Published online: 12 August 2020

Abstract

If we cannot store all edges in a dynamic graph, which edges should we store to estimate the triangle count accurately? Counting triangles (i.e., cliques of size three) is a fundamental graph problem with many applications in social network analysis, web mining, anomaly detection, etc. Recently, much effort has been made to accurately estimate the counts of global triangles (i.e., all triangles) and local triangles (i.e., all triangle incident to each node) in large dynamic graphs, especially with limited space. Although existing algorithms use sampling techniques without considering temporal dependencies in edges, we observe *temporal locality* in the formation of triangles in real dynamic graphs. That is, future edges are more likely to form triangles with recent edges than with older edges. In this work, we propose a family of single-pass streaming algorithms called *Waiting-Room Sampling* (WRS) for estimating the counts of global and local triangles in a fully dynamic graph, where edges are inserted and deleted over time, within a fixed memory budget. WRS exploits the temporal locality by always storing the most recent edges, which future edges are more likely to form triangles with, in the *waiting room*, while it uses reservoir sampling and its variant for the remaining edges. Our theoretical and empirical analyses show that WRS is: **(a) Fast and 'any time':** runs in linear time, always maintaining and updating estimates, while the input graph evolves, **(b) Effective**: yields up to *47% smaller estimation error* than its best competitors, and **(c) Theoretically sound**: gives unbiased estimates with small variances under the temporal locality.

Keywords Triangle counting · Graph stream · Waiting-room sampling · Temporal locality

1 Introduction

Consider a large dynamic graph where edges are inserted and deleted over time. If we cannot store every edge in memory, which edges should we store to estimate the count of triangles accurately?

Counting the triangles (i.e., cliques of size three) in a graph is a fundamental problem with many applications. For example, triangles in social networks have received much attention as an evidence of homophily (i.e., people choose friends similar to themselves) [28] and transitivity (i.e., people with common friends become friends) [51]. Thus, many concepts in social network analysis, such as social balance [51], the global/local clustering coefficient [52], and the transitivity ratio [30], are based on the triangle count. Moreover, the count of triangles has been used for spam and anomaly detection [6,27,40], web-structure analysis [9], degeneracy estimation [36], and query optimization [4].

Due to the importance of triangle counting, numerous algorithms have been developed in many different settings, including multi-core [20,41], external-memory [16,20], distributed-memory [3], and MapReduce [31,32,43] settings. The algorithms aim to accurately and rapidly count *global triangles* (i.e., all triangles in a graph) and/or *local triangles* (i.e., triangles that each node is involved with).

✉ Kijung Shin
kijungs@kaist.ac.kr

Dongjin Lee
dongjin.lee@kaist.ac.kr

Christos Faloutsos
christos@cs.cmu.edu

[1] School of Electrical Engineering, KAIST, Daejeon 34141, South Korea

[2] Graduate School of AI & School of Electrical Engineering, KAIST, Daejeon 34141, South Korea

[3] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

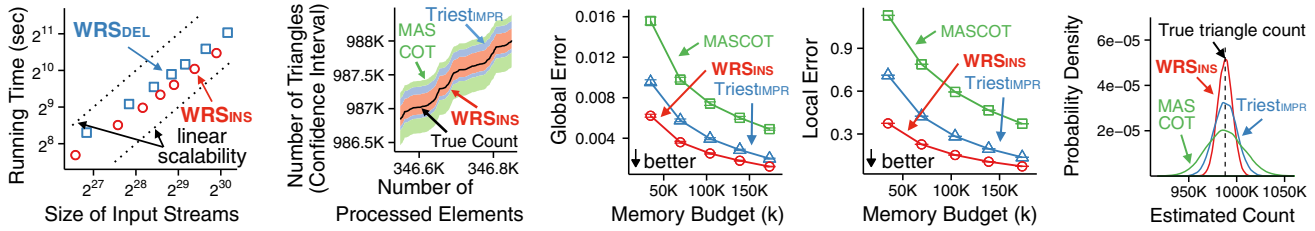


Fig. 1 Strengths of WRS. **a** WRS scales linearly with the size of the input stream. **b** WRS always maintains the estimates of the triangle counts while the input graph evolves. **c** WRS is more accurate than state-of-the-art streaming algorithms in both global and local triangle counting. **d** WRS gives unbiased estimates (Theorems 1 and 2) with small variances (Lemma 5)

Especially, as many real graphs, including social media and web, evolve over time, recent work has focused largely on streaming settings where graphs are given as a sequence of edge insertions and deletions. To accurately estimate the count of the triangles in large graph streams not fitting in memory, a number of sampling-based algorithms have been developed, as summarized in Table 1.

Notably, many real-world graphs exhibit *temporal locality* in triangle formation, i.e., the tendency that future edges are more likely to form triangles with recent edges than with older edges (see Sect. 4 for details). However, existing streaming algorithms sample edges without considering temporal dependencies between edges, and thus, they cannot exploit the temporal locality in triangle formation.

How can we exploit temporal locality for accurately estimating global and local triangle counts? As an answer to this question, we propose *Waiting-Room Sampling* (WRS),[1] a family of single-pass streaming algorithms for estimating the counts of global and local triangles in a fully dynamic graph stream, within a fixed memory budget. WRS always stores the most recent edges in the *waiting room*, while using standard reservoir sampling [49] and its variant, namely random pairing [13], for the remaining edges. Due to the temporal locality, when a new edge arrives, recent edges, which the new edge is likely to from triangles with, are always stored in the waiting room. Thus, the waiting room enables WRS to discover more triangles, which are useful for estimating the triangle counts more accurately, while reservoir sampling and random pairing enable WRS to give unbiased estimates.

Our theoretical and empirical analyses show that WRS has the following strengths:

- **Fast and 'any time'**: WRS scales linearly with the size of the input stream, and it gives the estimates of triangle counts at any time, not only at the end of the input stream (Fig. 1a, b).
- **Effective**: WRS produces up to *47% smaller* estimation error than its best competitors (Fig. 1c).
- **Theoretically sound**: we prove the unbiasedness of the estimates given by WRS and their small variances under temporal locality (Theorems 1 and 2; Fig. 1d).

Reproducibility The code and datasets used in the paper are available at http://dmlab.kaist.ac.kr/wrs/.

This paper is organized as follows. In Sect. 2, we review related work. In Sect. 3, we introduce notations and the problem definition. In Sect. 4, we discuss temporal locality in real graph streams. In Sect. 5, we propose our algorithm WRS and analyze its theoretical properties. After sharing experimental results in Sect. 6, we conclude in Sect. 7.

2 Related work

We discuss previous work on counting global and local triangles in a graph. After we briefly discuss counting triangles in a static graph, we focus on counting triangles with limited space in a graph stream, where edges arrive as a data stream. See Table 1 for a summary of the state-of-the-art streaming algorithms.

Triangle counting in static graphs There has been considerable interest in counting triangles in a static graph, and there have been numerous algorithms working in multi-core [20,41], external-memory [16,20], distributed-memory [3], and MapReduce [31,32,43] settings. While most of them are exact, several of them are approximate based on eigenvalues [44], sampled edges [3,11,19,41], and sampled wedges (i.e., paths of length 2) [34,46,47]. Notably a state-of-the-art method [46] estimates the global triangle count by sampling wedges after eliminating those that are less likely to participate in triangles. These approaches for static graphs are not directly applicable to graph streams, where edges should be processed in the order that they arrive.

[1] A preliminary version of WRS for insertion-only graph streams was presented in [35]. This work is an extended version of [35] with (a) a new algorithm for fully dynamic graph streams, (b) theoretical analyses on its accuracy and complexity, and (c) additional experiments with more datasets, competitors, and evaluation metrics.

Table 1 Comparison of state-of-the-art streaming algorithms for triangle counting

	WRS (Proposed)	THINKD [38,40]	TRIÈST$_{FD}$ [8]	ESD [15]	TRIÈST$_{IMPR}$ [8]	MASCOT [27]	Others [1,2,33]
Count Global Triangles	✓	✓	✓	✓	✓	✓	✓
Count Local Triangles	✓	✓	✓	✗	✓	✗	✗
Handling Large Graphs*	✓	✓	✓	✗	✓	✓	✓
Handle Edge Additions	✓	✓	✓	✓	✓	✓	✓
Handle Edge Deletions	✓	✓	✓	✓	✗	✗	✗
Exploit Temporal Patterns	✓	✗	✗	✗	✗	✗	✗

By exploiting temporal patterns, WRS achieves the best accuracy WRS also satisfies all other considered criteria
*Graphs that are too large to fit in memory

Global triangle counting in insertion-only graph streams For estimating the count of global triangles (i.e., all triangles in a graph), Tsourakakis et al. [45] proposed sampling each edge i.i.d. with probability p. Then, the global triangle count is estimated simply by multiplying that in the sampled graph by p^{-3}. Jha et al. [17] and Pavan et al. [33] proposed sampling wedges (i.e., paths of length 2) instead of edges for better space efficiency. Ahmed et al. [1] proposed an edge-sampling method where edges are sampled with different probabilities depending on whether an adjacency (i.e., an edge sharing a node) of them has been sampled or not. Ahmed et al. [2] proposed a priority-based edge sampling method where the priority is proportional to the number of the adjacencies of each edge or the number of triangles created by the edge.

Global triangle counting in fully dynamic graph streams Kutzkov and Pagh [25] combined edge and wedge sampling methods for global triangle counting in fully dynamic graph streams. For the same problem, Han and Sethu [15] proposed an incremental algorithm, which, however, requires the entire graph to be maintained in memory.

Local triangle counting in insertion-only graph streams For estimating the count of local triangles (i.e., triangles with each node), Lim and Kang [27] proposed MASCOT which samples each edge i.i.d with a fixed probability p but updating global and local counts whenever an edge arrives, even when the edge is not sampled. To properly set p, however, the number of edges in input streams should be known in advance. Likewise, randomly coloring nodes to sample the edges connecting nodes of the same color, as suggested by Kutzkov and Pagh [24], requires the number of nodes in advance to decide the number of colors. De Stefani et al. [8] proposed TRIÈST$_{IMPR}$ to address this problem using reservoir sampling [49], which fully utilizes given memory space, without requiring any prior knowledge of the input stream. Recently, Jung et al. [18] and Wang et al. [50] proposed streaming algorithms for counting global and local triangles, respectively, in a multigraph stream, which may have duplicated edges. Shin et al. [37,39] proposed distributed algorithms for triangle counting in a graph stream where edges are streamed from multiple sources. The streamed edges are processed and sampled across multiple workers.

Local triangle counting in fully dynamic graph streams De Stefani et al. [8] proposed TRIÈST$_{FD}$ for global and local triangle counting in fully dynamic graph streams. TRIÈST$_{FD}$ samples edges using random pairing [13], a variant of reservoir sampling for fully dynamic streams, without requiring any prior knowledge of the input stream. While TRIÈST$_{FD}$ simply discards unsampled edges, THINKD, proposed by Shin et al. [38,40] for the same problem, fully utilizes unsampled edges to update estimates before discarding them. Specifically, in response to each arrived change in the input stream, THINKD first updates corresponding estimates, which are shown to be unbiased, and then update the sampled graph

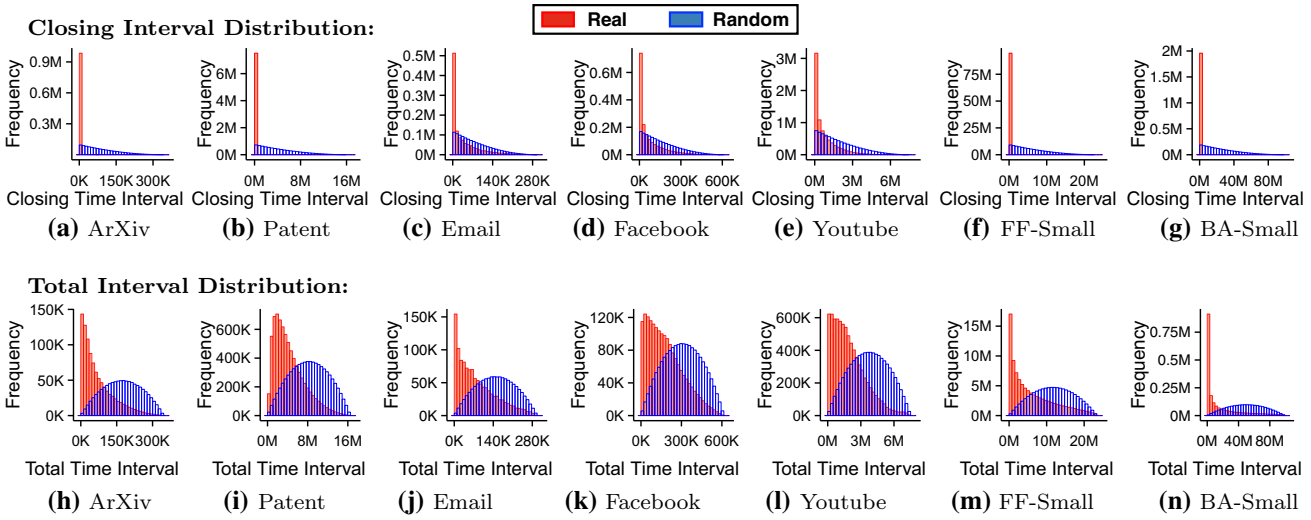


Fig. 2 Temporal locality in the formation of triangles in real graph streams. Closing and total intervals tend to be shorter in real graph streams than in randomly ordered ones. That is, future edges are more likely to form triangles with recent edges than with older edges

(i.e., sampled edges). When updating the sampled graph, THINKD$_{\text{FAST}}$, which is a simple and fast version, samples each edge i.i.d. with a fixed probability, while THINKD$_{\text{ACC}}$, which is an accurate version, employs random pairing [13].

Semi-streaming algorithms In addition to single-pass streaming algorithms, semi-streaming algorithms that require multiple passes over a graph were also explored [6,23].

Comparison with our proposed algorithm Our single-pass algorithm WRS estimates both global and local triangle counts in a fully dynamic graph stream without any prior knowledge of the input graph stream (see Sect. 3.3 for the detailed settings). Different from the existing approaches above, WRS exploits the temporal locality in real-world graph streams (see Fig. 2 in Sect. 4), which leads to higher accuracy.

3 Preliminaries and problem definition

In this section, we first introduce notations and concepts used in the paper. Then, we review two uniform sampling schemes that our proposed algorithms are based on. Lastly, we define the problem of global and local triangle counting in a real graph stream.

3.1 Notations and concepts

Symbols frequently used in the paper are listed in Table 2. Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of nodes $\mathcal{V}$ and the set of edges $\mathcal{E}$. We use the unordered pair $\{u, v\} \in \mathcal{E}$ to indicate the edge between two distinct nodes $u \neq v \in \mathcal{V}$, and the unordered triple $\{u, v, w\}$ to represent the triangle (i.e., clique of size three) with nodes u, v, and w.

Consider a dynamic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that evolves over time from an empty graph as edges (and adjacent nodes) are inserted and deleted. The changes in $\mathcal{G}$ can be represented as a *fully dynamic graph stream* $(\Delta^{(1)}, \Delta^{(2)}, \ldots)$, which is the sequence of edge insertions and deletions. For each $t \in \{1, 2, \ldots\}$, we use $e^{(t)}$ to denote the edge added or deleted at time t and use the pair $\Delta^{(t)} = (e^{(t)}, \delta^{(t)})$, where $\delta^{(t)} \in \{+, -\}$, to denote the change in $\mathcal{G}$ at time t. That is, $\Delta^{(t)} = (\{u, v\}, +)$ indicates the arrival of a new edge $\{u, v\} \notin \mathcal{E}$, and $\Delta^{(t)} = (\{u, v\}, -)$ indicates the removal of an existing edge $\{u, v\} \in \mathcal{E}$ at time t. We denote $\mathcal{G}$ after the t-th change $\Delta^{(t)}$ is applied by $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$. We denote the set of triangles in $\mathcal{G}^{(t)}$ by $\mathcal{T}^{(t)}$ and the set of triangles with a node u by $\mathcal{T}_u^{(t)} \subset \mathcal{T}^{(t)}$. We call $\mathcal{T}^{(t)}$ *global triangles* and $\mathcal{T}_u^{(t)}$ *local triangles* of each node u.

We use $t_{uv} \in \{1, 2, \ldots\}$ to indicate the last arrival time of each edge $\{u, v\}$. For example, in cases where the edge $\{u, v\}$ is deleted and reinserted in a fully dynamic graph stream (i.e., $\Delta^{(t_1)} = (\{u, v\}, +)$, $\Delta^{(t_2)} = (\{u, v\}, -)$, and $\Delta^{(t_3)} = (\{u, v\}, +)$, where $t_1 < t_2 < t_3$), t_{uv} is the arrival time of the reinserted edge (i.e., $t_{uv} = t_3$). For each triangle $\{u, v, w\} \in \mathcal{T}^{(t)}$, we let $e^{(1)}_{uvw}$, $e^{(2)}_{uvw}$, and $e^{(3)}_{uvw}$ be the edge arriving first, second, and last among $\{u, v\}$, $\{v, w\}$, and $\{w, u\}$, which together form $\{u, v, w\}$. We use $t^{(i)}_{uvw}$ to indicate the arrival time of $e^{(i)}_{uvw}$. That is, $t^{(1)}_{uvw} := \min\{t_{uv}, t_{vw}, t_{wu}\}$, $t^{(2)}_{uvw} := \text{median}\{t_{uv}, t_{vw}, t_{wu}\}$, and $t^{(3)}_{uvw} := \max\{t_{uv}, t_{vw}, t_{wu}\}$. These notations are used to account for the concept of temporal locality in Sect. 4.

3.2 Uniform sampling schemes for dynamic datasets

In this subsection, we give an overview of two uniform sampling schemes, namely reservoir sampling and random

Table 2 Table of symbols

Symbol	Definition
Notations for Graph Streams	
$\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$	Graph $\mathcal{G}$ at time t
$\{u, v\}$	Edge between nodes u and v
$\{u, v, w\}$	Triangle with nodes u, v, and w
$\Delta^{(t)} = (e^{(t)}, \delta^{(t)})$	Edge insertion or deletion at time t
$e^{(t)} = \{u, v\}$	Edge changed at time t
t_{uv}	Arrival time of edge $\{u, v\}$
$e^{(i)}_{uvw}$	Edge arrived i-th among $\{\{u, v\}, \{v, w\}, \{w, u\}\}$
$t^{(i)}_{uvw}$	Arrival time of $e^{(i)}_{uvw}$
$\mathcal{T}^{(t)}$	Set of triangles in $\mathcal{G}^{(t)}$
$\mathcal{T}^{(t)}_u$	Set of triangles with a node u in $\mathcal{G}^{(t)}$
Notations for Algorithms (defined in Sect. 5)	
$\mathcal{S}$	Given memory space
$\mathcal{W}$	Waiting room
$\mathcal{R}$	Reservoir
$\mathcal{E}_{\mathcal{R}}$	Set of edges flowing into $\mathcal{R}$ from $\mathcal{W}$
k	Maximum number of edges stored in $\mathcal{S}$
α	Relative size of the waiting room (i.e., $\lvert\mathcal{W}\rvert/\lvert\mathcal{S}\rvert$)
c	Estimated global triangle count
c_u	Estimated local triangle count of node u
$\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$	Graph composed of the edges in $\mathcal{S}$
$\hat{\mathcal{N}}_u$	Set of neighbors of a node u in $\hat{\mathcal{G}}$
Notations for Analyses (defined in Sect. 5)	
$\mathcal{A}^{(t)}$	Set of triangles added at time t or earlier
$\mathcal{D}^{(t)}$	Set of triangles deleted at time t or earlier
$\{u, v, w\}^{(t)}$	Triangle $\{u, v, w\}$ added or deleted at time t

pairing, for sampling as many items as possible from a dynamic dataset. Assume that a dataset $\mathcal{D}$ is initially empty, and it evolves over time as items are inserted and deleted. A sample $\mathcal{S} \subseteq \mathcal{D}$ is a set of items selected from $\mathcal{D}$ by a specific sampling rule. Following [7], a sampling scheme is *uniform* if all equal-sized subsets of $\mathcal{D}$ are equally likely to be $\mathcal{S}$, i.e., if

$$\mathbb{P}[\mathcal{S} = \mathcal{A}] = \mathbb{P}[\mathcal{S} = \mathcal{B}],\ \forall \mathcal{A} \neq \mathcal{B} \subseteq \mathcal{D}\ s.t.\ |\mathcal{A}| = |\mathcal{B}|. \quad (1)$$

The two sampling schemes described below satisfy this *uniformity*.

Algorithm 1: Reservoir Sampling

Input : (1) Insertion-only dataset: $\mathcal{D}$,
(2) Memory budget: k,
(3) Current sample: $\mathcal{S}$,
(4) Item addition: a
Output: Updated sample: $\mathcal{S}$

```
1 S ← ∅, |D| ← 0
2 Procedure INSERT(a)
3     |D| ← |D| + 1
4     if |S| < k then
5         S ← S ∪ {a}
6     else if Bernoulli(k/|D|) = 1 then
7         replace a randomly chosen item in S with a
```

3.2.1 Reservoir sampling

Given a sequence of item insertions into the dataset $\mathcal{D}$, reservoir sampling [49], which is described in Algorithm 1, maintains a bounded-size uniform sample $\mathcal{S}$. We use k to denote the maximum size of the maintained sample $\mathcal{S}$. Until the sample size $|\mathcal{S}|$ reaches k, each item inserted into $\mathcal{D}$ is stored in $\mathcal{S}$. Then, for each item inserted into $\mathcal{D}$, reservoir sampling replaces the newly inserted item with a randomly selected item from $\mathcal{S}$ with probability $k/|\mathcal{D}|$. Reservoir sampling guarantees that each item in $\mathcal{D}$ has the same probability $k/|\mathcal{D}|$ of being stored in $\mathcal{S}$, and thus, it guarantees uniformity [i.e., Eq. (1)]. However, it cannot handle item deletions, and thus, it can be used only for insertion-only datasets.

Algorithm 2: Random pairing (RP)

Input : (1) Fully dynamic dataset: $\mathcal{D}$,
(2) Memory budget: k,
(3) Current sample: $\mathcal{S}$,
(4) Item addition or deletion: a
Output: Updated sample: $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset, |\mathcal{D}| \leftarrow 0, n_b \leftarrow 0, n_g \leftarrow 0$
Procedure INSERT(a)
  $|\mathcal{D}| \leftarrow |\mathcal{D}| + 1$
  if $n_b + n_g = 0$ **then**
    if $|\mathcal{S}| < k$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}$
    else if $Bernoulli(k/|\mathcal{D}|) = 1$ **then**
      replace a randomly chosen item in $\mathcal{S}$ with a
  else
    if $Bernoulli(n_b/(n_b + n_g)) = 1$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{a\}, n_b \leftarrow n_b - 1$
    else $n_g \leftarrow n_g - 1$
Procedure DELETE(a)
  $|\mathcal{D}| \leftarrow |\mathcal{D}| - 1$
  if $a \in \mathcal{S}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{a\}, n_b \leftarrow n_b + 1$
  else $n_g \leftarrow n_g + 1$

3.2.2 Random pairing (RP)

Random pairing (RP) [13] is a uniform sampling method for fully dynamic datasets, where items are both added and deleted. The main idea behind RP is to make use of newly inserted items to 'compensate' for previous item deletions.

RP classifies item deletions into two categories: 'bad' uncompensated deletions and 'good' uncompensated deletions. An uncompensated deletion is 'bad' if the deleted item is in the sample, and thus, the deletion decreases the sample size by 1. On the other hand, an uncompensated deletion 'good' if the deleted item is not in the sample, and thus, the deletion does not affect the sample size. RP maintains counters n_b and n_g to record the number of 'bad' and 'good' uncompensated deletions, respectively. Undoubtedly, $n_b + n_g$ is equal to the total number of uncompensated deletions.

RP is described in Algorithm 2, where we use k to denote the maximum size of the maintained sample $\mathcal{S}$. For each item deletion from the fully dynamic dataset $\mathcal{D}$, RP checks whether the deleted item is included in the maintained sample $\mathcal{S}$ or not. If the deleted item is in $\mathcal{S}$, RP removes the item from $\mathcal{S}$ and increases n_b by 1; otherwise, RP simply increases n_g by 1. For each item inserted into $\mathcal{D}$, RP checks whether there are deletions that it needs to compensate for.

- **Case 1:** If there is no uncompensated deletion (i.e., if $n_b + n_g = 0$), then the insertion is proceeded as in reservoir sampling. Specifically, if $\mathcal{S}$ has less than k items, RP adds the inserted item to $\mathcal{S}$. Otherwise, RP replaces a uniformly random item in $\mathcal{S}$ with the newly inserted item with a certain probability.
- **Case 2:** On the other hand, if there are uncompensated deletions (i.e., if $n_b + n_g > 0$), then RP flips a coin. If the coin shows head, whose probability is $n_b/(n_b + n_g)$, RP adds the new item to the sample $\mathcal{S}$ and decreases n_b by 1. Otherwise, RP simply discards the new item and decreases n_g by 1.

Following the above procedure, RP guarantees uniformity [i.e., Eq. (1)] in the presence of arbitrary item insertions and deletions [13]. Unlike reservoir sampling, due to item deletions, RP does not guarantee that k samples are surely stored even if more than k items are processed.

3.3 Problem definition

In this work, we consider the problem of counting the global and local triangles in a graph stream assuming the following realistic conditions:

C1 **No Knowledge:** no information about the input stream (e.g., the node count, the edge count, etc.) is available in advance.
C2 **Real Dynamic:** in the input stream, edge insertions and edge deletions arrive in the order by which edges are created and removed in the input graph
C3 **Limited Memory Budget:** we store at most k edges in memory.
C4 **Single Pass:** edge additions and deletions are processed one by one in their order of arrival. Past changes cannot be accessed unless they are stored in memory (within the budget stated in C3).

Based on these conditions, we define the problem of global and local triangle counting in a real graph stream in Problem 1.

Problem 1 (Global and Local Triangle Counting in a Real Graph Stream)

(1) ***Given:***
- *a real graph stream* $\{\Delta^{(1)}, \Delta^{(2)}, \ldots\}$
- *a memory budget* k,

(2) ***Estimate:***
- *the global triangle count* $|\mathcal{T}^{(t)}|$
- *the local triangle counts* $\{(u, |\mathcal{T}_u^{(t)}|)\}_{u \in \mathcal{V}^{(t)}}$

at current time $t \in \{1, 2, \ldots\}$
(3) ***to Minimize:*** *the estimation errors.*

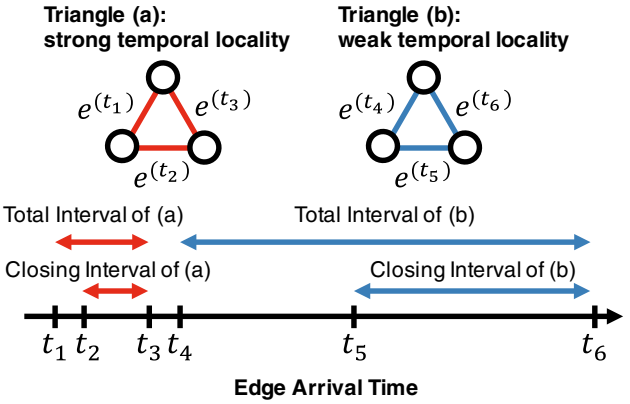


Fig. 3 Pictorial description of total intervals, closing intervals, and temporal locality

If $\delta^{(t)} = +$ for every $t \in \{1, 2, \ldots, \}$, we call Problem 1 *triangle counting in an insertion-only graph stream*. Otherwise, we call Problem 1 *triangle counting in a fully dynamic graph stream*. In Problem 1, instead of minimizing a specific measure of estimation error, we follow a general approach of reducing both bias and variance. This approach is robust to many measures of estimation error, as we show in the experiment section.

4 Empirical pattern: temporal locality

In this section, we discuss *temporal locality* (i.e., the tendency that future edges are more likely to form triangles with recent edges than with older edges) in real graph streams. To show the temporal locality, we investigate the distribution of *closing intervals* (see Definition 1) and *total intervals* (see Definition 2) in real graph streams. Figure 3 shows examples of closing and total intervals.

Definition 1 *(Closing Interval)* The *closing interval* of a triangle is defined as the time interval between the arrivals of the second and last edges. That is,

$$closing\_interval(\{u, v, w\}) := t^{(3)}_{uvw} - t^{(2)}_{uvw}.$$

Definition 2 *(Total Interval)* The *total interval* of a triangle is defined as the time interval between the arrivals of the first and last edges. That is,

$$total\_interval(\{u, v, w\}) := t^{(3)}_{uvw} - t^{(1)}_{uvw}.$$

Figure 2 shows the distributions of the closing and total intervals in real graph streams (see Sect. 6.1 for descriptions of the streams) and random ones obtained by randomly shuffling the orders of the edges in the corresponding real streams. In every dataset, both intervals tend to be much shorter in the real stream than in the randomly ordered one. That is, future edges do not form triangles with all previous edges with equal probability. They are more likely to form triangles with recent edges than with older edges.

Then, why does the temporal locality exist? It is related to *transitivity* [51], i.e., the tendency that people with common friends become friends. When an edge $\{u, v\}$ arrives, we can expect that edges connecting u and other neighbors of v or connecting v and other neighbors of u will arrive soon. These future edges form triangles with the edge $\{u, v\}$. The temporal locality is also related to new nodes. For example, in citation networks, when a new node arrives (i.e., a paper is published), many edges incident to the node (i.e., citations of the paper), which are likely to form triangles with each other, are created almost instantly. Likewise, in social media, new users make many connections within a short time by importing their friends from other social media or their address books during 'on-boarding' processes.

5 Proposed method: WRS

In this section, we propose *Waiting-Room Sampling* (WRS), a family of two single-pass streaming algorithms that exploit the temporal locality, presented in the previous section, for accurate global and local triangle counting in real graph streams. The two algorithms, namely WRS_{INS} and WRS_{DEL}, are designed for insertion-only graph streams and fully dynamic streams with edge deletions, respectively. Throughout the paper, we refer to WRS as the set of WRS_{INS} and WRS_{DEL}. We first discuss the intuition behind WRS in Sect. 5.1. Then, we describe the details of WRS_{INS} and WRS_{DEL} in Sects. 5.2 and 5.3, respectively. After that, we theoretically analyze their accuracy and complexity in Sects. 5.4 and 5.5, respectively.

5.1 Intuition behind WRS

For accurate estimation, WRS minimizes both the bias and variance of estimates. Reducing the variance is related to finding more triangles because, intuitively speaking, knowing more triangles is helpful to accurately estimate their count. This relation is more formally analyzed in Sect. 5.4. Thus, the following two goals should be considered when deciding which edges to store in memory:

- **Goal 1.** unbiased estimates of triangle counts should be able to be computed from the stored edges.
- **Goal 2.** when a new edge arrives, it should form many triangles with the stored edges.

Uniform random sampling, such as reservoir sampling and random pairing, achieves Goal 1 but fails to achieve Goal 2, ignoring the temporal locality, described in Sect. 4. Storing

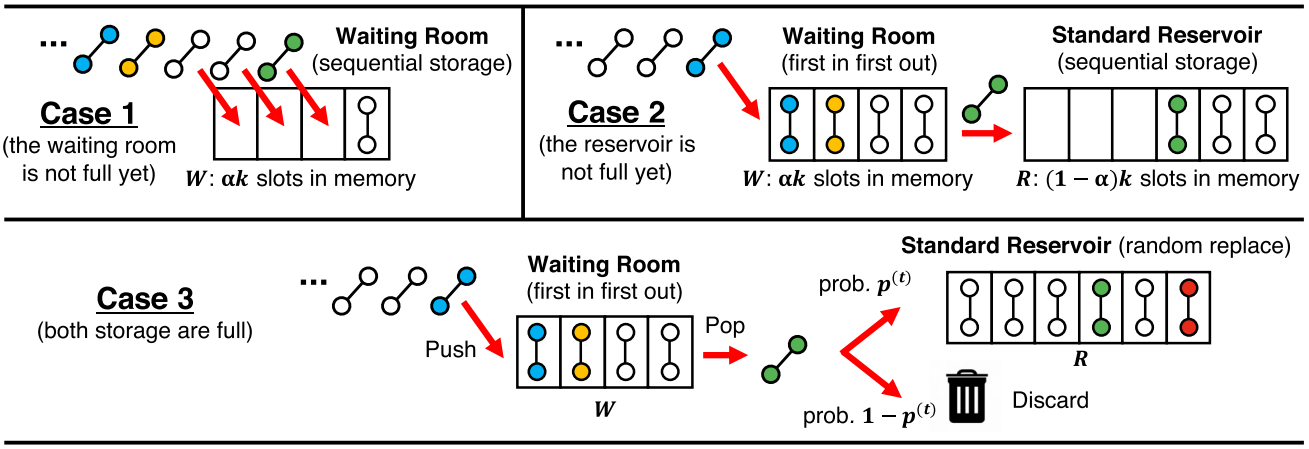


Fig. 4 Pictorial description of the sampling process in WRS_{INS}. Assume a new edge arrives. *Case 1* If the waiting room is not full, then the new edge is added to the waiting room. *Case 2* If the waiting room is full while the reservoir is not full, the oldest edge in the waiting room is moved from the waiting room to the reservoir, and the new edge is added to the waiting room. *Case 3* If both the waiting room and the reservoir are full, the oldest edge in the waiting room is evicted from the waiting room, and the new edge is added to the waiting room. Then, with probability $p^{(t)}$, the evicted edge replaces a uniformly random edge in the reservoir. Note that the latest $|\mathcal{W}|$ edges are stored in the waiting room, while the remaining older edges are uniformly sampled in the reservoir

the latest edges, while discarding the older ones, can be helpful to achieve Goal 2, as suggested by the temporal locality. However, simply discarding old edges makes unbiased estimation non-trivial.

To achieve both goals, WRS combines the two policies above. Specifically, it divides the memory space into the *waiting room* and the *reservoir*. The most recent edges are always stored in the waiting room, while the remaining edges are uniformly sampled in the reservoir using reservoir sampling or random pairing. The waiting room enables achieving Goal 2 since it exploits the temporal locality by storing the latest edges, which future edges are more likely to form triangles with. On the other hand, the reservoir enables achieving Goal 1, as described in detail in the following sections.

5.2 Details of WRS_{INS}

In this subsection, we describe WRS_{INS}, a simple version of WRS for insertion-only graph streams. WRS_{INS} provides unbiased estimates of the global and local triangle counts in insertion-only graph streams. We first describe the sampling policy of WRS_{INS}, which is presented in Fig. 4. Then, we describe how to estimate the triangle counts from sampled edges. A pseudocode of WRS_{INS} is given in Algorithm 3.

5.2.1 Sampling policy (lines 6–13 of Algorithm 3)

We use $\mathcal{S}$ to denote the given memory space, where at most k edges are stored. WRS divides $\mathcal{S}$ into the *waiting room* $\mathcal{W}$ and the *reservoir* $\mathcal{R}$, and we use α to denote the relative size of $\mathcal{W}$ (i.e., $|\mathcal{W}| = k\alpha$). For simplicity, we assume $k\alpha$ and $k(1-\alpha)$ are integers. In $\mathcal{W}$, the most recent $k\alpha$ edges in the input stream are stored, and some among the edges flowing from $\mathcal{W}$ are stored in the reservoir $\mathcal{R}$. Reservoir sampling, presented in Sect. 3.2.1, is used to decide which edges to be stored in $\mathcal{R}$. If we let $e^{(t)} = \{u, v\}$ be the edge arriving at time $t \in \{1, 2, \dots\}$ (i.e., $\Delta^{(t)} = (\{u, v\}, +)$), then the sampling scheme of WRS_{INS} is described as follows:

- **(Case 1).** If $\mathcal{W}$ is not full, add $e^{(t)}$ to $\mathcal{W}$ (line 6).
- **(Case 2).** If $\mathcal{W}$ is full and $\mathcal{R}$ is not full, WRS_{INS} first removes $e^{(t-k\alpha)}$ from $\mathcal{W}$, which is the oldest edge in $\mathcal{W}$, and then stores $e^{(t)}$ in $\mathcal{W}$, which is a queue with the 'first in first out' (FIFO) mechanism (lines 9–10). After that, WRS_{INS} stores $e^{(t-k\alpha)}$ in $\mathcal{R}$ (line 11).
- **(Case 3).** If both $\mathcal{W}$ and $\mathcal{R}$ are full, WRS_{INS} removes the oldest edge $e^{(t-k\alpha)}$ from $\mathcal{W}$ and inserts the new edge $e^{(t)}$ to $\mathcal{W}$ (lines 9–10). Then, WRS_{INS} replaces a uniformly random edge in $\mathcal{R}$ with $e^{(t-k\alpha)}$ with probability $p^{(t)}$, defined as

$$p^{(t)} := \frac{k(1-\alpha)}{|\mathcal{E}_{\mathcal{R}}^{(t)}|}, \tag{2}$$

  where $\mathcal{E}_{\mathcal{R}}^{(t)} := \{e^{(1)}, \dots, e^{(t-k\alpha)}\}$ is the set of edges flowing from $\mathcal{W}$ to $\mathcal{R}$ at time t or earlier (lines 12–13). That is, reservoir sampling, presented in Sect. 3.2.1, is used for $\mathcal{R}$, and thus, each edge in $\mathcal{E}_{\mathcal{R}}^{(t)}$ is stored in $\mathcal{R}$ with equal probability $p^{(t)}$.

In summary, when $\Delta^{(t)}$ arrives (or after $\Delta^{(t-1)}$ is processed), if $t \leq k + 1$, then each edge in $\{e^{(1)}, \dots, e^{(t-1)}\}$

Algorithm 3: WRS_{INS}: proposed algorithm for insertion-only graph streams

Input : (1) insertion-only graph stream: $\{\Delta^{(1)}, \Delta^{(2)}, \dots\}$,
(2) memory budget: k,
(3) relative size of the waiting room: α
Output: (1) estimated global triangle count: c,
(2) estimated local triangle counts: c_u for each node u

1 $|\mathcal{E}_{\mathcal{R}}| \leftarrow 0$
2 **foreach** $\Delta^{(t)} = (\{u, v\}, +)$ **do**
  /* Update estimates */
3 **foreach** *node* w *in* $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ **do**
    /* Discover the triangle $\{u, v, w\}$ */
4 initialize c, c_u, c_v, and c_w to 0 if they have not been set
5 increase c, c_u, c_v, and c_w by $1/p^{(t)}_{uvw}$
  /* Sample the inserted edge $\{u, v\}$ */
6 **if** $|\mathcal{W}| < k\alpha$ **then** $\mathcal{W} \leftarrow \mathcal{W} \cup \{\{u, v\}\}$
7 **else**
8 $|\mathcal{E}_{\mathcal{R}}| \leftarrow |\mathcal{E}_{\mathcal{R}}| + 1$
9 remove the oldest edge $\{x, y\}$ from $\mathcal{W}$
10 add $\{u, v\}$ to $\mathcal{W}$
11 **if** $|\mathcal{R}| < k(1-\alpha)$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{\{x, y\}\}$
12 **else if** Bernoulli$(k(1-\alpha)/|\mathcal{E}_{\mathcal{R}}|) = 1$ **then**
13 replace a uniformly random edge in $\mathcal{R}$ with $\{x, y\}$

is always stored in $\mathcal{W}$ or $\mathcal{R}$. If $t > k+1$, then each edge in $\{e^{(t-k\alpha)}, \dots, e^{(t-1)}\}$ is always stored in $\mathcal{W}$, while each edge in $\{e^{(1)}, \dots, e^{(t-k\alpha-1)}\}$ is stored in $\mathcal{R}$ with probability $p^{(t-1)}$.

5.2.2 Estimating triangle counts (lines 3–5 of Algorithm 3)

We use $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ to denote the sampled graph composed of the edges in $\mathcal{S}$ (i.e., in $\mathcal{W}$ or $\mathcal{R}$), and we use $\hat{\mathcal{N}}_u$ to denote the set of neighbors of each node $u \in \hat{\mathcal{V}}$ in $\hat{\mathcal{G}}$. We use c and c_u to denote the estimates of the global triangle count and the local triangle count of each node u, respectively, in the stream so far. That is, if we let $c^{(t)}$ and $c^{(t)}_u$ be c and c_u after processing $\Delta^{(t)}$, they are estimates of $|\mathcal{T}^{(t)}|$ and $|\mathcal{T}^{(t)}_u|$, respectively.

When each edge $e^{(t)} = \{u, v\}$ arrives, WRS_{INS} first finds the triangles composed of $\{u, v\}$ and two other edges in $\hat{\mathcal{G}}$. The set of such triangles is $\{\{u, v, w\} : w \in \hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v\}$. For each such triangle $\{u, v, w\}$, WRS_{INS} increases c, c_u, c_v, and c_w by $1/p^{(t)}_{uvw}$, where $p^{(t)}_{uvw}$ is the probability that WRS_{INS} discovers $\{u, v, w\}$ at time t, i.e., the probability that $\{v, w\}$ and $\{w, u\}$ are in $\mathcal{S}$ when $\{u, v\}$ arrives (see Lemma 1). Then, the expected increase of the counters by each triangle $\{u, v, w\}$ becomes 1 and the counters become unbiased estimates, as formalized in Theorem 1 in Sect. 5.4.

The remaining task is to compute $p^{(t)}_{uvw}$, the probability that WRS_{INS} discovers triangle $\{u, v, w\}$. To this end, we classify triangles into 3 types depending on where the first and second edges are stored when the third edge arrives, as in Definition 3. Recall that $e^{(i)}_{uvw}$ indicates the edge arriving i-th among the edges in $\{u, v, w\}$.

Definition 3 *(Types of Triangles)* We define the type of each triangle $\{u, v, w\}$, depending on where the first and the second edges in $\{u, v, w\}$ are stored when the third edge arrives, as follows:

$$type_{uvw} := \begin{cases} 1 & \text{if } e^{(1)}_{uvw} \in \mathcal{W} \text{ and } e^{(2)}_{uvw} \in \mathcal{W} \\ 2 & \text{else if } e^{(2)}_{uvw} \in \mathcal{W} \\ 3 & \text{otherwise.} \end{cases}$$

Note that a triangle is of Type 1 if its total interval is short. A triangle is of Type 2 if its closing interval is short while its total interval is not. If neither of its intervals is short, a triangle is of Type 3.

The probability that each triangle is discovered by WRS_{INS} depends on its type, as formalized in Lemma 1, where the superscript (t) is used to denote the value of each variable after the t-th element $\Delta^{(t)}$ is processed.

Lemma 1 (Triangle Discovering Probability in WRS_{INS}.) *Given the maximum size k of sample and the relative size α of the waiting room, the probability $p^{(t)}_{uvw}$ that each new triangle $\{u, v, w\} \in \mathcal{T}^{(t)} - \mathcal{T}^{(t-1)}$ is discovered in line 3 of Algorithm 3 at time t is as follows:*

$$p^{(t)}_{uvw} = \begin{cases} 1 & \text{if } type_{uvw} = 1 \\ \frac{z^{(t-1)}_{\mathcal{R}}}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|} & \text{if } type_{uvw} = 2 \\ \frac{z^{(t-1)}_{\mathcal{R}}}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|} \cdot \frac{z^{(t-1)}_{\mathcal{R}}-1}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|-1} & \text{if } type_{uvw} = 3, \end{cases} \tag{3}$$

where $z^{(t-1)}_{\mathcal{R}} = min(k(1-\alpha), |\mathcal{E}^{(t-1)}_{\mathcal{R}}|)$.

Proof Without loss of generality, we assume $e^{(1)}_{uvw} = \{v, w\}$, $e^{(2)}_{uvw} = \{w, u\}$, and $e^{(3)}_{uvw} = e^{(t)} = \{u, v\}$. That is, $\{v, w\}$ arrives earlier than $\{w, u\}$, and $\{w, u\}$ arrives earlier than $\{u, v\}$. When $e^{(t)} = \{u, v\}$ arrives at time t, the triangle $\{u, v, w\}$ is discovered if and only if $\{v, w\}$ and $\{w, u\}$ are in $\mathcal{S}^{(t-1)}$.

Note that, since WRS_{INS} performs updating the triangle counts before sampling edges, $\mathcal{E}_{\mathcal{R}} = \mathcal{E}^{(t-1)}_{\mathcal{R}}$ at the point of executing line 5 of Algorithm 3.

If $type_{uvw} = 1$, $\{v, w\}$ and $\{w, u\}$ are always stored in $\mathcal{W}^{(t-1)}$, when $\{u, v\}$ arrives. Thus, WRS_{INS} discovers $\{u, v, w\}$ with probability 1.

If $type_{uvw} = 2$, when $\{u, v\}$ arrives, $\{w, u\}$ is always stored in $\mathcal{W}^{(t-1)}$, while $\{v, w\}$ cannot be in $\mathcal{W}^{(t-1)}$ but can be in $\mathcal{R}^{(t-1)}$. For WRS_{INS} to discover $\{u, v, w\}$, $\{v, w\}$ should be in $\mathcal{R}^{(t-1)}$, and thus, the probability is

$$\mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)}] = \begin{cases} 1 & \text{if } |\mathcal{E}^{(t-1)}_{\mathcal{R}}| \le k(1-\alpha) \\ \frac{k(1-\alpha)}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|} & otherwise, \end{cases}$$

or equivalently,

$$\mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)}] = \frac{z_{\mathcal{R}}^{(t-1)}}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|},$$

where $z_{\mathcal{R}}^{(t-1)} = \min(k(1-\alpha), |\mathcal{E}_{\mathcal{R}}^{(t-1)}|)$

If $type_{uvw} = 3$, $\{v, w\}$ and $\{w, u\}$ cannot be in $\mathcal{W}^{(t-1)}$, when $\{u, v\}$ arrives. For WRS_{INS} to discover $\{u, v, w\}$, both $\{v, w\}$ and $\{w, u\}$ should be in $\mathcal{R}^{(t-1)}$. The probability of the event is

$$\begin{aligned}
&\mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)} \text{ and } \{w, u\} \in \mathcal{R}^{(t-1)}] \\
&= \mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)}] \\
&\cdot \mathbb{P}[\{w, u\} \in \mathcal{R}^{(t-1)} | \{v, w\} \in \mathcal{R}^{(t-1)}] \\
&= \frac{z_{\mathcal{R}}^{(t-1)}}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|} \cdot \frac{z_{\mathcal{R}}^{(t-1)} - 1}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}| - 1}.
\end{aligned}$$

□

Notice that no additional space is required to store the arrival times of sampled edges. This is because the type of each triangle and its discovering probability [i.e., Eq. (3)] can be computed from the size of $\mathcal{E}_{\mathcal{R}}$ and whether each edge is stored in $\mathcal{W}$ or $\mathcal{R}$ at time t, as explained in the proof of Lemma 1. Moreover, since only its size matters, $\mathcal{E}_{\mathcal{R}}$ does not have to be maintained.

5.3 Handling edge deletion: WRS_{DEL}

In real-world graphs, there can exist both edge insertions and deletions. There can even be cases where previously deleted edges are re-inserted. In this subsection, we present WRS_{DEL}, which extends WRS_{INS} for triangle counting in fully dynamic graph streams, which is the sequence of edge insertions and deletions. To this end, the Waiting-Room Sampling, described in the previous section, is also extended to handle edge deletions. WRS_{DEL} maintains unbiased estimates of both global and local triangle counts. Below, we first present the sampling policy of WRS_{DEL}, and then, we describe how to estimate the triangle counts from sampled edges. A pseudocode of WRS_{DEL} is given in Algorithm 4.

5.3.1 Sampling policy (lines 9–27 of Algorithm 4)

For sampling, WRS_{DEL} divides the given memory space $\mathcal{S}$, where up to k edges are stored, into the waiting room $\mathcal{W}$ and the reservoir $\mathcal{R}$, where up to $k\alpha$ and $k(1-\alpha)$ are stored, respectively. As in WRS_{INS}, the latest edges are stored in $\mathcal{W}$, and edges sampled among the other edges are stored in $\mathcal{R}$. Handling deletions of edges in $\mathcal{W}$ is straightforward, and RP, which is described in Sect. 3.2.2, is used for handling deletions of edges in $\mathcal{R}$ without losing uniformity. Let

Algorithm 4: WRS_{DEL}: proposed algorithm for fully dynamic graph streams

Input : (1) fully dynamic graph stream: $\{\Delta^{(1)}, \Delta^{(2)}, \ldots\}$,
(2) memory budget: k,
(3) relative size of the waiting room: α
Output: (1) estimated global triangle count: c,
(2) estimated local triangle counts: c_u for each node u

1 $|\mathcal{E}_{\mathcal{R}}| \leftarrow 0, n_b \leftarrow 0, n_g \leftarrow 0$
2 **foreach** $\Delta^{(t)} = (\{u, v\}, \delta)$ **do**
/* Update estimates */
3 **foreach** *node* w *in* $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ **do**
/* Discover the triangle $\{u, v, w\}$ */
4 initialize c, c_u, c_v, and c_w to 0 if they have not been set
5 **if** $\delta = +$ **then**
6 increase c, c_u, c_v, and c_w by $1/q_{uvw}^{(t)}$
7 **else if** $\delta = -$ **then**
8 decrease c, c_u, c_v, and c_w by $1/q_{uvw}^{(t)}$
/* Sample the inserted edge $\{u, v\}$ */
9 **if** $\delta = +$ **then**
10 **if** $|\mathcal{W}| < k\alpha$ **then** $\mathcal{W} \leftarrow \mathcal{W} \cup \{\{u, v\}\}$
11 **else**
12 $|\mathcal{E}_{\mathcal{R}}| \leftarrow |\mathcal{E}_{\mathcal{R}}| + 1$
13 remove the oldest edge $\{x, y\}$ from $\mathcal{W}$ and add $\{u, v\}$ to $\mathcal{W}$
14 **if** $n_b + n_g = 0$ **then**
15 **if** $|\mathcal{R}| < k(1-\alpha)$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{\{x, y\}\}$
16 **else if** Bernoulli$(k(1-\alpha)/|\mathcal{E}_{\mathcal{R}}|) = 1$ **then**
17 replace a randomly chosen edge in $\mathcal{R}$ with $\{x, y\}$
18 **else if** Bernoulli$(n_b/(n_b + n_g)) = 1$ **then**
19 $\mathcal{R} \leftarrow \mathcal{R} \cup \{\{x, y\}\}, n_b \leftarrow n_b - 1$
20 **else** $n_g \leftarrow n_g - 1$
/* Delete the removed edge $\{u, v\}$ */
21 **else if** $\delta = -$ **then**
22 **if** $\{u, v\} \in \mathcal{W}$ **then** $\mathcal{W} \leftarrow \mathcal{W} \setminus \{\{u, v\}\}$
23 **else**
24 $|\mathcal{E}_{\mathcal{R}}| \leftarrow |\mathcal{E}_{\mathcal{R}}| - 1$
25 **if** $\{u, v\} \in \mathcal{R}$ **then**
26 $\mathcal{R} \leftarrow \mathcal{R} \setminus \{\{u, v\}\}, n_b \leftarrow n_b + 1$
27 **else** $n_g \leftarrow n_g + 1$

$\Delta^{(t)} = (e^{(t)} = \{u, v\}, \delta^{(t)})$ be the edge addition or deletion at time $t \in \{1, 2, \ldots\}$. Let n_b and n_g be the counters of 'bad' and 'good' uncompensated deletions, respectively, which is used in RP to record each type of deletions. Below, we describe how WRS_{DEL} updates the maintained sample (i.e., edges stored in $\mathcal{W}$ and $\mathcal{R}$) in response to edge additions and deletions.

For Edge Insertions: Whenever an insertion of an edge arrives, WRS_{DEL} checks whether $\mathcal{W}$ is full or not.

- **(Case 1).** If $\mathcal{W}$ is not full, the new edge is stored at $\mathcal{W}$ (line 10).
- **(Case 2).** If $\mathcal{W}$ is full, WRS_{DEL} removes the oldest edge in $\mathcal{W}$ and inserts the new edge to $\mathcal{W}$ by the FIFO mechanism

(line 13), and then WRS_{DEL} decides whether to store the edge that is removed from $\mathcal{W}$ in $\mathcal{R}$ or not, following the RP procedure (lines 14–20), which is described in Sect. 3.2.2. Specifically, this case is further divided into two subcases.

- **(Case 2-1).** If there is no deletion to be compensated (line 14), RP processes each edge addition as reservoir sampling does (lines 15–17). That is, if the reservoir is not full (i.e., $|\mathcal{R}| < k(1-\alpha)$), RP adds the popped edge to $\mathcal{R}$, Otherwise, RP replaces a uniformly random edge in $\mathcal{R}$ with the popped edge with probability $(k(1-\alpha))/|\mathcal{E}_{\mathcal{R}}|$.
- **(Case 2-2).** If there are deletions that need to be compensated, then RP tosses a coin (line 18). If the coin shows head, whose probability is $n_b/(n_b+n_g)$, RP adds the popped edge to $\mathcal{R}$. Otherwise, RP discards the popped edge. Next, RP decreases either n_b or n_g by 1, depending on whether the popped edge has been added to $\mathcal{R}$ or not (lines 19–20).

For Edge Deletions: Whenever a deletion of an edge arrives, WRS_{DEL} first checks whether the edge is in $\mathcal{W}$ or not. If the edge is stored in $\mathcal{W}$, it is simply removed from $\mathcal{W}$ (line 22). Otherwise, WRS_{DEL} removes the edge from $\mathcal{R}$ (if it is in $\mathcal{R}$) and increases n_b or n_g depending on whether the edge was in $\mathcal{R}$ or not (lines 24–27).

5.3.2 Estimating triangle counting (lines 3–8 of Algorithm 4)

When each edge insertion or deletion $\Delta^{(t)} = (\{u, v\}, \delta^{(t)})$ arrives, WRS_{DEL} first finds the triangles composed of $\{u, v\}$ and two edges in $\hat{\mathcal{G}}$. The set of such triangles is $\{\{u, v, w\} : w \in \hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v\}$. For each triangle $\{u, v, w\}$, WRS_{DEL} increases or decreases c, c_u, c_v, and c_w by $1/q^{(t)}_{uvw}$, where $q^{(t)}_{uvw}$ is the probability that WRS_{DEL} discovers $\{u, v, w\}$ at time t (see Lemma 2) in line 3 of Algorithm 4, depending on whether the edge is inserted (i.e., $\delta^{(t)} = +$) or deleted (i.e., $\delta^{(t)} = -$). Then, the expected increase or decrease in each counter by each triangle $\{u, v, w\}$ becomes 1, which makes the counters unbiased estimates, as formalized in Theorem 2 in the following section.

In Lemma 2, the probability $q^{(t)}_{uvw}$ that WRS_{DEL} discovers the triangle $\{u, v, w\}$ at time t is formulated. To this end, in Definitions 4 and 5, we define two concepts: *added triangles* and *deleted triangles* in a fully dynamic graph stream. The added triangles and deleted triangles are the sets of all triangles that have been added to and deleted from the input graph, respectively. In a fully dynamic graph stream, if the same edge arrives again after being deleted, a triangle can be added or deleted multiple times. To distinguish the same triangle added and deleted at different times, we use $\{u, v, w\}^{(t)}$ to denote the triangle $\{u, v, w\}$ added or deleted at time t. Similarly, we use the superscript (t) to denote the value of each variable after the t-th element $\Delta^{(t)}$ is processed.

Definition 4 *(Added Triangles)* $\mathcal{A}^{(t)}$ is defined as the set of triangles that have been added to the graph $\mathcal{G}$ at time t or earlier. That is,

$$\mathcal{A}^{(t)} := \{\{u, v, w\}^{(s)} : \{u, v, w\} \notin \mathcal{T}^{(s-1)}, \{u, v, w\} \in \mathcal{T}^{(s)}, \textit{where } 1 \le s \le t\}.$$

Definition 5 *(Deleted Triangles)* $\mathcal{D}^{(t)}$ is defined as the set of triangles that have been deleted from the graph $\mathcal{G}$ at time t or earlier. That is,

$$\mathcal{D}^{(t)} := \{\{u, v, w\}^{(s)} : \{u, v, w\} \in \mathcal{T}^{(s-1)}, \{u, v, w\} \notin \mathcal{T}^{(s)}, \textit{where } 1 \le s \le t\}.$$

By the definition of added triangles, the set of triangles added at time t is $\mathcal{A}^{(t)} - \mathcal{A}^{(t-1)}$. Likewise, by the definition of deleted triangles, the set of triangles deleted at time t is $\mathcal{D}^{(t-1)} - \mathcal{D}^{(t)}$.

Lemma 2 (Triangle Discovering Probability in WRS_{DEL}.) *Given the maximum size k of sample and the relative size α of the waiting room, the probability $q^{(t)}_{uvw}$ that each triangle $\{u, v, w\} \in (\mathcal{A}^{(t)} - \mathcal{A}^{(t-1)}) \cup (\mathcal{D}^{(t-1)} - \mathcal{D}^{(t)})$ is discovered in line 3 of Algorithm 4 at time t is as follows:*

$$q^{(t)}_{uvw} = \begin{cases} 1 & \textit{if } type_{uvw} = 1 \\ \frac{y^{(t-1)}_{\mathcal{R}}}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|+d^{(t-1)}} & \textit{if } type_{uvw} = 2 \\ \frac{y^{(t-1)}_{\mathcal{R}}}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|+d^{(t-1)}} \cdot \frac{y^{(t-1)}_{\mathcal{R}}-1}{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|+d^{(t-1)}-1} & \textit{if } type_{uvw} = 3, \end{cases} \quad (4)$$

where $d^{(t-1)} = n^{(t-1)}_b + n^{(t-1)}_g$ *and* $y^{(t-1)}_{\mathcal{R}} = \min(k(1-\alpha), |\mathcal{E}^{(t-1)}_{\mathcal{R}}| + d^{(t-1)})$.

Proof WRS_{DEL} discovers a triangle $\{u, v, w\}^{(t)}$ added or removed at time t if any only if the other two edges are stored in $\mathcal{S}$ when an edge forming the triangle is added or removed at time t. Thus, the probability $q^{(t)}_{uvw}$ of discovering $\{u, v, w\}^{(t)}$ is equal to the probability that the other two edges are stored in $\mathcal{S}^{(t-1)}$. A full proof with the derivation of the probability is given in 'Appendix A.' □

Recall that the type of each triangle is determined simply by whether each edge is stored in $\mathcal{W}$ or $\mathcal{R}$. Moreover, Eq. (4) is directly calculated from the three counters (i.e., n_b, n_g and $|\mathcal{E}_{\mathcal{R}}|$). Thus, WRS_{DEL} requires no additional space to store any additional information or the arrival times of sampled edges. Especially, since only its size matters, $\mathcal{E}_{\mathcal{R}}$ does not have to be maintained.

5.4 Accuracy analysis

In this subsection, we prove that WRS maintains unbiased estimates, whose expected values are equal to the true global and local triangle counts. Then, we analyze the variance of the estimates provided by WRS_{INS}. Throughout this section, we use the superscript (t) to denote the value of each variable after the t-th element $\Delta^{(t)}$ is processed.

5.4.1 Bias analysis

We prove the unbiasedness of the estimates given by WRS_{INS} in Theorem 1.

Theorem 1 (Unbiasedness of WRS_{INS}) *If the input graph stream is insertion only and $k(1-\alpha) \geq 2$, then WRS_{INS} gives unbiased estimates of the global and local triangle counts at any time t. That is, if we let $c^{(t)}$ and $c_u^{(t)}$ be c and c_u after processing $\Delta^{(t)}$, respectively, then the followings hold:*

$$\mathbb{E}[c^{(t)}] = |\mathcal{T}^{(t)}|, \ \forall t \geq 1 \tag{5}$$

$$\mathbb{E}[c_u^{(t)}] = |\mathcal{T}_u^{(t)}|, \ \forall u \in \mathcal{V}^{(t)}, \ \forall t \geq 1. \tag{6}$$

Proof Let $x_{uvw}^{(s)}$ be the amount of increase in each of $c^{(t)}$, $c_u^{(t)}$, $c_v^{(t)}$, and $c_w^{(t)}$ due to the discovery of $\{u, v, w\}^{(s)}$ (i.e., $\{u, v, w\}$ added at time s) where $s \leq t$. Without loss of generality, we assume $e_{uvw}^{(3)} = e^{(s)} = \{u, v\}$, and thus, $\Delta^{(s)} = (\{u, v\}, +)$. Then, the following holds:

$$x_{uvw}^{(s)} = \begin{cases} 1/p_{uvw}^{(s)} & \text{if } \{v, w\} \textit{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & \text{otherwise,} \end{cases}$$

From this and Lemma 1, we have $\mathbb{E}[x_{uvw}^{(s)}] = 1/p_{uvw}^{(s)} \times p_{uvw}^{(s)} + 0 \times (1 - p_{uvw}^{(s)}) = 1$. Combining this and $c^{(t)} = \sum_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}} x_{uvw}^{(s)}$ gives

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

which proves Eq. (5). Likewise, combining $\mathbb{E}[x_{uvw}^{(s)}] = 1$ and $c_u^{(t)} = \sum_{\{u,v,w\}^{(s)} \in \mathcal{T}_u^{(t)}} x_{uvw}^{(s)}$ gives

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

which proves Eq. (6). □

Before showing the unbiasedness of WRS_{DEL}, we extend the concept of added triangles and deleted triangles in Definitions 4 and 5 to the node level. We denote the sets of added and deleted triangles with each node $u \in \mathcal{V}^{(t)}$ by $\mathcal{A}_u^{(t)} \subseteq \mathcal{A}^{(t)}$ and $\mathcal{D}_u^{(t)} \subseteq \mathcal{D}^{(t)}$, respectively. Then, from their counts, the number of triangles remaining without being deleted is obtained, as formalized in Lemma 3.

Lemma 3 (Count of Triangles [38]) *Subtracting the number of deleted triangles from the number of added triangles gives the number of triangles in the input stream. Formally,*

$$|\mathcal{T}^{(t)}| = |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}|, \ \forall t \geq 1, \tag{7}$$

$$|\mathcal{T}_u^{(t)}| = |\mathcal{A}_u^{(t)}| - |\mathcal{D}_u^{(t)}|, \ \forall t \geq 1, \ \forall u \in \mathcal{V}^{(t)}. \tag{8}$$

Based on Lemma 3, we prove the unbiasedness of the estimates given by WRS_{DEL} in fully dynamic graph streams in Theorem 2. Note that Theorem 2 is a generalization of Theorem 1.

Theorem 2 (Unbiasedness of WRS_{DEL}.) *If $k(1-\alpha) \geq 2$, then WRS_{DEL} gives unbiased estimates of the global and local triangle counts at any time t. That is, if we let $c^{(t)}$ and $c_u^{(t)}$ be c and c_u after processing $\Delta^{(t)}$, respectively, then the followings hold:*

$$\mathbb{E}[c^{(t)}] = |\mathcal{T}^{(t)}|, \ \forall t \geq 1, \tag{9}$$

$$\mathbb{E}[c_u^{(t)}] = |\mathcal{T}_u^{(t)}|, \ \forall u \in \mathcal{V}^{(t)}, \ \forall t \geq 1. \tag{10}$$

Proof Let $x_{uvw}^{(s)}$ be the amount of increase in each of $c^{(t)}$, $c_u^{(t)}$, $c_v^{(t)}$, and $c_w^{(t)}$ due to the discovery of $\{u, v, w\}^{(s)} \in \mathcal{A}^{(t)}$ (i.e., $\{u, v, w\}$ added at time s) where $s \leq t$. Without loss of generality, we assume $e_{uvw}^{(3)} = e^{(s)} = \{u, v\}$ and thus $\Delta^{(s)} = (\{u, v\}, +)$.

Then, the following holds:

$$x_{uvw}^{(s)} = \begin{cases} 1/q_{uvw}^{(s)} & \text{if } \{v, w\} \textit{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

Similarly, let $y_{uvw}^{(s)}$ be the amount of decrease in each of $c^{(t)}$, $c_u^{(t)}$, $c_v^{(t)}$, and $c_w^{(t)}$ due to the discovery of $\{u, v, w\}^{(s)} \in \mathcal{D}^{(t)}$ (i.e., $\{u, v, w\}$ deleted at time s) where $s \leq t$. Without loss of generality, we assume $e_{uvw}^{(3)} = e^{(s)} = \{u, v\}$ and thus $\Delta^{(s)} = (\{u, v\}, -)$.

Then, the following holds:

$$y_{uvw}^{(s)} = \begin{cases} -1/q_{uvw}^{(s)} & \text{if } \{v, w\} \textit{ and } \{w, u\} \in \mathcal{S}^{(s-1)} \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Equations (11), (12), and Lemma 2, which states that $\mathbb{P}[\{v, w\}, \{w, u\} \in \mathcal{S}^{(s-1)}]$ is equal to $q^{(s)}_{uvw}$, imply

$$\mathbb{E}[x^{(s)}_{uvw}] = 1 \text{ and } \mathbb{E}[y^{(s)}_{uvw}] = -1. \tag{13}$$

Definition 4, Definition 5, and Algorithm 4 imply

$$c^{(t)} = \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}} x^{(s)}_{uvw} + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}} y^{(s)}_{uvw}. \tag{14}$$

Combining this, linearity of expectation, Eq. (7) in Lemma 3, Eqs. (13) and (14) gives

$$\begin{aligned}\mathbb{E}[c^{(t)}] &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}} \mathbb{E}[x^{(s)}_{uvw}] + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}} \mathbb{E}[y^{(s)}_{uvw}] \\ &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}} 1 + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}} -1 \\ &= |\mathcal{A}^{(t)}| - |\mathcal{D}^{(t)}| = |\mathcal{T}^{(t)}|.\end{aligned}$$

Hence, Eq. (9) holds. Likewise, for each node $u \in \mathcal{V}^{(t)}$, the following equation holds:

$$c^{(t)}_u = \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}_u} x^{(s)}_{uvw} + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}_u} y^{(s)}_{uvw}. \tag{15}$$

Combining this, linearity of expectation, Eq. (8) in Lemma 3, Eqs. (13) and (15) gives

$$\begin{aligned}\mathbb{E}[c^{(t)}_u] &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}_u} \mathbb{E}[x^{(s)}_{uvw}] + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}_u} \mathbb{E}[y^{(s)}_{uvw}] \\ &= \sum_{\{u,v,w\}^{(s)} \in \mathcal{A}^{(t)}_u} 1 + \sum_{\{u,v,w\}^{(s)} \in \mathcal{D}^{(t)}_u} -1 \\ &= |\mathcal{A}^{(t)}_u| - |\mathcal{D}^{(t)}_u| = |\mathcal{T}^{(t)}_u|\end{aligned}$$

Hence, Eq. (10) holds. □

5.4.2 Variance analysis

We present a variance analysis through which we shed light on the effectiveness of WRS. For simplicity, we focus on the variance of the estimates provided by WRS_{INS} and specifically the following:

$$\tilde{\text{Var}}[c^{(t)}] = \sum\nolimits_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}} \text{Var}[x^{(s)}_{uvw}], \tag{16}$$

$$\tilde{\text{Var}}[c^{(t)}_u] = \sum\nolimits_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}_u} \text{Var}[x^{(s)}_{uvw}], \tag{17}$$

where the dependencies between $\{x^{(s)}_{uvw}\}_{\{u,v,w\}^{(s)} \in \mathcal{T}^{(t)}}$ are ignored. In real-world graphs, the variance $\text{Var}[c^{(t)}]$ and the simplified version $\tilde{\text{Var}}[c^{(t)}]$ [i.e., Eq. (16)] are strongly correlated ($R^2 > 0.99$), as discussed in detail in 'Appendix B.'

To show how the temporal locality, described in Sect. 4, is related to reducing $\tilde{\text{Var}}[c^{(t)}]$ and $\tilde{\text{Var}}[c^{(t)}_u]$, we first formulate $\text{Var}[x^{(t)}_{uvw}]$, which is summed in Eqs. (16) and (17), in Lemma 4.

Lemma 4 (Variance of x_{uvw} in WRS_{INS}) *If $k(1-\alpha) \geq 2$, the variance of the increment x_{uvw} in c by each triangle $\{u, v, w\} \in \mathcal{T}^{(t)} - \mathcal{T}^{(t-1)}$ in* WRS_{INS} *is*

$$\text{Var}[x^{(t)}_{uvw}] = \begin{cases} 0 & \text{if } type_{uvw} = 1 \\ \frac{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|}{z^{(t-1)}_{\mathcal{R}}} - 1 & \text{if } type_{uvw} = 2 \\ \frac{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|}{z^{(t-1)}_{\mathcal{R}}} \times \frac{|\mathcal{E}^{(t-1)}_{\mathcal{R}}|-1}{z^{(t-1)}_{\mathcal{R}}-1} - 1 & \text{if } type_{uvw} = 3, \end{cases} \tag{18}$$

where $z^{(t-1)}_{\mathcal{R}} = \min(k(1-\alpha), |\mathcal{E}^{(t-1)}_{\mathcal{R}}|)$.

Proof From $\text{Var}[x^{(t)}_{uvw}] = \mathbb{E}[(x^{(t)}_{uvw})^2] - \left(\mathbb{E}[x^{(t)}_{uvw}]\right)^2$, $\text{Var}[x^{(t)}_{uvw}] = (1/p^{(t)}_{uvw}) - 1$ holds. This and Eq. (3) imply that, if $k(1-\alpha) \geq 2$, the variance of $x^{(t)}_{uvw}$ is equal to Eq. (18). □

As formalized in Lemma 5, compared to $\text{Triès}_{\text{IMPR}}$ [8], where

$$\text{Var}[x^{(t)}_{uvw}] = \frac{|\mathcal{E}^{(t-1)}|}{m^{(t-1)}} \times \frac{|\mathcal{E}^{(t-1)}| - 1}{m^{(t-1)} - 1} - 1$$

and $m^{(t-1)} = \min(k, |\mathcal{E}^{(t-1)}|)$ for every triangle added at time t regardless of its type, WRS_{INS} reduces the variance regarding the triangles of Type 1 and 2, while increasing the variance regarding the triangles of Type 3. Note that, in both WRS_{INS} and $\text{Triès}_{\text{IMPR}}$, the variance regarding the triangles added at time $t = k + 1$ or earlier is 0 since every edge is stored without being discarded at time $t = k + 1$ or earlier.

Lemma 5 (Comparison of Variances between WRS_{INS} and $\text{Triès}_{\text{IMPR}}$) *At any time $t > k + 1$, for each triangle $\{u, v, w\} \in \mathcal{T}^{(t)} - \mathcal{T}^{(t-1)}$, $\text{Var}[x^{(t)}_{uvw}]$ is smaller in* WRS_{INS} *than in* $\text{Triès}_{\text{IMPR}}$ [8], i.e.,

$$\text{Var}[x^{(t)}_{uvw}] < \frac{|\mathcal{E}^{(t-1)}|}{m^{(t-1)}} \times \frac{|\mathcal{E}^{(t-1)}| - 1}{m^{(t-1)} - 1} - 1, \tag{19}$$

where $m^{(t-1)} = \min(k, |\mathcal{E}^{(t-1)}|)$, if **any** of the following conditions are satisfied:

- $type_{uvw} = 1$
- $type_{uvw} = 2$ and $t > 1 + \frac{\alpha}{1-\alpha}k$
- $type_{uvw} = 2$ and $\alpha < 0.5$.

Proof From $t > k+1$ and $|\mathcal{E}_{\mathcal{R}}^{(t-1)}| = |\mathcal{E}^{(t-1)}| - k\alpha$, the following (in)equalities hold:

$$|\mathcal{E}^{(t-1)}| = t-1 > k, \tag{20}$$

$$m^{(t-1)} = \min(k, |\mathcal{E}^{(t-1)}|) = k, \tag{21}$$

$$z_{\mathcal{R}}^{(t-1)} = \min(k(1-\alpha), |\mathcal{E}_{\mathcal{R}}^{(t-1)}|) = k(1-\alpha). \tag{22}$$

First, Eqs. (18), (20), and (21) imply

$$\mathrm{Var}[x_{uvw}^{(t)}] = 0 < \frac{|\mathcal{E}^{(t-1)}|}{m^{(t-1)}} \times \frac{|\mathcal{E}^{(t-1)}|-1}{m^{(t-1)}-1} - 1,$$

which proves Eq. (19) when $type_{uvw} = 1$.

Second, we prove Eq. (19) when $type_{uvw} = 2$ and $t > 1 + \frac{\alpha}{1-\alpha}k$. From $t > 1 + \frac{\alpha}{1-\alpha}k$, $|\mathcal{E}^{(t-1)}| > \frac{\alpha}{1-\alpha}k$ and thus $\left(1 + \frac{k}{|\mathcal{E}^{(t-1)}|}\right)\alpha < 1$ hold. This and Eq. (20) imply

$$\left(1 + \frac{k}{|\mathcal{E}^{(t-1)}|}\right)\left(1 - \frac{k-1}{|\mathcal{E}^{(t-1)}|-1}\right)\alpha < 1 - \frac{k-1}{|\mathcal{E}^{(t-1)}|-1}.$$

This and Eq. (20) again imply

$$\begin{aligned}&\left(1 - \frac{k(k-1)}{|\mathcal{E}^{(t-1)}|(|\mathcal{E}^{(t-1)}|-1)}\right)\alpha \\ &\leq \left(1 + \frac{k}{|\mathcal{E}^{(t-1)}|}\right)\left(1 - \frac{k-1}{|\mathcal{E}^{(t-1)}|-1}\right)\alpha \\ &< 1 - \frac{k-1}{|\mathcal{E}^{(t-1)}|-1}.\end{aligned}$$

This is equivalent to

$$(k-1)(|\mathcal{E}^{(t-1)}| - k\alpha) < |\mathcal{E}^{(t-1)}|(|\mathcal{E}^{(t-1)}|-1)(1-\alpha),$$

which is again equivalent to

$$\frac{|\mathcal{E}^{(t-1)}| - k\alpha}{k(1-\alpha)} - 1 < \frac{|\mathcal{E}^{(t-1)}|}{k} \times \frac{|\mathcal{E}^{(t-1)}|-1}{k-1} - 1.$$

Combining this, $|\mathcal{E}^{(t-1)}| = |\mathcal{E}_{\mathcal{R}}^{(t-1)}| + k\alpha$, Eqs. (18) and (22) gives

$$\mathrm{Var}[x_{uvw}^{(t)}] = \frac{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|}{k(1-\alpha)} - 1 < \frac{|\mathcal{E}^{(t-1)}|}{k} \times \frac{|\mathcal{E}^{(t-1)}|-1}{k-1} - 1,$$

which proves Eq. (19).

Lastly, the same conclusion holds when $type_{uvw} = 2$ and $\alpha < 0.5$ since Eq. (20) and $\alpha < 0.5$ imply $|\mathcal{E}^{(t-1)}| > \frac{\alpha}{1-\alpha}k$, which with $type_{uvw} = 2$ corresponds to the second case, which we prove above.

Hence, Eq. (19) holds under any of the given conditions. □

Therefore, the superiority of $\mathrm{WRS}_{\mathrm{INS}}$ in terms of small $\tilde{\mathrm{Var}}[c^{(t)}]$ and $\tilde{\mathrm{Var}}[c_u^{(t)}]$ depends on the distribution of the types of triangles in real graph streams. In the experiment section, we show that the triangles of Type 1 and 2 are abundant enough in real graph streams, as suggested by the temporal locality, so that $\mathrm{WRS}_{\mathrm{INS}}$ is more accurate than $\mathrm{Triest}_{\mathrm{IMPR}}$.

Similarly, the abundance of triangles of Type 1 and 2 explains why $\mathrm{WRS}_{\mathrm{DEL}}$ is more accurate than its best competitors in fully dynamic graph streams, as shown experimentally in Sect. 6.

5.5 Complexity analysis

In this subsection, we prove the time and space complexities of WRS. Especially, we show that WRS has the same time and space complexities as the state-of-the-art algorithms [8,27]. We assume that sampled edges are stored in the adjacency list format in memory, as in our implementation used for our experiments. However, storing them sequentially, as in Fig. 4, does not change the results below.

5.5.1 Time complexity analysis

The worst-case time complexity of WRS is linear in the memory budget and in the number of edges in the input stream, as formalized in Theorem 3.

Theorem 3 (Worst-Case Time Complexity of WRS) *Processing each element in the input stream by Algorithms 3 and 4 takes $O(k)$, and thus processing t elements in the input stream takes $O(kt)$.*

Proof In both Algorithms 3 and 4, the most expensive step of processing an inserted or deleted edge $\Delta^{(t)} = (\{u, v\}, \delta^{(t)})$ is to find the common neighbors $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ (line 3 in Algorithm 3 and line 3 in Algorithm 4). Assume $|\hat{\mathcal{N}}_u| \leq |\hat{\mathcal{N}}_v|$, without loss of generality, and a hash table is used to store all elements of $\hat{\mathcal{N}}_v$. $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$ is obtained by traversing all elements of $\hat{\mathcal{N}}_u$ and checking if each of them is in the hash table. Since checking the membership of an element in a hash table takes $O(1)$ time, the overall time complexity of finding the common neighbors is $O(|\hat{\mathcal{N}}_u|) = O(\min(|\hat{\mathcal{N}}_u|, |\hat{\mathcal{N}}_v|))$ time. Except for computing $\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v$, the others steps for processing $\Delta^{(t)} = (\{u, v\}, \delta^{(t)})$ take $O(|\hat{\mathcal{N}}_u \cap \hat{\mathcal{N}}_v|) = O(\min(|\hat{\mathcal{N}}_u|, |\hat{\mathcal{N}}_v|))$ time in total. Since $O(\min(|\hat{\mathcal{N}}_u|, |\hat{\mathcal{N}}_v|)) = O(k)$, the time complexity of processing each element is $O(k)$. □

Notice that this analysis assuming the worst-case graph stream is pessimistic for real graph streams, where $\min(|\hat{\mathcal{N}}_u|, |\hat{\mathcal{N}}_v|)$ is usually much smaller than k.

5.5.2 Space complexity analysis

Theorem 4 gives the space complexity of WRS. Note that, except for the space for outputs (specifically, estimates of the local triangle counts), WRS only requires $O(k)$ space.

Theorem 4 (Space Complexity of WRS) *Let $\mathcal{V}^{(t)}$ be the set of nodes that appear in the first t elements in the input stream. Processing t elements in the input stream by Algorithms 3 and 4 requires $O(k)$ space for global triangle counting and $O(k + |\mathcal{V}^{(t)}|)$ space for local triangle counting.*

Proof Algorithms 3 and 4 use $O(k)$ space for sampling edges, and they use $O(|\mathcal{V}^{(t)}|)$ space for maintaining the estimates of local triangle counts, which need not be maintained for global triangle counting. □

6 Experiments

We review experiments that we designed to answer the following questions:

- **Q1. Accuracy**: How accurately does WRS estimate global and local triangle counts? Is WRS more accurate than its state-of-the-art competitors?
- **Q2. Trends**: How do true triangle counts, estimates, and estimation errors change over time?
- **Q3. Illustration of Theorems**: Does WRS give unbiased estimates with variances smaller than its competitors'?
- **Q4. Scalability**: How does WRS scale with the number of edges in input streams?
- **Q5. Effects of the Size of the Waiting Room**: How does the relative size α of the waiting room affect the accuracy of WRS? What is the optimal value of α?
- **Q6. Effects of Temporal Locality**: How does the degree of temporal locality affect the accuracy of WRS?

6.1 Experimental settings

Machine: We ran all experiments on a PC with a 3.60GHz Intel i7-4790 CPU and 32GB memory.
Data: The real graph streams used in our experiments are summarized in Table 3 with the following details:

- **ArXiv** [12]: A citation network between papers in ArXiv's High Energy Physics. Each edge $\{u, v\}$ represents that paper u cited paper v. We used the submission time of u as the creation time of $\{u, v\}$.
- **Facebook** [48]: A friendship network between users of Facebook. Each edge $\{u, v\}$ represents that user v appeared in the friend list of user u. The edges whose creation times are unknown were ignored.
- **Email** [21]: An email network from Enron Corporation. Each edge $\{u, v\}$ represents that employee u sent to or received from person v (who may be a non-employee) at least one email. We used the creation time of the first email between u and v as the creation time of $\{u, v\}$.
- **YouTube** [29]: A friend network between users of YouTube. Each edge $\{u, v\}$ indicates that user u and user v are friends with each other. The edges created before 12/10/2006 were ignored since their exact creation times are unknown.
- **Patent** [14]: A citation network between patents. Each edge $\{u, v\}$ indicates that patent u cited patent v. We used the time when u was granted as the creation time of $\{u, v\}$.
- **Forest Fire** (FF) [26]: The forest fire model is a graph generator that reflects several structural properties (heavy-tailed degree distributions, small diameters, communities, etc.) and temporal properties (densification and shrinking diameters, etc.) of real-world graphs. Generated graphs grow over time with new nodes and edges.
- **Barabási-Albert** (BA) [5]: The Barabási-Albert model is a graph generator that reflects several structural properties of real-world graphs (heavy-tailed degree distributions, small diameters, giant connected components, etc.). Generated graphs grow over time with new nodes and edges.
- **Erdős-Rényi** (ER) [10]: The Erdős-Rényi model is a graph generator $G(n, p)$ that produces a random graph with n vertices where each possible pair of nodes are connected by an edge independently with probability p. Since generated graphs remain static over time, we used them only for test the scalability of WRS in Sect. 6.5.

The self-loops, the duplicated edges, and the directions of the edges were ignored in all the graph streams. In insertion-only streams, edges were streamed in the order by which they are created, as assumed in Sect. 3.3. For fully dynamic streams, however, since edge deletions are not included in the original datasets, we created edge deletions as follows: (a) choose β% of the edges and create the deletions of them, (b) locate each deletion in a random position after the corresponding edge addition. Throughout this section, we report experimental results when β was set to 20. However, the results were consistent with other β values.
Implementations: We compared the family of WRS to Trièst$_{\text{IMPR}}$ [8], Trièst$_{\text{FD}}$ [8], Mascot [27], ThinkD$_{\text{FAST}}$ [38,40], and ThinkD$_{\text{ACC}}$ [38,40]. They are all single-pass streaming algorithms that estimate both global and local triangle counts within a limited memory budget, as briefly described in Sect. 2. We implemented all the methods in Java, and we treated negative estimates as 0.
Parameter settings: In WRS, the relative size α of the waiting room was set to 0.1 unless otherwise stated.

Evaluation measures: To measure the accuracy of triangle counting, we used the following metrics:

- **Global Error**: Let x be the number of the global triangles at the end of the input stream and $\hat{x}$ be an estimated value of x. Then, the global error is defined as $|x - \hat{x}|/(x + 1)$.
- **Local Error** [27]: Let x_u be the number of the local triangles of each node $u \in \mathcal{V}$ at the end of the input stream and $\hat{x}_u$ be its estimate. Then, the local error is defined as

$$\frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} \frac{|x_u - \hat{x}_u|}{x_u + 1}.$$

- **Rank Correlation**: Spearman's rank correlation coefficient [42] measures the similarity of (a) the ranking of the nodes in terms of the true local triangle counts and (b) their ranking in terms of the estimated local triangle counts, at the end of the input stream.

In the following experiments, we computed each evaluation metric 1, 000 times for each algorithm, unless otherwise stated, and reported the mean.[2]

6.2 Q1: Accuracy

Figures 5 and 6 show the accuracies of the considered methods in the insertion-only and fully dynamic graph streams, respectively, within different memory budgets. The memory budget k was changed from 50% of the number of elements in the input stream to 0.1% until no algorithm is better than settings all estimates to 0 (i.e., the dashed line) in terms of each evaluation measure.[3, 4]

In all the graph streams, WRS was most accurate in terms of all the evaluation metrics, regardless of the memory budget. The accuracy gaps between WRS and the second best method were especially large in the ArXiv, Patent, FF-Large, and BA-Small datasets, which showed a strong degree of temporal locality (see Sect. 6.7 for its effects on accuracy). In the ArXiv dataset, for example, WRS_{INS} gave up to *47% smaller local error* and *40% smaller global error* than the second-best method. For the same dataset, WRS_{DEL} gave up to *28% smaller local error* and *36% smaller global error* than the second-best approach. WRS was most accurate since its estimates are based on the largest number of discovered triangles, which intuitively mean more information. Specifically, due to its effective sampling scheme, WRS discovered up to 2.9× more triangles than its competitors while processing the same streams, as seen in Fig. 7a, d.

6.3 Q2: Trends

Figure 8 shows how (1) the estimate of the global triangle count, (2) the global error, (3) the local error, and (4) the number of discovered triangles changed over time in each algorithm. We set the memory budget (i.e., k) to the 10% of the number of elements in the input stream. The dashed line denotes the end of the stream. WRS was most accurate at all times both in terms of global and local error, and it discovered the largest number of triangles at all times. Since MASCOT and $\text{THINKD}_{\text{FAST}}$ gradually use more space over time, their global and local errors gradually decreased over time. The other algorithms were exact until the memory is full, and after that, their global and local errors gradually increased over time.

6.4 Q3: Illustration of theorems

We ran experiments illustrating our analyses in Sect. 5.4. Figures 1b, d, and 9 show the distributions and the 95% confidence intervals of 10, 000 estimates of the global triangle count in the ArXiv dataset obtained by different algorithms. We set the memory budget k to the 10% of the number of elements in the dataset. The average of the estimates of WRS was close to the true triangle count. Moreover, the estimates of WRS had smaller variances and confidence intervals than those of the competitors. These results are consistent with Theorems 1–2 and Lemma 5.

6.5 Q4: Scalability

We measured how the running time of WRS scales with the number of edges in the input streams. To measure the scalability independently of the speed of the input stream, we measured the time taken by WRS to process all the elements while ignoring the time taken to wait for the arrivals of elements. Figure 10a shows the results in graph streams that we created by sampling different numbers of edges in the Patent dataset. We set k to the 10% of the number of edges in the entire dataset. Each reported value is the average over 1, 000 runs. The same experiment was performed with several other datasets, and the results are shown in Fig. 10. In all the considered datasets, the running time of WRS scaled linearly with the number of edges. That is, the time taken by WRS to

[2] That is, for each measure, we computed the measure of the estimates on each trial, and then, we reported the mean of the computed values. We did not report each measure of the mean of the estimates obtained from all trials.

[3] For MASCOT and $\text{THINKD}_{\text{FAST}}$, we set the sampling probability p so that the expected number of sampled edges is equal to the memory budget.

[4] All the considered algorithms were always better than setting all estimates to zero in terms of global error and rank correlation. However, all the considered algorithms were not in terms local error when the memory budget k was extremely small, and thus, the variances of estimates of local triangle counts were very large.

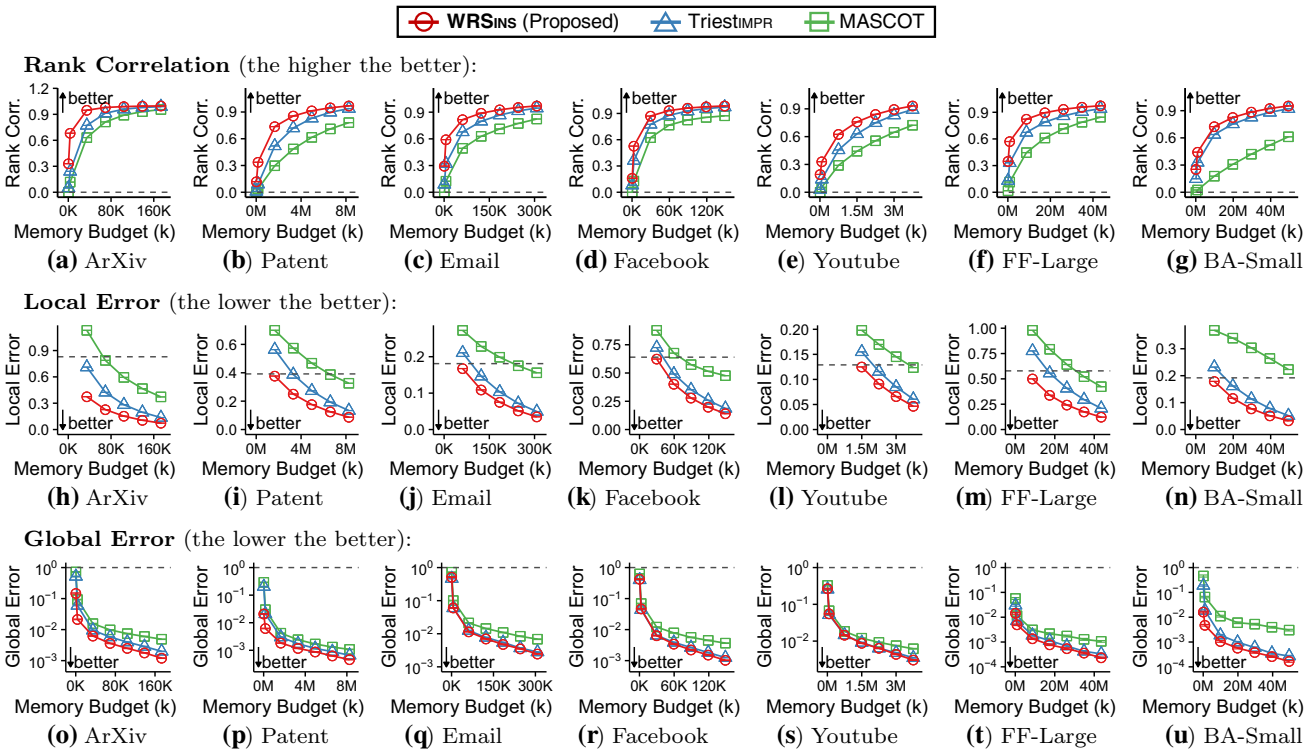


Fig. 5 WRS_{INS} is accurate. M: million, K: thousand. In all insertion-only graph streams, WRS_{INS} is most accurate for both global and local triangle counting, regardless of the memory budget k. The error bars indicate estimated standard errors

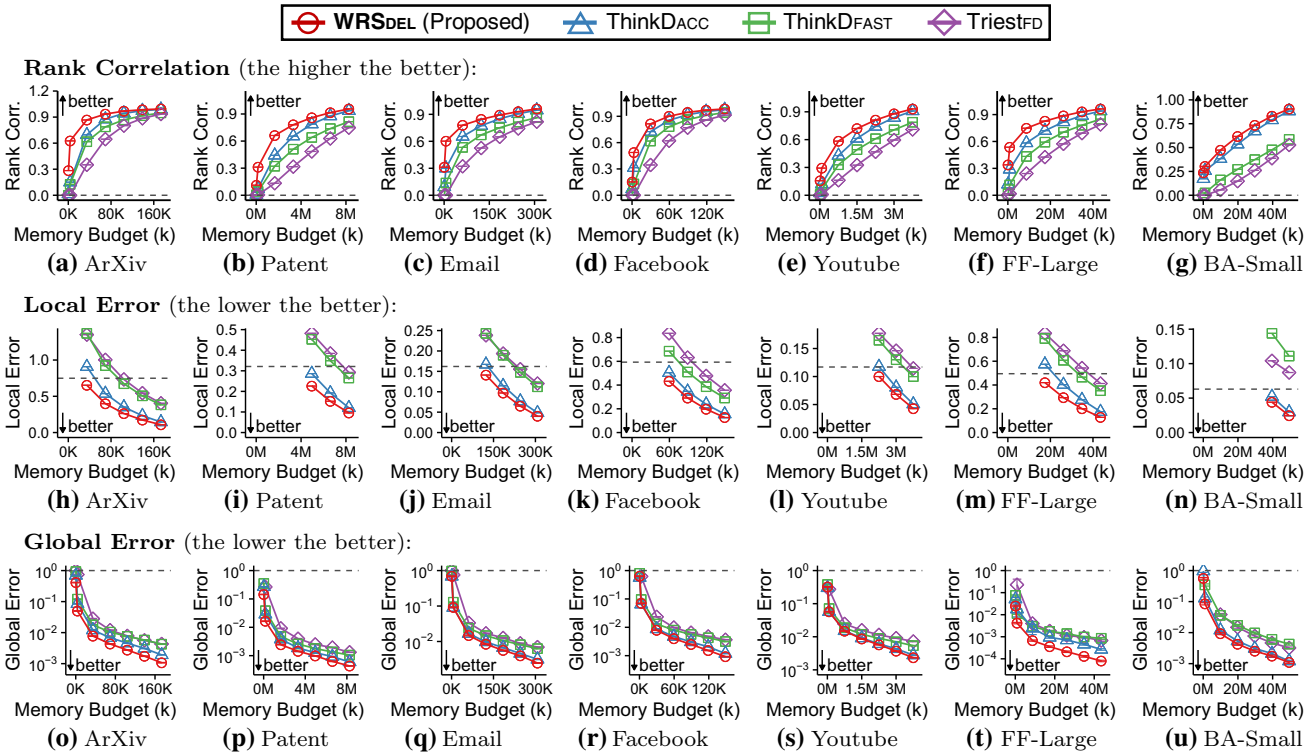
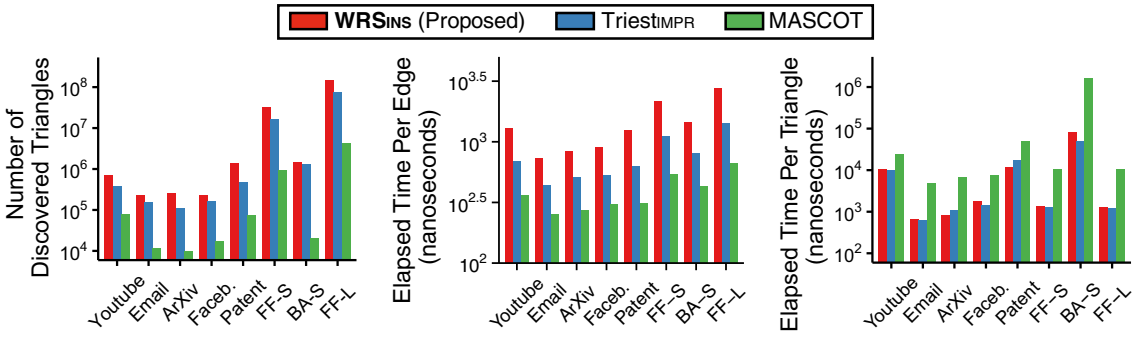


Fig. 6 WRS_{DEL} is accurate. M: million, K: thousand. In all fully dynamic graph streams, WRS_{DEL} is most accurate for both global and local triangle counting, regardless of the memory budget k. The error bars indicate estimated standard errors

Table 3 Summary of real-world graph streams

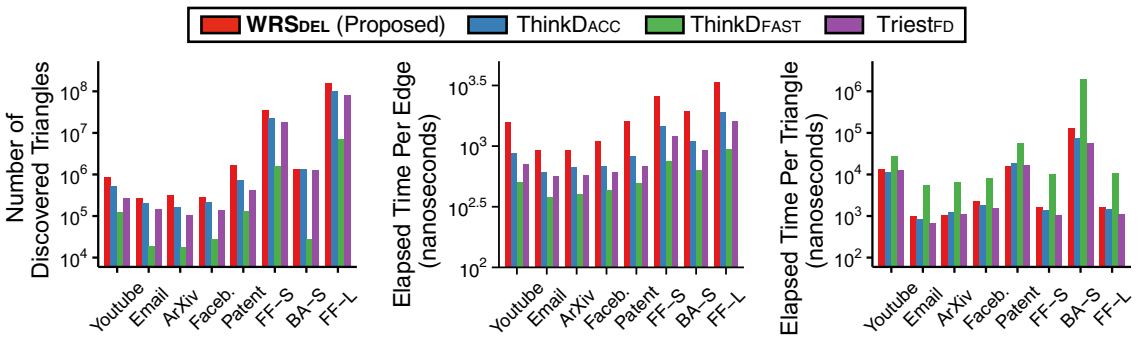
Name	# Nodes	# Edges	# Triangles	Description
ArXiv	30, 565	346, 849	988, 009	Citation
Facebook	61, 096	614, 797	1, 756, 259	Friendship
Email	86, 978	297, 456	1, 180, 387	Email
Youtube	3, 181, 831	7, 505, 218	7, 766, 821	Friendship
Patent	3, 774, 768	16, 518, 947	7, 515, 023	Citation
FF-Small	3, 000, 000	23, 345, 764	94, 648, 815	Synthetic
FF-Large	10, 000, 000	87, 085, 880	426, 338, 480	Synthetic
BA-Small	3, 000, 000	10^8	1, 958, 656	Synthetic
BA-Large	10, 000, 000	10^9	60, 117, 894	Synthetic
ER	1, 000, 000	10^{11}	1.333×10^{15}	Synthetic



(a) Number of Discovered Triangles (Insertion Only)

(b) Running Time (Insertion Only)

(c) Elapsed Time Per Triangle(Insertion Only)



(d) Number of Discovered Triangles (Fully Dynamic)

(e) Running Time (Fully Dynamic)

(f) Elapsed Time Per Triangle (Fully Dynamic)

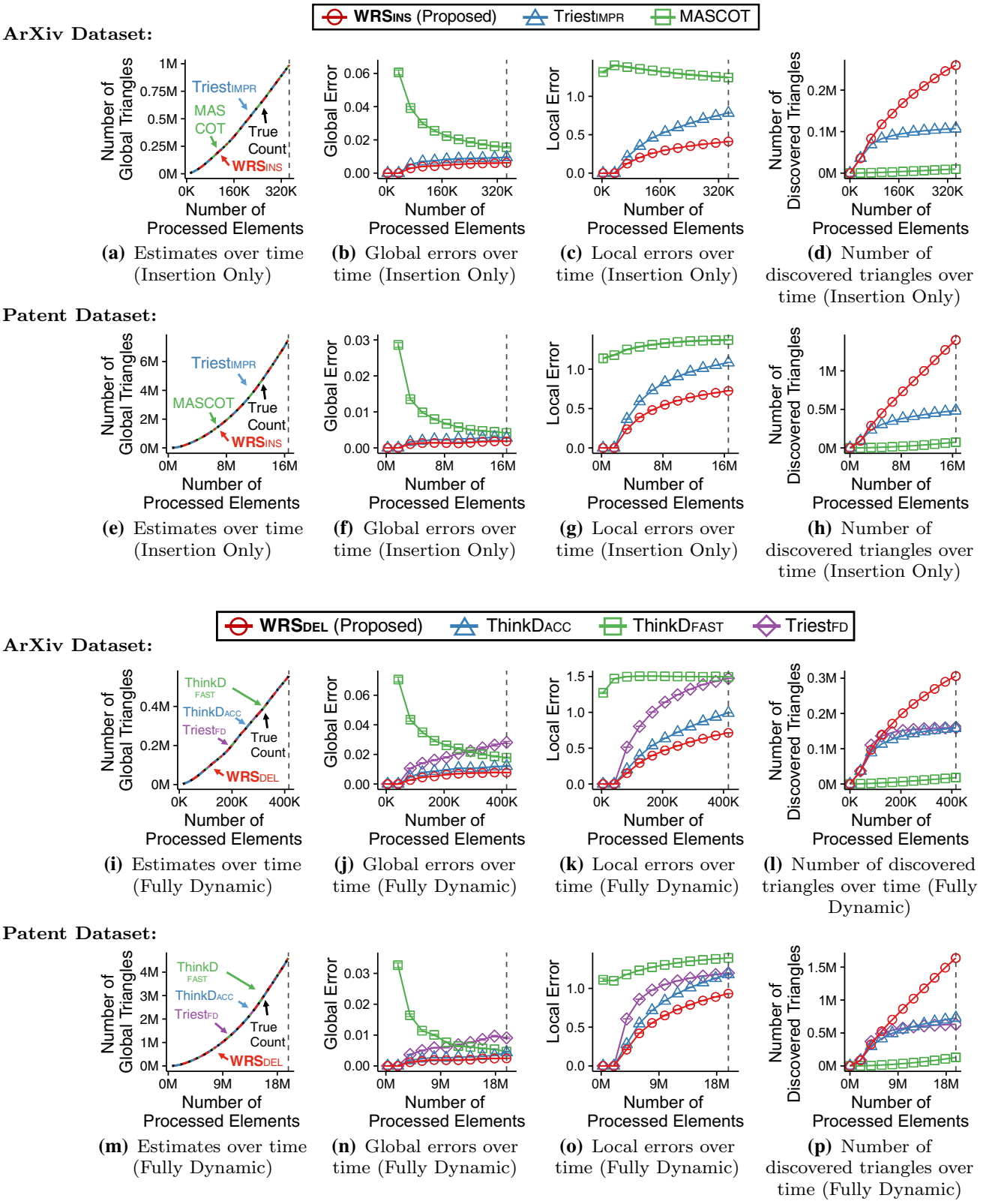
Fig. 7 The sampling scheme of WRS is effective. k is set to 10% of the number of the edges in each dataset, and α is set to 0.1. **a, d** WRS discovers up to 2.9× more triangles than the second best method, in the same streams. **b, e** The running times and the numbers of discovered triangles show similar trends. **c, f** WRS is comparable to its competitors in terms of running time per discovered triangle

process each element in the input stream was almost constant regardless of the number of edges that have arrived so far.

Despite its linear scalability, WRS took more time than its competitors to process the entire stream, as seen in Fig. 7b, e, mainly because it discovered and processed more triangles, as seen in Fig. 7a, d. In terms of running time per discovered triangle, WRS was comparable to the competitors, as seen in Fig. 7c, f.

6.6 Q5: Effects of the size of the waiting room

We measured how the accuracy of WRS depends on α, the relative size of the waiting room. Figure 11 shows the results



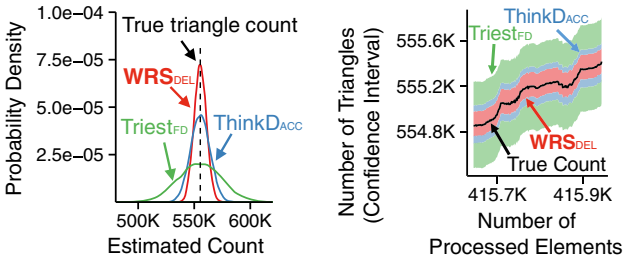
(a) Estimates over time (Insertion Only) **(b)** Global errors over time (Insertion Only) **(c)** Local errors over time (Insertion Only) **(d)** Number of discovered triangles over time (Insertion Only)

(e) Estimates over time (Insertion Only) **(f)** Global errors over time (Insertion Only) **(g)** Local errors over time (Insertion Only) **(h)** Number of discovered triangles over time (Insertion Only)

(i) Estimates over time (Fully Dynamic) **(j)** Global errors over time (Fully Dynamic) **(k)** Local errors over time (Fully Dynamic) **(l)** Number of discovered triangles over time (Fully Dynamic)

(m) Estimates over time (Fully Dynamic) **(n)** Global errors over time (Fully Dynamic) **(o)** Local errors over time (Fully Dynamic) **(p)** Number of discovered triangles over time (Fully Dynamic)

Fig. 8 WRS is 'any time' and accurate at all times. **a, e, i, m** WRS is 'any time,' maintaining estimates, while the input graph evolves. **b, c, f, g, j, k, n, o** WRS is most accurate at all times. **d, h, l, p** WRS discovers the most triangles at all times



(a) WRS_{DEL} is unbiased **(b)** 95% confidence intervals over time

Fig. 9 WRS_{DEL} is unbiased. **a** WRS_{DEL} provides unbiased estimates, and their variances are small. **b** WRS_{DEL} is the most accurate with the smallest confidence intervals over the entire stream. The ArXiv dataset is used for both (**a**) and (**b**)

with different memory budgets. Here, we used global error as the accuracy metric, and the average values over 1,000 runs were reported. In all the datasets, using proper amount of memory space for the waiting room gave better accuracy than using no space for the waiting room ($\alpha = 0$) and using half the space for the waiting room ($\alpha = 0.5$), regardless of memory budgets. Although proper α values depended on datasets and memory budgets, the accuracy was maximized when α was about 0.1 in most of the cases.

6.7 Q6: Effects of temporal locality

We investigated how the degree of temporal locality affects the effectiveness of WRS. To this end, we quantified the degree of temporal locality as the fraction of the number of Type 1 or 2 triangles to the number of all discovered triangles after processing the graph stream until the given memory space is full (i.e., after processing the first k non-deleted edges in the input graph stream). Table 4 shows the number of discovered triangles of each type and their proportions when we used WRS_{INS}. We set the memory budget k to 10% of the number of elements in each dataset, and we set α to 0.1, following the result of Sect. 6.6. In the ArXiv, Patent, FF-Large, and BA-Small datasets, more than 99.9% of the discovered triangles are of Type 1 or Type 2, whose closing intervals are smaller than the size of the waiting room. In these datasets, which show strong temporal locality, WRS_{INS} was especially more accurate than its competitors, as seen in Fig. 5h, i, m, n, o, p, t, u. The other datasets (i.e., the Email, Facebook, and YouTube datasets) show weaker temporal locality with smaller fractions of Type 1 or 2 triangles. In them, the accuracy gaps between WRS_{INS} and its competitors were relatively small, as seen in Fig. 5j, k, l, q, r, s.

Figure 12 shows how the global and local errors of WRS_{INS} and its best competitor (i.e., $\text{Triès}_{\text{IMPR}}$) changed according to the degree of temporal locality. To obtain graph streams with different degrees of temporal locality, we chose different fractions (from 10% to 50%) of the edges in the ArXiv dataset and swapped them with randomly cho-

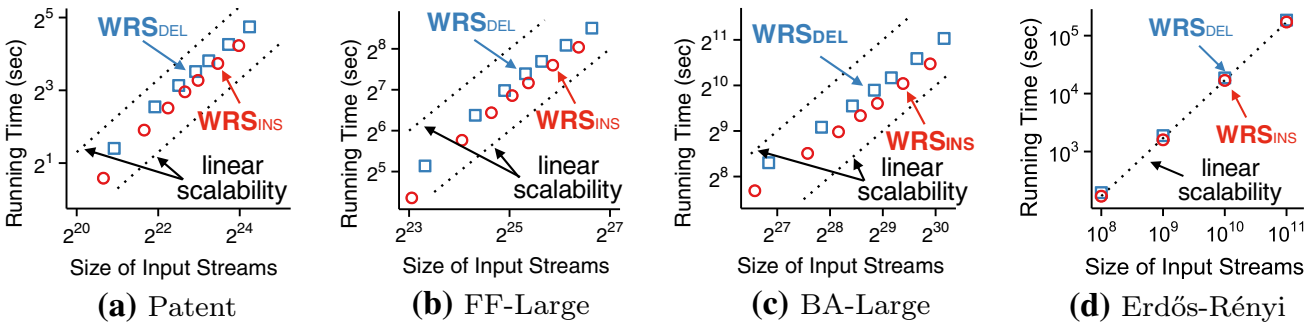


Fig. 10 WRS is scalable. The running time of WRS is linear in the number of edges in the input graph stream

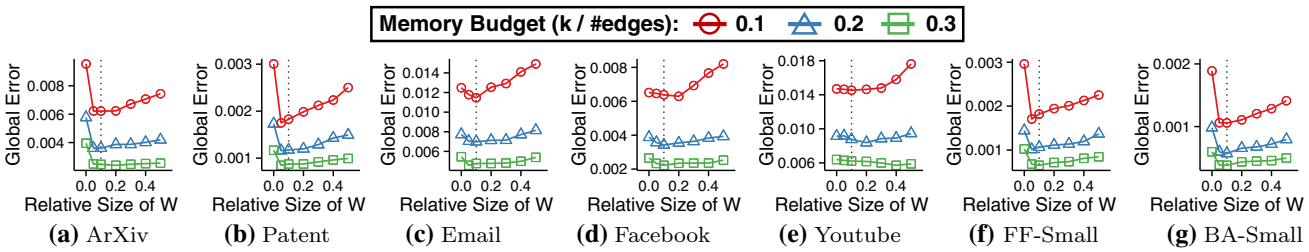
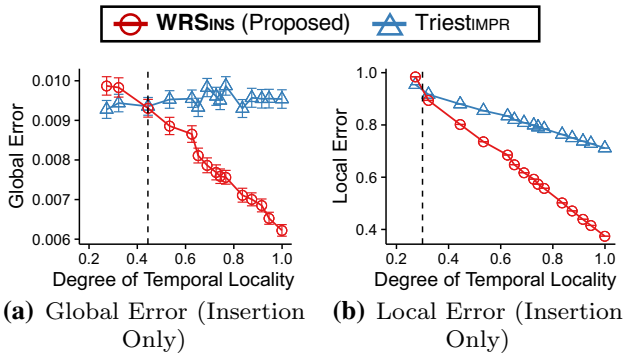


Fig. 11 Effects of α on the accuracy of WRS. In most cases, the accuracy of WRS is maximized when about 10% of memory space is used for the waiting room (i.e., $\alpha = 0.1$)

Table 4 The degree of temporal locality in the input stream affects the accuracy of WRS

Dataset	#Δ	#T1 (#T1/#Δ)	#T2 (#T2/#Δ)	#T3 (#T3/#Δ)	(#T1 + #T2)/#Δ
ArXiv	35, 937	6, 754 (**0.1879**)	29, 161 (**0.8114**)	22 (0.0006)	**0.9994**
Patent	92, 511	345 (**0.0037**)	92, 166 (**0.9963**)	0 (0)	**1.0000**
Email	57, 319	16, 363 (0.2855)	15, 477 (0.2700)	25, 479 (0.4445)	0.5555
Facebook	48, 831	8, 918 (0.1826)	22, 004 (0.4506)	17, 909 (0.3668)	0.6332
Youtube	112, 386	26, 954 (0.2398)	42, 466 (0.3779)	42, 966 (0.3823)	0.6177
FF-Large	38, 135, 620	13, 397, 029 (**0.3513**)	24, 738, 591 (**0.6487**)	0 (0)	**1.0000**
BA-Small	1, 131, 350	645, 321 (**0.5704**)	486, 029 (**0.4296**)	0 (0)	**1.0000**

The table below reports the number of triangles of each type and their proportions after processing each graph stream until the given memory space is full. The ArXiv, Patent, FF-Large, and BA-Small datasets, where WRS was especially more accurate than its competitors, show strong temporal locality with large fractions of Type 1 or 2 triangles. The other datasets show weaker temporal locality with smaller fractions of Type 1 or 2 triangles
Δ: all types of triangles when the given memory space is full.
T1: Type 1 triangles among Δ, T2: Type 2 triangles among Δ, T3: Type 3 triangles among Δ.
Note that the bold values represent the proportions of triangles in datasets that exhibit strong temporal locality



(a) Global Error (Insertion Only) **(b)** Local Error (Insertion Only)

Fig. 12 Effects of temporal locality on the accuracy of WRS_{INS} WRS_{INS} is significantly more accurate than its best competitor when the input graph stream exhibits strong temporal locality

sen edges.[5] WRS_{INS} was significantly more accurate than $\text{Triès}_{\text{IMPR}}$ when the input graph stream exhibited strong temporal locality. As the temporal locality became weaker, the accuracy gap decreased, and eventually $\text{Triès}_{\text{IMPR}}$ became more accurate than WRS_{INS}.

As shown in the results, the degree of temporal locality in the input stream affects the accuracy of WRS. Thus, if the degree of temporal locality in the input stream is not known in advance, we can first measure the fraction of Type 1 or 2 triangles until the given memory space is full, as in Table 4 and Fig. 12, and use WRS only if the fraction is high enough (e.g., the fraction is greater than 0.5). If the fraction is low, we can continue without the waiting room, which is equivalent to $\text{Triès}_{\text{IMPR}}$, or use any other method.

[5] Specifically, we ran the Knuth shuffle [22] while skipping different fractions of iterations.

7 Conclusion

In this work, we propose WRS, a family of single-pass streaming algorithms for global and local triangle counting in insertion-only and fully dynamic graph streams. WRS divides the memory space into the waiting room, where the latest edges are stored, and the reservoir, where the remaining edges are uniformly sampled. By doing so, WRS exploits the temporal locality in real graph streams for reducing variances, while giving unbiased estimates. To sum up, WRS has the following strengths:

- **Fast and 'any time'**: WRS scales linearly with the number of edges in the input graph stream, and it maintains estimates at any time, while the input graph evolves over time (Figs. 1, 9b).
- **Effective**: Estimation error in WRS is up to 47% *smaller* than that in its best competitors (Figs. 5, 6).
- **Theoretically sound**: WRS gives unbiased estimates with small variances under the temporal locality (Theorems 1, 2; Lemma 5; and Figs. 1, 9).

Reproducibility: The code and datasets used in the paper are available at http://dmlab.kaist.ac.kr/wrs/.

Acknowledgements This research was supported by Disaster-Safety Platform Technology Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (Grant Number: 2019M3D7A1094364) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)). This research was also supported by the National Science Foundation under Grant No. CNS-1314632 and IIS-1408924. This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National

Science Foundation, or other funding parties. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

A Appendix: Proof of Lemma 2

We provide a proof of Lemma 2, which is based on several properties of RP. We first introduce the uniformity of RP [38] in Lemma 6. Then, we present the mean and variance of the size of the reservoir $\mathcal{R}$ [13] in Lemma 7. Throughout this section, we use the superscript (t) to indicate the value of each variable after the t-th element $\Delta^{(t)}$ is processed. We let $\mathcal{E}_{\mathcal{R}}^{(t)}$ be the set of edges flowing to the reservoir $\mathcal{R}$ from the waiting room $\mathcal{W}$ at time t or earlier in Algorithm 4. We also let $y_{\mathcal{R}}^{(t)} = \min(k(1-\alpha), |\mathcal{E}_{\mathcal{R}}^{(t)}| + n_b^{(t)} + n_g^{(t)})$.

Lemma 6 (Uniformity of Random Pairing [38]) *At any time t, all equally sized subsets of the $\mathcal{E}_{\mathcal{R}}^{(t)}$ are equally likely to be a subset of the reservoir $\mathcal{R}^{(t)}$. Formally,*

$$\mathbb{P}[\mathcal{A} \subseteq \mathcal{R}^{(t)}] = \mathbb{P}[\mathcal{B} \subseteq \mathcal{R}^{(t)}], \quad \forall t \geq 1,\ \forall A \neq B \subset \mathcal{E}_{\mathcal{R}}^{(t)},\ s.t.\ |\mathcal{A}| = |\mathcal{B}|. \tag{23}$$

Lemma 7 (Expectation, Variance of the Reservoir Size of Random Pairing [13]) *The expected value and the variance of the size of the reservoir $\mathcal{R}$ at any time t in Algorithm 4 are formulated as follows:*

$$\mathbb{E}[|\mathcal{R}^{(t)}|] = \frac{|\mathcal{E}_{\mathcal{R}}^{(t)}|}{|\mathcal{E}_{\mathcal{R}}^{(t)}| + d^{(t)}} \cdot y_{\mathcal{R}}^{(t)}, \tag{24}$$

$$\mathrm{Var}[|\mathcal{R}^{(t)}|] = \frac{d^{(t)} \cdot y_{\mathcal{R}}^{(t)} \cdot (|\mathcal{E}_{\mathcal{R}}^{(t)}| + d^{(t)} - y_{\mathcal{R}}^{(t)}) \cdot |\mathcal{E}_{\mathcal{R}}^{(t)}|}{(|\mathcal{E}_{\mathcal{R}}^{(t)}| + d^{(t)})^2 \cdot (|\mathcal{E}_{\mathcal{R}}^{(t)}| + d^{(t)} - 1)}, \tag{25}$$

where $d^{(t)} = n_b^{(t)} + n_g^{(t)}$.

In Lemma 8, we formulate the probability that each edge in $\mathcal{E}_{\mathcal{R}}$ is stored in the reservoir $\mathcal{R}$ in $\mathrm{WRS_{DEL}}$.

Lemma 8 (Sampling Probability of Each Edge in Random Pairing) *At any time t, the probability that each edge in $\mathcal{E}_{\mathcal{R}}$ is stored in $\mathcal{R}$ after the t-th element is processed in Algorithm 4 is*

$$\mathbb{P}[\{u, v\} \in \mathcal{R}^{(t)}] = \frac{y_{\mathcal{R}}^{(t)}}{|\mathcal{E}_{\mathcal{R}}^{(t)}| + n_b^{(t)} + n_g^{(t)}}. \tag{26}$$

Proof Let $\mathbb{1}(\{u, v\} \in \mathcal{R}^{(t)})$ be a random variable that becomes 1 if $\{u, v\} \in \mathcal{R}^{(t)}$ and 0 otherwise. By definition,

$$|\mathcal{R}^{(t)}| = \sum_{\{u,v\} \in \mathcal{E}_{\mathcal{R}}^{(t)}} \mathbb{1}(\{u, v\} \in \mathcal{R}^{(t)}). \tag{27}$$

Then, by linearity of expectation and Eq. (27),

$$\begin{aligned}\mathbb{E}[|\mathcal{R}^{(t)}|] &= \sum_{\{u,v\} \in \mathcal{E}_{\mathcal{R}}^{(t)}} \mathbb{E}[\mathbb{1}(\{u, v\} \in \mathcal{R}^{(t)})] \\ &= \sum_{\{u,v\} \in \mathcal{E}_{\mathcal{R}}^{(t)}} \mathbb{P}[\{u, v\} \in \mathcal{R}^{(t)}].\end{aligned} \tag{28}$$

Then, Eq. (26) is obtained as follows:

$$\begin{aligned}\mathbb{P}[\{u, v\} \in \mathcal{R}^{(t)}] &= \frac{1}{|\mathcal{E}_{\mathcal{R}}^{(t)}|} \sum_{\{w,x\} \in \mathcal{E}_{\mathcal{R}}^{(t)}} \mathbb{P}[\{w, x\} \in \mathcal{R}^{(t)}] \\ &= \frac{\mathbb{E}[|\mathcal{R}^{(t)}|]}{|\mathcal{E}_{\mathcal{R}}^{(t)}|} = \frac{y_{\mathcal{R}}^{(t)}}{|\mathcal{E}_{\mathcal{R}}^{(t)}| + n_b^{(t)} + n_g^{(t)}},\end{aligned}$$

where the first, second, and last equalities are from Eqs. (23), (28) and (24), respectively. □

Proof of Lemma 2 We prove Lemma 2 based on Lemma 8. □

Proof Without loss of generality, we assume $e_{uvw}^{(1)} = \{v, w\}$, $e_{uvw}^{(2)} = \{w, u\}$, and $e_{uvw}^{(3)} = e^{(t)} = \{u, v\}$. That is, $\{v, w\}$ arrives earlier than $\{w, u\}$, and $\{w, u\}$ arrives earlier than $\{u, v\}$. When $e^{(t)} = \{u, v\}$ arrives at time t, the triangle $\{u, v, w\}$ is discovered if and only if both $\{v, w\}$ and $\{w, u\}$ are in $\mathcal{S}^{(t-1)}$.

Note that, since $\mathrm{WRS_{DEL}}$ updates the triangle counts before sampling or deleting edges, $\mathcal{E}_{\mathcal{R}} = \mathcal{E}_{\mathcal{R}}^{(t-1)}, n_b = n_b^{(t-1)}$ and $n_g = n_g^{(t-1)}$ when executing lines 6 and 8 of Algorithm 4.

If $type_{uvw} = 1$, $\{v, w\}$ and $\{w, u\}$ are always stored in $\mathcal{W}^{(t-1)}$, when $\{u, v\}$ arrives. Thus, $\mathrm{WRS_{DEL}}$ discovers $\{u, v, w\}$ with probability 1.

If $type_{uvw} = 2$, when $\{u, v\}$ arrives, $\{w, u\}$ is always stored in $\mathcal{W}^{(t-1)}$, while $\{v, w\}$ cannot be in $\mathcal{W}^{(t-1)}$ but can be in $\mathcal{R}^{(t-1)}$. For $\mathrm{WRS_{DEL}}$ to discover $\{u, v, w\}$, $\{v, w\}$ should be in $\mathcal{R}^{(t-1)}$, and from Eq. (26), the probability of the event is

$$\mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)}] = \frac{y_{\mathcal{R}}^{(t-1)}}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}| + d^{(t-1)}},$$

where $d^{(t-1)} = n_b^{(t-1)} + n_g^{(t-1)}$.

If $type_{uvw} = 3$, $\{v, w\}$ and $\{w, u\}$ cannot be in $\mathcal{W}^{(t-1)}$, when $\{u, v\}$ arrives. For $\mathrm{WRS_{DEL}}$ to discover $\{u, v, w\}$, both $\{v, w\}$ and $\{w, u\}$ should be in $\mathcal{R}^{(t-1)}$. The probability of the event is $\mathbb{P}[\{v, w\} \in \mathcal{R}^{(t-1)}$ and $\{w, u\} \in \mathcal{R}^{(t-1)}]$.

Below, we formulate this joint probability by expanding the covariance sum $\sum_{\{v,w\} \neq \{w,u\}} Cov\big(\mathbb{1}(\{v, w\} \in \mathcal{R}^{(t-1)}), \mathbb{1}(\{w, u\} \in \mathcal{R}^{(t-1)})\big)$ in two ways and compare them. Each random variable $\mathbb{1}(\{u, v\} \in \mathcal{R}^{(t-1)})$ is 1 if $\{u, v\} \in \mathcal{R}^{(t-1)}$ and 0 otherwise.

First, we expand the variance of $\mathcal{R}^{(t-1)}$. From Eq. (27),

$$Var\big[|\mathcal{R}^{(t-1)}|\big] = \sum_{\{u,v\}\in\mathcal{E}_{\mathcal{R}}^{(t-1)}} Var\big[\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)})\big] + \sum_{\{u,v\}\neq\{x,y\}} Cov\Big(\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)}), \mathbb{1}(\{x,y\}\in\mathcal{R}^{(t-1)})\Big),$$

and hence, the covariance sum is be expanded as

$$\sum_{\{u,v\}\neq\{x,y\}} Cov\Big(\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)}), \mathbb{1}(\{x,y\}\in\mathcal{R}^{(t-1)})\Big) = Var\big[|\mathcal{R}^{(t-1)}|\big] - \sum_{\{u,v\}\in\mathcal{E}_{\mathcal{R}}^{(t-1)}} Var\big[\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)})\big]. \tag{29}$$

From $Var[x] = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$, we have

$$Var\big[\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)})\big] = \mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\big] - \mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\big]^2. \tag{30}$$

Applying Eqs. (26) and (30) to Eq. (29) results in

$$\begin{aligned}
&\sum_{\{u,v\}\neq\{x,y\}} Cov\Big(\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)}), \mathbb{1}(\{x,y\}\in\mathcal{R}^{(t-1)})\Big) \\
&= Var\big[|\mathcal{R}^{(t-1)}|\big] - \sum_{\{u,v\}\in\mathcal{E}_{\mathcal{R}}^{(t-1)}} Var\big[\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)})\big] \\
&= Var\big[|\mathcal{R}^{(t-1)}|\big] \\
&\quad - |\mathcal{E}_{\mathcal{R}}^{(t-1)}| \cdot \frac{y_{\mathcal{R}}^{(t-1)} \cdot (|\mathcal{E}_{\mathcal{R}}^{(t-1)}| + d^{(t-1)} - y_{\mathcal{R}}^{(t-1)})}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}| + d^{(t-1)})^2}.
\end{aligned} \tag{31}$$

Then, we directly expand the covariance sum. With $Cov(x,y) = \mathbb{E}[xy] - \mathbb{E}[x]\cdot\mathbb{E}[y]$ and Eq. (26), the covariance sum is expanded as

$$\begin{aligned}
&\sum_{\{u,v\}\neq\{x,y\}} Cov\Big(\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)}), \mathbb{1}(\{x,y\}\in\mathcal{R}^{(t-1)})\Big) \\
&= \sum_{\{u,v\}\neq\{x,y\}} \Big(\mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big] \\
&\quad - \mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\big]\cdot\mathbb{P}\big[\{x,y\}\in\mathcal{R}^{(t-1)}\big]\Big) \\
&= \sum_{\{u,v\}\neq\{x,y\}} \mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big] \\
&\quad - \frac{y_{\mathcal{R}}^{(t-1)}\cdot y_{\mathcal{R}}^{(t-1)}\cdot|\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|-1)}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)})^2}.
\end{aligned} \tag{32}$$

Next, we obtain the sum of joint probabilities by comparing the two expansions [i.e., Eqs. (31) and (32)] as follows:

$$\begin{aligned}
&\sum_{\{u,v\}\neq\{x,y\}} \mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big] \\
&= \sum_{\{u,v\}\neq\{x,y\}} Cov\Big(\mathbb{1}(\{u,v\}\in\mathcal{R}^{(t-1)}), \mathbb{1}(\{x,y\}\in\mathcal{R}^{(t-1)})\Big) \\
&\quad + \frac{y_{\mathcal{R}}^{(t-1)}\cdot y_{\mathcal{R}}^{(t-1)}\cdot|\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|-1)}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)})^2} \\
&= Var\big[|\mathcal{R}^{(t-1)}|\big] \\
&\quad - |\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot\frac{y_{\mathcal{R}}^{(t-1)}\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)}-y_{\mathcal{R}}^{(t-1)})}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)})^2} \\
&\quad + \frac{y_{\mathcal{R}}^{(t-1)}\cdot y_{\mathcal{R}}^{(t-1)}\cdot|\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|-1)}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)})^2} \\
&= \frac{y_{\mathcal{R}}^{(t-1)}\cdot(y_{\mathcal{R}}^{(t-1)}-1)\cdot|\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|-1)}{(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)})\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)}-1)}.
\end{aligned} \tag{33}$$

Lastly, each joint probability $\mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big]$ is implied from Eqs. (23) and (33) as follows:

$$\begin{aligned}
&\mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big] \\
&= \frac{\sum_{\{u,v\}\neq\{x,y\}}\mathbb{P}\big[\{u,v\}\in\mathcal{R}^{(t-1)}\cap\{x,y\}\in\mathcal{R}^{(t-1)}\big]}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|\cdot(|\mathcal{E}_{\mathcal{R}}^{(t-1)}|-1)} \\
&= \frac{y_{\mathcal{R}}^{(t-1)}}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)}} \times \frac{y_{\mathcal{R}}^{(t-1)}-1}{|\mathcal{E}_{\mathcal{R}}^{(t-1)}|+d^{(t-1)}-1},
\end{aligned}$$

which proves Eq. (4) in Lemma 2 when $type_{uvw} = 3$. □

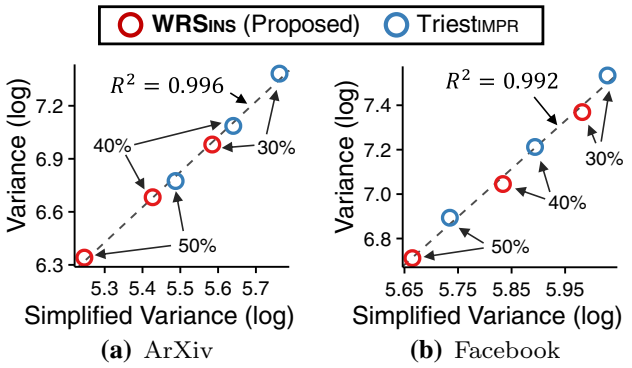


Fig. 13 The true and simplified variances are strongly correlated ($R^2 > 0.99$) on a log-log scale. Both true and simplified variances of WRS_{INS} are smaller than those of $\text{Triès}_{\text{IMPR}}$ within the same memory budget

B Appendix: Variance analysis

We measured the variance $\text{Var}[c^{(t)}]$ of the estimate of the global triangle count and the simplified version $\tilde{\text{Var}}[c^{(t)}]$ [i.e., Eq. (16)] while changing the memory budget k from 30% to 50% of the number of elements in the input stream. As seen in Fig. 13, while there is a clear difference between the true and simplified variances, the two values are strongly correlated ($R^2 > 0.99$) on a log-log scale both in the ArXiv and Facebook datasets. This strong correlation supports the validity of

our simple and illuminating analysis in Sect. 5.4.2, where we compare the simplified variances of WRS_{INS} and $\text{Triès}_{\text{IMPR}}$, instead of their true variances, to provide an intuition why WRS_{INS} is more accurate than $\text{Triès}_{\text{IMPR}}$.

References

1. Ahmed, N.K., Duffield, N., Neville, J., Kompella, R.: Graph sample and hold: a framework for big-graph analytics. In: KDD, pp. 1446–1455 (2014)
2. Ahmed, N.K., Duffield, N., Willke, T.L., Rossi, R.A.: On sampling from massive graph streams. PVLDB **10**(11), 1430–1441 (2017)
3. Arifuzzaman, S., Khan, M., Marathe, M.: Patric: A parallel algorithm for counting triangles in massive networks. In: CIKM, pp. 529–538 (2013)
4. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: SODA, pp. 623–632 (2002)
5. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
6. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient algorithms for large-scale local triangle counting. ACM Trans. Knowl. Discov. Data **4**(3), 13 (2010)
7. Brown, P.G., Haas, P.J.: Techniques for warehousing of sample data. In: ICDE, pp. 6–6 (2006)
8. De Stefani, L., Epasto, A., Riondato, M., Upfal, E.: Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Trans. Knowl. Discov. Data **11**(4), 43 (2017)
9. Eckmann, J.P., Moses, E.: Curvature of co-links uncovers hidden thematic layers in the world wide web. PNAS **99**(9), 5825–5829 (2002)
10. Erdös, P.: On the structure of linear graphs. Isr. J. Math. **1**(3), 156–160 (1963)
11. Etemadi, R., Lu, J., Tsin, Y.H.: Efficient estimation of triangles in very large graphs. In: CIKM, pp. 1251–1260 (2016)
12. Gehrke, J., Ginsparg, P., Kleinberg, J.: Overview of the 2003 kdd cup. ACM SIGKDD Explor. Newsl. **5**(2), 149–151 (2003)
13. Gemulla, R., Lehner, W., Haas, P.J.: Maintaining bounded-size sample synopses of evolving datasets. VLDB J. **17**(2), 173–201 (2008)
14. Hall, B.H., Jaffe, A.B., Trajtenberg, M.: The nber patent citation data file: Lessons, insights and methodological tools. Tech. rep., National Bureau of Economic Research (2001)
15. Han, G., Sethu, H.: Edge sample and discard: a new algorithm for counting triangles in large dynamic graphs. In: ASONAM, pp. 44–49 (2017)
16. Hu, X., Tao, Y., Chung, C.W.: I/O-efficient algorithms on triangle listing and counting. ACM Trans. Database Syst. **39**(4), 27 (2014)
17. Jha, M., Seshadhri, C., Pinar, A.: A space efficient streaming algorithm for triangle counting using the birthday paradox. In: KDD, pp. 589–597 (2013)
18. Jung, M., Lim, Y., Lee, S., Kang, U.: Furl: Fixed-memory and uncertainty reducing local triangle counting for multigraph streams. Data Min. Knowl. Discov. **33**(5), 1225–1253 (2019)
19. Kallaugher, J., Price, E.: A hybrid sampling scheme for triangle counting. In: SODA, pp. 1778–1797 (2017)
20. Kim, J., Han, W.S., Lee, S., Park, K., Yu, H.: Opt: A new framework for overlapped and parallel triangulation in large-scale graphs. In: SIGMOD, pp. 637–648 (2014)
21. Klimt, B., Yang, Y.: Introducing the enron corpus. In: CEAS (2004)
22. Knuth, D.E.: Seminumerical algorithms. Art Comput. Program. **2**, 139–140 (1997)
23. Kolountzakis, M.N., Miller, G.L., Peng, R., Tsourakakis, C.E.: Efficient triangle counting in large graphs via degree-based vertex partitioning. In: WAW, pp. 15–24 (2010)
24. Kutzkov, K., Pagh, R.: On the streaming complexity of computing local clustering coefficients. In: WSDM, pp. 677–686 (2013)
25. Kutzkov, K., Pagh, R.: Triangle counting in dynamic graph streams. In: SWAT, pp. 306–318 (2014)
26. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: densification and shrinking diameters. ACM Trans. Knowl. Discov. Data **1**(1), 2 (2007)
27. Lim, Y., Kang, U.: Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In: KDD, pp. 685–694 (2015)
28. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: homophily in social networks. Annu. Rev. Sociol. **27**(1), 415–444 (2001)
29. Mislove, A.: Online social networks: measurement, analysis, and applications to distributed information systems. Ph.D. thesis, Rice University (2009)
30. Newman, M.E.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
31. Park, H.M., Myaeng, S.H., Kang, U.: Pte: Enumerating trillion triangles on distributed systems. In: KDD, pp. 1115–1124 (2016)
32. Park, H.M., Silvestri, F., Kang, U., Pagh, R.: Mapreduce triangle enumeration with guarantees. In: CIKM, pp. 1739–1748 (2014)
33. Pavan, A., Tangwongsan, K., Tirthapura, S., Wu, K.L.: Counting and sampling triangles from a graph stream. PVLDB **6**(14), 1870–1881 (2013)
34. Seshadhri, C., Pinar, A., Kolda, T.G.: Triadic measures on graphs: the power of wedge sampling. In: SDM, pp. 10–18 (2013)
35. Shin, K.: Wrs: Waiting room sampling for accurate triangle counting in real graph streams. In: ICDM, pp. 1087–1092 (2017)
36. Shin, K., Eliassi-Rad, T., Faloutsos, C.: Patterns and anomalies in k-cores of real-world graphs with applications. Knowl. Inf. Syst. **54**(3), 677–710 (2018)
37. Shin, K., Hammoud, M., Lee, E., Oh, J., Faloutsos, C.: Tri-fly: Distributed estimation of global and local triangle counts in graph streams. In: PAKDD, pp. 651–663 (2018)
38. Shin, K., Kim, J., Hooi, B., Faloutsos, C.: Think before you discard: accurate triangle counting in graph streams with deletions. In: ECML/PKDD, pp. 141–157 (2018)
39. Shin, K., Lee, E., Oh, J., Hammoud, M., Faloutsos, C.: Cocos: Fast and accurate distributed triangle counting in graph streams. arXiv preprint arXiv:1802.04249 (2018)
40. Shin, K., Oh, S., Kim, J., Hooi, B., Faloutsos, C.: Fast, accurate and provable triangle counting in fully dynamic graph streams. ACM Trans. Knowl. Discov. Data **14**(2), 1–39 (2020)
41. Shun, J., Tangwongsan, K.: Multicore triangle computations without tuning. In: ICDE, pp. 149–160 (2015)
42. Spearman, C.: The proof and measurement of association between two things. Am. J. Psychol. **15**(1), 72–101 (1904)
43. Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: WWW, pp. 607–614 (2011)
44. Tsourakakis, C.E.: Fast counting of triangles in large real networks without counting: algorithms and laws. In: ICDM, pp. 608–617 (2008)
45. Tsourakakis, C.E., Kang, U., Miller, G.L., Faloutsos, C.: Doulion: counting triangles in massive graphs with a coin. In: KDD, pp. 837–846 (2009)
46. Turk, A., Türkoglu, D.: Revisiting wedge sampling for triangle counting. In: TheWebConf, pp. 1875–1885 (2019)
47. Türkoglu, D., Turk, A.: Edge-based wedge sampling to estimate triangle counts in very large graphs. In: ICDM, pp. 455–464 (2017)
48. Viswanath, B., Mislove, A., Cha, M., Gummadi, K.P.: On the evolution of user interaction in facebook. In: WOSN, pp. 37–42 (2009)

49. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)
50. Wang, P., Qi, Y., Sun, Y., Zhang, X., Tao, J., Guan, X.: Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. PVLDB **11**(2), 162–175 (2017)
51. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications, vol. 8. Cambridge University Press, Cambridge (1994)
52. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-world networks. Nature **393**(6684), 440–442 (1998)

ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages

MATTEO RIONDATO, Two Sigma Investments, LP
ELI UPFAL, Brown University

ΑΒΡΑΞΑΣ (ABRAXAS): Gnostic word of mystic meaning.

We present ABRA, a suite of algorithms to compute and maintain probabilistically guaranteed high-quality approximations of the betweenness centrality of all nodes (or edges) on both static and fully dynamic graphs. Our algorithms use progressive random sampling and their analysis rely on Rademacher averages and pseudodimension, fundamental concepts from statistical learning theory. To our knowledge, ABRA is the first application of these concepts to the field of graph analysis. Our experimental results show that ABRA is much faster than exact methods, and vastly outperforms, in both runtime number of samples, and accuracy, state-of-the-art algorithms with the same quality guarantees.

CCS Concepts: • **Mathematics of computing → Probabilistic algorithms**; • **Human-centered computing → Social networks**; • **Theory of computation → Shortest paths**; **Dynamic graph algorithms**; **Sketching and sampling**; **Sample complexity and generalization bounds**;

Additional Key Words and Phrases: Centrality measures, pseudodimension, statistical learning theory, uniform bounds

ACM Reference format:
Matteo Riondato and Eli Upfal. 2018. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *ACM Trans. Knowl. Discov. Data.* 12, 5, Article 61 (July 2018), 38 pages.
https://doi.org/10.1145/3208351

1 INTRODUCTION

Centrality measures are fundamental concepts in graph analysis: they assign to each node or edge a score that quantifies some notion of the importance of the node/edge in the network [40]. Betweenness Centrality (BC) is a very popular centrality measure that, informally, defines the importance of a node or edge z in the network as proportional to the fraction of shortest paths (SPs) in the network that go through z [3, 19] (see Section 3 for formal definitions).

Brandes [14] presented an algorithm (denoted BA) to compute the exact BC values for all nodes or edges in a graph $G = (V, E)$ in time $O(|V||E|)$ if the graph is unweighted, or time

A preliminary version of this work appeared in the proceedings of ACM KDD'16 as [49].
This work was supported in part by NSF grant IIS-1247581 (https://www.nsf.gov/awardsearch/showAward?AWD_ID=1247581) and NIH grant R01-CA180776 (https://projectreporter.nih.gov/project_info_details.cfm?icde=0&aid=8685211), and by funding from Two Sigma Investments, LP. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflects the views of Two Sigma Investments, LP, or the National Science Foundation.
Authors' addresses: M. Riondato, Two Sigma Investments, LP, 100 Avenue of the Americas, 16th Fl. New York, NY 10013; email: matteo@twosigma.com; E. Upfal, Brown University, Department of Computer Science, 115 Waterman St. Providence, RI 02912; email: eli@cs.brown.edu.

https://doi.org/10.1145/3208351

Table 1. Variants of ABRA

Variant	Description
ABRA-s	Progressive sampling algorithm for static graphs (Section 4.1)
ABRA-s-set	Sampling algorithm for a specific set of nodes (Section 4.1.3)
ABRA-s-fix	Fixed-size sampling algorithm for static graphs (Section 6.4)
ABRA-s-k	Sampling algorithm for the top-k nodes with highest BC (Section 4.5)
ABRA-d	Sampling algorithm for fully-dynamic graphs (Section 5)

$O(|V||E| + |V|^2 \log |V|)$ if the graph has positive weights. The cost of BA is excessive on modern networks with millions of nodes and tens of millions of edges. Moreover, having the exact BC values may often not be needed, given the exploratory nature of the task. A high-quality approximation of the values is usually sufficient, provided it comes with stringent guarantees.

Today's networks are not only large, but also *dynamic*: edges are added and removed continuously. Keeping the BC values up-to-date after edge insertions and removals is a challenging task, and proposed algorithms [21, 28, 32, 33, 38, 39, 44] may improve the running time for some specific class of input graphs and update models, but in general cannot offer worst-case time and space complexities better than from-scratch-recomputation using BA [1]. Maintaining a high-quality approximation up-to-date is more feasible and more *sensible*: there is little informational gain in keeping track of exact BC values that change continuously.

Contributions. We focus on developing algorithms for approximating the BC of all nodes and edges in static and dynamic graphs. Our contributions are the following.

— We present ABRA (for "Approximating Betweenness with Rademacher Averages"), the first family of algorithms based on *progressive sampling* for approximating the BC of all nodes in static and dynamic graphs, where node and edge insertions and deletions are allowed. The BC approximations computed by ABRA are *probabilistically guaranteed* to be within a user-specified additive error ε from their exact values. We also present variants using a fixed amount of samples, with relative error (i.e., within a multiplicative factor ε of the true value) for the top-k nodes with highest BC, and variants that use refined estimators to give better approximations with a slightly larger sample size. Additionally, we also show a *fixed* sampling variant that performs exactly as many sample operations as requested by the user. Table 1 shows a summary of these variants.
— Our analysis relies on Rademacher averages [29, 51] and pseudodimension [43], fundamental concepts from the field of statistical learning theory [53]. Building on known and novel results using these concepts, ABRA computes the approximations without having to keep track of any global property of the graph, in contrast with existing algorithms [8, 10, 47]. A byproduct of our analysis is new general results on pseudodimension (Lemmas 3.7 and 3.8), which show properties that can be used to bound the pseudodimension of *any* problem. ABRA performs only "real work" toward the computation of the approximations, without having to obtain such global properties or update them after modifications of the graph. To the best of our knowledge, ours is the first application of Rademacher averages and pseudodimension to graph analysis problems, and the first to use *progressive* random sampling for BC computation. Using pseudodimension, we derive new analytical results on the sample complexity of the BC computation task (see Table 2 for our results and a comparison with existing bounds) generalizing previous contributions [47], and formulating a conjecture on the connection between pseudodimension and the distribution of SP lengths. Our

Table 2. Comparison of Sample-Based Algorithms for BC Estimation on Graphs

Works	Sample space	Sample size for ε-approximation* with confidence $\geq 1-\delta$	Analysis techniques
[15, 24, 25]	Nodes	$O\left(\frac{1}{\varepsilon^2}\left(\ln\lvert V\rvert + \ln\frac{1}{\delta}\right)\right)$	Hoeffding's inequality, union bound
[9, 47]	Shortest paths	$O\left(\frac{1}{\varepsilon^2}\left(\log_2 \mathrm{VD}(G) + \ln\frac{1}{\delta}\right)\right)^{\dagger}$	VC dimension
This work	Pairs of nodes	Variable, at most $O\left(\frac{1}{\varepsilon^2}\left(\log_2 \mathrm{L}(G) + \ln\frac{1}{\delta}\right)\right)^{\ddagger}$ but usually much less	Rademacher averages, pseudodimension

*See Definition 3.2 for the formal definition.
†$\mathrm{VD}(G)$ is the vertex-diameter of the graph G.
‡$\mathrm{L}(G)$ is the size of the largest weakly connected component of G. See Section 4.2 for tighter bounds.

work hence also showcases the usefulness of these highly theoretical concepts developed in the setting of supervised learning to develop practical algorithms for important problems in unsupervised settings.
— The results of our experimental evaluation on real networks show that ABRA outperforms, in both speed, number of samples, and accuracy the state-of-the-art methods offering the same guarantees [47], and it is significantly faster than exact methods [14].

The present paper extends the conference version [49] along multiple directions. The most significant new contributions are the following:

— A revised version of the algorithm with a new proof of correctness, where we fixed a subtle mistake in the algorithm presented in the conference version, due to the presence of randomly stopped sequences of random variables.
— A new variant of the algorithm using a fixed amount of samples (instead of progressive sampling), which returns much better approximations than previously existing algorithms using a fixed amount of samples [15, 47].
— A stricter bound to the maximum approximation error, which allows ABRA's stopping condition to be satisfied at smaller sample sizes than before.
— A completely new upper bound to the number of samples needed by ABRA to compute an approximation of the desired quality, which allows ABRA to deterministically stop after the number of samples suggested by the upper bound. This upper bound is based on pseudodimension [43], and we show upper and, in some cases, matching lower bounds to the pseudodimension of the problem of estimating betweenness centralities, shedding new light on its sample complexity. We additionally formulate an open conjecture (see Conjecture 4.9) that we show true for fundamental specific cases, and, if proved true, would greatly reduce the needed number of samples.
— We also reworked our algorithm for relative-error approximation of the top-k highest betweenness values, improving its stopping condition so it will use fewer samples.
— We also present all the proofs of our theoretical results, and additional experimental results, which give insights to the betweenness estimation problem and to the behavior of our algorithms. Moreover, we have added examples throughout the text, with the goal of improving the clarity of the presentation and to make the paper more self-contained.

Outline. We discuss related works in Section 2. The formal definitions of the concepts we use in the work can be found in Section 3. Our algorithms for approximating BC on static graphs are presented in Section 4, while the dynamic case is discussed in Section 5. The results of our extensive experimental evaluation are presented in Section 6. We draw conclusions in Section 7.

2 RELATED WORK

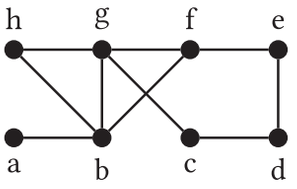
The definition of BC comes from the sociology literature [3, 19], but the study of efficient algorithms to compute it started only when graphs of substantial size became available to the analysts, following the emergence of the Web. The BA algorithm by Brandes [14] is currently the asymptotically fastest algorithm for computing the exact BC values for all nodes in the network. A number of works also explored heuristics to improve BA [18, 50], but retained the same worst-case time complexity.

The use of random sampling to approximate the BC values in static graphs was proposed independently by Jacob et al. [25] and Brandes and Pich [15], and successive works explored the tradeoff space of sampling-based algorithms [8–10, 47]. Other works focused on estimating the BC of a single target node, rather than on obtaining uniform guarantees for all the nodes [6, 26]. We focus here on related works that offer approximation guarantees similar to ours. For an in-depth discussion of previous contributions approximating BC on static graphs but not offering guarantees, we refer the reader to the comments by Riondato and Kornaropoulos [47, Section 2]. Table 2 shows a comparison of the sample space, sample size, and analysis techniques for the different works discussed in this section.

Riondato and Kornaropoulos [46, 47] present algorithms that employ the Vapnik–Chervonenkis (VC) dimension [53] to compute what is currently the tightest upper bound on the sample size sufficient to obtain guaranteed approximations of the BC of all nodes in a static graph. Their algorithms offer the same guarantees as ABRA but, to compute the sample size, they need to compute an upper bound on a characteristic quantity of the graph (the vertex-diameter, namely the maximum number of nodes on any SP). A progressive sampling algorithm based on the vertex-diameter was recently introduced [11]. Thanks to our use of Rademacher averages in a progressive random sampling setting, ABRA does not need to compute any characteristic quantity of the graph, and instead uses an efficient-to-evaluate stopping condition to determine when the approximated BC values are close to the exact ones. This allows ABRA to use smaller samples and be much faster than the algorithms by Riondato and Kornaropoulos [47].

A number of works [21, 28, 32, 33, 38, 39, 44] focused on computing the *exact* BC for all nodes in a dynamic graph, taking into consideration different update models. None of these algorithm is provably asymptotically faster than a complete computation from scratch using Brandes' algorithm [14] on general graphs (some of them are faster than BA on some specific classes of input and under some specific update models), and they all require significant amount of space (more details about these works can be found in [8, Section 2]). In contrast, Bergamini and Meyerhenke [8, 9] built on the work by Riondato and Kornaropoulos [47] to derive an algorithm for maintaining high-quality approximations of the BC of all nodes when the graph is dynamic and both additions and deletions of edges are allowed. Due to the use of the algorithm by Riondato and Kornaropoulos [47] as a building block, the algorithm must keep track of the vertex-diameter after an update to the graph. Our algorithm for dynamic graphs, instead, does not need this piece of information, and therefore can spend more time in computing the approximations, rather than in keeping track of global properties of the graph.

Hayashi et al. [24] recently proposed a data structure called *Hypergraph Sketch* to maintain the SP DAGs between pairs of nodes following updates to the graph. Their algorithm uses random sampling and this novel data structure allows them to maintain a high-quality, probabilistically



(a) Example graph

(b) BC values

Vertex v	a	b	c	d	e	f	g	h
$\mathrm{b}(v)$	0	0.125	0.0536	0.0179	0.0357	0.0893	0.125	0

Fig. 1. Example of BC.

guaranteed approximation of the BC of all nodes in a dynamic graph. Their guarantees come from an application of the simple uniform deviation bounds (i.e., the union bound) to determine the sample size, as previously done by Jacob et al. [25] and Brandes and Pich [15]. As a result, the resulting sample size is excessively large, as it depends on the *number of nodes in the graph*. Our improved analysis using the Rademacher averages allows us to develop an algorithm that uses the Hypergraph Sketch with a much smaller number of samples, and is therefore faster.

Progressive random sampling with Rademacher Averages has been used by Elomaa and Kääriäinen [17] and Riondato and Upfal [48] in completely different settings, i.e., to train classification trees and to mine frequent itemsets, respectively.

3 PRELIMINARIES

We now introduce the formal definitions and basic results that we use throughout the paper.

3.1 Graphs and Betweenness Centrality

Let $G = (V, E)$ be a graph. G may be directed or undirected and may have nonnegative weights on the edges. For any ordered pair (u, v) of different nodes $u \neq v$, let $\mathcal{S}_{uv}$ be the set of SPs from u to v, and let $\sigma_{uv} = |\mathcal{S}_{uv}|$. Given a path p between two nodes $u, v \in V$, a node $w \in V$ is *internal to* p if and only if $w \neq u$, $w \neq v$, and p goes through w. We denote as $\sigma_{uv}(w)$ the number of SPs from u to v that w is internal to.

Definition 3.1 (BC [3, 19]). Given a graph $G = (V, E)$, the BC *of a node* $w \in V$ is defined as

$$\mathrm{b}(w) = \frac{1}{|V|(|V|-1)} \sum_{\substack{(u,\, v) \in V \times V \\ u \neq v}} \frac{\sigma_{uv}(w)}{\sigma_{uv}} \quad (\in [0, 1]).$$

An example of a graph and the associated BC values, taken from [47, Section 3] is shown in Figure 1.[1]

Many variants of BC have been proposed in the literature, including, e.g., one for edges [40] and one limited to random walks of a fixed length [31]. Our results can be extended to many of these variants, following the same discussion as in [47, Section 6].

In this work, we focus on computing an *ε-approximation* of the collection $B = \{\mathrm{b}(w), w \in V\}$.

Definition 3.2 (ε-approximation). Given $\varepsilon \in (0, 1)$, an *ε-approximation to* B is a collection

$$\widetilde{B} = \left\{\widetilde{\mathrm{b}}(w), w \in V\right\},$$

such that, for all $w \in V$,

$$\left|\tilde{\mathrm{b}}(w) - \mathrm{b}(w)\right| \leq \varepsilon.$$

[1] The BC values reported by Riondato and Kornaropoulos [47] are not correct. We report corrected values.

In Section 4.5, we show a relative (i.e., multiplicative) error variant for the k nodes with highest BC.

3.2 Rademacher Averages

Rademacher Averages [29] are fundamental concepts to study the rate of convergence of a set of sample averages to their expectations. They are at the core of statistical learning theory [53] but their usefulness extends way beyond the learning framework [48]. We present here only the definitions and results that we use in our work, and we refer the readers to, e.g., the book by Shalev-Shwartz and Ben-David [51] for in-depth presentation and discussion.

While the Rademacher complexity can be defined on an arbitrary measure space, we restrict our discussion here to a sample space that consists of a finite domain $\mathcal{D}$ and the uniform distribution over the elements of $\mathcal{D}$. Let $\mathcal{F}$ be a family of functions from $\mathcal{D}$ to the *interval* $[0, 1]$,[2] and let $\mathcal{S} = \{s_1, \ldots, s_\ell\}$ be a collection of ℓ independent uniform samples from $\mathcal{D}$. For each $f \in \mathcal{F}$, define

$$\mathrm{m}_{\mathcal{D}}(f) = \frac{1}{|\mathcal{D}|} \sum_{c \in \mathcal{D}} f(c) \quad \text{and} \quad \mathrm{m}_{\mathcal{S}}(f) = \frac{1}{\ell} \sum_{i=1}^{\ell} f(s_i). \tag{1}$$

It holds

$$\mathrm{m}_{\mathcal{D}}(f) = \mathbb{E}[f] \quad \text{and} \quad \mathbb{E}\left[\mathrm{m}_{\mathcal{S}}(f)\right] = \mathrm{m}_{\mathcal{D}}(f).$$

Given $\mathcal{S}$, we are interested in bounding the *maximum deviation of* $\mathrm{m}_{\mathcal{S}}(f)$ *from* $\mathrm{m}_{\mathcal{D}}(f)$ *among all* $f \in \mathcal{F}$, i.e., the quantity

$$\sup_{f \in \mathcal{F}} \left|\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)\right|. \tag{2}$$

For $1 \le i \le \ell$, let λ_i be a Rademacher random variable (r.v.), i.e., a r.v. that takes value 1 with probability $1/2$ and -1 with probability $1/2$. The r.v.'s λ_i are independent. Consider the quantity

$$\mathrm{R}_{\mathcal{F}}(\mathcal{S}) = \mathbb{E}_{\lambda}\left[\sup_{f \in \mathcal{F}} \frac{1}{\ell} \sum_{i=1}^{\ell} \lambda_i f(s_i)\right], \tag{3}$$

where the expectation is taken only w.r.t. the Rademacher r.v.'s, i.e., conditioning on $\mathcal{S}$. The quantity $\mathrm{R}_{\mathcal{F}}(\mathcal{S})$ is known as the *(conditional) Rademacher average of* $\mathcal{F}$ *on* $\mathcal{S}$.[3]

The connection between $\mathrm{R}_{\mathcal{F}}(\mathcal{S})$ and the maximum deviation (2) is a key result in statistical learning theory. Classically, e.g., in textbooks and surveys, the connection has been presented using suboptimal bounds that are useful for conveying the intuition behind the connection, but inappropriate for practical use (see, e.g., [51, Thm. 26.5], and compare the bounds presented therein with the ones presented in the following). Tighter although more complex bounds are available [41, 42]. Specifically, we use Theorem 3.3, which is an extension of [41, Thm. 3.11] to a probabilistic tail bound for the supremum of the *absolute value* of the deviation for functions with codomain $[0, 1]$.

[2] The fact that the codomain of the functions in $\mathcal{F}$ is the interval $[0, 1]$ is of crucial importance, as many of the results presented in this section are valid *only* for such functions, although they can be extended to general *nonnegative* functions.
[3] In this work, we deal, for the most part, with the conditional Rademacher average, rather than with its expectation over the possible samples (which is known as the "Rademacher average," without specializing adjectives). Hence, we usually omit the specification "conditional," unless it is needed to avoid confusion.

Theorem 3.3. *Let $\mathcal{S}$ be a collection of ℓ independent uniform samples from $\mathcal{D}$. Let $\eta \in (0,1)$. Then, with probability at least $1-\eta$,*

$$\sup_{f\in\mathcal{F}} |\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| \leq 2\mathrm{R}_{\mathcal{F}}(\mathcal{S}) + \frac{\ln\frac{3}{\eta} + \sqrt{\left(\ln\frac{3}{\eta} + 4\ell\mathrm{R}_{\mathcal{F}}(\mathcal{S})\right)\ln\frac{3}{\eta}}}{\ell} + \sqrt{\frac{\ln\frac{3}{\eta}}{2\ell}}. \tag{4}$$

Even more refined bounds than the ones presented above are available [42] but, as observed by Oneto et al., in practice they do not seem to perform better than the one presented in (4).

Computing, or even estimating, the expectation in (3) w.r.t. the Rademacher r.v.'s is not straightforward and can be computationally expensive, requiring a time-consuming Monte Carlo simulation [12]. For this reason, *upper bounds to the Rademacher average* are usually employed in (4) in place of $\mathrm{R}_{\mathcal{F}}(\mathcal{S})$. A powerful and efficient-to-compute bound is presented in Theorem 3.4. Given $\mathcal{S}$, consider, for each $f \in \mathcal{F}$, the vector $\mathrm{v}_{f,\mathcal{S}} = (f(s_1), \ldots, f(s_\ell))$, and let $\mathcal{V}_{\mathcal{S}} = \{\mathrm{v}_{f,\mathcal{S}}, f \in \mathcal{F}\}$ be the *set* of such vectors ($|\mathcal{V}_{\mathcal{S}}| \leq |\mathcal{F}|$, as there may be distinct functions of $\mathcal{F}$ with identical vectors).

Theorem 3.4. *Let $\mathrm{w} : \mathbb{R}^+ \to \mathbb{R}^+$ be the function*

$$\mathrm{w}(r) = \frac{1}{r}\ln\left(\sum_{\mathrm{v}\in\mathcal{V}_{\mathcal{S}}} \exp\left[\frac{r^2\|\mathrm{v}\|_2^2}{2\ell^2}\right]\right), \tag{5}$$

where $\|\cdot\|_2$ denotes the ℓ_2-norm (Euclidean norm). Then

$$\mathrm{R}_{\mathcal{F}}(\mathcal{S}) \leq \min_{r\in\mathbb{R}^+} \mathrm{w}(r). \tag{6}$$

This result is obtained from a careful reading of the proof of Massart's Lemma [51, Lemma 26.8].

The function w is convex, continuous in $\mathbb{R}^+$, and has first and second derivatives w.r.t. r everywhere in its domain, so it is possible to minimize it efficiently using standard convex optimization methods [13]. More refined bounds can be derived but are more computationally expensive to compute [2].

3.2.1 Rademacher Averages for Relative-Error Approximation. In this section we discuss how to obtain an upper bound to the supremum of a specific relative (i.e., multiplicative) deviation of sample means from their expectations, for a family $\mathcal{F}$ of functions from a domain $\mathcal{D}$ to $[0,1]$.

Let $\mathcal{S} = \{s_1, \ldots, s_\ell\}$ be a collection of ℓ elements from $\mathcal{D}$. Given a parameter $\theta \in (0,1]$, we are interested specifically in giving probabilistic bounds to the quantity

$$\sup_{f\in\mathcal{F}} \frac{|\mathrm{m}_{\mathcal{D}}(f) - \mathrm{m}_{\mathcal{S}}(f)|}{\max\{\theta, \mathrm{m}_{\mathcal{D}}(f)\}}. \tag{7}$$

Li et al. [36] used *pseudodimension* to study the quantity

$$\sup_{f\in\mathcal{F}} \frac{|\mathrm{m}_{\mathcal{D}}(f) - \mathrm{m}_{\mathcal{S}}(f)|}{\mathrm{m}_{\mathcal{D}}(f) + \mathrm{m}_{\mathcal{S}}(f) + \theta}. \tag{8}$$

Har-Peled and Sharir [22] derived their concept of relative (θ,ε)-approximation from this quantity and were only concerned with binary functions. The quantity in (8) has been studied often in the literature of statistical learning theory [23], [4, Section 5.5], [12, Section 5.1], while other works [5, 7, 16], [12, Section 5.1] focused on the quantity

$$\sup_{f\in\mathcal{F}} \frac{|\mathrm{m}_{\mathcal{D}}(f) - \mathrm{m}_{\mathcal{S}}(f)|}{\sqrt{\mathrm{m}_{\mathcal{D}}(f)}}.$$

We study the quantity in (7) because it applies to our specific case.

It is easy to see that

$$\sup_{f\in\mathcal{F}} \frac{|\mathsf{m}_{\mathcal{D}}(f) - \mathsf{m}_{\mathcal{S}}(f)|}{\max\{\theta, \mathsf{m}_{\mathcal{D}}(f)\}} \leq \sup_{f\in\mathcal{F}} \frac{|\mathsf{m}_{\mathcal{D}}(f) - \mathsf{m}_{\mathcal{S}}(f)|}{\theta}. \tag{9}$$

Therefore, a bound to the r.h.s. of this equation implies a bound to the quantity from (7) that we are interested in. We can use Theorem 3.3 to obtain a bound to the supremum of the absolute deviations of the sample means from their expectations for the functions in $\mathcal{F}$, and then divide this bound by θ.

Theorem 3.5. *Let $\eta \in (0,1)$. Let $\mathcal{S}$ be a collection of ℓ elements of $\mathcal{D}$ sampled independently. Then, with probability at least $1-\eta$,*

$$\sup_{f\in\mathcal{F}} \frac{|\mathsf{m}_{\mathcal{D}}(f) - \mathsf{m}_{\mathcal{S}}(f)|}{\max\{\theta, \mathsf{m}_{\mathcal{D}}(f)\}} \leq \frac{1}{\theta}\left(2\mathrm{R}_{\mathcal{F}}(\mathcal{S}) + \frac{\ln(3/\eta) + \sqrt{\ln(3/\eta)\,(4\ell\mathrm{R}_{\mathcal{F}}(\mathcal{S}) + \ln(3/\eta))}}{\ell} + \sqrt{\frac{\ln(3/\eta)}{2\ell}}\right).$$

3.3 Pseudodimension

Before introducing the pseudodimension, we must recall some notions and results about the VC dimension. We refer the reader to the books by Shalev-Shwartz and Ben-David [51] and by Anthony and Bartlett [4] for an in-depth exposition of VC dimension and pseudodimension.

VC dimension. Let D be a (potentially infinite) domain and let $\mathcal{R}$ be a collection of subsets of D ($\mathcal{R} \subseteq 2^D$). We call $\mathcal{R}$ a *rangeset on D*, and its elements are called *ranges*. Given $A \subseteq D$, the *projection of $\mathcal{R}$ on A* is $P_{\mathcal{R}}(A) = \{R \cap A, R \in \mathcal{R}\}$. When $P_{\mathcal{R}}(A) = 2^A$, we say that A is *shattered* by $\mathcal{R}$. The *VC dimension* of $\mathcal{R}$, denoted as $\mathsf{VC}(\mathcal{R})$ is the size of the largest subset of B that can be shattered.

For example, let $D = \mathbb{R}$ and let $\mathcal{R}$ be the collection of closed intervals of $\mathbb{R}$, i.e.,

$$\mathcal{R} = \{[a,b], a < b \in \mathbb{R}\}.$$

A set A of two different points $A = \{c,d\}$ s.t. $c,d \in \mathbb{R}$ can be shattered as follows. W.l.o.g., let $c < d$, and define $g = c + (d-c)/2$, and let h_1 and h_2 be such that $h_1 < h_2 < c$. Consider the ranges $[h_1,h_2]$, $[c,g]$, $[g,d]$, $[c,d]$. Each intersection of each of one of these ranges with A is a different subset of $\{c,d\}$, and for each B of the four subsets of A there is one range R_B of the four above such that $A \cap R_A = B$. Thus, $P_{\mathcal{R}}(A) = 2^A$, i.e., the set A is shattered by $\mathcal{R}$.

Consider now a set $C = \{c,d,f\}$ of three different points $c < d < f \in \mathbb{R}$. There is no range $R \in \mathcal{R}$, such that $R \cap C = \{c,f\}$. Indeed all intervals that contain c and f must also contain d. Thus, C cannot be shattered, because it must be $P_{\mathcal{R}}(C) \neq 2^C$. This fact holds for all sets C of three points, so the VC dimension of $\mathcal{R}$ is $\mathsf{VC}(\mathcal{R}) = 2$.

Pseudodimension. Let $\mathcal{F}$ be a class of functions from some domain U to $[0,1]$. Consider, for each $f \in \mathcal{F}$, the subset R_f of $D = U \times [0,1]$ defined as

$$R_f = \{(x,t), x \in U \text{ and } t \leq f(x)\}.$$

We define a rangeset $\mathcal{F}^+$ on D as

$$\mathcal{F}^+ = \{R_f, f \in \mathcal{F}\}.$$

The pseudodimension of $\mathcal{F}$ [43], denoted as $\mathsf{PD}(\mathcal{F})$ is the VC dimension of $\mathcal{F}^+$ [4, Section 11.2],

$$\mathsf{PD}(\mathcal{F}) = \mathsf{VC}(\mathcal{F}^+).$$

For example, consider the family $\mathcal{F}$ of functions from $U = (0,1]$ to $[0,1]$

$$\mathcal{F} = \{f_k(x) = kx, \text{ for } 0 < k \leq 1\}.$$

The pseudodimension of $\mathcal{F}$ is $\mathrm{PD}(\mathcal{F}) = 1$. Indeed, for each $f_k \in \mathcal{F}$, i.e., for each $0 < k \leq 1$, the set $R_{f_k} = R_k$ is

$$R_k = \{(x, y), 0 \leq x \leq 1 \text{ and } y \leq kx\}.$$

It is a useful exercise to check how to shatter a set containing a single point (x, y), $0 \leq x, y \leq 1$.

To show that $\mathrm{PD}(\mathcal{F}) = 1$, we need to show that no set A of two pairs (x_1, y_1) and (x_2, y_2) can be shattered by $\mathcal{F}^+$. First of all, notice that it must be $y_1 \leq x_1$ and $y_2 \leq x_2$ because there is no range R_k that contains (x, y) if $y > x$. Assume now w.l.o.g. that $x_1 \leq x_2$. If $y_1 > y_2$, then there is no $k \in [0, 1]$ such that $kx_1 \geq y_1$ and $kx_2 < y_2$, thus there is no range R_k such that $A \cap R_k = \{(x_1, y_1)\}$. If instead $y_1 \leq y_2$, then let $z = y_2/x_2$. We have to consider two subcases:

(1) if $y_1 > zx_1$, then there is no $k \in [0, 1]$, such that $kx_1 \geq y_1$ and $kx_2 < y_2$, thus there is no range R_k such that $A \cap R_k = \{(x_1, y_1)\}$. To see this, assume that such a k exists. Then, it would hold that $k > z$ because $kx_1 \geq y_1 > zx_1$, thus $kx_2 > zx_2 = y_2$, which is a contradiction.
(2) if $y_1 \leq zx_2$, then there is no $k \in [0, 1]$, such that $kx_1 < y_1$ and $kx_2 \geq y_2$, thus there is no range R_k such that $A \cap R_k = \{(x_2, y_2)\}$. To see this, assume that such a k exists. Then, it would hold that $k < z$ because $kx_1 < y_1 \leq zx_1$, thus $kx_2 < zx_2 = y_2$, which is a contradiction.

Hence, the set A cannot be shattered, implying $\mathrm{PD}(\mathcal{F}) = 1$.

The fundamental result that we use is that having an upper bound on the pseudodimension allows to bound the supremum of the deviations from (2), as stated in the following theorem.

Theorem 3.6 ([36]). *Let U be a domain and $\mathcal{F}$ be a family of functions from U to $[0, 1]$. Let $\mathrm{PD}(\mathcal{F}) \leq d$. Given $\varepsilon, \eta \in (0, 1)$, let $\mathcal{S}$ be a collection of elements sampled independently and uniformly at random from D, with size*

$$|\mathcal{S}| = \frac{c}{\varepsilon^2}\left(d + \log\frac{1}{\eta}\right). \tag{10}$$

Then

$$\Pr\left(\text{there is } f \in \mathcal{F} \text{ s.t. } |\mathrm{m}_D(f) - \mathrm{m}_{\mathcal{S}}(f)| > \varepsilon\right) < \eta.$$

The constant c is universal, i.e., it does not depend on U, $\mathcal{F}$, or $\mathcal{S}$. It is estimated to be less than 0.5 [37].

The following two technical but completely general lemmas are, to the best of our knowledge, new. They present constraints on the sets that can actually be shattered by a rangeset, allowing the analyst to focus only on such set to prove bounds on the pseudodimension. We use them later to show upper bounds to the number of samples needed by ABRA.

Lemma 3.7. *Let $B \subseteq D$ ($D = U \times [0, 1]$) be a set that is shattered by $\mathcal{F}^+$. Then, B can contain* at most one *element $(d, x) \in D$ for each $d \in U$.*

Proof. Let $d \in U$ and consider any two distinct values $x_1, x_2 \in [0, 1]$. Let, w.l.o.g., $x_1 < x_2$ and let $B = \{(\tau, x_1), (\tau, x_2)\}$. From the definitions of the ranges, there is no $R \in \mathcal{F}^+$ such that $R \cap B = \{(d, x_1)\}$, therefore B cannot be shattered, and so neither can any of its supersets. □

Lemma 3.8. *Let $B \subseteq D$ ($D = U \times [0, 1]$) be a set that is shattered by $\mathcal{F}^+$. Then, B does not contain* any *element in the form $(d, 0)$, for any $d \in U$.*

Proof. For any $d \in U$, $(d, 0)$ is contained in every $R \in \mathcal{F}^+$, hence given a set $B = \{(d, 0)\}$, it is impossible to find a range $R_\emptyset$, such that $B \cap R_\emptyset = \emptyset$, therefore B cannot be shattered, nor can any of its supersets, hence the thesis. □

4 APPROXIMATING BETWEENNESS CENTRALITY IN STATIC GRAPHS

We now present and analyze ABRA-s, our *progressive sampling algorithm* that computes, with probability at least $1 - \delta$, an ε-approximation to the collection of exact BC values in a *static* graph. Many of the details and properties of ABRA-s are shared with the other ABRA algorithms we present in later sections.

Progressive sampling. Progressive sampling algorithms are intrinsically *iterative*. At a high level, they work as follows. At iteration i, the algorithm extracts an approximation of the values of interest (in our case, of the BC of all nodes) from a collection $\mathcal{S}_i$ of $S_i = |\mathcal{S}_i|$ random samples from a suitable domain $\mathcal{D}$ (in our case, the samples are pairs of different nodes). Then, the algorithm checks a specific *stopping condition* that uses information obtained from the sample $\mathcal{S}_i$ and from the computed approximation. If the stopping condition is satisfied, then the approximation has, with at least the required probability (in our case $1 - \delta$), the desired quality (in our case, it is an ε-approximation). The approximation is then returned in output and the algorithm terminates. If the stopping condition is not satisfied, the algorithm builds a collection $\mathcal{S}_{i+1}$ by adding random samples to $\mathcal{S}_i$ until it has size S_{i+1}. Then, it computes a new approximation from the so-created collection $\mathcal{S}_{i+1}$, and checks the stopping condition again and so on.

There are two main challenges for the designer of progressive sampling algorithm: deriving a good stopping condition and determining good choices for the initial sample size S_1 and the subsequent sample sizes S_{i+1}, $i \geq 1$.

An ideal stopping condition is such that

(1) when satisfied, it guarantees that the computed approximation has the desired quality properties (in our case, the approximation is, with probability at least $1 - \delta$, an ε-approximation); and
(2) it can be evaluated efficiently; and
(3) it is "weak," in the sense that is satisfied at small sample sizes.

The stopping condition for ABRA-s (presented in the following) is based on Theorem 3.3 and Theorem 3.4, and has all the above desirable properties.

The second challenge is determining the *sample schedule* $(S_i)_{i>0}$. Any monotonically increasing sequence of positive numbers can act as sample schedule, but the goal in designing a good sample schedule is to minimize the number of iterations that are needed before the stopping condition is satisfied, while minimizing the sample size S_i at the iteration i at which this happens. The sample schedule is fixed in advance, but an *adaptive approach* allows to find a reasonable initial sample size, and then skip directly to a sample size at which the stopping condition is likely satisfied. We developed such a general adaptive approach, which can be used also in other progressive sampling algorithms and is not specific to ABRA (see Section 4.1.2.)

4.1 Algorithm Description and Analysis

ABRA-s takes as input a graph $G = (V, E)$, which may be directed or undirected and may have nonnegative weights on the edges, a sample schedule $(S_i)_{i\geq 1}$, and two parameters $\varepsilon, \delta \in (0, 1)$. It outputs a collection $\widetilde{B} = \{\widetilde{\mathrm{b}}(w), w \in V\}$ that is, with probability at least $1 - \delta$, an ε-approximation of the betweenness centralities $B = \{\mathrm{b}(w), w \in V\}$. Let $\mathcal{D} = \{(u, v) \in V \times V, u \neq v\}$ be the set of all pairs of distinct nodes. For each node $w \in V$, let $f_w : \mathcal{D} \to [0, 1]$ be the function

$$f_w(u, v) = \frac{\sigma_{uv}(w)}{\sigma_{uv}}, \tag{11}$$

i.e., $f_w(u,v)$ is the fraction of SPs from u to v that go through w (i.e., that w is internal to.) Let $\mathcal{F} = \{f_w, w \in V\}$ be the set of these functions. Given this definition, we have that

$$\mathrm{m}_{\mathcal{D}}(f_w) = \frac{1}{|\mathcal{D}|}\sum_{(u,v)\in\mathcal{D}} f_w(u,v) = \frac{1}{|V|(|V|-1)}\sum_{\substack{(u,\,v)\in V\times V\\ u\neq v}} \frac{\sigma_{uv}(w)}{\sigma_{uv}} = \mathrm{b}(w).$$

The intuition behind ABRA-s is the following. Let $\mathcal{S} = \{(u_i, v_i), 1 \le i \le \ell\}$ be a collection of ℓ pairs (u,v) sampled independently and uniformly from $\mathcal{D}$. For the sake of clarity, we define

$$\widetilde{\mathrm{b}}(w) = \mathrm{m}_{\mathcal{S}}(f_w) = \frac{1}{\ell}\sum_{i=1}^{\ell} f_w(u_i,v_i) = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{\sigma_{u_iv_i}(w)}{\sigma_{u_iv_i}}.$$

For each $w \in V$ consider the vector

$$\mathbf{v}_w = (f_w(u_1,v_1),\ldots,f_w(u_\ell,v_\ell))\,.$$

It is easy to see that $\widetilde{\mathrm{b}}(w) = \|\mathbf{v}_w\|_1/\ell$. Let now $\mathcal{V}_{\mathcal{S}}$ be the *set* of these vectors, i.e.,

$$\mathcal{V}_{\mathcal{S}} = \{\mathbf{v}_w, w \in V\}.$$

It is possible that $|\mathcal{V}_{\mathcal{S}}| \le |V|$ as there may be two different nodes u and v with $\mathbf{v}_u = \mathbf{v}_v$. This is indeed the usual case in practice. If we have complete knowledge of the set $\mathcal{V}_{\mathcal{S}}$ (i.e., of its elements), then we can compute the quantity

$$\omega^* = \min_{r\in\mathbb{R}^+}\frac{1}{r}\ln\left(\sum_{\mathbf{v}\in\mathcal{V}_{\mathcal{S}}}\exp\left[\frac{r^2\|\mathbf{v}\|_2^2}{2\ell^2}\right]\right),$$

which, from Theorem 3.4, is an *upper bound* to $\mathrm{R}_{\mathcal{F}}(\mathcal{S})$. We can use ω^* to obtain an upper bound $\xi_{\mathcal{S}}$ to the supremum of the absolute deviation by plugging ω^* in (4). It follows from Theorem 3.3 that the collection $\widetilde{B} = \{\widetilde{\mathrm{b}}(w) = \|\mathbf{v}_w\|_1/\ell, w \in V\}$ is, with probability at least $1-\phi$, a $\xi_{\mathcal{S}}$-approximation to the exact betweenness values.

ABRA-s builds on this intuition and works as follows. The algorithm builds a collection $\mathcal{S}$ by sampling pairs (u,v) independently and uniformly at random from $\mathcal{D}$ until $\mathcal{S}$ has size S_1. After sampling a pair (u,v), ABRA-s performs an $s-t$ SP computation from u to v, and then backtracks from v to u along the SPs just computed, to keep track of the set $\mathcal{V}_{\mathcal{S}}$ of vectors (details given below). For clarity of presentation, let $\mathcal{S}_1$ denote $\mathcal{S}$ when it has size exactly S_1, and analogously for $\mathcal{S}_i$ and S_i, $i > 1$. Once $\mathcal{S}_i$ has been "built," ABRA-s computes $\xi_{\mathcal{S}_i}$ as described earlier, using $\phi_i = \delta/2^i$. It then checks whether $\xi_{\mathcal{S}_i} \le \varepsilon$. This is ABRA-s *stopping condition*:[4] when it holds, ABRA-s returns

$$\widetilde{B} = \left\{\widetilde{\mathrm{b}}(w) = \|\mathbf{v}_w\|_1/S_i, w \in V\right\}.$$

Otherwise, ABRA-s iterates and continues adding samples from $\mathcal{D}$ to $\mathcal{S}$ until it has size S_2, and so on until $\xi_{\mathcal{S}_i} \le \varepsilon$ holds. The pseudocode for ABRA-s is presented in Algorithm 1, while the steps to update $\mathcal{V}_{\mathcal{S}}$, described in the following, are in Algorithm 2.

Computing and maintaining the set $\mathcal{V}_{\mathcal{S}}$. We now discuss in details how ABRA-s efficiently maintains the set $\mathcal{V}_{\mathcal{S}}$ of vectors, which is used to compute the value $\xi_{\mathcal{S}}$ and the values $\widetilde{\mathrm{b}}(w) = \|\mathbf{v}_w\|_1/|\mathcal{S}|$ in $\widetilde{B}$. In addition to $\mathcal{V}_{\mathcal{S}}$, ABRA-s also maintains a map M from V to $\mathcal{V}_{\mathcal{S}}$ (i.e., $M[w]$ is a vector $\mathbf{v}_w \in \mathcal{V}_{\mathcal{S}}$), and a counter $\mathrm{c}_{\mathbf{v}}$ for each $\mathbf{v} \in \mathcal{V}_{\mathcal{S}}$, denoting how many nodes $w \in V$ have $M[w] = \mathbf{v}$.

[4] A different stopping condition for generic progressive sampling using Rademacher average was presented by Koltchinskii et al. [30] and used by Elomaa and Kääriäinen [17]. We choose to use ours because it is more efficient to compute.

ALGORITHM 1: ABRA-s: absolute error approximation of BC on static graphs

Input: Graph $G = (V, E)$, sample schedule $(S_i)_{i \geq 1}$, accuracy parameter $\varepsilon \in (0, 1)$, confidence parameter $\delta \in (0, 1)$.

Output: Pair $(\widetilde{B}, \xi)$ such that $\xi \leq \varepsilon$ and $\widetilde{B}$ is a set of BC approximations for all nodes in V, which is, with probability at least $1 - \delta$, an ξ-approximation to $B = \{\mathrm{b}(w), w \in v\}$.

1 $\mathcal{D} \leftarrow \{(u, v) \in V \times V, u \neq v\}$
2 $S_0 \leftarrow 0$
3 $\mathbf{0} = (0)$
4 $\mathcal{V} = \{\mathbf{0}\}$
5 **foreach** $w \in V$ **do** $M[w] = \mathbf{0}$
6 $\mathrm{c}_{\mathbf{0}} \leftarrow |V|$
7 $i \leftarrow 1, j \leftarrow 1$
8 **while** *True* **do**
9 **for** $\ell \leftarrow 1$ **to** $S_i - S_{i-1}$ **do**
10 $(u, v) \leftarrow$ `uniform_random_sample`$(\mathcal{D})$
11 `compute_SPs`(u, v) `//Truncated SP computation`
12 **if** *reached* v **then**
13 **foreach** $z \in \mathrm{P}_u[v]$ **do** $\sigma_{zv} \leftarrow 1$
14 **foreach** *node* w *on a SP from* u *to* v*, in reverse order by* $\mathrm{d}(u, w)$ **do**
15 $\sigma_{uv}(w) \leftarrow \sigma_{uw}\sigma_{wv}$
16 `update_structures()` `//See Algorithm 2`
17 **foreach** $z \in \mathrm{P}_u[w]$ **do** $\sigma_{zv} \leftarrow \sigma_{zv} + \sigma_{wv}$
18 $j \leftarrow j + 1$
19 $\omega_i \leftarrow \min_{r \in \mathbb{R}^+} \frac{1}{r} \ln\left(\sum_{\mathbf{v} \in \mathcal{V}} \exp\left[r^2 \|\mathbf{v}\|^2 / (2S_i^2)\right]\right)$
20 $\phi_i \leftarrow \delta / 2^i$
21 $\xi_i \leftarrow 2\omega_i + \frac{\ln(3/\phi_i) + \sqrt{(\ln(3/\phi_i) + 4S_i\omega_i)\ln(3/\phi_i)}}{S_i} + \sqrt{\frac{\ln(3/\phi_i)}{2S_i}}$
22 **if** $\xi_i \leq \varepsilon$ **then**
23 **break**
24 **else**
25 $i \leftarrow i + 1$
26 $\widetilde{B} \leftarrow \{\widetilde{\mathrm{b}}(w) \leftarrow \|M[w]\|_1 / S_i, w \in V\}$
27 **return** $(\widetilde{B}, \xi_i)$

At the beginning of the execution of the algorithm, $\mathcal{S} = \emptyset$ and $\mathcal{V}_{\mathcal{S}} = \emptyset$. Nevertheless, ABRA-s initializes $\mathcal{V}_{\mathcal{S}}$ to contain one special empty vector $\mathbf{0}$, with no components, and M so that $M[w] = \mathbf{0}$ for all $w \in V$, and $\mathrm{c}_{\mathbf{0}} = |V|$ (lines 3 and following in Algorithm 1).

After having sampled a pair (u, v) from $\mathcal{D}$, ABRA-s updates $\mathcal{V}_{\mathcal{S}}$, M and the counters as follows. First, it performs (line 11) a $s - t$ SP computation from u to v using any SP algorithm (e.g., BFS, Dijkstra, or even any bidirectional search SP algorithm) modified, as discussed by Brandes [14, Lemma 3], to keep track, for each node w encountered during the computation, of the SP distance $\mathrm{d}(u, w)$ from u to w, of the number σ_{uw} of SPs from u to w, and of the set $\mathrm{P}_u(w)$ of (immediate) predecessors of w along the SPs from u.[5] Once v has been reached (and only if it has been reached), the algorithm starts backtracking from v toward u along the SPs it just computed (line 14 in

[5]Storing the set of immediate predecessors is not necessary. By not storing it, we can reduce the space complexity from $O(|E|)$ to $O(|V|)$, at the expense of some additional computation at runtime.

ALGORITHM 2: `update_structures`

```
1 v ← M[w]
2 v' ← v ∪ {(j, σ_uv(w)/σ_uv)}
3 if v' ∉ V_S then
4 |   c_v' ← 1
5 |   V_S ← V_S ∪ {v'}
6 else c_v' ← c_v' + 1
7 M[w] ← v'
8 if c_v > 1 then c_v ← c_v − 1
9 else V_S ← V_S \ {v}
```

Algorithm 1). During this backtracking, the algorithm visits the nodes along the SPs in *inverse* order of SP distance from u, ties broken arbitrarily. For each visited node w different from u and v, ABRA-s computes the value $f_w(u,v) = \sigma_{uv}(w)/\sigma_{uv}$ of SPs from u to v that go through w, which is obtained as

$$\sigma_{uv}(w) = \sigma_{uw} \times \sum_{z\,:\,w \in \mathrm{P}_u(z)} \sigma_{zv},$$

where the value σ_{uw} is obtained during the $s-t$ SP computation, and the values σ_{zw} are computed recursively during the backtracking (line 17), as described by Brandes [14]. After computing $\sigma_{uv}(w)$, the algorithm takes the vector $\mathbf{v} \in \mathcal{V}_S$ such that $M[w] = \mathbf{v}$ and creates a new vector $\mathbf{v}'$ by appending $\sigma_{uv}(w)/\sigma_{uv}$ to the end of $\mathbf{v}$.[6] Then, it adds $\mathbf{v}'$ to the *set* $\mathcal{V}_S$, updates $M[w]$ to $\mathbf{v}'$, and increments the counter $\mathrm{c}_{\mathbf{v}'}$ by one (lines 1–7 of Algorithm 2). Finally, the algorithm decrements the counter $\mathrm{c}_{\mathbf{v}}$ by one, and if $\mathrm{c}_{\mathbf{v}}$ becomes equal to zero, ABRA-s removes $\mathbf{v}$ from $\mathcal{V}_S$ (line 9 of Algorithm 2). At this point, the algorithm moves to analyzing another node w' with distance from u less or equal to the distance of w from u. It is easy to see that when the backtracking reaches u, the set $\mathcal{V}_S$, the map M, and the counters have been correctly updated. An example of how the data structures evolves from one sample to the other is shown in Figure 2.

We remark that to compute $\xi_{\mathcal{S}_i}$ and $\widetilde{B}$ and to keep the map M up to date, ABRA-s does not actually need to store the vectors in $\mathcal{V}_S$ (even in sparse form), but it is sufficient to maintain their ℓ_1 and ℓ_2 (i.e., Euclidean) norms, which require much less space, at the expense of some additional bookkeeping.

4.1.1 Quality Guarantees. The following theorem shows the guarantees given by ABRA-s.

Theorem 4.1. *Let r be the index of the last iteration of the algorithm, and let $(\widetilde{B}, \xi_{\mathcal{S}_r})$ be the output. With probability at least $1-\delta$, $\widetilde{B}$ is an $\xi_{\mathcal{S}_r}$-approximation to the set $B = \{\mathrm{b}(w), w \in V\}$.*

Since $\xi_{\mathcal{S}_r} \leq \varepsilon$, we have that $\widetilde{B}$ is at least an ε-approximation, as required by the user.

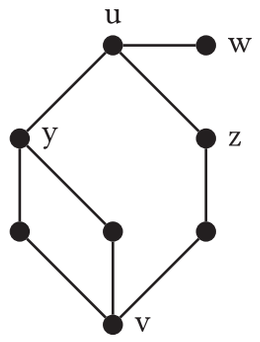
We now prove Theorem 4.1.

Consider a sequence $(X_j)_{j\geq 1}$ of random variables, where each X_j is a pair of distinct nodes (u,v) sampled independently and uniformly at random from $\mathcal{D} \subset V \times V$. We can reason about the sequence $(X_j)_{j\geq 1}$ *independently from the algorithm.*

Let $(S_i)_{i\geq 1}$ be the sample schedule fixed by the user. Consider the sequences $(X_j)_{j\geq 1}$ and $(S_i)_{i\geq 1}$ and let $(\mathcal{S}_i)_{i\geq 1}$ be the sequence s.t.

$$\mathcal{S}_i = \{X_1, \ldots, X_{S_i}\}, \text{ for all } i \geq 1.$$

[6] The pseudocode of ABRA-s uses a sparse representation for the vectors $\mathbf{v} \in \mathcal{V}_S$, storing only the nonzero components of each $\mathbf{v}$ as pairs (j, g), where j is the component index and g is the value of that component.



(a) A portion of the graph

Vertex x	$M[x]$
w	**a**
y	**a**
z	**a**

Vector **v**	$c_{\mathbf{v}}$
$\mathbf{a} = (a_1, \ldots, a_\ell)$	3

(b) Before the update

Vertex x	$M[x]$
w	**a**
y	**b**
z	**c**

Vector **v**	$c_{\mathbf{v}}$
$\mathbf{a} = (a_1, \ldots, a_\ell, 0)$	1
$\mathbf{b} = (a_1, \ldots, a_\ell, 2/3)$	1
$\mathbf{c} = (a_1, \ldots, a_\ell, 1/3)$	1

(c) After the update

Fig. 2. Example of the evolution of the data structures. Figure 2(a) shows the relevant *portion* of the graph. The algorithm samples the pair (u, v). Figures 2(b) and (c) show the data structures M and S, and the counter $c_{\mathbf{v}}$ for the relevant nodes w, z, and y before and after the update, respectively.

Fact 1. *For every $i \geq 1$, $\mathcal{S}_i = \{X_1, \ldots, X_{S_i}\}$ is a collection of S_i independent uniform samples from $\mathcal{D}$.*

Given $\delta \in (0, 1)$, let $(\phi_i)_{i \geq 1}$ be the sequence s.t.

$$\phi_i = \frac{\delta}{2^i}, \text{ for all } i \geq 1.$$

Given any collection $\mathcal{S} = \{(u_1, v_1), \ldots, (u_k, v_k)\}$ of elements from $\mathcal{D}$, let $\mathcal{V}_{\mathcal{S}}$ be the *set* of vectors

$$\mathcal{V}_{\mathcal{S}} = \left\{ \mathbf{v}_w = \left(\frac{\sigma_{u_1 v_i}(w)}{\sigma_{u_1, v_1}}, \ldots, \frac{\sigma_{u_k v_k}(w)}{\sigma_{u_k v_k}} \right), w \in V \right\},$$

and let

$$\omega(\mathcal{S}) = \min_{r \in \mathbb{R}^+} \frac{1}{r} \ln \left(\sum_{\mathbf{v} \in \mathcal{V}_{\mathcal{S}}} \exp\left[r^2 \|\mathbf{v}\|^2 / (2|\mathcal{S}|^2) \right] \right).$$

For $\lambda \in (0, 1)$, define

$$\xi(\mathcal{S}, \lambda) = 2\omega(\mathcal{S}) + \frac{\ln(3/\lambda) + \sqrt{(\ln(3/\lambda) + 4|\mathcal{S}|\omega(\mathcal{S}))\ln(3/\lambda)}}{|\mathcal{S}|} + \sqrt{\frac{\ln(3/\lambda)}{2|\mathcal{S}|}}.$$

Lemma 4.2. *Let $\delta \in (0, 1)$. Then*

$$\Pr\left(\exists i > 0 \text{ s.t. } \sup_{w \in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i) \right) \leq \delta. \tag{12}$$

The probability in (12) is taken over all realizations of the sequence $(\mathcal{S}_i)_{i \geq 1}$, i.e., over all realizations of the sequence $(X_j)_{j \geq 1}$.

Proof of Lemma 4.2. From the union bound, we have

$$\Pr\left(\exists i > 0 \text{ s.t. } \sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i)\right) \leq \sum_{i=1}^{\infty} \Pr\left(\sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i)\right). \tag{13}$$

Since Fact 1 holds, we can apply Theorem 3.3 to each $\mathcal{S}_i$. Using the definition of ϕ_i we have, for any fixed i:

$$\Pr\left(\sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i)\right) \leq \phi_i.$$

Continuing from (13) using the above inequality, we then have

$$\begin{aligned}\Pr\left(\exists i > 0 \text{ s.t. } \sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i)\right) &\leq \sum_{i=1}^{\infty} \Pr\left(\sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \geq \xi(\mathcal{S}_i, \phi_i)\right) \\ &\leq \sum_{i=1}^{\infty} \phi_i = \sum_{i=1}^{\infty} \frac{\delta}{2^i} = \delta.\end{aligned}$$

□

Consider now a realization of the sequence $(X_j)_{j\geq 1}$, and therefore of the sequence $(\mathcal{S}_i)_{i\geq 1}$. Imagine a variant of ABRA-s that, instead of performing the sampling of pairs of nodes independently and uniformly at random from $\mathcal{D}$ during the execution (line 10 of Algorithm 1), is given the realizations of the collections $\mathcal{S}_i$ as input, and at the ith iteration of the loop starting on line 8 of Algorithm 1 "operates" on $\mathcal{S}_i \setminus \mathcal{S}_{i-1}$ (i.e., the loop on lines 9–18 skips line 10 and uses the elements of $\mathcal{S}_i \setminus \mathcal{S}_{i-1}$), so that the quantity ω_i computed on line 19 is exactly $\omega(\mathcal{S}_i)$, and the quantity ξ_i computed on line 21 is exactly $\xi(\mathcal{S}, \phi_i)$.

Lemma 4.3. *Let r denote the last iteration after which this* variant *of* ABRA-s *stops, i.e., r is the minimal i such that $\xi(\mathcal{S}_i, \phi_i) \leq \varepsilon$. With probability at least $1 - \delta$, the output of* this variant *of* ABRA-s *is an $\xi(\mathcal{S}_r, \phi_r)$-approximation to the set of exact betweenness centralities of all nodes.*

The probability in the lemma is taken over all possible realizations of the sequence $(\mathcal{S}_i)_{i\geq 1}$, i.e., of $(X_j)_{j\geq 1}$.

Proof of Lemma 4.3. Lemma 4.2 says that with probability at least $1 - \delta$, the sequence $(\mathcal{S}_i)_{i\geq 1}$ is such that it holds

$$\sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_i}(w) - \mathrm{b}(w)| \leq \xi(\mathcal{S}_i, \phi_i), \text{ for all } i \geq 1. \tag{14}$$

Whether this property holds or not for the realization of $(\mathcal{S}_i)_{i\geq 1}$, i.e., of $(X_j)_{j\geq 1}$, that is "fed" to the algorithm is completely independent from what the algorithm does. Indeed, the realization is fixed before the algorithm even starts.

Suppose (14) holds, which happens with probability at least $1 - \delta$. The collection of pairs of nodes that the algorithm has seen is exactly the realization of $\mathcal{S}_r$. The property in (14) holds particularly for $i = r$, where, as stated in the hypothesis, r is the last iteration actually done by the algorithm, i.e., the earliest iteration i such that $\xi(\mathcal{S}_i, \phi_i) \leq \varepsilon$. Thus, it holds that

$$\sup_{w\in V} |\widetilde{\mathrm{b}}_{\mathcal{S}_r}(w) - \mathrm{b}(w)| \leq \xi(\mathcal{S}_r, \phi_r) \leq \varepsilon.$$

□

Fact 2. *The distribution of the collection of pairs of nodes sampled by "vanilla" ABRA-s is the same as the distribution of the collection of transactions sampled by the variant of the algorithm described above because, given the same random bits, "vanilla" ABRA-s and the variant generate exactly the same sequence of pairs of nodes.*

By combining Lemma 4.3 and Fact 2, we obtain the correctness of the vanilla version of ABRA-s, thus proving Theorem 4.1.

4.1.2 Choosing a Sample Schedule. We now discuss how to choose a reasonable sample schedule $(S_i)_{i\geq 1}$, so that the algorithm terminates after having performed a small number of iterations at small sample sizes.

Initial sample size. The initial sample size S_1 should be such that

$$S_1 \geq \frac{(1 + 8\varepsilon + \sqrt{1 + 16\varepsilon})\ln(6/\delta)}{4\varepsilon^2}. \tag{15}$$

To understand the intuition behind the lower bound, recall (4), and consider that, at the beginning of the algorithm, there is obviously no information available about $\mathrm{R}_{\mathcal{F}}(\mathcal{S}_1)$, except that it is *non-negative*, i.e., $\mathrm{R}_{\mathcal{F}}(\mathcal{S}_1) \geq 0$. It follows that, for the r.h.s. of (4) to be at most ε at the end of the first iteration (i.e., for the stopping condition to be satisfied at this time), it is necessary that

$$2\frac{\ln(6/\delta)}{S_1} + \sqrt{\frac{\ln(6/\delta)}{2S_1}} \leq \varepsilon. \tag{16}$$

Solving for S_1 under the constraints $S_1 \geq 1$, $\delta \in (0, 1)$, $\varepsilon \in (0, 1)$, gives the unique solution in (15).

One may not want to set S_1 equal to the r.h.s. of (15) because its derivation basically assumes that $\mathrm{R}_{\mathcal{F}}(\mathcal{S}_1) = 0$, which is unlikely if not impossible in practice, and therefore it is almost guaranteed that, when S_1 equals the r.h.s., the stopping condition would not be satisfied at this sample size. A more reasonable approach would be to multiply the r.h.s. of (15) by some quantity greater than one.

Successive sample sizes. Any increasing sequence can be used as a sample schedule, but there is evidence that a geometrically increasing sample schedule, i.e., a sample schedule such that $S_i = c^i S_1$, for some $c > 1$, may be optimal [45].

4.1.3 Targeting a Specific Set of Nodes. In some situations, one may be interested in estimating the BC of only a subset $R \subset V$ of the nodes of the graph. ABRA-s can be easily adapted to this scenario. W.r.t. the pseudocode presented in Algorithms 1 and 2, the changes are the following:

(1) the map M is initialized only with elements of R (line 5 of Algorithm 1);
(2) $\mathtt{c_0}$ is initialized to $|R|$ (line 6);
(3) `update_structures` on line 16 is only called if the node w belongs to R.
(4) The output collection $\widetilde{B}$ only contains values for the nodes in R.

We denote this variant as ABRA-s-set, and we use it in Section 4.5.

Restricting to a specific set R of nodes can only have a positive impact on the running time, as the stopping condition may be satisfied earlier than in ABRA-s.

4.2 Upper Bounds on the Number of Samples

It is natural to ask whether, given a graph $G = (V, E)$, there exists an integer s such that ABRA-s can stop and output $(\widetilde{B}, \xi)$ after having sampled s pair of nodes and $\widetilde{B}$ will be, with probability at least $1 - \delta$, a ξ-approximation with $\xi \leq \varepsilon$, independently from whether the stopping condition is satisfied or not at that point in the execution. If such a sample size s exists, we can modify the stopping condition of ABRA-s to just stop after having examined a sample of that size, as we describe in Section 4.2.1. Such a sample size exists and it is a function of a characteristic quantity of the graph G and of ε and δ. Its derivation uses *pseudodimension* [43], an extension of the VC dimension to real-valued functions (see Section 3.3).

4.2.1 Using the Upper Bounds. The upper bounds to the pseudodimension presented in the following subsections can be used in a variant of ABRA-s as follows.

Before starting the sampling process (i.e., before entering the loop on line 8 in Algorithm 1), an upper bound d to the pseudodimension is computed (in the worst case, computing d requires finding the weakly connected components (WCC) of a slightly modified graph G', see the discussion after Theorem 4.4). Let now, for any $i \geq 1$

$$W(i) = \frac{1}{\varepsilon^2}\left(d + \ln\frac{2^i \cdot 3}{\delta}\right) \text{ (compare with (10)),}$$

and let

$$i^* = \min\{i, S_{i-1} \leq W(i) \leq S_i\}.$$

The quantity i^* is computed immediately after having computed d. ABRA-s is further modified to always set $\xi_{S_{i^*}}$ equal to ε and exit the main loop at iteration i^*. In terms of the pseudocode from Algorithm 1, this changes correspond to wrapping lines 19 to 25 in a **if** block with the condition "$i < i^*$." Line 25 is then followed by an **else** block that sets $\xi_{S_{i^*}}$ to ε and calls **break**.

The correctness of this variant of ABRA-s comes from Theorems 3.6 and 4.1.

4.2.2 General Cases. We now show a general upper bound on the pseudodimension of $\mathcal{F}$. The derivation of this upper bound follows the one for VC dimension in [47, Section 4], adapted to our settings.

Let $G = (V, E)$ be a graph, and consider the family

$$\mathcal{F} = \{f_w, w \in V\},$$

where f_w goes from $\mathcal{D} = \{(u, v) \in V \times V, u \neq v\}$ to $[0, 1]$ and is defined in (11). The rangeset $\mathcal{F}^+$ contains one range R_w for each node $w \in V$. The set $R_w \subseteq \mathcal{D} \times [0, 1]$ contains pairs in the form $((u, v), x)$, with $(u, v) \in \mathcal{D}$ and $x \in [0, 1]$. The pairs $((u, v), x) \in R_w$ with $x > 0$ are all and only the pairs in this form such that

(1) w is on a SP from u to v; and
(2) $x \leq \sigma_{uv}(w)/\sigma_{uv}$.

For any SP p let $\mathsf{Int}(p)$ be the *set* of nodes that are internal to p, i.e., not including the extremes of p. For any pair (u, v) of distinct nodes, let

$$N_{uv} = \bigcup_{p \in \mathcal{S}_{uv}} \mathsf{Int}(p)$$

be the set of nodes in the SP DAG from u to v, *excluding u and v*, and let $s_{uv} = |N_{uv}|$. Let $\mathsf{H}(G)$ be the maximum integer h such that there are at least $\lfloor \log_2 h \rfloor + 1$ pairs (u, v), such that is $s_{uv} \geq h$. Except in trivial cases, $\mathsf{H}(G) > 0$.

Theorem 4.4. *It holds* $\mathsf{PD}(\mathcal{F}) \leq \lfloor \log_2 \mathsf{H}(G) \rfloor + 1$.

Proof. Let $k > \lfloor \log_2 \mathsf{H}(G) \rfloor + 1$ and assume for the sake of contradiction that $\mathsf{PD}(\mathcal{F}) = k$. From the definition of pseudodimension, we have that there is a set Q of k elements of the domain of $\mathcal{F}^+$ that is shattered.

From the definition of $\mathsf{H}(G)$ and from Lemma 3.7, we have that Q must contain an element $a = ((u, v), x)$, $x > 0$, of the domain of $\mathcal{F}^+$ such that $s_{uv} < \mathsf{H}(G)$.

There are 2^{k-1} nonempty subsets of Q containing a. Let us label these nonempty subsets of Q containing a as $S_1, \ldots, S_{2^{k-1}}$, where the labeling is arbitrary. Given that Q is shattered, for each set S_i there must be a range R_i in $\mathcal{F}^+$ such that $S_i = Q \cap R_i$. Since all the S_i's are different from each other, then all the R_i's must be different from each other. Given that a is a member of every S_i, a must also belong to each R_i, that is, there are 2^{k-1} distinct ranges in $\mathcal{F}^+$ containing a. But a

belongs only to (not necessarily all) the ranges R_w for $w \in N_{uv}$. This means that a belongs to at most s_{uv} ranges in $\mathcal{F}^+$.

But $s_{uv} < \mathsf{H}(G)$, by definition of $\mathsf{H}(G)$, so a can belong to at most $\mathsf{H}(G)$ ranges from $\mathcal{R}_G$. Given that $2^{k-1} > \mathsf{H}(G)$, we reached a contradiction, and there cannot be 2^{k-1} distinct ranges containing a, hence not all the sets S_i can be expressed as $Q \cap R_i$ for some $R_i \in \mathcal{F}^+$.

Then, Q cannot be shattered, and we have

$$\mathsf{PD}(\mathcal{F}) = \mathsf{VC}(\mathcal{F}^+) \leq \lfloor \log_2 \mathsf{H}(G) \rfloor + 1. \qquad \square$$

Computing $\mathsf{H}(G)$ exactly is not practical as it would defeat the purpose of using sampling. Instead, we now present looser but efficient-to-compute upper bounds on the pseudodimension of $\mathcal{F}$ which can be used in practice.

Let $G = (V, E)$ be a graph and let $G' = (V', E')$ be the graph obtained by removing from V some nodes and from E the edges incident to any of the removed nodes. Specifically:

- If G is undirected, we obtain V' by removing all nodes of degree 1 from V.[7]
- If G is directed, we obtain V' by removing all nodes u such that the elements of E involving u are either all in the form (u, v) or are all in the form (v, u).

Consider now the largest (in terms of number of nodes) WCC of G', and let L be its size (number of nodes in it).

Lemma 4.5. *It holds* $\mathsf{PD}(\mathcal{F}) \leq \lfloor \log_2 L \rfloor + 1$.

Proof. Let's consider *undirected* graphs first. Each WCC of G' is a subset (potentially improper) of one and only one WCC of G. Let W be a WCC of G (W is a set of nodes, $W \subseteq V$) and let W' be the corresponding WCC of G' ($W' \subseteq V'$). Let (u, v) be a pair of nodes in W. It holds $N_{uw} \subseteq W$, i.e., $W \cap N_{uw} = N_{uw}$. We want to show that $N_{uw} \subseteq W'$.

Let v be any node in $W \setminus W'$ (if such a node exists, otherwise it must be $W' = W$ and therefore it must be $N_{uw} \subseteq W'$, since $N_{uw} \subseteq W$). It must be that $v \in V \setminus V'$, i.e., v is one of the removed nodes, which must have had degree 1 in G. The node v is not *internal* to any SP between any two nodes in G, i.e., $v \notin N_{zy}$ for any pair of nodes $(z, y) \in V \times V$, and particularly $v \notin N_{uw}$. This is true for any $v \in W \setminus W'$, hence $(W \setminus W') \cap N_{uw} = \emptyset$. It holds

$$\begin{aligned} W' \cap N_{uw} &= (W \cap N_{uw}) \setminus ((W \setminus W') \cap N_{uw}) \\ &= N_{uw} \setminus \emptyset = N_{uw}, \end{aligned}$$

i.e., $N_{uw} \subseteq W'$. Thus, $|N_{uw}| \leq |W'|$, and therefore $\mathsf{H}(G) \leq L$, from which we obtain the thesis, given Theorem 4.4.

Let's now consider *directed* graphs. It is no longer true that each WCC of G' is a subset (potentially improper) of one and only one WCC of G: there may be multiple WCCs of G' that are subsets of a WCC of G, hence we cannot proceed as in the case of undirected graphs.

Let $\{u, v, w, z\}$ be a set of nodes in V', such that at least three of them are distinct (if two of them are the same, we can assume w.l.o.g. that they are neither u and v nor w and z), and such that there is a path (and hence a SP) in G from u to v and from w to z, and that all these nodes belong to the same WCC of G but to two or more different WCCs of G'. We want to show that no set containing both $((u, v), x)$ and $((w, z), y)$ for some $x, y \in (0, 1)$ could have been shattered by $\mathcal{F}^+$.

Let $S = \{((u, v), x), ((w, z), y)\}$, for u, v, w, z as above. If $\mathcal{F}^+$ cannot shatter S, then it cannot shatter any superset of S, so we can focus on S. We assumed that there is a SP from u to v and a SP from w to z in G. Any SP from u to v and from w to z still exists in G', as the removed nodes are

[7] This removal operation is done only once, not iteratively.

not internal to any SPs in G. Hence, u and v belong to the same WCC A in G' and w and z belong to the same WCC B in G'. We have, by construction of u, v, w, z that $A \neq B$.

Assume that S is shattered by $\mathcal{F}^+$. Then, there must be a node h that is internal to both a SP from u to v and a SP from w to z. If there was not such a node h then S could not be shattered, as there would not be a node ℓ such that the intersection between S and the range R_ℓ associated to ℓ is S. Since h exists, and it is internal to two SPs, then it must belong to V'. Since all SPs from u to v and from w to z still exist in G' then so do those that go through h. This means that there is a path from each of u, v, w, z to the others (e.g., from u to each of v, w, and z), hence they should all belong to the same WCC of G', but this is a contradiction. Hence, S cannot be shattered by $\mathcal{F}^+$.

This implies that sets that can be shattered by $\mathcal{F}^+$ are only sets in the form $\{((u_i, v_i), x_i), i = k\}$ such that all nodes u_i and v_i (for all i) belong to the same WCC of G'. Hence, we can proceed as in the undirected graphs case and obtain the thesis. □

The upper bound derived in Lemma 4.5 is somewhat disappointing, and sometimes noninformative: if G is undirected and has a single connected component, then the same bound to the sample size that can be obtained using the pseudodimension could be easily obtained using the union bound. We conjecture that it should be possible to obtain better bounds (see Conjecture 4.9).

4.2.3 Special Cases. In this section, we consider some special restricted settings that make computing an high-quality approximation of the BC of all nodes easier. One example of such restricted settings is when the graph is *undirected* and every pair of distinct nodes is either connected with a *single* SP or there is no path between the two nodes (because they belong to different connected components). Examples of these settings are many road networks, where the unique SP condition is often enforced [20]. Riondato and Kornaropoulos [47, Lemma 2] showed that, in this case, the number of samples needed to compute a high-quality approximation of the BC of all nodes is *independent* of any property of the graph, and only depends on the quality controlling parameters ε and δ. The algorithm by Riondato and Kornaropoulos [47] works differently from ABRA-s, as it samples one SP at a time and only updates the BC estimation of nodes along this path, rather than sampling a pair of nodes and updating the estimation of all nodes on any SPs between the sampled nodes. Nevertheless, we can actually even generalize the result by Riondato and Kornaropoulos [47], as shown in Theorem 4.6.

THEOREM 4.6. *Let $G = (V, E)$ be a graph such that it is possible to partition the set $\mathcal{D} = \{(u, v) \in V \times V, u \neq v\}$ in two classes: a class $A = \{(u^*, v^*)\}$ containing a single pair of different nodes (u^*, v^*) such that $\sigma_{u^*v^*} \leq 2$ (i.e., connected by either at most two SPs or not connected), and a class $B = \mathcal{D} \setminus A$ of pairs (u, v) of nodes with $\sigma_{uv} \leq 1$ (i.e., either connected by a single SP or not connected). Then, the pseudodimension of the family of functions*

$$\{f_w : \mathcal{D} \to [0, 1], w \in V\},$$

where f_w is defined as in (11), is at most 3.

To prove Theorem 4.6, we show in Lemma 4.7 that some subsets of $\mathcal{D} \times [0, 1]$ cannot be shattered by $\mathcal{F}^+$, on any graph G. Theorem 4.6 follows immediately from this result.

LEMMA 4.7. *There exists no undirected graph $G = (V, E)$, such that it is possible to shatter a set*

$$B = \{((u_i, v_i), x_i), 1 \leq i \leq 4\} \subseteq \mathcal{D} \times [0, 1]$$

if there are at least three distinct values $j', j'', j''' \in [1, 4]$ for which

$$\sigma_{u_{j'}v_{j'}} = \sigma_{u_{j''}v_{j''}} = \sigma_{u_{j'''}v_{j'''}} = 1.$$

Proof. First of all, according to Lemmas 3.7 and 3.8, for B to be shattered it must be

$$(u_i, v_i) \neq (u_j, v_j) \text{ for } i \neq j$$

and $x_i \in (0, 1]$, $1 \leq i \leq 4$.

Riondato and Kornaropoulos [47, Lemma 2] showed that there exists no undirected graph $G = (V, E)$ such that it is possible to shatter B if

$$\sigma_{u_1 v_1} = \sigma_{u_2 v_2} = \sigma_{u_3 v_3} = \sigma_{u_4 v_4} = 1.$$

Hence, what we need to show to prove the thesis is that it is impossible to build an undirected graph $G = (V, E)$ such that $\mathcal{F}^+$ can shatter B when the elements of B are such that

$$\sigma_{u_1 v_1} = \sigma_{u_2 v_2} = \sigma_{u_3 v_3} = 1 \quad \text{and} \quad \sigma_{u_4 v_4} = 2.$$

Assume now that such a graph G exists, and therefore B is shattered by $\mathcal{F}^+$.

For $1 \leq i \leq 3$, let p_i be the *unique* SP from u_i to v_i, and let p_4' and p_4'' be the two SPs from u_4 to v_4.

First of all, notice that if any two of p_1, p_2, p_3 meet at a node a and separate at a node b, then they cannot meet again at any node before a or after b, as otherwise there would be multiple SPs between their extreme nodes, contradicting the hypothesis. Let this fact be denoted as F_1.

Since B is shattered, its subset

$$A = \{((u_i, v_i), x_i), 1 \leq i \leq 3\} \subset B$$

is also shattered, and in particular it can be shattered by a collection of ranges that is a subset of a collection of ranges that shatters B. We now show some facts about the properties of this shattering, which we will use later in the proof.

Define

$$i^+ = \begin{cases} i + 1 & \text{if } i = 1, 2 \\ 1 & \text{if } i = 3 \end{cases}$$

and

$$i^- = \begin{cases} 3 & \text{if } i = 1 \\ i - 1 & \text{if } i = 2, 3 \end{cases}.$$

Let w_A be a node such that $R_{w_A} \cap A = A$. For any set $L = \{k_1, k_2, \ldots\} \subseteq \{1, 2, 3, 4\}$ of indices, let $w_L = w_{k_1, k_2, \ldots}$ be the node such that

$$R_L \cap A = \{((u_{k_\ell}, v_{k_\ell}), x_{k_\ell}), k_\ell \in L\}.$$

For example, for $i \in \{1, 2, 3\}$, w_{i, i^+} is the node such that

$$R_{w_{i, i^+}} \cap A = \{((u_i, v_i), x_i), ((u_{i^+}, v_{i^+}), x_{i^+})\}.$$

Analogously, w_{i, i^-} is the node such that

$$R_{w_{i, i^-}} \cap A = \{((u_i, v_i), x_i), ((u_{i^-}, v_{i^-}), x_{i^-})\}.$$

We want to show that w_A is on the SP connecting w_{i, i^+} to w_{i, i^-} (such a SP must exist because the graph is undirected and w_{i, i^+} and w_{i, i^-} must be on the same connected component, as otherwise they could not be used to shatter A.) Assume w_A was not on the SP connecting w_{i, i^+} to w_{i, i^-}. Then, we would have that either w_{i, i^+} is "between" w_A and w_{i, i^-} (i.e., along the SP connecting these nodes) or w_{i, i^-} is between w_A and w_{i, i^+}. Assume it was the former (the latter follows by symmetry). Then

(1) there must be a SP p' from u_{i^-} to v_{i^+} that goes through w_{i, i^-};
(2) there must be a SP p'' from u_{i^-} to v_{i^+} that goes through w_A;
(3) there is no SP from u_{i^-} to v_{i^+} that goes through w_{i, i^+}.

Since there is only one SP from u_{i^-} to v_{i^-}, it must be that $p' = p''$. But, then p' is a SP that goes through $w_{i,i-}$ and through w_A but not through w_{i,i^+}, and p_i is a SP that goes through w_{i,i^-}, through w_{i,i^+}, and through w_A (either in this order or in the opposite). This means that there are at least two SPs between w_{i,i^-} and w_A, and therefore there would be two SPs between u_i and v_i, contradicting the hypothesis that there is only one SP between these nodes. Hence, it must be that w_A is between w_{i,i^-} and w_{i,i^+}. This is true for all i, $1 \leq i \leq 3$. Denote this fact as F_2.

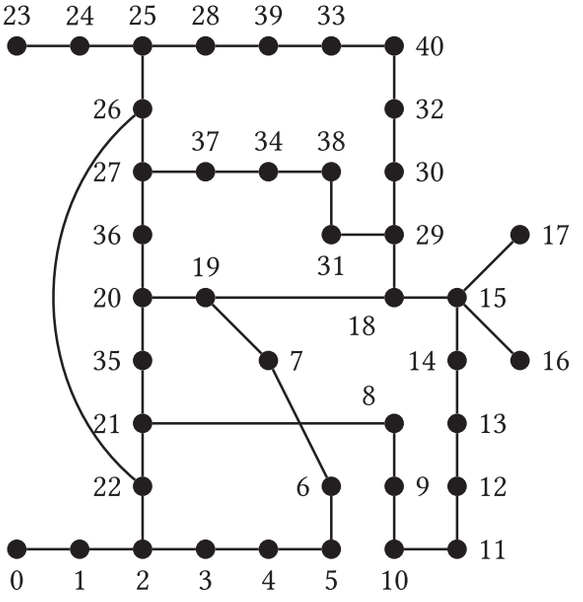
Consider now the nodes $w_{i,4}$ and $w_{j,4}$, for $i, j \in \{1, 2, 3\}$, $i \neq j$. We now show that they cannot belong to the same SP from u_4 and v_4.

- Assume that $w_{i,4}$ and $w_{j,4}$ are on the same SP p from u_4 to v_4 and assume that $w_{i,j,4}$ is also on p. Consider the possible orderings of $w_{i,4}$, $w_{j,4}$, and $w_{i,j,4}$ along p.
  - If the ordering is $w_{i,4}$, then $w_{j,4}$, then $w_{i,j,4}$ or $w_{j,4}$, then $w_{j,4}$, then $w_{i,j,4}$, or the reverses of these orderings (for a total of four orderings), then it is easy to see that fact F_1 would be contradicted, as there are two different SPs from the first of these nodes to the last, one that goes through the middle one, and one that does not, but then there would be two SPs between the pair of nodes (u_k, v_k), where k is the index in $\{1, 2, 3\}$ different than 4 that is in common between the first and the last nodes in this ordering, and this would contradict the hypothesis, so these orderings are not possible.
  - Assume instead the ordering is such that $w_{i,j,4}$ is between $w_{i,4}$ and $w_{j,4}$ (two such ordering exist). Consider the paths p_i and p_j. They must meet at some node $w_{f_{i,j}}$ and separate at some node $w_{l_{i,j}}$. From the ordering, and fact F_1, $w_{i,j,4}$ must be between these two nodes. From fact F_2 we have that also w_A must be between these two nodes. Moreover, neither $w_{i,4}$ nor $w_{j,4}$ can be between these two nodes. But, then consider the SP p. This path must go together with p_i (resp. p_j) from at least $p_{i,4}$ (resp. $p_{j,4}$) to the farthest between $w_{f_{i,j}}$ and $w_{l_{i,j}}$ from $p_{i,4}$ (resp. $p_{j,4}$). Then, in particular p goes through all nodes between $w_{f_{i,j}}$ and $w_{l_{i,j}}$ that p_i and p_j go through. But, since w_A is among these nodes, and w_A cannot belong to p, this is impossible, so these orderings of the nodes $w_{i,4}$, $w_{j,4}$, and $w_{i,j,4}$ are not possible.

  Hence, we showed that $w_{i,4}$, $w_{j,4}$, and $w_{i,j,4}$ cannot be on the same SP from u_4 to v_4.
- Assume now that $w_{i,4}$ and $w_{j,4}$ are on the same SP from u_4 to v_4, but $w_{i,j,4}$ is on the other SP from u_4 to v_4 (by hypothesis there are only two SPs from u_4 to v_4). Since what we show in the previous point must be true for all choices of i and j, we have that all nodes $w_{h,4}$, $1 \leq h \leq 3$, must be on the same SP from u_4 to v_4, and all nodes in the form $w_{i,j,4}$, $1 \leq i < j \leq 3$ must be on the other SP from u_4 to v_4. Consider now these three nodes, $w_{1,2,4}$, $w_{1,3,4}$, and $w_{2,3,4}$ and consider their ordering along the SP from u_4 to v_4 that they lay on. No matter what the ordering is, there is an index $h \in \{1, 2, 3\}$ such that the SP p_h must go through the extreme two nodes in the ordering but not through the middle one. But this would contradict fact F_1, so it is impossible that we have $w_{i,4}$ and $w_{j,4}$ on the same SP from u_4 to v_4, but $w_{i,j,4}$ is on the other SP, for any choice of i and j.

We showed that the nodes $w_{i,4}$ and $w_{j,4}$ cannot be on the same SP from u_4 to v_4. But, this is true for any choice of the unordered pair (i, j) and there are three such choices, but only two SPs from u_4 to v_4, so it is impossible to accommodate all the constraints requiring $w_{i,4}$ and $w_{j,4}$ to be on different SPs from u_4 to v_4. Hence, we reach a contradiction, and B cannot be shattered. □

The bound in Theorem 4.6 is tight, i.e., there exists a graph for which the pseudodimension is exactly 3 [47, Lemma 4]. Moreover, as soon as we relax the requirement in Theorem 4.6 and allow two pairs of nodes to be connected by two SPs, there are graphs with pseudodimension 4, as shown in the following lemma.



(a) Graph for Lemma 4.8

(b) How to shatter $Q = \{a, b, c, d\}$ from Lemma 4.8.

$P \subseteq Q$	Node v s.t. $P = Q \cap R_v$
$\emptyset$	0
$\{a\}$	1
$\{b\}$	24
$\{c\}$	40
$\{d\}$	38
$\{a, b\}$	20
$\{a, c\}$	2
$\{a, d\}$	21
$\{b, c\}$	25
$\{b, d\}$	27
$\{c, d\}$	29
$\{a, b, c\}$	19
$\{a, b, d\}$	15
$\{a, c, d\}$	22
$\{b, c, d\}$	26
$\{a, b, c, d\}$	18

Fig. 3. Supporting figures and table for Lemma 4.8.

Lemma 4.8. *There is an undirected graph $G = (V, E)$ such that there is a set $\{(u_i, v_i), u_i, v_i \in V, u_i \neq v_i, 1 \leq i \leq 4\}$ with $|S_{u_1,v_1}| = |S_{u_2,v_2}| = 2$ and $|S_{u_3,v_3}| = |S_{u_4,v_4}| = 1$ that is shattered.*

Proof. Consider the undirected graph $G = (V, E)$ in Figure 3(a). There is a single SP from 0 to 16, i.e.,

$$0, 1, 2, 22, 21, 35, 20, 19, 18, 15, 16.$$

There is a single SP from 23 to 17, i.e.,

$$23, 24, 25, 26, 27, 36, 20, 19, 18, 15, 17.$$

There are exactly two SPs from 5 to 33,

$$5, 4, 3, 2, 22, 26, 25, 28, 39, 33 \quad \text{and} \quad 5, 6, 7, 19, 18, 29, 30, 32, 40, 33.$$

There are exactly two SPs from 11 to 34,

$$11, 10, 9, 8, 21, 22, 26, 27, 37, 34 \quad \text{and} \quad 11, 12, 13, 14, 15, 18, 29, 31, 38, 34.$$

Let $a = ((0, 16), 1)$, $b = ((23, 17), 1)$, $c = ((5, 33), 1/2)$, and $d = ((11, 34), 1/2)$. We can shatter the set $Q = \{a, b, c, d\}$, as shown in Figure 3(b). □

The significance of this lemma is in the fact that it highlights how the distribution of the numbers of SPs between vertices is indeed the quantity that governs the pseudodimension: a small change in this distribution (from at most a single SP for all pairs of node in Theorem 4.6, to at most a single SP for all-but-one pairs and two SP for that single pair in Lemma 4.8) causes a change in the pseudodimension. We use this evidence to formulate Conjecture 4.9.

For the case of *directed* networks, it is currently an open question whether a high-quality (i.e., within ε) approximation of the BC of all nodes can be computed from a sample whose size is independent of properties of the graph, but it is known that, even if possible, the constant would not be the same as for the undirected case [47, Section 4.1].

We conjecture that, given some information on how many pairs of nodes are connected by x SPs, for $x \geq 0$, it should be possible to derive a strict bound on the pseudodimension associated to the graph. Formally, we pose the following conjecture, which would allow us to generalize Lemma 4.7, and develop an additional stopping rule for ABRA-s based on the (empirical) pseudodimension.

Conjecture 4.9. *Let $G = (V, E)$ be a graph and let ℓ be the maximum positive integer for which there exists a set $L = \{(u_1, v_i), \ldots, (u_\ell, v_\ell)\}$ of ℓ distinct pairs of distinct nodes such that*

$$\sum_{i=1}^{\ell} \sigma_{u_i v_i} \geq \binom{\ell}{\lfloor \ell/2 \rfloor}.$$

then $\mathrm{PD}(\mathcal{F}) \leq \ell$.

The conjecture is tight in the sense that, e.g., for the graph in Figure 3(a), it holds that $\ell = 4$ and the pseudodimension is exactly ℓ, as proven in Lemma 4.8.

4.3 Alternative Estimators

Geisberger et al. [20] present an alternative estimator for BC using random sampling. Their experimental results show that the quality of the approximation is significantly improved, but they do not present any theoretical analysis. Their algorithm, which follows the work of Brandes and Pich [15], differs from ours as it samples nodes and performs a Single-Source-Shortest-Paths (SSSP) computation from each of the sampled nodes. We can use an adaptation of their estimator in a variant of ABRA-s, and we can prove that this variant still computes, with probability at least $1 - \delta$, a ρ-approximation of the BC of all nodes with $\rho \leq \varepsilon$, therefore removing the main limitation of the original work, which offered no quality guarantees. We now present this variant considering, for ease of discussion, the special case of the linear scaling estimator by Geisberger et al. [20]. This technique can be extended to the generic parameterized estimators they present.

The intuition behind the alternative estimator is to increase the estimation of the BC for a node w proportionally to the ratio between the SP distance $\mathrm{d}(u, w)$ from the first component u of the pair (u, v) to w and the SP distance $\mathrm{d}(u, v)$ from u to v. Rather than sampling pairs of nodes, the algorithm samples triples (u, v, d), where d is a *direction*, (either $\leftarrow$ or $\rightarrow$), and updates the betweenness estimation differently depending on d, as follows. Let $\mathcal{D}' = \mathcal{D} \times \{\leftarrow, \rightarrow\}$ and for each $w \in V$, define the function g_w from $\mathcal{D}'$ to $[0, 1]$ as

$$g_w(u, v, d) = \begin{cases} \frac{\sigma_{uv}(w)}{\sigma_{uv}} \frac{\mathrm{d}(u,w)}{\mathrm{d}(u,v)} & \text{if } d = \rightarrow \\ \frac{\sigma_{uv}(w)}{\sigma_{uv}} \left(1 - \frac{\mathrm{d}(u,w)}{\mathrm{d}(u,v)}\right) & \text{if } d = \leftarrow \end{cases}$$

Let $\mathcal{S}$ be a collection of ℓ elements of $\mathcal{D}'$ sampled uniformly and independently at random with replacement. The alternative estimator $\widetilde{\mathrm{b}}(w)$ of the BC of a node w is

$$\widetilde{\mathrm{b}}(w) = \frac{2}{\ell} \sum_{(u,v,d) \in \mathcal{S}} g_w(u, v, d) = 2\mathrm{m}_{\mathcal{S}}(g_w) = \mathrm{m}_{\mathcal{S}}(2g_w).$$

The presence of the factor 2 in the estimator calls for two minor adjustments in this variant of the algorithm w.r.t. the "vanilla" ABRA-s, since, in a nutshell, we now want to estimate the expectations of functions with codomain in $[0, 2]$:

— the update to the vector $\mathbf{v}$ to obtain $\mathbf{v}'$ on line 2 of Algorithm 2 becomes

$$\mathbf{v}' \leftarrow \mathbf{v} \cup \{(j, g_w(u, v, d))\};$$

—the definition of ξ_i on line 21 of Algorithm 1 becomes

$$\xi_i = 2\omega_i^* + \frac{2\gamma_i + \sqrt{2\gamma_i(2\gamma_i + 4\ell\omega_i^*)}}{\ell} + 2\sqrt{\frac{\gamma_i}{2\ell}}.$$

These changes ensure that the output of this variant of ABRA-s is still a high-quality approximation of the BC of all nodes, i.e., that Theorem 4.1 still holds. This is due to the fact that the results on the Rademacher averages presented in Section 3.2 can be extended to families of functions whose codomain is an interval $[0, b]$ (in this case, $b = 2$). Other details such the starting sample size and exact expression to compute the next sample size, described in a previous section, or the relative-error variant described in the following section, can also be adapted to this case.

It is important to mention that, despite having solved the main drawback of the work by Geisberger et al. [20], i.e., its lack of guarantees, the solution is not entirely satisfactory: the presence of the 2 in the estimator results in larger stopping sample sizes than the "vanilla" ABRA-s. This drawback is due to the fact that the size of the co-domain of the functions, which in this case is 2, is used in the proof of Theorem 3.3 in place of the variance, which is suboptimal. This sub-optimality is evident in this case: the alternative estimator is supposed to have a lower variance than the vanilla one, but technical limitations in the proof do not allow us to exploit this fact. Removing this limitation is an interesting direction for future work.

4.4 Fixed Sample Size

In this section, we introduce a variant ABRA-s-fix of ABRA-s that uses a *fixed* sample size, rather than using progressive sampling.

Instead of specifying ε and δ as part of the input, the user specifies δ and a positive integer value S, representing the number of samples (i.e., random pairs of nodes) that the algorithm will take. The algorithm will *always* perform S (and only S) iterations of the loop on lines 9–18 in Algorithm 1, then computes ω^*, γ, and ξ as in lines 19–21, and it will output, *no matter the value of* ξ, the pair $(\widetilde{B}, \xi)$.

THEOREM 4.10. *With probability at least* $1 - \delta$*, the set* $\widetilde{B}$ *is a* ξ*-approximation to the set of exact betweenness centralities for all nodes.*

The proof is immediate from Theorem 3.3 and the definition of ξ.

An advantage of this algorithm with respect to other fixed sampling algorithms, such as the one by Riondato and Kornaropoulos [47], is that it can compute the quality of the approximation directly from the sample, without having to compute characteristic quantities from the graph. Additionally, the parameter S is more interpretable, for an end user, than the parameter ε. On the other hand, the quality of the approximation that will be obtained is not known (or even set by the user) in advance before running the algorithm.

4.5 Relative-Error Top-k Approximation

In practical applications it is usually sufficient to identify the nodes with highest BC, as they act, in some sense, as the "primary information gateways" of the network. In this section, we present a variant ABRA-s-k of ABRA-s to compute a high-quality approximation of the set $\mathrm{TOP}(k, G)$ of the top-k nodes with highest BC in a graph G.

The approximation $\widetilde{\mathrm{b}}(w)$ returned by ABRA-s-k for a node w is within a *multiplicative* factor $\rho \leq \varepsilon$ from its exact value $\mathrm{b}(w)$, rather than an additive factor $\xi \leq \varepsilon$ as probabilistically guaranteed by ABRA-s (see Theorem 4.12 for ABRA-s-k's guarantees). Achieving such higher accuracy guarantees has a cost in terms of the number of samples needed to compute the approximations.

Formally, assume to order the nodes in the graph in decreasing order by BC, ties broken arbitrarily, and let b_k be the BC of the kth node in this ordering. The set $\mathrm{TOP}(k, G)$ of the top-k nodes with highest betweenness in G is defined as the set of nodes with BC at least b_k, and can contain more than k nodes,

$$\mathrm{TOP}(k, G) = \{(w, \mathrm{b}(w)), v \in V \text{ and } \mathrm{b}(w) \geq b_k\}.$$

The algorithm ABRA-s-k follows an approach similar to the one taken by the algorithm for the same task by Riondato and Kornaropoulos [47, Section 5.2] and works in two phases. The pseudocode for ABRA-s-k is presented in Algorithm 3, and we now describe how the algorithm works.

ALGORITHM 3: ABRA-s-k: relative-error approximation of top-k BC nodes on static graph

Input: Graph $G = (V, E)$, accuracy parameter $\varepsilon \in (0, 1)$, confidence parameter $\delta \in (0, 1)$, value $k \geq 1$, sample schedule $(S_i)_{i \geq 1}$
Output: Pair $\left(\widetilde{B}, \rho\right)$, where $\rho \leq \varepsilon$ and $\widetilde{B}$ is a set of approximations of the BC of the top-k nodes in V with highest BC

$\delta', \delta'' \leftarrow$ reals such that $(1 - \delta')(1 - \delta'') = 1 - \delta$
$\left(\widetilde{B}', \xi\right) \leftarrow$ output of ABRA-s with input $G, \varepsilon, \delta', (S_i)_{i \geq 1}$
$\tilde{b}'_k \leftarrow k$th highest value $\widetilde{\mathrm{b}}'(w)$ in $\widetilde{B}'$
$y' \leftarrow \tilde{b}'_k - \xi$
$C \leftarrow \left\{v \in V, \widetilde{\mathrm{b}}(v) \geq \tilde{b}'_k - 2\xi\right\}$
$\left(\widetilde{B}'', \rho\right) \leftarrow$ output of ABRA-s-set-r with input $G, \varepsilon, \delta'', C, y', (S_i)_{i \geq 1}$
$\tilde{b}''_k \leftarrow k$th highest value $\widetilde{\mathrm{b}}''(w)$ in $\widetilde{B}''$
$y'' \leftarrow \frac{\tilde{b}''_k}{1+\rho}$
$z \leftarrow \max\{y', y''\}$
$\widetilde{\mathrm{TOP}}(k, G) = \left\{\left(w, \widetilde{\mathrm{b}}''(w)\right), w \in C \text{ and } \widetilde{\mathrm{b}}''(w) \geq z(1 - \rho)\right\}$
return $\left(\widetilde{\mathrm{TOP}}(k, G), \rho\right)$

Let δ' and δ'' be such that $(1 - \delta')(1 - \delta'') = 1 - \delta$. In the first phase, ABRA-s is run with input G, ε, δ', and $(S_i)_{i \geq 1}$, and it returns the pair $(\widetilde{B}', \xi)$.

Let now $\tilde{b}'_k$ be the kth highest value $\widetilde{\mathrm{b}}'(w)$ in $\widetilde{B}'$, ties broken arbitrarily, and let $y' = \tilde{b}'_k - \xi$. Also let C be the set

$$C = \left\{v \in V, \widetilde{\mathrm{b}}'(v) \geq \tilde{b}'_k - 2\xi\right\}.$$

Before describing the second phase of ABRA-s-k, we introduce a variant ABRA-s-set-r of ABRA-s-set (see Section 4.1.3), which takes an additional parameter $\theta \in (0, 1)$. This parameter plays a role in the stopping condition of ABRA-s-set-r as follows. The only difference between ABRA-s-set-r and ABRA-s-set is in the definition of the quantity ξ_i, which in ABRA-s-set-r becomes

$$\xi_i = \frac{1}{\theta}\left(2\omega_i^* + \frac{\ln(3/\phi_i) + \sqrt{(\ln(3/\phi_i) + 4\ell\omega_i^*)\ln(3/\phi_i)}}{\ell} + \sqrt{\frac{\ln(3/\phi_i)}{2\ell}}\right). \tag{17}$$

In the second phase, ABRA-s-k runs ABRA-s-set-r with input G, ε, δ'', C as the specific set of interest, $\theta = y'$, and $(S_i)_{i \geq 1}$.[8] It returns a pair $(\widetilde{B}'', \rho)$, with $\rho \leq \varepsilon$.

[8]The sample schedules may be different between the first and second phases, but we do not discuss this case for ease of presentation.

Let $\tilde{b}''_k$ be the kth highest value $\widetilde{\mathrm{b}}''(w)$ in $\widetilde{B}''$, ties broken arbitrarily, and let $y'' = \tilde{b}''_k/(1+\rho)$.
ABRA-s-k first computes $z = \max\{y', y''\}$, and then computes the set

$$\widetilde{\mathrm{TOP}}(k, G) = \left\{\left(w, \widetilde{\mathrm{b}}(w)\right), w \in C \text{ and } \widetilde{\mathrm{b}}(w) \geq z(1-\rho)\right\},$$

and returns $(\widetilde{\mathrm{TOP}}(k, G), \rho)$.

Guarantees. We now discuss the quality guarantees of the output of ABRA-s-k.
We first state a result on the output of ABRA-s-set-r.

Theorem 4.11. *Let ε, δ, and θ in $(0,1)$ and let $Z \subset V$. Let $(\widetilde{B} = \{\widetilde{\mathrm{b}}(w), w \in Z\}, \rho)$ be the output of ABRA-s-set-r. With probability at least $1-\delta$ it holds that $\rho \leq \varepsilon$ and*

$$\frac{\left|\widetilde{\mathrm{b}}(w) - \mathrm{b}(w)\right|}{\max\{\theta, \mathrm{b}(w)\}} < \rho, \text{ for all } w \in Z.$$

The proof follows the same steps as the proof for Theorem 4.1, using the definition of ξ_i from (17) and applying Theorem 3.5 instead of Theorem 3.3.
We have the following result showing the properties of the collection $\widetilde{\mathrm{TOP}}(k, G)$.

Theorem 4.12. *With probability at least $1-\delta$, the pair $(\widetilde{\mathrm{TOP}}(k, G), \rho)$ returned by ABRA-s-k is such that $\rho \leq \varepsilon$ and:*

(1) for any pair $(v, \mathrm{b}(v)) \in \mathrm{TOP}(k, G)$, there is a pair $(v, \widetilde{\mathrm{b}}(v)) \in \widetilde{\mathrm{TOP}}(k, G)$ and $\widetilde{\mathrm{b}}(v)$ is such that $|\widetilde{\mathrm{b}}(v) - \mathrm{b}(v)| \leq \rho \mathrm{b}(v)$;
(2) for any pair $(w, \widetilde{\mathrm{b}}(w)) \in \widetilde{\mathrm{TOP}}(k, G)$ such that $(w, \mathrm{b}(w)) \notin \mathrm{TOP}(k, G)$, it holds that $\widetilde{\mathrm{b}}(w) \leq (1+\rho) b_k$.

Proof. With probability at least $1-\delta'$, we have, from the properties of ABRA-s, that $(\widetilde{B}', \xi)$ are such that

$$\left|\widetilde{\mathrm{b}}(w) - \mathrm{b}(w)\right| \leq \xi, \text{ for all } w \in V. \tag{18}$$

With probability at least $1-\delta''$, we have, from the properties of ABRA-s-set-r in Theorem 4.11 that

$$\frac{\left|\widetilde{\mathrm{b}}(w) - \mathrm{b}(w)\right|}{\max\{y', \mathrm{b}(w)\}} \leq \rho, \text{ for all } w \in C. \tag{19}$$

Suppose both these events occur, which happens with probability at least $1-\delta$.
Consider the value y'. Since (18) holds, it is straightforward to see that $y' \leq b_k$. Thus, it also holds that all nodes appearing in $\mathrm{TOP}(k, G)$ belong to C, which may contain other nodes.
Since (19) holds, we have that for all $w \in C$ such that $\mathrm{b}(v) \geq y'$, it holds $\widetilde{\mathrm{b}}''(w)/(1+\rho) \leq b(v)$. Similarly, for all $w \in C$ such that $\mathrm{b}(v) < y$, it holds $\widetilde{\mathrm{b}}''(w)/(1+\rho) \leq y \leq b_k$. It follows from these properties of the nodes in C that $y'' \leq b_k$.
Since $y' \leq b_k$ and $y'' \leq b_k$, it holds $z \leq b_k$. Any w appearing in $\mathrm{TOP}(k, G)$ therefore has $\widetilde{\mathrm{b}}(w) \geq z(1-\rho)$. It follows that all nodes appearing in $\mathrm{TOP}(k, G)$ will be in $\widetilde{\mathrm{TOP}}(k, G)$, satisfying the first part of (1) in the statement of the theorem.
The second part of (1) follows from (19), since for all nodes v appearing in $\mathrm{TOP}(k, G)$ it holds that $\mathrm{b}(v) \geq y'$.
Property (2) in the statement follows from (19), noticing that some of the nodes under consideration may have $\mathrm{b}(v) < y' \leq b_k$. □

5 APPROXIMATING BETWEENNESS CENTRALITY IN DYNAMIC GRAPHS

Fully dynamic graphs are graphs that evolve over time, with nodes and edges added and removed at each time step. In this section, we show how the analysis based on Rademacher averages and pseudodimension presented in the previous section can be used to improve an algorithm by Hayashi et al. [24] that computes and keeps up-to-date an high-quality approximation of the BC of all nodes in a fully dynamic graph. The improvement consists in a large reduction in the number of samples needed by Hayashi et al.'s algorithm.

Hayashi et al. [24] introduced two fast data structures called the Hypergraph Sketch and the Two-Ball Index: the Hypergraph Sketch stores the BC estimations for all nodes, while the Two-Ball Index is used to store the SP DAGs and to understand which parts of the Hypergraph Sketch needs to be modified after an update to the graph (i.e., an edge or node insertion or deletion). Hayashi et al. [24] show how to use these data structures to maintain a set of estimation that is, with probability at least $1 - \delta$, an ε-approximation of the BC of all nodes in a fully dynamic graph.

Using the novel data structures results in orders-of-magnitude speedups w.r.t. previous contributions [8, 9]. The algorithm by Hayashi et al. [24] is based on a random sampling where pairs of nodes are sampled and the BC estimation of the nodes along the SPs between the two nodes are updated as necessary. The same sampling scheme is used by ABRA-s, but the Hayashi et al.'s algorithm uses a *fixed* number of samples computed at the beginning of the algorithm and never changed during execution. Hayashi et al.'s analysis of the number of samples necessary to obtain, with probability at least $1 - \delta$, an ε-approximation of the BC of all nodes uses the union bound, resulting in a number of samples that depends on the logarithm of the number of nodes in the graph, i.e., $O(\varepsilon^{-2}(\log(|V|/\delta)))$ pairs of nodes must be sampled.

The progressive sampling approach and the stopping condition used by ABRA-s can be used in Hayashi et al.'s algorithm in place of a fixed sample size. As a result, the algorithm can decide when it has sampled enough to first populate the Hypergraph Sketch and the Two-Ball Index at the beginning of the algorithm.

After each update to the graph, the algorithm, in addition to performing bookkeeping operations on the Hypergraph Sketch and the Two-Ball Index, must keep up-to-date the set $\mathcal{V}_S$ and the map M (already used in ABRA-s). Once the data structures are up-to-date, the algorithm checks whether the stopping condition is still satisfied. If it is not, additional pairs of nodes are sampled and the Hypergraph Sketch and the Two-Ball Index are updated with the estimations resulting from these additional samples. The sampling of additional pairs continues until the stopping condition is satisfied, according to a sample schedule. There is no need to discard samples when the stopping condition is satisfied: the value ξ returned by the algorithm would just be even smaller than ε.

The overhead of additional checks of the stopping condition is minimal because checking the stopping condition is extremely efficient, as we show in our experimental evaluation. On the other hand, the use of the progressive sampling scheme based on the Rademacher averages allows the algorithm to sample much fewer pairs of nodes than in the static sampling case based on the union bound: Riondato and Kornaropoulos [47] already showed that it is possible to sample much less than $O(\log |V|)$ nodes and, as we show in our experiments, the sample sizes required by the progressive sampling approach are even smaller than the ones used in the algorithm by Riondato and Kornaropoulos [47], and thus in the one by Hayashi et al. [24]. Reducing the number of samples naturally leads to a speedup, as the running time of the algorithms are, in a first approximation, linear in the number of samples. Additionally, the amount of space required to store the data structures would also decrease, as they now store information about fewer SP DAGs.

THEOREM 5.1. *The pair* $(\widetilde{B} = \{\widetilde{\mathrm{b}}(w), w \in V\}, \xi)$ *returned by this variant of the algorithm by Hayashi et al. [24] after each update to the graph has been processed, is such that* $\xi \leq \varepsilon$ *and*

Table 3. Characteristics of the Graphs

Graph	Directed	Vertices	Edges
Cit-HepPh	N	34,546	421,578
Email-Enron	N	36,682	183,831
p2p-Gnutella31	Y	62,586	147,892
Soc-Epinions1	Y	75,879	508,837

$$\Pr(\exists w \in V \text{ s.t. } |\widetilde{\mathrm{b}}(w) - \mathrm{b}(w)| > \xi) < \delta.$$

The proof follows from the correctness of the algorithm by Hayashi et al. [24] and of ABRA-s (Theorem 4.1).

6 EXPERIMENTAL EVALUATION

In this section, we presents the results of our experimental evaluation. We measure and analyze the performances of ABRA-s in terms of its runtime and sample size and accuracy, and compared them with those of the exact algorithm BA [14] and the approximation algorithm RK [47], which offers the same guarantees as ABRA-s, i.e., it computes, with probability at least $1 - \delta$, an ε-approximation of the BC of all nodes.

Implementation and environment. We implement ABRA-s and ABRA-s-fix in C++11, as an extension of the NetworKit library [52]. We use NLopt [27] for the optimization steps. The code is available from http://matteo.rionda.to/software/ABRA-radebetw.tbz2. We performed the experiments on a machine with a AMD Phenom^{TM} II X4 955 processor and 16GB of RAM, running FreeBSD 12.

Datasets and parameters. We use graphs of various nature (communication, citations, P2P, and social networks) from the SNAP repository [35]. The characteristics of the graphs are reported in the Table 3.

In our experiments, we set ε in the range $[0.01, 0.03]$. In all the results, we report that δ is fixed to 0.1. We experimented with different values for this parameter, and, as expected, it has a very limited impact on the nature of the results, given the logarithmic dependence of the sample size on δ. For the sample schedule, we used a geometrically increasing sample schedule with $S_i = c^i S_0$ for all $i \geq 1$, where S_0 is the r.h.s. of (16) and $c \in \{1.2, 1.5, 2.0, 3.0\}$.

We performed five runs for each combination of parameters. The variance between the different runs was insignificant, so we report, unless otherwise specified, the results for a random run.

The "Change" column in some of the following tables reports the *percentage change* in the metric of interest w.r.t. a baseline, computed as

$$\text{change} = 100\frac{\text{ABRA-s} - \text{baseline}}{\text{baseline}}.$$

For example, if the metric of interest were the runtime, and the measured runtime of ABRA-s was 6 seconds and the one for the baseline BA was 9 seconds, then the percentage change would be -33.33.

6.1 Accuracy

We evaluate the accuracy of ABRA-s by measuring the absolute error $|\widetilde{\mathrm{b}}(v) - \mathrm{b}(v)|$ for al nodes v in the graph. The theoretical analysis guarantees that this quantity should be at most ξ and therefore at most ε for all nodes, with probability at least $1 - \delta$.

Table 4. Comparison between ε and ξ, and Error Distribution

Graph	Error bound		Error distribution		
	$\varepsilon \times 10^3$	$\xi \times 10^3$	Avg $\times 10^3$	Stdev $\times 10^3$	Max $\times 10^3$
Cit-HepPh	10	8.816	0.014	0.027	0.920
	15	13.175	0.021	0.040	1.374
	20	17.582	0.027	0.052	2.044
	25	21.892	0.033	0.067	2.153
	30	26.322	0.039	0.082	2.956
Email-Enron	10	8.689	0.005	0.023	1.008
	15	13.047	0.007	0.034	1.393
	20	17.349	0.009	0.048	2.112
	25	21.757	0.011	0.061	3.119
	30	25.992	0.013	0.070	2.515
P2p-Gnutella31	10	8.978	0.009	0.026	0.445
	15	13.438	0.014	0.038	0.797
	20	17.945	0.018	0.050	0.795
	25	22.645	0.022	0.063	1.514
	30	27.102	0.026	0.077	1.671
Soc-Epinions1	10	9.872	0.007	0.021	0.839
	15	14.815	0.009	0.030	0.843
	20	19.678	0.012	0.041	1.595
	25	24.637	0.014	0.052	2.135
	30	29.738	0.016	0.063	2.467

Note: Geometric schedule with $c = 1.5$.

An important result is that in *all* the thousands of runs of ABRA-s, the maximum error was *always* smaller than ξ, not just with probability $>1 - \delta$.

We report statistics about the absolute error in Table 4, computed for runs with a geometric schedule with $c = 1.5$ (results for other values of c were qualitatively equivalent). The minimum error was always zero, so we do not report it in the table. The maximum error is almost always *an order of magnitude smaller than* ξ, and the average error is around *three orders of magnitude* smaller. The standard deviation, which is very close to the average, suggests that most of the errors are almost two orders of magnitude smaller than ε, as can be verified by considering the average error plus three standard deviations, which include approximately 95% of the nodes. As expected, the maximum error grows as ε^{-2}.

These results show that *ABRA-sis very accurate, more than what is guaranteed by the theoretical analysis*. This can be explained by the fact that the bounds to the sampling size, the stopping condition, and the sample schedule are *conservative*, in the sense that ABRA-s may be sampling more than necessary to obtain an ε-approximation with probability at least $1 - \delta$. Tightening any of these components would result in a less conservative algorithm that offers the same approximation quality guarantees and is an interesting research direction.

6.2 Runtime and Sample Size

We evaluate the runtime of ABRA-s and compare it with the ones of BA and RK. Doing so also gives us a chance to comment on the sample sizes used by ABRA-s and compare them with those used by RK. The results are reported in Tables 5–8 (one table per graph).

Table 5. Runtime and Sample Size Comparison for the Cit-HepPh Graph

		Runtime (seconds)						
			Change vs.		Sample size		Err. bound $\xi \times 10^3$	
$\varepsilon \times 10^3$	Schedule	ABRA-s	BA	RK	ABRA-s	Change vs. RK	ABRA-s	Change vs. RK
10	$c = 1.2$	346.25	−32.14	−27.45	52,961	−27.48	9.746	−2.54
	$c = 1.5$	314.39	−38.38	−34.12	47,888	−34.42	8.816	−11.84
	$c = 2.0$	276.15	−45.88	−42.14	42,566	−41.71	8.778	−12.22
	$c = 3.0$	417.32	−18.21	−12.56	63,849	−12.57	7.119	−28.81
15	$c = 1.2$	156.97	−69.23	−11.63	23,982	−26.11	14.645	−2.37
	$c = 1.5$	139.28	−72.70	−34.44	21,684	−33.19	13.175	−12.16
	$c = 2.0$	123.92	−75.71	−41.67	19,274	−40.61	13.186	−12.09
	$c = 3.0$	187.97	−63.16	−11.53	28,911	−10.92	10.634	−29.11
20	$c = 1.2$	88.73	−82.61	−25.78	13,737	−24.76	19.393	−3.04
	$c = 1.5$	80.42	−84.24	−32.73	12,420	−31.97	17.582	−34.12
	$c = 2.0$	71.88	−85.91	−39.88	11,040	−39.53	17.601	−11.99
	$c = 3.0$	107.18	−78.99	−10.35	16,560	−9.30	14.238	−28.81
25	$c = 1.2$	57.80	−88.67	−24.40	8,949	−23.41	24.346	−2.61
	$c = 1.5$	51.43	−89.92	−32.74	8,091	−30.76	21.892	−29.67
	$c = 2.0$	46.65	−90.86	−38.99	7,192	−38.45	21.835	−12.66
	$c = 3.0$	68.98	−86.48	−9.78	10,787	−7.69	17.702	−29.19
30	$c = 1.2$	41.22	−91.92	−22.27	6,324	−22.06	29.478	−1.74
	$c = 1.5$	37.15	−92.72	−29.95	5,717	−29.54	26.322	−12.26
	$c = 2.0$	32.80	−93.57	−38.16	5,081	−37.38	26.374	−12.09
	$c = 3.0$	48.90	−90.42	−7.80	7,621	−6.08	21.223	−29.26

General comments. We can observe that the runtime is essentially proportional to $1/\varepsilon^2$ and to the sample size, as expected from the theoretical results. There is no clear dependency between the sample schedule and the resulting runtime, a phenomenon that we analyze in detail in the following paragraphs.

We broke down the runtime in three components: "sampling" (inclusive of the time needed to run s-t SP computations and updating the data structures), "stopping condition" (inclusive of computing ω^* and ξ), and "other." In all the runs, "sampling" accounted for more than 99% of the total runtime.[9] In other words, ABRA-s spent almost all its execution doing useful work, i.e., collecting samples, running s-t SP computations, and updating the data structures. This is contrast with other algorithms, e.g., RK, that need to compute some characteristic quantity of the graph at the beginning of the execution. Evaluating the stopping condition is extremely efficient: it accounted for less than one percent of the runtime.

Comparison with BA . The change w.r.t. the exact algorithm BA is significant, reaching at times values smaller than −90 (essentially a 10x speedup) and almost always smaller than −50 (a 2x speedup). The change tends to be smaller in magnitude as ε becomes smaller, as expected. It happened only once, on the email-Enron graph for $\varepsilon = 0.010$ and $c = 2.0$ that BA was faster than ABRA-s. This event, like similar ones we discuss later in this section, happens due to the fact that the sampling schedule must be fixed in advance, i.e., it cannot be adaptive or the correctness of the

[9]These results are not reported in any table, as they were essentially the same for all graphs and all combinations of the parameters, so we limit ourselves to commenting them.

Table 6. Runtime and Sample Size Comparison for the Email-Enron Graph

		Runtime (seconds)			Sample size		Err. bound $\xi \times 10^3$	
			Change vs.					
$\varepsilon \times 10^3$	Schedule	ABRA-s	BA	RK	ABRA-s	Change vs. RK	ABRA-s	Change vs. RK
10	$c = 1.2$	218.03	−9.10	−2.17	76,265	**4.44**	9.863	−1.37
	$c = 1.5$	206.13	−14.06	−7.51	71,832	−1.64	8.689	−13.11
	$c = 2.0$	246.06	**2.59**	**10.40**	85,132	**16.58**	7.585	−24.15
	$c = 3.0$	183.76	−23.38	−17.55	63,849	−12.57	8.319	−16.81
15	$c = 1.2$	99.89	−58.35	**0.25**	34,535	**6.41**	14.853	−0.98
	$c = 1.5$	93.29	−61.10	−6.09	32,526	**0.22**	13.047	−13.02
	$c = 2.0$	110.57	−53.90	**11.30**	38,548	**18.77**	11.368	−24.21
	$c = 3.0$	82.95	−65.42	−16.50	28,911	−10.92	12.446	−17.03
20	$c = 1.2$	57.08	−76.20	**2.01**	19,782	**8.35**	19.698	−1.51
	$c = 1.5$	53.63	−77.64	−4.15	18,630	**2.04**	17.349	−34.76
	$c = 2.0$	63.17	−73.66	**12.89**	22,080	**20.94**	15.105	−24.47
	$c = 3.0$	47.05	−80.38	−16.02	16,560	−9.30	16.623	−16.89
25	$c = 1.2$	37.38	−84.42	**5.18**	12,887	**10.29**	24.647	−1.41
	$c = 1.5$	34.81	−85.49	−2.06	12,137	**3.87**	21.757	−30.61
	$c = 2.0$	40.98	−82.92	15.30	14,384	**23.10**	18.756	−24.98
	$c = 3.0$	30.82	−87.15	−13.28	10,787	−7.69	20.801	−16.80
30	$c = 1.2$	26.23	−89.06	**6.27**	9,107	**12.24**	29.619	−1.27
	$c = 1.5$	24.77	−89.67	**0.38**	8,576	**5.69**	25.992	−13.36
	$c = 2.0$	29.29	−87.79	**18.68**	10,162	**25.24**	22.461	−25.13
	$c = 3.0$	21.86	−90.88	−11.42	7,621	−6.08	24.781	−17.40

Note: The cases when ABRA-s was worse than BA and/or RK are in bold font.

algorithm is compromised. The consequence is that the algorithm may sample much more than needed. Specifically, when the value ξ computed at the end of an iteration is very close but still larger than ε, the algorithm must move to the next iteration, which will have a much larger sample size (c times larger), and where almost surely ξ will be much smaller than ε and the algorithm will terminate. This is indeed the case for email-Enron, $\varepsilon = 0.010$ and $c = 2.0$, as can be seen by looking at the error bound ξ computed by the algorithm and reported in the second-to-last column from the right in Table 6: ξ is significantly less than ε, and at the iteration before the last, we checked that ξ was very close but larger than ε. Indeed ABRA-s with other sample schedules is able to stop much earlier.

Comparison with RK. Comparing the runtime of ABRA-s with that of RK, we can observe that the former is almost always much faster than the latter, reaching up to 6x speedups. The one exception is again the email-Enron graph (Table 6), where ABRA-s is at times slower than RK. We already mentioned how this phenomenon can be traced back to the use of a static sampling schedule. We will show in Section 6.4 that, when given the same amount of samples, the ABRA-s-fix variant performs much better than RK.

This relative slowdown is compensated by a reduction, often significantly greater than the slowdown, in the error bound ξ (which for RK is ε): the additional time taken by ABRA-s to collect more samples is not "wasted," as it results in a better upper bound on the maximum error, as can be seen in the two rightmost columns in the tables. Hence, the "rigidness" due to the fixed sample schedule is significantly softened.

Table 7. Runtime and Sample Size Comparison for the p2p-Gnutella31 Graph

		Runtime (sec.)			Sample size		Err. bound $\xi \times 10^3$	
			Change vs.					
$\varepsilon \times 10^3$	Schedule	ABRA-s	BA	RK	ABRA-s	Change vs. RK	ABRA-s	Change vs. RK
10	$c = 1.2$	33.06	−81.47	−69.27	30,648	−63.09	9.859	−1.41
	$c = 1.5$	34.11	−80.89	−68.29	31,925	−61.55	8.978	−10.22
	$c = 2.0$	45.80	−74.34	−57.43	42,566	−48.73	7.732	−22.68
	$c = 3.0$	68.51	−61.61	−36.32	63,849	−23.10	6.292	−37.08
15	$c = 1.2$	15.49	−91.32	−30.27	13,878	−62.39	14.901	−0.66
	$c = 1.5$	15.29	−91.43	−68.18	14,456	−60.82	13.438	−10.42
	$c = 2.0$	20.55	−88.49	−57.24	19,274	−47.77	11.636	−22.43
	$c = 3.0$	31.18	−82.53	−35.12	28,911	−21.65	9.433	−37.12
20	$c = 1.2$	8.60	−95.18	−68.78	7,949	−61.70	19.943	−0.29
	$c = 1.5$	8.87	−95.03	−67.79	8,280	−60.11	17.945	−32.81
	$c = 2.0$	11.81	−93.38	−57.12	11,040	−46.81	15.474	−22.63
	$c = 3.0$	17.67	−90.10	−35.82	16,560	−95.01	12.513	−37.44
25	$c = 1.2$	5.60	−96.86	−68.09	5,178	−61.02	24.719	−1.12
	$c = 1.5$	5.88	−96.70	−66.47	5,394	−59.40	22.645	−28.22
	$c = 2.0$	7.88	−95.58	−55.08	7,192	−45.86	19.200	−23.20
	$c = 3.0$	11.30	−93.67	−35.59	10,787	−18.80	15.669	−37.32
30	$c = 1.2$	4.01	−97.75	−67.85	3,659	−60.34	29.879	−0.40
	$c = 1.5$	4.19	−97.65	−66.44	3,811	−58.69	27.102	−9.66
	$c = 2.0$	5.56	−96.88	−55.43	5,081	−44.93	23.430	−21.90
	$c = 3.0$	8.28	−95.36	−33.65	7,621	−17.40	18.790	−37.37

We observe the slowdown only on email-Enron. The reason is that email-Enron is the graph we analyzed with the smallest diameter, therefore RK can use a small sample size. We show in Section 6.3 that RK scale very badly as the diameter grows, while ABRA-s does not. On the other hand, for ABRA-s we do not have practical bounds to the pseudodimension of the problem (see Section 4.2), thus the algorithm cannot deterministically stop after having sample a certain number of pairs of nodes, like RK does. Finding better bounds to the pseudodimension of the problem, for example by proving Conjecture 4.9, would allow ABRA-s to stop earlier. A possible workaround, until that happens, is to run RK in parallel with ABRA-s, running both with $\delta' = \delta/2$, and making RK use the same random samples and the same s-t SP computations, and stopping as either algorithm stops and returning the corresponding approximation.

6.3 Scalability

In this section, we compare the scalability of ABRA-s to that of RK as the vertex-diameter of the graph grows. This choice is motivated by the fact that approximate betweenness estimation algorithms tend[10] to scale well as the numbers of nodes and edges grows because in many graph evolution models the vertex-diameter of the network tends to shrink as these quantities increase [34]. On the other hand, RK is known to be susceptible to growth in the vertex-diameter because its

[10]The one algorithm that does not scale well being the algorithm by Brandes and Pich [15] due to the fact that its sample size depends on the number of nodes in the graph.

Table 8. Runtime and Sample Size Comparison for the Soc-Epinions1 Graph

		Runtime (seconds)						
			Change vs.		Sample size		Err. bound $\xi \times 10^3$	
$\varepsilon \times 10^3$	Schedule	ABRA-s	BA	RK	ABRA-s	Change vs. RK	ABRA-s	Change vs. RK
10	$c = 1.2$	185.48	−71.79	−55.03	44,134	−39.56	9.976	−0.24
	$c = 1.5$	132.62	−79.83	−67.85	31,925	−56.28	9.872	−1.28
	$c = 2.0$	178.07	−72.91	−56.83	42,566	−41.71	8.495	−15.05
	$c = 3.0$	267.00	−59.39	−35.27	63,849	−12.57	6.908	−30.92
15	$c = 1.2$	83.80	−87.25	−35.60	19,985	−38.42	14.994	−0.04
	$c = 1.5$	60.02	−90.87	−73.98	14,456	−55.46	14.815	−1.24
	$c = 2.0$	80.13	−87.81	−65.26	19,274	−40.61	12.648	−15.68
	$c = 3.0$	120.28	−81.70	−47.85	28,911	−10.92	10.422	−30.52
20	$c = 1.2$	48.22	−92.67	−71.08	11,447	−37.30	19.982	−0.09
	$c = 1.5$	34.29	−94.78	−79.43	8,280	−54.65	19.678	−25.93
	$c = 2.0$	46.78	−92.88	−71.94	11,040	−39.53	16.930	−15.35
	$c = 3.0$	69.24	−89.47	−58.46	16,560	−9.30	13.831	−30.85
25	$c = 1.2$	31.34	−95.23	−77.14	7,457	−36.18	24.949	−0.20
	$c = 1.5$	22.93	−96.51	−83.27	5,394	−53.84	24.637	−21.29
	$c = 2.0$	29.89	−95.45	−78.20	7,192	−38.45	20.892	−16.43
	$c = 3.0$	45.32	−93.11	−66.94	10,787	−7.69	17.351	−30.59
30	$c = 1.2$	22.35	−96.60	−81.50	5,270	−35.05	29.691	−1.03
	$c = 1.5$	15.88	−97.58	−86.86	3,811	−53.03	29.738	−0.87
	$c = 2.0$	21.48	−96.73	−82.22	5,081	−37.38	25.196	−16.01
	$c = 3.0$	32.11	−95.12	−73.42	7,621	−6.08	20.495	−31.68

sample size depends on this quantity. We here evaluate the resilience of ABRA-s to changes in the vertex-diameter.

We create artificial graphs using the email-Enron graph as the starting point, then selecting a node v at random, and adding a *tail* (or chain) of k edges starting from v. Thinking of the original graph as a balloon, the tail can be imagined as the string to hold the balloon. We use $k = 20, 40, 80, 160$. The email-Enron graph is undirected and has a (vertex)-diameter of 11, so by adding the tail the resulting graph had a much larger vertex-diameter, hence RK would have to collect more samples.

The results are presented in Table 9. ABRA-s used a geometric sample schedule with $c = 1.5$. It is evident that ABRA-s's runtime and sample size scale well and actually are independent from changes in the tail length, while this is clearly not the case for RK. This phenomena can be explained by the fact that ABRA-s's analysis is based on Rademacher averages, which take into account the distribution of the path lengths, while RK uses VC dimension, which considers only the worst case.

6.4 Fixed Sample Size

In this section, we compare the performances of ABRA-s-fix, presented in Section 4.4, and RK. We report the results on email-Enron with $\delta = 0.1$ in Table 10. The sample sizes are chosen as a result of the way RK computes its sample size: we fix ε and compute the sample size such that, with probability at least $1 - \delta$, RK outputs an ε-approximation. We then run ABRA-s-fix with the

Table 9. Scalability as the Vertex-Diameter Grows Due to the Addition of a Tail of Length k.

		Runtime (seconds)			Sample size		
ε	k	ABRA-s	RK	Change	ABRA-s	RK	Change
0.005	20	800.44	914.67	−12.49	281,870	332,104	−15.13
	40	802.54	1027.48	−21.89	281,870	372,104	−24.25
	80	803.72	1140.89	−29.55	281,870	412,104	−31.60
	160	800.98	1254.28	−36.14	281,870	452,104	−37.65
0.010	20	202.89	228.98	−11.39	71,832	83,026	−13.48
	40	203.79	256.92	−20.68	71,832	93,026	−22.78
	80	201.63	284.96	−29.24	71,832	103,026	−30.28
	160	203.34	313.55	−35.15	71,832	113,026	−36.45
0.015	20	92.13	101.30	−9.06	32,526	36,901	−11.86
	40	91.76	114.27	−19.69	32,526	41,345	−21.33
	80	91.13	126.80	−28.13	32,526	45,790	−28.97
	160	91.37	139.45	−34.47	32,526	50,234	−35.25
0.020	20	52.92	57.10	−7.32	18,630	20,757	−10.25
	40	51.84	64.35	−19.43	18,630	23,257	−19.90
	80	52.08	71.40	−27.06	18,630	25,757	−27.67
	160	52.81	78.56	−32.77	18,630	28,257	−34.07
0.025	20	34.13	36.73	−7.06	12,137	13,285	−8.64
	40	34.32	41.11	−16.51	12,137	14,885	−18.46
	80	33.98	45.65	−25.57	12,137	16,485	−26.38
	160	34.39	50.05	−31.29	12,137	18,085	−32.89
0.030	20	24.25	25.45	−4.73	8,576	9,226	−7.05
	40	16.12	28.61	−43.64	5,717	10,337	−44.69
	80	24.02	31.70	−24.24	8,576	11,448	−25.09
	160	24.44	34.71	−29.57	8,576	12,559	−31.71

Table 10. Comparison between ε and ξ, and Change for ABRA-s-fix and RK on Email-Enron

	Error bound		Change	
Sample size	$\varepsilon \times 10^3$	$\xi \times 10^3$	Error bound	Sample size
73,026	10	7.74	−22.52	−66.56
32,456	15	11.71	−21.87	−63.83
18,257	20	15.71	−21.45	−62.09
11,685	25	19.76	−20.94	−59.98
8,114	30	23.79	−20.67	−58.90

same sample size and value δ, and observe the value for ξ, the second component of the output of ABRA-s-fix, to compare it with ε.

The fourth column from the left reports the percentage change between ξ and ε. The fifth column from the left shows the percentage change between the sample size used by ABRA-s-fix and the sample size that RK would need to achieve an ε equal to the ξ reported in the third column. The reduction in the bound to the maximum error exceeds 20%, and the reduction in sample size is around 60%.

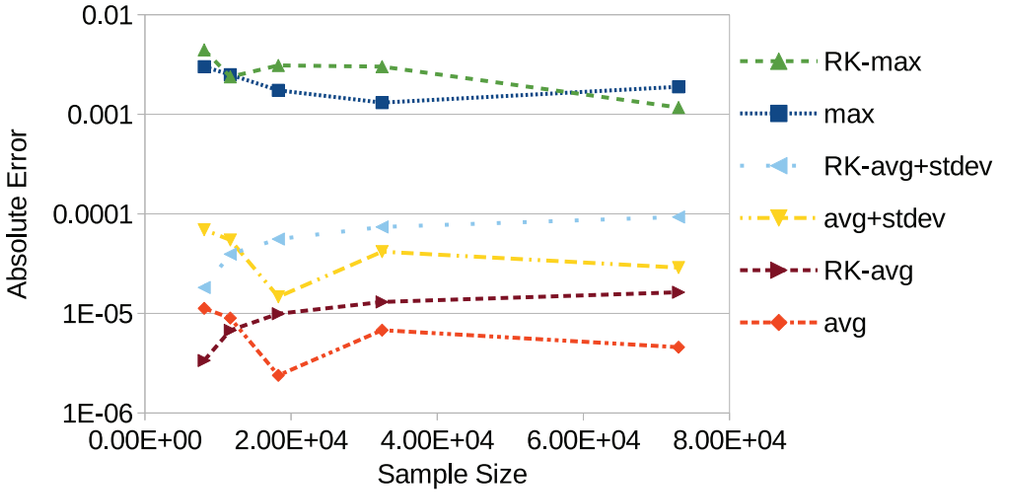


Fig. 4. Accuracy comparison at fixed sample size between ABRA-s-fix and RK.

In Figure 4, we show a comparison of statistics of the absolute error achieved by ABRA-s-fix and RK, at the same sample sizes. Once again, we show the results for a single run for each sample size, which explains the nonmonotonicity of some of the reported quantities as the sample size increases. This comparison is effectively a comparison of the different estimators used by the two algorithms. The one used by ABRA-s-fix takes into account all SPs between a sampled pair of nodes, while the one used by RK only considers a randomly-chosen one between such SPs. Thus, ABRA-s-fix essentially extracts "more information per sample." It is therefore not surprising that in general the average, maximum, and average-plus-standard-deviation of the absolute errors of ABRA-s-fix are lower than the corresponding quantities for RK.

These results are even more remarkable when one considers the fact that RK is given the big advantage of computing the diameter of the graph, which is needed by RK to compute the sample size. Without this knowledge, RK would not even be able to start. On the other hand, ABRA-s-fix has no need for any information about the complete graph, and can just start sampling and compute its quality guarantees only on the sample, thanks to the use of Rademacher averages.

7 CONCLUSIONS

We presented ABRA, a family of sampling-based algorithms for computing and maintaining high-quality approximations of (variants of) the BC of all nodes in static and dynamic graphs with updates (both deletions and insertions). We discussed a number of variants of our basic algorithm, including finding the top-k nodes with higher BC, using improved estimators, using a fixed sample size, and special cases when there is a single SP betwen nodes. ABRA greatly improves, theoretically and experimentally, the current state of the art. The analysis relies on Rademacher averages and on pseudodimension, fundamental concepts from statistical learning theory. To our knowledge this is the first application of these results and ideas to graph mining, and we believe that they should be part of the toolkit of any algorithm designer interested in efficient algorithms for data analysis.

ACKNOWLEDGMENTS

The authors are thankful to Elisabetta Bergamini and Christian Staudt for their help with the NetworKit code, to Emanuele Natale and Michele Borassi for finding a mistake, now corrected, in one of our proofs, and to Nikolaj Tatti and Fabio Vandin for the stimulating discussions.

REFERENCES

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. 2015. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1681–1697.

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, and Sandro Ridella. 2014. A deep connection between the Vapnik–Chervonenkis entropy and the Rademacher complexity. *IEEE Transactions on Neural Networks and Learning Systems* 25, 12 (2014), 2202–2211.

[3] Jac M. Anthonisse. 1971. *The Rush in a Directed Graph*. Technical Report BN 9/71. Stichting Mathematisch Centrum, Amsterdam, Netherlands.

[4] Martin Anthony and Peter L. Bartlett. 1999. *Neural Network Learning – Theoretical Foundations*. Cambridge University Press.

[5] Martin Anthony and John Shawe-Taylor. 1993. A result of Vapnik with applications. *Discrete Applied Mathematics* 47, 3 (1993), 207–217.

[6] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. 2007. Approximating betweenness centrality. In *Algorithms and Models for the Web-Graph*, Anthony Bonato and Fan R. K. Chung (Eds.), Lecture Notes in Computer Science, Vol. 4863. Springer, Berlin, 124–137.

[7] Peter L. Bartlett and Gábor Lugosi. 1999. An inequality for uniform deviations of sample averages from their means. *Statistics & Probability Letters* 44, 1 (1999), 55–62.

[8] Elisabetta Bergamini and Henning Meyerhenke. 2015. Fully-dynamic approximation of betweenness centrality. In *Proceedings of the 23rd European Symposium on Algorithms (ESA'15)*. 155–166.

[9] Elisabetta Bergamini and Henning Meyerhenke. 2016. Approximating betweenness centrality in fully-dynamic networks. *Internet Mathematics* 12, 5 (2016), 281–314.

[10] Elisabetta Bergamini, Henning Meyerhenke, and Christian L. Staudt. 2015. Approximating betweenness centrality in large evolving networks. In *Proceedings of the 17th Workshop on Algorithm Engineering and Experiments (ALENEX'15)*. SIAM, 133–146.

[11] Michele Borassi and Emanuele Natale. 2016. KADABRA is an adaptive algorithm for betweenness via random approximation. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA'16)*. 20:1–20:18.

[12] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. 2005. Theory of classification: A survey of some recent advances. *ESAIM: Probability and Statistics* 9 (2005), 323–375.

[13] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[14] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* 25, 2 (2001), 163–177.

[15] Ulrik Brandes and Christian Pich. 2007. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos* 17, 7 (2007), 2303–2318.

[16] Corinna Cortes, Spencer Greenberg, and Mehryar Mohri. 2013. Relative deviation learning bounds and generalization with unbounded loss functions. *arXiv:1310.5796v4* (Oct. 2013).

[17] Tapio Elomaa and Matti Kääriäinen. 2002. Progressive Rademacher sampling. In *Proceedings of AAAI/IAAI*, Rina Dechter and Richard S. Sutton (Eds.). AAAI Press/MIT Press, 140–145.

[18] Dóra Erdős, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. 2015. A divide-and-conquer algorithm for betweenness centrality. In *Proceedings of SIAM International Conference on Data Mining (SDM'15)*. SIAM, 433–441.

[19] Linton C. Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* 40 (1977), 35–41.

[20] Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08)*. SIAM, 90–100.

[21] O. Green, R. McColl, and David A. Bader. 2012. A fast algorithm for streaming betweenness centrality. In *Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust (PASSAT'12)*. IEEE, 11–20.

[22] Sariel Har-Peled and Micha Sharir. 2011. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry* 45, 3 (2011), 462–496.

[23] David Haussler. 1992. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation* 100, 1 (1992), 78–150.

[24] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully dynamic betweenness centrality maintenance on massive networks. *Proceedings of the VLDB Endowment* 9, 2 (2015), 48–59.

[25] Riko Jacob, Dirk Koschützki, KatharinaAnna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. 2005. Algorithms for centrality indices. In *Network Analysis*, Ulrik Brandes and Thomas Erlebach (Eds.), Lecture Notes in Computer Science, vol. 3418. Springer, Berlin, 62–82.

[26] Shiyu Ji and Zenghui Yan. 2016. Refining approximating betweenness centrality based on samplings. *a*rXiv:1608.04472 (2016).

[27] Steven G. Johnson. 2014. The NLopt Nonlinear-Optimization Package. (2014). Retrieved https://nlopt.readthedocs.io/en/latest/.

[28] Miray Kas, Matthew Wachs, Kathleen M. Carley, and L. Richard Carley. 2013. Incremental algorithm for updating betweenness centrality in dynamically growing networks. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'13)*. IEEE/ACM, 33–40.

[29] Vladimir Koltchinskii. 2001. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory* 47, 5 (Jul. 2001), 1902–1914.

[30] Vladimir Koltchinskii, C. T. Abdallah, Marco Ariola, Peter Dorato, and Dmitry Panchenko. 2000. Improved sample complexity estimates for statistical learning control of uncertain systems. *IEEE Transactions on Automatic Control* 45, 12 (Dec. 2000), 2383–2388. DOI: http://dx.doi.org/10.1109/9.895579

[31] Nicolas Kourtellis, Tharaka Alahakoon, Ramanuja Simha, Adriana Iamnitchi, and Rahul Tripathi. 2012. Identifying high betweenness centrality nodes in large social networks. *Social Network Analysis and Mining* 3, 4 (2012), 899–914.

[32] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. 2015. Scalable online betweenness centrality in evolving graphs. *IEEE Transactions on Knowledge and Data Engineering* 27, 9 (2015), 2494–2506.

[33] Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. 2012. QUBE: A quick algorithm for updating betweenness centrality. In *Proceedings of the 21st International Conference on World Wide Web (WWW'12)*. IW3C2, 351–360.

[34] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007), Article 2.

[35] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. (June 2014) Retrieved http://snap.stanford.edu/data.

[36] Yi Li, Philip M. Long, and Aravind Srinivasan. 2001. Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences* 62, 3 (2001), 516–527.

[37] Maarten Löffler and Jeff M. Phillips. 2009. Shape fitting on point sets with probability distributions. In *Proceedings of Algorithms - ESA 2009*, Amos Fiat and Peter Sanders (Eds.), Lecture Notes in Computer Science, vol. 5757. Springer, Berlin, 313–324. DOI: http://dx.doi.org/10.1007/978-3-642-04128-0_29

[38] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. 2014. Betweenness centrality – Incremental and faster. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science (MFCS'14)*. 577–588.

[39] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. 2014. Decremental all-pairs ALL shortest paths and betweenness centrality. In *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC'14)*. 766–778.

[40] Mark E. J. Newman. 2010. *Networks – An Introduction*. Oxford University Press.

[41] Luca Oneto, Alessandro Ghio, Davide Anguita, and Sandro Ridella. 2013. An improved analysis of the Rademacher data-dependent bound using its self bounding property. *Neural Networks* 44 (2013), 107–111.

[42] Luca Oneto, Alessandro Ghio, Sandro Ridella, and Davide Anguita. 2016. Global Rademacher complexity bounds: From slow to fast convergence rates. *Neural Processing Letters* 43, 2 (2016), 567–602.

[43] David Pollard. 1984. *Convergence of Stochastic Processes*. Springer-Verlag.

[44] Matteo Pontecorvi and Vijaya Ramachandran. 2015. Fully dynamic betweenness centrality. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC'15)*. 331–342.

[45] Foster Provost, David Jensen, and Tim Oates. 1999. Efficient progressive sampling. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery Data Mining (KDD'99)*. ACM, New York, NY, 23–32. DOI: http://dx.doi.org/10.1145/312129.312188

[46] Matteo Riondato and Evgenios M. Kornaropoulos. 2014. Fast approximation of betweenness centrality through sampling. In *Proceedings of WSDM*, Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler (Eds.). ACM, 413–422.

[47] Matteo Riondato and Evgenios M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475.

[48] Matteo Riondato and Eli Upfal. 2015. Mining frequent itemsets through progressive sampling with Rademacher averages. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, 1005–1014.

[49] Matteo Riondato and Eli Upfal. 2016. ABRA: Approximating betweenness centrality in static and dynamic graphs with Rademacher averages. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 1145–1154.

[50] Ahmet Erdem Sarıyüce, Erik Saule, Kamer Kaya, and Ümit V. Çatalyürek. 2013. Shattering and compressing networks for betweenness centrality. In *Proceedings of SIAM International Conference on Data Mining (SDM'13)*. SIAM, 686–694.

[51] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
[52] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. NetworKit: A tool suite for large-scale complex network analysis. *Network Science* 4, 4 (2016), 508–530.
[53] Vladimir N. Vapnik. 1999. *The Nature of Statistical Learning Theory*. Springer-Verlag.

Received July 2017; revised April 2018; accepted April 2018

KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation

MICHELE BORASSI, IMT School for Advanced Studies Lucca
EMANUELE NATALE, Max-Planck-Institut für Informatik

We present KADABRA, a new algorithm to approximate betweenness centrality in directed and undirected graphs, which significantly outperforms all previous approaches on real-world complex networks. The efficiency of the new algorithm relies on two new theoretical contributions, of independent interest.

The first contribution focuses on sampling shortest paths, a subroutine used by most algorithms that approximate betweenness centrality. We show that, on realistic random graph models, we can perform this task in time $|E|^{\frac{1}{2}+o(1)}$ with high probability, obtaining a significant speedup with respect to the $\Theta(|E|)$ worst-case performance. We experimentally show that this new technique achieves similar speedups on real-world complex networks, as well.

The second contribution is a new rigorous application of the adaptive sampling technique. This approach decreases the total number of shortest paths that need to be sampled to compute all betweenness centralities with a given absolute error, and it also handles more general problems, such as computing the k most central nodes. Furthermore, our analysis is general, and it might be extended to other settings.

CCS Concepts: • **Theory of computation** → *Graph algorithms analysis*;

Additional Key Words and Phrases: Betweenness centrality, shortest path algorithm, graph mining, sampling, network analysis

ACM Reference format:
Michele Borassi and Emanuele Natale. 2019. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. *J. Exp. Algorithmics* 24, 1, Article 1.2 (February 2019), 35 pages.
https://doi.org/10.1145/3284359

1 INTRODUCTION

In this work, we focus on estimating the *betweenness centrality*, which is one of the most famous measures of *centrality* for nodes and edges of real-world complex networks [24, 36]. The rigorous definition of betweenness centrality has its roots in sociology, dating back to the 1970s, when Freeman formalized the informal concept discussed in the previous decades in different scientific communities [6, 17, 22, 44, 45], although the definition already appeared in [3]. Since then, this notion has been very successful in network science [28, 36, 37, 51].

This work was done while the authors were visiting the Simons Institute for the Theory of Computing.
Authors' addresses: M. Borassi, IMT School for Advanced Studies Lucca, Piazza S. Francesco 19 - 55100 Lucca (LU) - Italy; email: michele.borassi@gmail.com; E. Natale, COATI Team, I3S, 2004 route des Lucioles - B.P. 93 - F-06902 Sophia Antipolis Cedex - France; email: emanuele.natale@inria.fr.

1084-6654/2019/02-ART1.2 $15.00
https://doi.org/10.1145/3284359

A probabilistic way to define the betweenness centrality[1] $\mathrm{bc}(v)$ of a node v in a graph $G = (V, E)$ is the following. We choose two nodes s and t, and we go from s to t through a shortest path π; if the choices of s, t, and π are made uniformly at random, the betweenness centrality of a node v is the probability that we pass through v.

In a seminal paper [18], Brandes showed that it is possible to exactly compute the betweenness centrality of all the nodes in a graph in time $O(mn)$, where n is the number of nodes and m is the number of edges. A corresponding lower bound was proved in [15]: if we are able to compute the betweenness centrality of a single node in time $O(mn^{1-\epsilon})$ for some $\epsilon > 0$, then the Strong Exponential Time Hypothesis [29] is false.

This result further motivates the rich line of research on computing approximations of betweenness centrality, with the goal of trading precision with efficiency. The main idea is to define a probability distribution over the set of all paths, by choosing two uniformly random nodes s, t, and then a uniformly distributed st-path $\boldsymbol{\pi}$, so that $\Pr(v \in \boldsymbol{\pi}) = \mathrm{bc}(v)$. As a consequence, we can approximate $\mathrm{bc}(v)$ by sampling paths $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\tau$ according to this distribution, and estimating $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau} \sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$ (and $v \neq s, t$), 0 otherwise.

The tricky part of this approach is to provide probabilistic guarantees on the quality of this approximation: the goal is to obtain a $1 - \delta$ confidence interval $\boldsymbol{I}(v) = [\tilde{\boldsymbol{b}}(v) - \lambda_L, \tilde{\boldsymbol{b}}(v) + \lambda_U]$ for $\mathrm{bc}(v)$, which means that $\Pr(\forall v \in V, \mathrm{bc}(v) \in \boldsymbol{I}(v)) \geq 1 - \delta$. Thus, the research for approximating betweenness centrality has been focusing on obtaining, as fast as possible, the smallest possible $\boldsymbol{I}$.

Our Contribution. In this work, we propose a new and faster algorithm to approximate betweenness centrality in directed and undirected graphs, named KADABRA. In the standard task of approximating betweenness centralities with absolute error at most λ, we show that, on average, the new algorithm is more than 100 times faster than the previous ones, on graphs with approximately 10,000 nodes. Moreover, differently from previous approaches, our algorithm can perform more general tasks, since it does not need all confidence intervals to be equal. As an example, we consider the computation of the k most central nodes: all previous approaches compute all centralities with an error λ, and use this approximation to obtain the ranking. Conversely, our approach allows us to use small confidence intervals only when they are needed, and allows bigger confidence intervals for nodes whose centrality values are "well separated.'. This way, we can compute for the first time an approximation of the k most central nodes in networks with millions of nodes and hundreds of millions of edges, like the Wikipedia citation network and the IMDB actor collaboration network.

Our results rely on two main theoretical contributions, which are interesting in their own right, since their generality naturally extends to other applications.

Balanced Bidirectional Breadth-First Search. By leveraging on recent advanced results, we prove that, on many realistic random models of real-world complex networks, it is possible to sample a random path between two nodes s and t in time $m^{\frac{1}{2}+o(1)}$ if the degree distribution has finite second moment, or $m^{\frac{4-\beta}{2}+o(1)}$ if the degree distribution is power law with exponent $2 < \beta < 3$. The models considered are the Configuration Model [12], and all Rank-1 Inhomogeneous Random Graph models [48, Chapter 3], such as the Chung-Lu model [35], the Norros-Reittu model [38], and the Generalized Random Graph [48, Chapter 3]. Our proof techniques have the merit of adopting a unified approach that simultaneously works in all models considered. These models well represent metric properties of real-world networks [16]: indeed, our results are confirmed by practical experiments.

[1] As explained in Section 2, to simplify notation we consider the *normalized* betweenness centrality.

The algorithm used is simply a balanced bidirectional BFS (bb-BFS): we perform a BFS from each of the two endpoints s and t, in such a way that the two BFSs are likely to explore about the same number of edges, and we stop as soon as the two BFSs "touch each other." Rather surprisingly, this technique was never implemented to approximate betweenness centrality, and it is rarely used in the experimental algorithm community. Our theoretical analysis provides a clear explanation of the reason why this technique improves over the standard BFS: this means that many state-of-the-art algorithms for real-world complex networks can be improved by the bb-BFS.

Adaptive Sampling Made Rigorous. To speed up the estimation of the betweenness centrality, previous works make use of the technique of adaptive sampling, which consists in testing during the execution of the algorithm whether some condition on the sample obtained so far has been met, and terminating the execution of the algorithm as soon as this happens. However, this technique introduces a subtle stochastic dependence between the time in which the algorithm terminates and the correctness of the given output, which previous papers claiming a formal analysis of the technique did not realize (see Section 3 for details). With an argument based on martingale theory, we provide a general analysis of this useful technique. Through this result, we do not only improve previous estimators, but we also make it possible to define more general stopping conditions, that can be decided "on the fly": this way, with little modifications, we can adapt our algorithm to perform more general tasks than previous ones.

To better illustrate the power of our techniques, we focus on the unweighted, static graphs, and to the centrality of nodes. However, our algorithm can be easily adapted to compute the centrality of edges, to handle weighted graphs and, since its core part consists merely in sampling paths, we conjecture that it may be coupled with the existing techniques in [9, 10] to handle dynamic graphs.

Related Work

Computing Betweenness Centrality. With the recent event of big data, the major shortcoming of betweenness centrality has been the lack of efficient methods to compute it [18]. In the worst case, the best exact algorithm to compute the centrality of all the nodes is due to Brandes [18], and its time complexity is $O(mn)$: the basic idea of the algorithm is to define the dependency $\delta_s(v) = \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where $\sigma_{st}(v)$ is the number of st-shortest paths passing through v, and σ_{st} is the total number of st-shortest paths. The idea of the Brandes algorithm is that the dependency can be computed in time $O(m)$ for each $v \in V$. In [15], it is also shown that the Brandes algorithm is almost optimal, provided the Strong Exponential Time Hypothesis [29] holds, which is widely believed to be the case. The latter result further motivates the already rich line of research on approaches that overcome this barrier. A first possibility is to use heuristics, which do not provide analytical guarantees on their performance [26, 47, 49]. Another line of research has defined variants of betweenness centrality, which might be easier to compute [19, 23, 39]. Finally, a third line of research has investigated approximation algorithms, which trade accuracy for speed [20, 21, 28, 30, 32]. Our work follows the latter approach. The first approximation algorithm proposed in the literature [30] adapts Eppstein and Wang's approach for computing closeness centrality [25], using Hoeffding's inequality and the union bound technique. This way, it is possible to obtain an estimate of the betweenness centrality of every node that is correct up to an additive error λ with probability $1 - \delta$, by sampling $O(\frac{D^2}{\lambda^2} \log \frac{n}{\delta})$ nodes, where D is the diameter of the graph. In [28], it is shown that this can lead to an overestimation. Riondato and Kornaropoulos improve this sampling-based approach by sampling single shortest paths instead of the whole dependency of a node [42], introducing the use of the VC-dimension. As a result, the number of samples is decreased to $\frac{c}{\lambda^2}(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log(\frac{1}{\delta}))$, where VD is the vertex diameter, that is, the minimum number of nodes in a shortest path in G (it can be different from $D + 1$ if the graph is weighted).

This use of the VC-dimension is further developed and generalized in [43]. Finally, many of these results were adapted to handle dynamic networks [9, 43].

Approximating the Top-k Betweenness Centrality Set. Let us order the nodes $v_1, \dots, v_n$ such that $\text{bc}(v_1) \geq \dots \geq \text{bc}(v_n)$ and define $TOP(k) = \{(v_i, \text{bc}(v_i)) : i \leq k\}$. In [42] and [43], the authors provide an algorithm that, for any given δ, ϵ, with probability $1 - \delta$ outputs a set $\widetilde{TOP}(k) = \{(v_i, \tilde{\boldsymbol{b}}(v_i))\}$ such that (i) if $v \in TOP(k)$, then $v \in \widetilde{TOP}(k)$ and $|\text{bc}(v) - \tilde{\boldsymbol{b}}(v)| \leq \epsilon\,\text{bc}(v)$; (ii) if $v \in \widetilde{TOP}(k)$ but $v \notin TOP(k)$, then $\tilde{\boldsymbol{b}}(v) \leq (\mathbf{b}_k - \epsilon)(1 + \epsilon)$ where $\mathbf{b}_k$ is the k-th largest betweenness given by a preliminary phase of the algorithm.

Adaptive Sampling. In [4, 43], the number of samples required is substantially reduced using the adaptive sampling technique introduced by Lipton and Naughton in [33, 34]. Let us clarify that, by adaptive sampling, we mean that the termination of the sampling process depends on the sample observed so far (in other cases, the same expression refers to the fact that the distribution of the new samples is a function of the previous ones [2], while the sample size is fixed in advance). Except for [40], previous approaches tacitly assume that there is little dependency between the stopping time and the correctness of the output: indeed, they prove that, for each *fixed* τ, the probability that the estimate is wrong at time τ is below δ. However, the stopping time $\boldsymbol{\tau}$ is a random variable, and in principle there might be dependency between the event $\boldsymbol{\tau} = \tau$ and the event that the estimate is correct at time τ. As for [40], they consider a specific stopping condition and their proof technique does not seem to extend to other settings. For a more thorough discussion of this issue, we defer the reader to Section 3.

Bidirectional BFS. The possibility of speeding up a breadth-first search for the shortest-path problem by performing, at the same time, a BFS from the final end-point, has been considered since the 1970s [41], and has been used for the implementation of the plateau heuristic (see, e.g., [1] and [5]). In the context of random graphs, similar ideas have been used to derive upper bounds on the diameter [14]. However, while for computing shortest-paths in weighted directed graphs it is customary to employ the bidirectional version of Dijkstra's algorithm, on undirected unweighted graphs the bidirectional BFS has often not been considered an important heuristic improvement [31], perhaps partly because of the lack of theoretical results dealing with its efficiency. On the other hand, in [42] (and in some public talks by M. Riondato), the bidirectional BFS was proposed as a possible way to improve the performance of betweenness centrality approximation algorithms.

Structure of the Article. In Section 2, we describe our algorithm, and in Section 3 we discuss the main difficulty of the adaptive sampling, and the reasons why our techniques are not affected. In Section 4, we define the balanced bidirectional BFS, and we sketch the proof of its efficiency on random graphs. In Section 5, we show that our algorithm can be adapted to compute the k most central nodes. In Sections 6–8, we provide full proofs of our results. Finally, in Section 9 we experimentally show the effectiveness of our new algorithm.

2 ALGORITHM OVERVIEW

To simplify notation, we always consider the *normalized* betweenness centrality of a node v, which is defined by[2]

$$\text{bc}(v) = \frac{1}{n(n-1)} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

[2]Note that sometimes the chosen normalizing factor for $\text{bc}(v)$ is $(n-1)(n-2)$. We follow the convention of [43] (where the sum is over all $s6$, $t \in V$ with $\sigma_{st}(v) = 0$ whenever $s = v$ or $t = v$).

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through v. Furthermore, to simplify the exposition, we use bold symbols to denote random variables, and light symbols to denote deterministic quantities. On the same line of previous works, our algorithm samples random paths $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\tau$, where $\boldsymbol{\pi}_i$ is chosen by selecting uniformly at random two nodes s, t, and then selecting uniformly at random one of the shortest paths from s to t. Then, it estimates $\mathrm{bc}(v)$ with $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau}\sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$, 0 otherwise. By definition of $\boldsymbol{\pi}_i$, $\mathbb{E}[\tilde{\boldsymbol{b}}(v)] = \mathrm{bc}(v)$.

The tricky part is to bound the distance between $\tilde{\boldsymbol{b}}(v)$ and its expected value. In this regard, recall Hoeffding's inequality.

Lemma 1 (Hoeffding's inequality). *Let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$ be independent random variables such that $a_i < \boldsymbol{X}_i < b_i$, and let $\boldsymbol{X} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X}_i$. Then,*

$$\Pr(|\boldsymbol{X} - \mathbb{E}[X]| \geq \lambda) \leq \exp\left\{-\frac{2k^2\lambda^2}{\sum_{i=1}^{k}(b_i - a_i)}\right\}.$$

With a straightforward application of Hoeffding's inequality, it is possible to prove that

$$\Pr(|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \geq \lambda) \leq 2e^{-2\tau\lambda^2}. \tag{1}$$

A direct application of Equation (1) considers a union bound on all possible nodes v, obtaining $\Pr(\forall v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \geq \lambda) \leq 2ne^{-2\tau\lambda^2}$. This means that the algorithm can safely stop as soon as $2ne^{-2\tau\lambda^2} \leq \delta$, that is, after $\tau = \frac{1}{2\lambda^2}\log(\frac{2n}{\delta})$ steps.

In order to improve this idea, we can start from the Chernoff bound instead of the Hoeffding inequality.

Lemma 2 (Chernoff bound ([35])). *Let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$ be independent random variables such that $\boldsymbol{X}_i \leq M$ for each $1 \leq i \leq k$, and let $\boldsymbol{X} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{X}_i$. Then,*

$$\Pr(\boldsymbol{X} \geq \mathbb{E}[\boldsymbol{X}] + \lambda) \leq \exp\left\{-\frac{k\lambda^2}{2(\frac{1}{k}\sum_{i=1}^{k}\mathbb{E}[\boldsymbol{X}_i^2] + M\lambda/3)}\right\}.$$

From Lemma 2 we thus obtain that

$$\Pr(|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \geq \lambda) \leq 2\exp\left(-\frac{\tau\lambda^2}{2(\mathrm{bc}(v) + \lambda/3)}\right).$$

If we assume the error λ to be small, this inequality is stronger than the previous one for all values of $\mathrm{bc}(v) < \frac{1}{4} - \frac{\lambda}{3} < \frac{1}{4}$ (a condition which holds for almost all nodes, in almost all graphs considered). However, in order to apply this inequality, we have to deal with the fact that we do not know $\mathrm{bc}(v)$ in advance, and hence we do not know when to stop. Intuitively, to solve this problem, we make a "change of variable," and we rewrite the previous inequality as

$$\Pr(\mathrm{bc}(v) \leq \tilde{\boldsymbol{b}}(v) - f) \leq \delta_L^{(v)} \quad \text{and} \quad \Pr(\mathrm{bc}(v) \geq \tilde{\boldsymbol{b}}(v) + g) \leq \delta_U^{(v)}, \tag{2}$$

for some functions $f = f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau), g = g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$. Our algorithm fixes at the beginning the values $\delta_L^{(v)}, \delta_U^{(v)}$ for each node v, and, at each step, it tests if $f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau)$ and $g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$ are small enough. If this condition is satisfied, the algorithm stops. Note that this approach lets us define very general stopping conditions, that might depend on the centralities computed until now, on the single nodes, and so on.

Remark. Instead of fixing the values $\delta_L^{(v)}, \delta_U^{(v)}$ at the beginning, one might want to decide them during the algorithm, depending on the outcome. However, this is not formally correct, because of

dependency issues (e.g., (2) does not even make sense, if $\delta_L^{(v)}, \delta_U^{(v)}$ are random). Finding a way to overcome this issue is left as a challenging open problem (more details are provided in Section 3).

In order to implement this idea, we still need to solve an issue: Equation (2) holds for each *fixed* time τ, but the stopping time of our algorithm is a random variable $\boldsymbol{\tau}$, and there might be dependency between the value of $\boldsymbol{\tau}$ and the probability in Equation (2). To this purpose, we use a stronger inequality, that holds even if $\boldsymbol{\tau}$ is a random variable.

Theorem 3 (McDiarmid'98 ([35])). *Let X be a martingale associated with a filter $\mathcal{F}$, satisfying*

- $\mathrm{Var}(X_i | \mathcal{F}_i) \leq \sigma_i$ *for* $1 \leq i \leq \ell$,
- $|X_i - X_{i-1}| \leq M$, *for* $1 \leq i \leq \ell$.

Then, we have

$$\Pr(X - \mathbb{E}(X) \geq \lambda) \leq \exp\left(-\frac{\lambda^2}{2(\sum_{i=1}^{\ell} \sigma_i^2 + M\lambda/3)}\right).$$

However, to use Equation (2), we need to assume that $\boldsymbol{\tau} < \omega$ for some deterministic ω: in our algorithm, we choose $\omega = \frac{c}{\lambda^2}(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log(\frac{2}{\delta}))$, because, by the results in [42], after ω samples, the maximum error is at most λ, with probability $1 - \frac{\delta}{2}$. Furthermore, also f and g should be modified, since they now depend on the value of ω. The pseudocode of the algorithm obtained is available in Algorithm 1 (as was done in previous approaches, we can easily parallelize the while loop in line 5).

ALGORITHM 1: Our algorithm for approximating betweenness centrality.

Input: a graph $G = (V, E)$
Output: for each $v \in V$, an approximation $\tilde{\boldsymbol{b}}(v)$ of $\mathrm{bc}(v)$ such that
$\Pr(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda) \geq 1 - \delta$

1 $\omega \leftarrow \frac{c}{\lambda^2}\left(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log\left(\frac{2}{\delta}\right)\right)$;
2 $(\delta_L^{(v)}, \delta_U^{(v)}) \leftarrow$ `computeDelta()`;
3 $\tau \leftarrow 0$;
4 **foreach** $v \in V$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow 0$;
5 **while** $\tau < \omega$ *and not* `haveToStop` $(\tilde{\boldsymbol{b}}, \delta_L, \delta_U, \omega, \tau)$ **do**
6 $\quad \boldsymbol{\pi} =$ `samplePath()`;
7 $\quad$ **foreach** $v \in \boldsymbol{\pi}$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
8 $\quad \tau \leftarrow \tau + 1$;
9 **end**
10 **foreach** $v \in V$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\tau$;
11 **return** $\tilde{\boldsymbol{b}}$

The correctness of the algorithm follows from the following theorem, which is the base of our adaptive sampling, and which we prove in Section 6 (where we also define the functions f and g).

Theorem 4. *Let $\tilde{\boldsymbol{b}}(v)$ be the output of Algorithm 1, and let $\boldsymbol{\tau}$ be the number of samples at the end of the algorithm. Then, with probability $1 - \delta$, the following conditions hold:*

- *if $\boldsymbol{\tau} = \omega$, $|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| < \lambda$ for all v;*
- *if $\boldsymbol{\tau} < \omega$, $-f(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \omega) \leq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \leq g(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \omega)$ for all v.*

Remark. This theorem says that, at the beginning of the algorithm, we know that, with probability $1 - \delta$, one of the two conditions will hold when the algorithm stops, independently on the final value of $\boldsymbol{\tau}$. This is essential to avoid the stochastic dependence that we discuss in Section 3.

In order to apply this theorem, we choose λ such that our goal is reached if all centralities are known with error at most λ. Then, we choose the function `haveToStop`in a way that our goal is reached if the stopping condition is satisfied. This way, our algorithm is correct, both if $\boldsymbol{\tau} = \omega$ and if $\boldsymbol{\tau} < \omega$. For example, if we want to compute all centralities with bounded absolute error, we simply choose λ as the bound we want to achieve, and we plug the stopping condition $f, g \leq \lambda$ in the function `haveToStop`. Instead, if we want to compute an approximation of the k most central nodes, we need a different definition of f and g, which is provided in Section 5.

To complete the description of this algorithm, we need to specify the following functions.

`computeDelta` The algorithm works for any choice of the $\delta_L^{(v)}, \delta_U^{(v)}$, but a good choice yields better running times. We propose a heuristic way to choose them in Section 7.

`samplePath` In order to sample a path between two random nodes s and t, we use a balanced bidirectional BFS, which is defined in Section 8.

3 ADAPTIVE SAMPLING

In this section, we highlight the main technical difficulty in the formalization of adaptive sampling, which previous works claiming analogous results did not address. Furthermore, we sketch the way we overcome this difficulty: our argument is quite general, and it could be easily adapted to formalize these claims.

As already said, the problem is the stochastic dependence between the time $\boldsymbol{\tau}$ in which the algorithm terminates and the event $\boldsymbol{A}_\tau$ = "at time τ, the estimate is within the required distance from the true value", since both $\boldsymbol{\tau}$ and $\boldsymbol{A}_\tau$ are functions of the same random sample. Since it is typically possible to prove that $\Pr(\neg \boldsymbol{A}_\tau) \leq \delta$ for every fixed τ, one may be tempted to argue that also $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) \leq \delta$, by applying these inequalities at time $\boldsymbol{\tau}$. However, this is not correct: indeed, if we have no assumptions on $\boldsymbol{\tau}$, $\boldsymbol{\tau}$ could even be defined as the smallest τ such that $\boldsymbol{A}_\tau$ does not hold!

More formally, if we want to link $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$ to $\Pr(\neg \boldsymbol{A}_\tau)$, we have to use the law of total probability, that says that

$$\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) = \sum_{\tau=1}^{\infty} \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} = \tau) \Pr(\boldsymbol{\tau} = \tau) \tag{3}$$

$$= \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} \leq \tau) \Pr(\boldsymbol{\tau} \leq \tau) + \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} \geq \tau) \Pr(\boldsymbol{\tau} \geq \tau). \tag{4}$$

Then, if we want to bound $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$, we need to assume that

$$\Pr(\neg A_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} = \tau) \leq \Pr(\neg A_\tau) \quad \text{or that} \quad \Pr(\neg A_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} \geq \tau) \leq \Pr(\neg A_\tau), \tag{5}$$

which would allow one to bound Equation (3) or (4) from above. The equations in (5) are implicitly assumed to be true in previous works adopting adaptive sampling techniques. Unfortunately, because of the stochastic dependence, it is quite difficult to prove such inequalities. Some approaches managed to overcome these difficulties by bounding Equation (3) with a union bound over a sequence of τ's of exponentially increasing size [40]. The latter approach suffices in most cases: aside from the factor which one loses by employing exponentially growing upper bounds, the union bound costs a multiplicative factor of order $\log \log n$ on the functions f and g in Equation (2). In this work, we opt for the more general approach of adaptive sampling, which allows a more parsimonious estimate of Equation (3), at the price of a more sophisticated analysis.

In some sense, our proofs consist in circumventing the problem of dealing with inequalities such as Equation (5): in the proof of Theorem 4, we fix a deterministic time ω, we impose that $\boldsymbol{\tau} \leq \omega$, and we apply the inequalities with $\tau = \omega$. Then, using the martingale theory, we convert results that hold at time ω to results that hold at the stopping time $\boldsymbol{\tau}$ (see Section 6).

4 BALANCED BIDIRECTIONAL BFS

A major improvement of our algorithm, with respect to previous counterparts, is that we sample shortest paths through a balanced bidirectional BFS, instead of a standard BFS. In this section, we describe this technique, and we bound its running time on realistic models of random graphs, with high probability. The idea behind this technique is very simple: if we need to sample a uniformly random shortest path from s to t, instead of performing a full BFS from s until we reach t, we perform at the same time a BFS from s and a BFS from t, until the two BFSs touch each other (if the graph is directed, we perform a "forward" BFS from s and a "backward" BFS from t).

More formally, assume that we have visited up to level l_s from s and to level l_t from t, let $\Gamma^{l_s}(s)$ be the set of nodes at distance l_s from s, and similarly let $\Gamma^{l_t}(t)$ be the set of nodes at distance l_t from t. If $\sum_{v \in \Gamma^{l_s}(s)} \deg(v) \leq \sum_{v \in \Gamma^{l_t}(t)} \deg(v)$, we process all nodes in $\Gamma^{l_s}(s)$, otherwise we process all nodes in $\Gamma^{l_t}(t)$ (since the time needed to process level l_s is proportional to $\sum_{v \in \Gamma^{l_s}(s)} \deg(v)$, this choice minimizes the time needed to visit the next level). Assume that we are processing the node $v \in \Gamma^{l_s}(s)$ (the other case is analogous). For each neighbor w of v we do the following:

- if w was never visited, we add w to $\Gamma^{l_s+1}(s)$;
- if w was already visited in the BFS from s, we do not do anything;
- if w was visited in the BFS from t, we add the edge (v, w) to the set Π of candidate edges in the shortest path.

After we have processed a level, we stop if $\Gamma^{l_s}(s)$ or $\Gamma^{l_t}(t)$ is empty (in this case, s and t are not connected), or if Π is not empty. In the latter case, we select an edge from Π, so that the probability of choosing the edge (v, w) is proportional to $\sigma_{sv}\sigma_{wt}$ (we recall that σ_{xy} is the number of shortest paths from x to y, and it can be computed during the BFS as in [20]). Then, the path is selected by considering the concatenation of a random path from s to v, the edge (v, w), and a random path from w to t (these random paths can be easily chosen by backtracking, as shown in [42]).

4.1 Analysis on Random Graphs

In order to show the effectiveness of the balanced bidirectional BFS, we bound its running time in several models of random graphs: the Configuration Model (CM, [12]), and Rank-1 Inhomogeneous Random Graph models (IRG, [48, Chapter 3]), such as the Chung-Lu model [35], the Norros-Reittu model [38], and the Generalized Random Graph [48, Chapter 3]. In these models, we fix the number n of nodes, and we give a weight ρ_u to each node. In the CM, we create edges by giving ρ_u half-edges to each node u, and pairing these half-edges uniformly at random; in IRG we connect each pair of nodes (u, v) independently with probability close to $\rho_u\rho_v / \sum_{w \in V} \rho_w$.

In this section, we only provide an informal sketch of the main proof ideas. The formal proof is given in Section 8, where we look at the random graph G as a sequence of graphs G_i whose degree distribution satisfies three general conditions, discussed in that section.

Theorem 5. *Let G be a graph generated through the aforementioned models. More precisely, let $\{G_i\}_i$ be a sequence of graphs whose number of nodes n_i tends to infinity, and whose degree distribution Λ_i satisfy the following three properties[3]:*

[3]See Section 8 for details regarding how the weights ρ_v of the aforementioned random graph models relate to these assumptions.

(1) there is a probability distribution Λ such that the Λ_i's tend to Λ in distribution;
(2) $M_1(\Lambda_i)$ tends to $M_1(\Lambda) < \infty$, where $M_1(\Lambda)$ is the first moment of Λ;
(3) one of the following two conditions hold:
  (a) $M_2(\Lambda_i)$ tends to $M_2(\Lambda) < \infty$, where $M_2(\Lambda)$ is the second moment of Λ;
  (b) Λ is a power-law distribution with $2 < \beta < 3$, and there is a global constant C such that, for each d, $\Pr(\Lambda_i \geq d) \leq \frac{C}{d^{\beta-1}}$.

Then, for each fixed $\epsilon > 0$, and for each pair of nodes s, t, with high probability (w.h.p.), the time needed to compute an st-shortest path through a bidirectional BFS is $O(n^{\frac{1}{2}+\epsilon})$ if the distribution Λ has finite second moment, $O(n^{\frac{4-\beta}{2}} + \epsilon)$ if Λ is a power-law distribution with $2 < \beta < 3$.

Sketch of Proof of Theorem 5. The idea of the proof is that the time needed by a bidirectional BFS is proportional to the number of visited edges, which is close to the sum of the degrees of the visited nodes, which are very close to their weights. Hence, we have to analyze the weights of the visited edges: for this reason, if V' is a subset of V, we define the volume of V' as $\rho_{V'} = \sum_{v \in V'} \rho_v$.

Our visit proceeds by "levels" in the BFS trees from s and t: if we never process a level with total weight at least $n^{\frac{1}{2}+\epsilon}$, since the diameter is $O(\log n)$, the volume of the set of processed vertices is $O(n^{\frac{1}{2}+\epsilon} \log n)$, and the number of visited edges cannot be much bigger (e.g., this happens if s and t are not connected). Otherwise, assume that, at some point, we process a level l_s in the BFS from s with total weight $n^{\frac{1}{2}+\epsilon}$: then, the corresponding level l_t in the BFS from t has also weight $n^{\frac{1}{2}+\epsilon}$ (otherwise, we would have expanded from t, because weights and degrees are strongly correlated). We use the "birthday paradox": levels $l_s + 1$ in the BFS from s, and level $l_t + 1$ in the BFS from t are random sets of nodes with size close to $n^{\frac{1}{2}+\epsilon}$, and hence there is a node that is common to both, w.h.p. This means that the time needed by the bidirectional BFS is proportional to the volume of all levels in the BFS tree from s, until l_s, plus the volume of all levels in the BFS tree from t, until l_t (note that we do not expand levels $l_s + 1$ and $l_t + 1$). All levels except the last have volume at most $n^{\frac{1}{2}+\epsilon}$, and there are $O(\log n)$ such levels because the diameter is $O(\log n)$: it only remains to estimate the volume of the last level.

By definition of the models, the probability that a node v with weight ρ_v belongs to the last level is about $\frac{\rho_v \rho_{\Gamma^{l_s-1}(s)}}{M} \leq \rho_v n^{-\frac{1}{2}+\epsilon}$: hence, the expected volume of $\Gamma^{l_s}(s)$ is at most $\sum_{v \in V} \rho_v \Pr(v \in \Gamma^{l_s-1}(s)) \leq \sum_{v \in V} \rho_v^2 n^{-\frac{1}{2}+\epsilon}$. Through standard concentration inequalities, we prove that this random variable is concentrated: hence, we only need to compute this expected value. If the degree distribution has finite second moment, then $\sum_{v \in V} \rho_v^2 = O(n)$, concluding the proof. If the degree distribution is power law with $2 < \beta < 3$, then we have to consider separately nodes v such that $\rho_v < n^{\frac{1}{2}}$ and such that $\rho_v > n^{\frac{1}{2}}$. In the first case, $\sum_{\rho_v < n^{\frac{1}{2}}} \rho_v^2 \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^2\Lambda(d) \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^{2-\beta} \approx n^{1+\frac{3-\beta}{2}}$. In the second case, we prove that the volume of the set of nodes with weight bigger than $n^{\frac{1}{2}}$ is at most $n^{\frac{4-\beta}{2}}$. Hence, the total volume of $\Gamma^{l_s}(s)$ is at most $n^{-\frac{1}{2}+\epsilon} n^{1+\frac{3-\beta}{2}} + n^{\frac{4-\beta}{2}} \approx n^{\frac{4-\beta}{2}}$. □

5 COMPUTING THE k MOST CENTRAL NODES

Differently from previous works, our algorithm is more flexible, making it possible to compute the betweenness centrality of different nodes with different precision. This feature can be exploited if we only want to rank the nodes: for instance, if v is much more central than all the other nodes, we do not need a very precise estimation on the centrality of v to say that it is the top node. Following this idea, in this section we adapt our approach to the approximation of the ranking of the k most central nodes: as far as we know, this is the first approach which computes the ranking without computing a λ-approximation of all betweenness centralities, allowing significant

speedups. Clearly, we cannot expect our ranking to be always correct, otherwise the algorithm does not terminate if two of the k most central nodes have the same centrality. For this reason, the user fixes a parameter λ, and, for each node v, the algorithm does one of the following:

— it provides the exact position of v in the ranking;
— it guarantees that v is not in the top-k;
— it provides a value $\tilde{b}(v)$ such that $|\mathrm{bc}(v) - \tilde{b}(v)| \leq \lambda$.

In other words, similarly to what is done in [42], the algorithm provides a set of $k' \geq k$ nodes containing the top-k nodes, and for each pair of nodes v, w in this subset, either we can rank correctly v and w, or v and w are almost even, that is, $|\mathrm{bc}(v) - \mathrm{bc}(w)| \leq 2\lambda$. In order to obtain this result, we plug into Algorithm 1 the aforementioned conditions in the function `haveToStop` (see Algorithm 2).

ALGORITHM 2: The function `haveToStop` to compute the top-k nodes.

Input: for each node v, the values of $\tilde{b}(v), \delta_L^{(v)}, \delta_U^{(v)}$, and the values of ω and τ
Output: True if the algorithm should stop, False otherwise

Sort nodes in decreasing order of $\tilde{b}(v)$, obtaining $v_1, \ldots, v_n$;
for $i \in [1, \ldots, k]$ **do**
  if $f(\tilde{b}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ *or* $g(\tilde{b}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ **then**
    if $\tilde{b}(v_{i-1}) - f(\tilde{b}(v_{i-1}), \delta_L^{(v_{i-1})}, \omega, \tau) < \tilde{b}(v_i) + g(\tilde{b}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ *or* $\tilde{b}(v_i) - f(\tilde{b}(v_i), \delta_L^{(v_i)}, \omega, \tau) < \tilde{b}(v_{i+1}) + g(\tilde{b}(v_{i+1}), \delta_U^{(v_{i+1})}, \omega, \tau)$ **then**
      return *False*;
    end
  end
end
for $i \in [k+1, \ldots, n]$ **do**
  if $f(\tilde{b}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ *or* $g(\tilde{b}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ **then**
    if $\tilde{b}(v_k) - f(\tilde{b}(v_k), \delta_L^{(v_k)}, \omega, \tau) < \tilde{b}(v_i) + g(\tilde{b}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ **then**
      return *False*;
    end
  end
end
return *True*;

Then, we have to adapt the function `computeDelta` to optimize the $\delta_L^{(v)}$'s and the $\delta_U^{(v)}$'s to the new stopping condition: in other words, we have to choose the values of $\lambda_L^{(v)}$ and $\lambda_U^{(v)}$ that should be plugged into the function `computeDelta` (we recall that the heuristic `computeDelta` chooses the $\delta_L^{(v)}$'s so that we can guarantee as fast as possible that $\tilde{b}(v) - \lambda_L^{(v)} \leq \mathrm{bc}\, v \leq \tilde{b}(v) + \lambda_U^{(v)}$). To this purpose, we estimate the betweenness of all nodes with few samples and we sort all nodes according to these approximate values $\tilde{b}(v)$, obtaining $v_1, \ldots, v_n$. The basic idea is that, for the first k nodes, we set $\lambda_U^{(v_i)} = \frac{\tilde{b}(v_{i-1}) - \tilde{b}(v_i)}{2}$, and $\lambda_L^{(v_i)} = \frac{\tilde{b}(v_i) - \tilde{b}(v_{i+1})}{2}$ (the goal is to find confidence intervals that separate the betweenness of v_i from the betweenness of v_{i+1} and v_{i-1}). For nodes that are not in the top-k, we choose $\lambda_L^{(v)} = 1$ and $\lambda_U^{(v)} = \tilde{b}(v_k) - \lambda_L^{(v_k)} - \tilde{b}(v_i)$ (the goal is to prove that v_i is not in the top-k). Finally, if $\tilde{b}(v_i) - \tilde{b}(v_{i+1})$ is small, we simply set $\lambda_L^{(v_i)} = \lambda_U^{(v_i)} = \lambda_L^{(v_{i+1})} = \lambda_U^{(v_{i+1})} = \lambda$, because we do not know if $\mathrm{bc}(v_{i+1}) > \mathrm{bc}(v_i)$, or vice versa.

6 PROOF OF THEOREM 4

In our algorithm, we sample $\boldsymbol{\tau}$ shortest paths $\boldsymbol{\pi}_i$, where $\boldsymbol{\tau}$ is a random variable such that $\boldsymbol{\tau} = \tau$ can be decided by looking at the first τ paths sampled (see Algorithm 1). Furthermore, thanks to Equation (3) in [42], we assume that $\boldsymbol{\tau} \leq \omega$ for some fixed $\omega \in \mathbb{R}^+$ such that, after ω steps,

$$\Pr(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda) \geq 1 - \frac{\delta}{2}.$$

When the algorithm stops, our estimate of the betweenness is $\tilde{\boldsymbol{b}}(v) := \frac{1}{\boldsymbol{\tau}} \sum_{i=1}^{\boldsymbol{\tau}} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v)$ is 1 if v belongs to $\boldsymbol{\pi}_i$, 0 otherwise.

To estimate the error, we prove the following theorem.

Theorem 6. *For each node v and for every fixed real numbers δ_L, δ_U, it holds that*

$$\Pr\left(\mathrm{bc}(v) \leq \boldsymbol{b}(v) - f(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau})\right) \leq \delta_L \quad \textit{and}$$
$$\Pr\left(\mathrm{bc}(v) \geq \tilde{\boldsymbol{b}}(v) + g(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau})\right) \leq \delta_U,$$

where

$$f(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau}) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_L} \left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_L}}} \right) \quad \textit{and} \tag{6}$$

$$g(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau}) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_U} \left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_U}}} \right). \tag{7}$$

Proof. Since this theorem deals with a single node v, let us simply write $\mathrm{bc} = \mathrm{bc}(v), \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{b}}(v), \boldsymbol{X}_i = \boldsymbol{X}_i(v)$. Let us consider $\boldsymbol{Y}^\tau = \sum_{i=1}^{\tau} (\boldsymbol{X}_i - \mathrm{bc})$ (we recall that $\boldsymbol{X}_i = 1$ if v is in the i-th path sampled, $\boldsymbol{X}_i = 0$ otherwise). Clearly, $\boldsymbol{Y}^\tau$ is a martingale, and $\boldsymbol{\tau}$ is a stopping time for $\boldsymbol{Y}^\tau$: this means that also $\boldsymbol{Z}^\tau = \boldsymbol{Y}^{\min(\boldsymbol{\tau}, \tau)}$ is a martingale.

Let us apply Theorem 3 to the martingales $\boldsymbol{Z}$ and $-\boldsymbol{Z}$: for each fixed $\lambda_L, \lambda_U > 0$ we have

$$\Pr\left(\boldsymbol{Z}^\omega \geq \lambda_L\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \geq \lambda_L\right) \leq \exp\left(-\frac{\lambda_L^2}{2(\omega\,\mathrm{bc} + \lambda_L/3)}\right) = \delta_L \quad \text{and} \tag{8}$$

$$\Pr\left(-\boldsymbol{Z}^\omega \geq \lambda_U\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \leq -\lambda_U\right) \leq \exp\left(-\frac{\lambda_U^2}{2(\omega\,\mathrm{bc} + \lambda_U/3)}\right) = \delta_U. \tag{9}$$

We now show how to prove Equation (6) from Equation (8). The way to derive Equation (7) from Equation (9) is analogous.

If we express λ_L as a function of δ_L we get

$$\lambda_L^2 = 2 \log \frac{1}{\delta_L} \left(\omega\,\mathrm{bc} + \frac{\lambda_L}{3}\right) \iff \lambda_L^2 - \frac{2}{3}\lambda_L \log \frac{1}{\delta_L} - 2\omega\,\mathrm{bc} \log \frac{1}{\delta_L} = 0,$$

which implies that

$$\lambda_L = \frac{1}{3} \log \frac{1}{\delta_L} \pm \sqrt{\frac{1}{9}\left(\log \frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc} \log \frac{1}{\delta_L}}.$$

Since Equation (8) holds for any positive value λ_L, it also holds for the value corresponding to the positive solution of this equation, that is,

$$\lambda_L = \frac{1}{3} \log \frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log \frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc} \log \frac{1}{\delta_L}}.$$

Plugging this value into Equation (8), we obtain

$$\Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \geq \frac{1}{3}\log\frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}\right) \leq \delta_L. \tag{10}$$

By assuming $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\boldsymbol{\tau}}\log(\frac{1}{\delta_L})$, the event in Equation (10) can be rewritten as

$$(\boldsymbol{\tau}\,\mathrm{bc})^2 - 2\,\mathrm{bc}\left(\boldsymbol{\tau}^2\tilde{\boldsymbol{b}} + \omega\log\frac{1}{\delta_L} - \frac{1}{3}\boldsymbol{\tau}\log\frac{1}{\delta_L}\right) - \frac{2}{3}\log\frac{1}{\delta_L}\boldsymbol{\tau}\tilde{\boldsymbol{b}} + \left(\boldsymbol{\tau}\tilde{\boldsymbol{b}}\right)^2 \geq 0.$$

By solving the previous quadratic equation w.r.t. bc we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}} + \frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 - \left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}\right)^2 + \frac{2}{3\boldsymbol{\tau}}\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}}\right),$$

where we only considered the solution which upper bounds bc, since we assumed $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\tau}\log(\frac{1}{\delta_L})$. After simplifying the terms under the square root in the previous expression, we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}\omega}{\boldsymbol{\tau}^2\log\frac{1}{\delta_L}}}\right),$$

which means that

$$\Pr(\mathrm{bc} \leq \tilde{\boldsymbol{b}} - f(\tilde{\boldsymbol{b}}, \delta_L, \omega, \boldsymbol{\tau})) \leq \delta_L,$$

concluding the proof. □

We now show how Theorem 6 implies Theorem 4. To simplify notation, we often omit the arguments of the functions f and g.

Proof of Theorem 4. Let $\boldsymbol{E}_1$ be the event

$$(\boldsymbol{\tau} = \omega \wedge \exists v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| > \lambda),$$

and let $\boldsymbol{E}_2$ be the event

$$(\boldsymbol{\tau} < \omega \wedge (\exists v \in V, -f \geq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \vee \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \geq g)).$$

Let us also denote $\tilde{\boldsymbol{b}}_\tau(v) = \frac{1}{\tau}\sum_{i=1}^{\tau}\boldsymbol{X}_i(v)$ (note that $\tilde{\boldsymbol{b}}_{\boldsymbol{\tau}}(v) = \tilde{\boldsymbol{b}}(v)$).

By our choice of ω and Equation (3) in [42],

$$\Pr(\boldsymbol{E}_1) \leq \Pr(\exists v \in V, |\tilde{\boldsymbol{b}}_\omega(v) - \mathrm{bc}(v)| > \lambda) \leq \frac{\delta}{2},$$

where $\tilde{\boldsymbol{b}}_\omega(v)$ is the approximate betweenness of v after ω samples. Furthermore, by Theorem 6,

$$\begin{aligned}\Pr(\boldsymbol{E}_2) &\leq \sum_{v\in V}\Pr(\boldsymbol{\tau} < \omega \wedge -f \geq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v)) + \Pr(\boldsymbol{\tau} < \omega \wedge \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \leq g)\\ &\leq \sum_{v\in V}\delta_L^{(v)} + \delta_U^{(v)} \leq \frac{\delta}{2}.\end{aligned}$$

By a union bound, $\Pr(\boldsymbol{E}_1 \vee \boldsymbol{E}_2) \leq \Pr(\boldsymbol{E}_1) + \Pr(\boldsymbol{E}_1) \leq \delta$, concluding the proof of Theorem 4. □

7 HOW TO CHOOSE $\delta_L^{(v)}, \delta_U^{(v)}$

In Section 6, we proved that our algorithm works for any choice of the values $\delta_L^{(v)}, \delta_U^{(v)}$. In this section, we show how we can heuristically compute such values, in order to obtain the best performances.

For each node v, let $\lambda_L^{(v)}, \lambda_U^{(v)}$ be the lower and the upper maximum error that we want to obtain on the betweenness of v: if we simply want all errors to be smaller than λ, we choose $\lambda_L^{(v)}, \lambda_U^{(v)} = \lambda$, but for other purposes different values might be needed. We want to minimize the time τ such that the approximation of the betweenness at time τ is in the confidence interval required. In formula, we want to minimize

$$\min\left\{\tau \in \mathbb{N} : \forall v \in V, \left(f\left(\tilde{\boldsymbol{b}}_\tau(v), \delta_L^{(v)}, \omega, \tau\right) \leq \lambda_L^{(v)} \wedge g\left(\tilde{\boldsymbol{b}}_\tau(v), \delta_U^{(v)}, \omega, \tau\right) \leq \lambda_U^{(v)}\right)\right\}, \tag{11}$$

where $\tilde{\boldsymbol{b}}_\tau(v)$ is the approximation of $\mathrm{bc}(v)$ obtained at time τ, and

$$f\left(\tau, \tilde{\boldsymbol{b}}_\tau, \delta_L, \omega\right) = \frac{1}{\tau}\log\frac{1}{\delta_L}\left(\frac{1}{3} - \frac{\omega}{\tau} + \sqrt{\left(\frac{1}{3} - \frac{\omega}{\tau}\right)^2 + \frac{2\tilde{\boldsymbol{b}}_\tau\omega}{\log\frac{1}{\delta_L}}}\right) \quad \text{and}$$

$$g\left(\tau, \tilde{\boldsymbol{b}}_\tau, \delta_U, \omega\right) = \frac{1}{\tau}\log\frac{1}{\delta_U}\left(\frac{1}{3} + \frac{\omega}{\tau} + \sqrt{\left(\frac{1}{3} + \frac{\omega}{\tau}\right)^2 + \frac{2\tilde{\boldsymbol{b}}_\tau\omega}{\log\frac{1}{\delta_U}}}\right).$$

The goal of this section is to provide deterministic values of $\delta_L^{(v)}, \delta_U^{(v)}$ that minimize the value in Equation (11), and such that $\sum_{v\in V}\delta_L^{(v)} + \delta_U^{(v)} < \frac{\delta}{2}$. To obtain our estimate, we replace $\tilde{\boldsymbol{b}}_\tau(v)$ with an approximation $\tilde{b}(v)$, that we compute by sampling α paths, before starting the algorithm (in our code, $\alpha = \frac{\omega}{100}$). Furthermore, we consider a simplified version of Equation (11): in most cases, λ_L is much smaller than all other quantities in play, and since ω is proportional to $\frac{1}{\lambda_L^2}$, we can safely assume

$$f(\tau, \tilde{b}(v), \delta_L^{(v)}, \omega) \approx \sqrt{\frac{2\tilde{b}(v)\omega}{\tau^2}\log\frac{1}{\delta_L}} \qquad \text{and} \qquad g(\tau, \tilde{b}(v), \delta_U^{(v)}, \omega) \approx \sqrt{\frac{2\tilde{b}(v)\omega}{\tau^2}\log\frac{1}{\delta_U}}.$$

Hence, in place of the value in Equation (11), our heuristic tries to minimize

$$\min\left\{\tau \in \mathbb{N} : \forall v \in V, \sqrt{\frac{2\tilde{b}(v)\omega}{\tau^2}\log\frac{1}{\delta_L^{(v)}}} \leq \lambda_L^{(v)} \wedge \sqrt{\frac{2\tilde{b}(v)\omega}{\tau^2}\log\frac{1}{\delta_U^{(v)}}} \leq \lambda_U^{(v)}\right\}.$$

Solving with respect to τ, we are trying to minimize

$$\max_{v\in V}\left(\max\left(\sqrt{\frac{2\tilde{b}(v)\omega}{\left(\lambda_L^{(v)}\right)^2}\log\frac{1}{\delta_L^{(v)}}}, \sqrt{\frac{2\tilde{b}(v)\omega}{\left(\lambda_U^{(v)}\right)^2}\log\frac{1}{\delta_U^{(v)}}}\right)\right),$$

which is the same as minimizing $\max_{v\in V}\max(c_L(v)\log\frac{1}{\delta_L^{(v)}}, c_U(v)\log\frac{1}{\delta_U^{(v)}})$ for some constants $c_L(v), c_U(v)$, conditioned on $\sum_{v\in V}\delta_L^{(v)} + \delta_U^{(v)} < \frac{\delta}{2}$. We claim that, among the possible choices of $\delta_L^{(v)}, \delta_U^{(v)}$, the best choice makes all the terms in the maximum equal: otherwise, if two terms were different, we would be able to slightly increase and decrease the corresponding values, in order to decrease the maximum. This means that, for some constant C, for each v, $c_L(v)\log\frac{1}{\delta_L^{(v)}} = c_U(v)\log\frac{1}{\delta_L^{(v)}} = C$, that is, $\delta_L^{(v)} = \exp(-\frac{C}{c_L(v)})$, $\delta_U^{(v)} = \exp(-\frac{C}{c_U(v)})$. In order to find the largest

constant C such that $\sum_{v \in V} \delta_L^{(v)} + \delta_U^{(v)} \leq \frac{\delta}{2}$, we use a binary search procedure on all possible constants C.

Finally, if $c_L(v) = 0$ or $c_U(v) = 0$, this procedure chooses $\delta_L^{(v)} = 0$: to avoid this problem, we impose $\sum_{v \in V} \delta_L^{(v)} + \delta_U^{(v)} \leq \frac{\delta}{2} - \epsilon\delta$, and we add $\frac{\epsilon\delta}{2n}$ to all the $\delta_L^{(v)}$'s and all the $\delta_U^{(v)}$'s (in our code, we choose $\epsilon = 0.001$). The pseudocode of the algorithm is available in Algorithm 3.

ALGORITHM 3: The function `computeDelta` .

Input: a graph $G = (V, E)$, and two values $\lambda_L^{(v)}, \lambda_U^{(v)}$ for each $v \in V$
Output: for each $v \in V$, two values $\delta_L^{(v)}, \delta_U^{(v)}$

$\alpha \leftarrow \frac{\omega}{100}$;
$\epsilon \leftarrow 0.0001$;
foreach $i \in [1, \alpha]$ **do**
  $\boldsymbol{\pi}$ = `samplePath()`;
  foreach $v \in \boldsymbol{\pi}$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
end
foreach $v \in V$ **do**
  $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\alpha$;
  $c_L(v) \leftarrow \frac{2\tilde{b}(v)\omega}{(\lambda_L^{(v)})^2}$;
  $c_U(v) \leftarrow \frac{2\tilde{b}(v)\omega}{(\lambda_U^{(v)})^2}$;
end
Binary search to find C such that $\sum_{v \in V} \exp\left(-\frac{C}{c_L(v)}\right) + \exp\left(-\frac{C}{c_U(v)}\right) = \frac{\delta}{2} - \epsilon\delta$;
foreach $v \in V$ **do**
  $\delta_L^{(v)} \leftarrow \exp\left(-\frac{C}{c_L(v)}\right) + \frac{\epsilon\delta}{2n}$;
  $\delta_U^{(v)} \leftarrow \exp\left(-\frac{C}{c_U(v)}\right) + \frac{\epsilon\delta}{2n}$;
end
return $\boldsymbol{b}$;

8 BALANCED BIDIRECTIONAL BFS ON RANDOM GRAPHS

In this section, we prove Theorem 5, following the sketch of proof given in Section 4.1.

We start by providing the formal details regarding the efficiency of the bidirectional BFS in several models of random graphs: the Configuration Model (CM, [12]), and Rank-1 Inhomogeneous Random Graph models (IRG, [48, Chapter 3]), such as the Chung-Lu model [35], the Norros-Reittu model [38], and the Generalized Random Graph [48, Chapter 3]. All these models are defined by fixing the number n of nodes and n weights ρ_v, and by creating edges at random, in a way that node v gets degree close to ρ_v.

More formally, the edges are generated as follows:

- In the CM, each node is associated to ρ_v half-edges, or stubs; edges are created by randomly pairing these $M = \sum_{v \in V} \rho_v$ stubs (we assume the number of stubs to be even, by adding a stub to a random node if necessary).
- In IRG, an edge between a node v and a node w exists with probability $f(\frac{\rho_v \rho_w}{M})$, where $M = \sum_{v \in V} \rho_v$, and the existence of different edges is independent. Different choices of the function f create different models.

- In general, we assume that f satisfies the following conditions:
  * f is derivable at least twice in 0;
  * f is increasing;
  * $f'(0) = 1$;
- in the Chung-Lu model, $f(x) = \min(x, 1)$;
- in the Norros-Reittu model, $f(x) = 1 - e^{-x}$;
- in the Generalized Random Graph model, $f(x) = \frac{x}{1+x}$.

It remains to define how we choose the weights ρ_v, when the number of nodes n tends to infinity. In the line of previous works [27, 38, 48], we consider a sequence of graphs G_i, whose number of nodes n_i tends to infinity, and whose degree distribution Λ_i satisfy the following:

(1) there is a probability distribution Λ such that the Λ_i's tend to Λ in distribution;
(2) $M_1(\Lambda_i)$ tends to $M_1(\Lambda) < \infty$, where $M_1(\Lambda)$ is the first moment of Λ;
(3) one of the following two conditions hold:
   (a) $M_2(\Lambda_i)$ tends to $M_2(\Lambda) < \infty$, where $M_2(\Lambda)$ is the second moment of Λ;
   (b) Λ is a power-law distribution with $2 < \beta < 3$, and there is a global constant C such that, for each d, $\Pr(\Lambda_i \geq d) \leq \frac{C}{d^{\beta-1}}$.

For example, these assumptions are satisfied with probability 1 if we choose the degrees independently, according to a distribution Λ with finite mean [48, Section 6.1,7.2].

Remark. These assumptions cover the Erdös-Renyi random graph with constant average degree, and all power-law distributions with $\beta > 2$ (because, if $\beta > 3$, then $M_2(\Lambda)$ is finite).

Remark. Assumption 3b seems less natural than the other assumptions. However, it is necessary to exclude pathological cases: for example, assume that G_i has $n - 2$ nodes chosen according to a power-law distribution, and two nodes u, v with weight $n^{1-\epsilon}$. All assumptions except 3b are satisfied, but the bidirectional BFS is not efficient, because if s is a neighbor of u with degree 1, and t is a neighbor of v with degree 1, then a bidirectional BFS from s and t needs to visit all neighbors of u or all neighbors of v, and the time needed is $\Omega(n^{1-\epsilon})$.

We say that a random graph has a property π asymptotically almost surely (a.a.s.) if $\Pr(\pi(G_i))$ tends to 1 when n tends to infinity. We say that a random graph has a property π w.h.p. if $\frac{\Pr(\pi(G_i))}{n_i^k}$ tends to 0 for some $k > 0$.

Before proving the main theorem, we need two more definitions and a technical assumption.

Definition 8.1. In the CM, let $\rho_{\text{res}} = \frac{M_2(\Lambda)}{M_1(\Lambda)} - 1$. In IRG, let $\rho_{\text{res}} = \frac{M_2(\Lambda)}{M_1(\Lambda)}$ (if Λ is a power-law distribution with $2 < \beta < 3$, we simply define $\rho_{\text{res}} = +\infty$).

Definition 8.2. Given a set $V' \subseteq V$, the volume of V' is $\rho_{V'} = \sum_{v \in V'} \rho_v$. Furthermore, if $V' = \Gamma^d(s)$, we abbreviate $\rho_{\Gamma^d(s)}$ with $\boldsymbol{r}^l(s)$.

The value ρ_{res} is closely related to $\frac{\boldsymbol{r}^{l+1}(s)}{\boldsymbol{r}^l(s)}$: informally, the expected value of this fraction is ρ_{res}. For this reason, if $\rho_{\text{res}} < 1$, then the size of neighbors tends to decrease, and all connected components have $O(\log n)$ nodes. Conversely, if $\rho_{\text{res}} > 1$, then the size of neighbors tends to increase, and there is a *giant component* of size $\Theta(n)$ (for a proof of these facts, see [48, Section 2.3 and Chapter 4]). Our last assumption is that $\rho_{\text{res}} > 1$, in order to ensure the existence of the giant component.

Under these assumptions, we prove Theorem 5, following the sketch in Section 4. We start by linking the degrees and the weights of nodes.

Lemma 7. *For each node v, $\rho_v n^{-\epsilon} \leq \deg(v) \leq \rho_v n^{\epsilon}$ w.h.p.*

Proof. We use [16, Lemmas 32 and 37][4]: these lemmas imply that, for each $\epsilon > 0$, if $\rho_v > n^\epsilon$, $(1-\epsilon)\rho_v \le \deg(v) \le (1+\epsilon)\rho_v$ w.h.p. We have to handle the case where $\rho_v < n^\epsilon$: one of the two inequalities is empty, while for the other inequality we observe that, if we decrease the weight of v, the degree of v can only decrease. Hence, if $\rho_v < n^\epsilon$, $\deg(v) < (1+\epsilon)n^\epsilon$, and the result follows by changing the value of ϵ. □

Following the intuitive proof, we have linked the number of visited edges with their weights. Let us define an abbreviation for the volume of the nodes at distance l from s.

Definition 8.3. We denote by $\boldsymbol{r}^l(s)$ the volume of nodes at distance exactly l from s. In the CM, we denote by $\boldsymbol{R}^l(s)$ the set of stubs at distance l from s.

Now, we need to show that, if $\boldsymbol{r}^{l_s}(s), \boldsymbol{r}^{l_t}(t) > n^{\frac{1}{2}+\epsilon}$, then $d(s,t) \le l_s + l_t + 2$ w.h.p.

Lemma 8. *Assume that $\boldsymbol{r}^{l_s}(s) > n^{\frac{1}{2}+\epsilon}$, $\boldsymbol{r}^{l_t}(t) > n^{\frac{1}{2}+\epsilon}$, and $\boldsymbol{r}^{l_s-1}(s), \boldsymbol{r}^{l_t-1}(t) < (1-\epsilon)n^{\frac{1}{2}+\epsilon}$. Then, $d(s,t) \le l_s + l_t + 2$.*

Proof. Let us assume that we know the structure of $\boldsymbol{N}^{l_s}(s)$ and $\boldsymbol{N}^{l_t}(t)$, that is, for each possible structure S of the subgraph induced by all nodes at distance l_s from s and distance l_t from t, let E_S be the event that $\boldsymbol{N}^{l_s}(s)$ and $\boldsymbol{N}^{l_t}(t)$ are exactly S. If we prove that $\Pr(d(s,t) \le l + l' + 2 | E_S) < \epsilon$, then $\Pr(\boldsymbol{r}^{l+1}(s) > \boldsymbol{r}^l(s)) = \sum_S \Pr(\boldsymbol{r}^{l+1}(s) > \boldsymbol{r}^l(s) | E_S) \Pr(E_S) < \sum_S \epsilon \Pr(E_S) = \epsilon$. First of all, if S is such that the two neighborhoods touch each other, $\Pr(d(s,t) \le l + l' + 2 | E_S) = 0 < \epsilon$. Otherwise, we consider separately the CM and IRG.

In the CM, conditioned on E_S, the stubs that are paired with stubs in $\boldsymbol{R}^{l_s}(s)$ are a random subset of the set of stubs that are not paired in S. This random subset has size at least $\epsilon n^{\frac{1}{2}+\epsilon} \ge n^{\frac{1+\epsilon}{2}}$ (because ϵ is a fixed constant, and n tends to infinity). Since the total number of stubs is $O(n)$, and since the number of stubs in $\boldsymbol{R}^{l_t}(t)$ is at least $\epsilon n^{\frac{1+\epsilon}{2}}$, one of the stubs in $\boldsymbol{R}^{l_t}(t)$ is paired with a stub in $\boldsymbol{r}^{l_s}(s)$ w.h.p., and $d(s,t) \le l_s + l_t + 1$.

In IRG, the probability that a node v is not connected to any node in $\Gamma^{l_s}(s)$ is at most $\prod_{w \in \Gamma^{l_s}(s)} (1 - f(\frac{\rho_v \rho_w}{M})) = \prod_{w \in \Gamma^{l_s}(s)} (1 - \Omega(\frac{\rho_w}{M})) = \exp(-\sum_{w \in \Gamma^{l_s}(s)} \Omega(\frac{\rho_w}{M})) = \exp(-\Omega(\frac{\boldsymbol{r}^{l_s}(s)}{M})) = 1 - \Omega(\frac{\boldsymbol{r}^{l_s}(s)}{M}) = 1 - \Omega(n^{-\frac{1}{2}+\epsilon})$. This means that v belongs to $\Gamma^{l_s+1}(s)$ with probability $\Omega(n^{-\frac{1}{2}+\epsilon})$, and similarly it belongs to $\Gamma^{l_t+1}(t)$ with probability $\Omega(n^{-\frac{1}{2}+\epsilon})$. Since the two events are independent, the probability that v belongs to both is $\Omega(n^{-1+2\epsilon})$. Since, for each node v, the events that v belongs to $\Gamma^{l_s+1}(s) \cap \Gamma^{l_t+1}(t)$ are independent, by a straightforward application of Hoeffding's inequality, w.h.p., there is a node v that belongs to $\Gamma^{l_s+1}(s) \cap \Gamma^{l_t+1}(s)$, and $d(s,t) \le l_s + l_t + 2$ w.h.p., concluding the proof. □

The next ingredient is used to bound the first integers l_s, l_t such that $\boldsymbol{r}^{l_s}(s), \boldsymbol{r}^{l_t}(t) > n^{\frac{1}{2}+\epsilon}$.

Theorem 9 (Theorem 5.1 in [27] for the CM, Theorem 14.8 in [13] for IRG (see also [16, 48])). *The diameter of a graph generated through the aforementioned models is $O(\log n)$.*

The last ingredient of our proof is an upper bound on the size of $\boldsymbol{r}^{l_s}(s)$ and $\boldsymbol{r}^{l_t}(t)$.

Lemma 10. *With high probability, for each $s \in V$ and for each l such that $\sum_{i=0}^{l} \boldsymbol{r}^l(s) < n^{\frac{1}{2}+\epsilon}$, $\boldsymbol{r}^{l+1}(s) < n^{\frac{1}{2}+3\epsilon}$ if Λ has finite second moment, $\boldsymbol{r}^{l+1}(s) < n^{\frac{4-\beta}{2}+3\epsilon}$ if Λ is a power law with $2 < \beta < 3$.*

Proof. We consider separately nodes with weight at most $n^{\frac{1}{2}-2\epsilon}$ from nodes with bigger weights: in the former case, we bound the number of such nodes that are in $\boldsymbol{R}^{l+1}(s)$, while in

[4]This article uses a further assumption on IRG, but the proofs of Lemmas 32 and 39 do not rely on this assumption.

the latter case we bound the total number of nodes with weight at least $n^{\frac{1}{2}-2\epsilon}$. Let us start with nodes with the latter case. □

Claim. for each ϵ, $\sum_{\rho_v \geq n^{\frac{1}{2}-\epsilon}} \rho_v$ is smaller than $n^{\frac{1}{2}+3\epsilon}$ if Λ has finite second moment, and it is smaller than $n^{\frac{4-\beta}{2}+3\epsilon}$ if Λ is a power law with $2 < \beta < 3$.

Proof of claim. If Λ has finite second moment, by Chebyshev inequality, for each α,

$$\Pr\left(\Lambda_i > n^{\frac{1}{2}+\alpha}\right) \leq \frac{\mathrm{Var}(\Lambda_i)}{n^{1+2\alpha}} \leq \frac{M_2(\Lambda_i)}{n^{1+2\alpha}} = O\left(\frac{M_2(\Lambda)}{n^{1+2\alpha}}\right) = O(n^{-1-2\alpha}).$$

For $\alpha = \epsilon$, this means that no node has weight bigger than $n^{\frac{1}{2}+\epsilon}$, and for $\alpha = -\epsilon$, this means that the number of nodes with weight bigger than $n^{\frac{1}{2}-\epsilon}$ is at most $n^{2\epsilon}$. We conclude that $\sum_{\rho_v \geq n^{\frac{1}{2}-\epsilon}} \rho_v \leq \sum_{\rho_v \geq n^{\frac{1}{2}-\epsilon}} n^{\frac{1}{2}+\epsilon} \leq n^{\frac{1}{2}+3\epsilon}$.

If Λ is a power law with $2 < \beta < 3$, by Assumption 3b the number of nodes with weight at least d is at most $Cnd^{-\beta+1}$. Consequently, using Abel's summation technique,

$$\begin{aligned}
\sum_{\rho_v \geq n^{\frac{1}{2}-\epsilon}} \rho_v &= \sum_{d=\rho_v}^{+\infty} d|\{v : \rho_v = d\}| \\
&= \sum_{d=n^{\frac{1}{2}-\epsilon}}^{+\infty} d(|\{v : \rho_v \geq d\}| - |\{v : \rho_v \geq d+1\}|) \\
&= \sum_{d=n^{\frac{1}{2}-\epsilon}}^{+\infty} d|\{v : \rho_v \geq d\}| - \sum_{d=n^{\frac{1}{2}-\epsilon}+1}^{+\infty} (d-1)|\{v : \rho_v \geq d\}| \\
&= n^{\frac{1}{2}-\epsilon}|\{v : \rho_v \geq n^{\frac{1}{2}-\epsilon}\}| + \sum_{d=n^{\frac{1}{2}-\epsilon}+1}^{+\infty} |\{v : \rho_v \geq d\}| \\
&\leq Cn^{\frac{1}{2}-\epsilon} n^{1-\left(\frac{1}{2}-\epsilon\right)(\beta-1)} + \sum_{d=n^{\frac{1}{2}-\epsilon}+1}^{+\infty} Cnd^{-\beta+1} \\
&= O\left(n^{\frac{4-\beta}{2}+\epsilon\beta} + n^{1-\left(\frac{1}{2}-\epsilon\right)(\beta-2)}\right) = O\left(n^{\frac{4-\beta}{2}+\epsilon\beta}\right).
\end{aligned}$$

□

By this claim, $\sum_{v \in \Gamma l+1 s, \rho_v \geq n^{\frac{1}{2}-2\epsilon}} \rho_v$ is smaller than $n^{\frac{1}{2}+6\epsilon}$ if Λ has finite second moment, and it is smaller than $n^{\frac{4-\beta}{2}+6\epsilon}$ if Λ is a power law with $2 < \beta < 3$. To conclude the proof, we only have to bound $\sum_{v \in \Gamma l+1 s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \rho_v$.

Claim. With high probability, $\sum_{v \in \Gamma l+1 s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \rho_v < n^{\frac{1}{2}+\epsilon}$ if Λ has finite second moment, $\sum_{v \in \Gamma l+1 s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \rho_v < n^{\frac{4-\beta}{2}+\epsilon}$ if Λ is a power law with $2 < \beta < 3$.

Proof of Claim, CM. As in the proof of Lemma 8, we can safely assume that we know the structure S of $\boldsymbol{N}^l(s)$. Let us sort the stubs in $\boldsymbol{R}^l(s)$, not paired by S, obtaining $a_1, \ldots, a_k$, and let $\boldsymbol{a}_i$ be the stub paired with a_i. Let $\mathrm{res}(a)$ be the number of stubs of the node a, minus a, and let $\boldsymbol{X}_i = \mathrm{res}(\boldsymbol{a}_i)$ if $\mathrm{res}(\boldsymbol{a}_i) < n^{\frac{1}{2}-2\epsilon}$, 0 otherwise: clearly, $\sum_{v \in \Gamma l+1 s, \rho_v \leq n^{\frac{1}{2}-2\epsilon}} \rho_v \leq \sum_{i=1}^{k} \boldsymbol{X}_i$ (with equality if

there are no horizontal or diagonal edges in the BFS tree). After the first $i-1$ stubs are paired, since $i < n^{\frac{1}{2}+\epsilon}$ and since the number of stubs paired in S is $O(n^{\frac{1}{2}+\epsilon}\log n)$, for each $k < n^{\frac{1}{2}-2\epsilon}$,

$$\begin{aligned}\Pr(X_i = k) &= \Pr(\text{res}(a_i) = k)\\ &= \frac{|\{a \in A : a \text{ unpaired after } i \text{ rounds}, \text{res}(a) = k\}|}{|\{a \in A : a \text{ unpaired after } i \text{ rounds}\}|}\\ &= \frac{|\{a \in A : \text{res}(a) = k\}| + O(n^{\frac{1}{2}+\epsilon})}{|A| + O(n^{\frac{1}{2}+\epsilon})}\\ &= \frac{(k+1)\Lambda(k+1)}{M_1(\Lambda)} + O(n^{-\frac{1}{2}+\epsilon}).\end{aligned}$$

Consequently, conditioned on all pairings of a_j for $j < i$, $\mathbb{E}[X_i] = \sum_{k=0}^{n^{\frac{1}{2}-2\epsilon}} k\frac{(k+1)\Lambda(k+1)}{M_1(\Lambda)} + O(n^{-\frac{1}{2}+\epsilon}\log n) = \alpha(n)$, where $\alpha(n) = O(1)$ if Λ has finite second moment, and $\alpha(n) = O(n^{\frac{3-\beta}{2}})$ if Λ is a power law with $2 < \beta < 3$. Hence, for each ϵ, $\sum_{i=1}^{k} X_i - i(M_1(\Lambda)+\epsilon)$ is a supermartingale, and by Azuma's inequality

$$\Pr\left(\sum_{i=1}^{k} X_i - k\alpha(n) \geq \alpha(n)\right) \leq \exp\left(-\frac{\alpha(n)^2}{2\sum_{i=1}^{k} n^{\frac{1}{2}-2\epsilon}}\right) \leq \exp(-n^{\epsilon}).$$

Then, w.h.p., $\sum_{i=1}^{k} X_i \leq n^{\frac{1}{2}+\epsilon}(\alpha(n)+2)$, concluding the proof of the claim. □

Proof of Claim, IRG. The number of nodes w with weight at most $n^{\frac{1}{2}-2\epsilon}$ that belong to $\Gamma l+1 s$ is at most

$$\sum_{v \in \Gamma l s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \sum_{w \in V} \rho_w X_{v,w},$$

where $X_{v,w} = 1$ with probability $f(\frac{\rho_v\rho_w}{M}) = O(\frac{\rho_v\rho_w}{M})$ because $\rho_v\rho_w < n^{1-\epsilon}$. Moreover,

$$\mathbb{E}\left[\sum_{v \in \Gamma l s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \sum_{w \in V} \rho_w X_{v,w}\right] = O\left(r^l(s)\frac{\sum_{v\in V}\rho_v^2}{n}\right) = r^l(s)\alpha(n),$$

where $\alpha(n) = O(1)$ if Λ has finite second moment, and $\alpha(n) = O(n^{\frac{3-\beta}{2}})$ if Λ is a power law with $2 < \beta < 3$.

By Hoeffding's inequality,

$$\Pr\left(\sum_{v \in \Gamma l s, \rho_v < n^{\frac{1}{2}-2\epsilon}} \sum_{w \in V} \rho_w X_{v,w} - r^l(s)\alpha(n) \geq r^l(s)\alpha(n)\right) \leq n^{\frac{r^l(s)\alpha(n)}{r^l(s)n^{\frac{1}{2}-2\epsilon}}} \leq n^{-\epsilon}.$$

This concludes the proof. □

This claim let us conclude the proof of the lemma.

Proof of Theorem 5. Let $D_s^i = \sum_{v\in\Gamma i s}\deg(v)$, $D_t^j = \sum_{w\in\Gamma j t}\deg(w)$, and let us suppose that we have visited until level l_s from s, until level l_t from t, and that $D_s^{l_s}, D_t^{l_t} > n^{\frac{1}{2}+2\epsilon}$. If this situation never occurs, by Theorem 9, the total number of visited edges is at most $O(\log n)n^{\frac{1}{2}+2\epsilon} = O(n^{\frac{1}{2}+3\epsilon})$, and the conclusion follows. Otherwise, again by Theorem 9, the number of edges visited in the two BFS trees before levels l_s and l_t is $O(n^{\frac{1}{2}+3\epsilon})$. Furthermore, by Lemma 7, $r^{l_s}(s), r^{l_t}(t) > n^{\frac{1}{2}+2\epsilon}$. We claim that, without loss of generality, we can assume $r^{l_s-1}(s) < \epsilon r^{l_s}(s)$, to apply Lemma 8.

Indeed, if $\boldsymbol{r}^{l_s-1}(s)$ is too big, we iteratively decrease l_s until we find a neighbor verifying $\boldsymbol{r}^{l_s}(s) > (1-\epsilon')\boldsymbol{r}^{l_s-1}(s)$. This process can last at most $O(\log n)$ steps, and hence it is stopped at a point l_s such that $\boldsymbol{r}^{l_s}(s) > n^{\frac{1}{2}+2\epsilon}(1-\epsilon')^{O(\log n)} \geq n^{\frac{1}{2}+\epsilon'}$ if ϵ' is small enough. Similarly, we can suppose without loss of generality that $\boldsymbol{r}^{l_t}(t) > (1-\epsilon')\boldsymbol{r}^{l_t-1}(t)$. By Lemma 8, $d(s,t) \leq l_s + l_t + 2$, and the number of nodes needed to conclude the BFS is at most $D_s^{l_s} + D_t^{l_t}$ (note that, if we extend twice the visit from s, it means that $D_s^{l_s+1} < D_t^{l_t}$). By Lemma 7, $D_s^{l_s} \leq n^{\epsilon}\boldsymbol{r}^{l_s}(s)$, and by Lemma 10 this value is at most $n^{\frac{1}{2}+3\epsilon}$ if λ has finite second moment, and $n^{\frac{4-\beta}{2}+3\epsilon}$ if λ is power law with $2 < \beta < 3$. We conclude that the total number of visited nodes is at most $n^{\frac{1}{2}+3\epsilon} + D_s^{l_s} + D_t^{l_t} \leq n^{\frac{1}{2}+3\epsilon} + \boldsymbol{r}^{l_s}(s) + \boldsymbol{r}^{l_t}(t) \leq n^{\frac{1}{2}+4\epsilon}$ ($n^{\frac{4-\beta}{2}+4\epsilon}$, respectively) if λ has finite second moment (if λ is power law with $2 < \beta < 3$, respectively). The theorem follows by changing the value of ϵ. □

9 EXPERIMENTAL RESULTS

In this section, we test our algorithm on several real-world networks, in order to evaluate its performances. The platform for our tests is a server with 1515GB RAM and 48 Intel(R) Xeon(R) CPU E7-8857 v2 cores at 3.00GHz, running Debian GNU Linux 8. The algorithms are implemented in C++, and they are compiled using gcc 5.3.1. The source code of our algorithm is available at https://github.com/natema/kadabra.

Comparison with the State of the Art. The first experiment compares the performances of our algorithm KADABRA with the state of the art. The first competitor is the RK algorithm [42], available in the open source *NetworKit* framework [46]. This algorithm uses the same estimator as our algorithm, but the stopping condition is different: it simply stops after sampling $k = \frac{c}{\epsilon^2}(\lfloor \log_2(\mathrm{VD}-2) \rfloor + 1 + \log(\frac{1}{\delta}))$, and it uses a heuristic to upper bound the vertex diameter. Following suggestions by the author of the *NetworKit* implementation, we set to 20 the number of samples used in the latter heuristic [7].

The second competitor is the ABRA algorithm [43], available at http://matteo.rionda.to/software/ABRA-radebetw. tbz2. This algorithm samples pairs of nodes (s,t), and it adds the fraction of st-paths passing from v to the approximation of the betweenness of v, for each node v. The stopping condition is based on a key result in statistical learning theory, and there is a scheduler that decides when it should be tested. Following the suggestions by the authors, we use both the automatic scheduler ABRA-Aut, which uses a heuristic approach to decide when the stopping condition should be tested, and the geometric scheduler ABRA-1.2, which tests the stopping condition after $(1.2)^i k$ iterations, for each integer i.

The test is performed on a dataset made by 15 undirected and 15 directed real-world networks, taken from the datasets SNAP (snap.stanford.edu/), LASAGNE (piluc.dsi.unifi.it/lasagne), and KONECT (http://konect.uni-koblenz.de/networks/). As in [43], we have considered all values of $\lambda \in \{0.03, 0.025, 0.02, 0.015, 0.01, 0.005\}$, and $\delta = 0.1$. All the algorithms have to provide an approximation $\tilde{\boldsymbol{b}}(v)$ of $\mathrm{bc}(v)$ for each v such that $\Pr(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda) \geq 1-\delta$. In Figure 1, we report the time needed by the different algorithms on every graph for $\lambda = 0.005$ (the behavior with different values of λ is very similar).

From the figure, we see that KADABRA is much faster than all the other algorithms, on all graphs: on average, our algorithm is about 100 times faster than RK in undirected graphs, and about 70 times faster in directed graphs; it is also more than 1,000 times faster than ABRA. The latter value is due to the fact that the ABRA algorithm has large running times on few networks: in some cases, it did not even conclude its computation within 1 hour. The authors confirmed that this behavior might be due to some bugs in the code, which seems to affect it only on specific graphs:

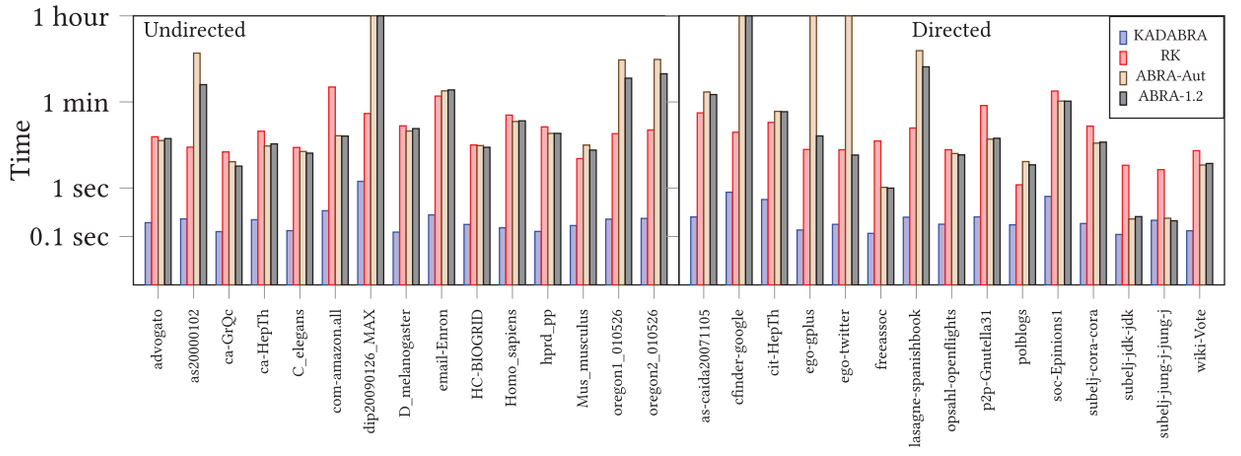


Fig. 1. The time needed by the different algorithms (y axis), on all the graphs of our dataset (x axis).

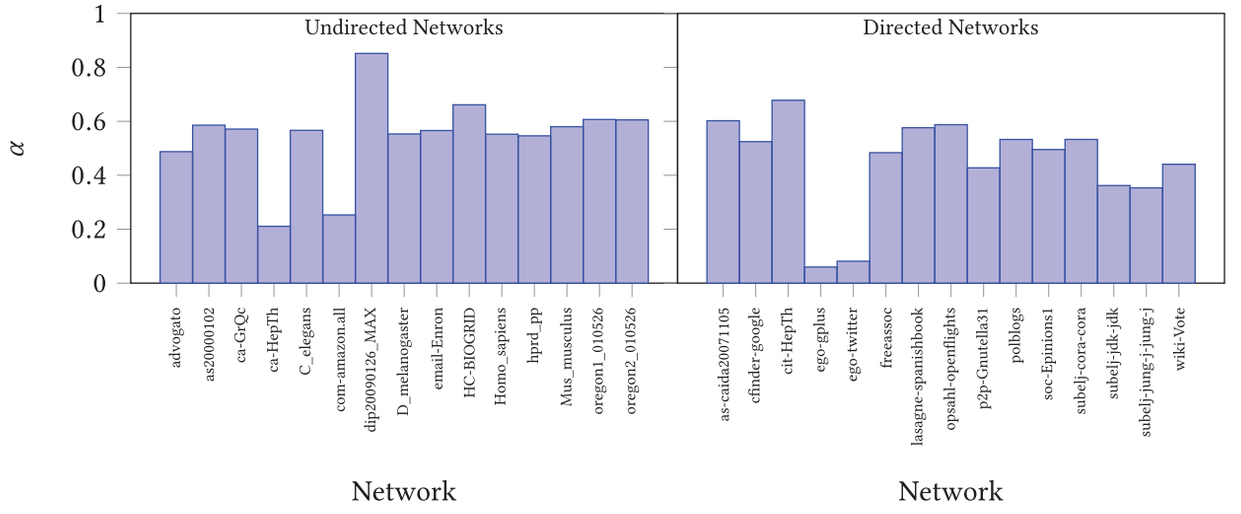


Fig. 2. The exponent α such that the average number of edges visited during a bidirectional BFS is n^α.

indeed, in most networks, the performances of ABRA are better than those of the RK algorithm (but, still, not better than KADABRA).

In order to explain these data, we take a closer look at the improvements obtained through the bidirectional BFS, by considering the average number of edges m_{avg} that the algorithm visits in order to sample a shortest path (for all our competitors, $m_{\text{avg}} = m$, since they perform a full BFS). In Figure 2, for each graph in our dataset, we plot $\alpha = \frac{\log(m_{\text{avg}})}{\log(m)}$ (intuitively, this means that the average number of edges visited is m^α).

The figure shows that, apart from few cases, the number of edges visited is close to $n^{\frac{1}{2}}$, confirming the results in Section 4. This means that, since many of our networks have approximately 10,000 edges, the bidirectional BFS is about 100 times faster than the standard BFS. Finally, for each value of λ, we report in Figure 3 the number of samples needed by all the algorithms, averaged over all the graphs in the dataset.

From the figure, KADABRA needs to sample the smallest amount of shortest paths, and the average improvement over RK grows when λ tends to 0, from a factor 1.14 (1.14, respectively) if $\lambda = 0.03$, to a factor 1.79 (2.05, respectively) if $\lambda = 0.005$ in the case of undirected (directed, respectively) networks. Again, the behavior of ABRA is highly influenced by the behavior on few networks, and as a consequence the average number of samples is higher. In any case, also in the graphs where ABRA has good performances, KADABRA still needs a smaller number of samples.

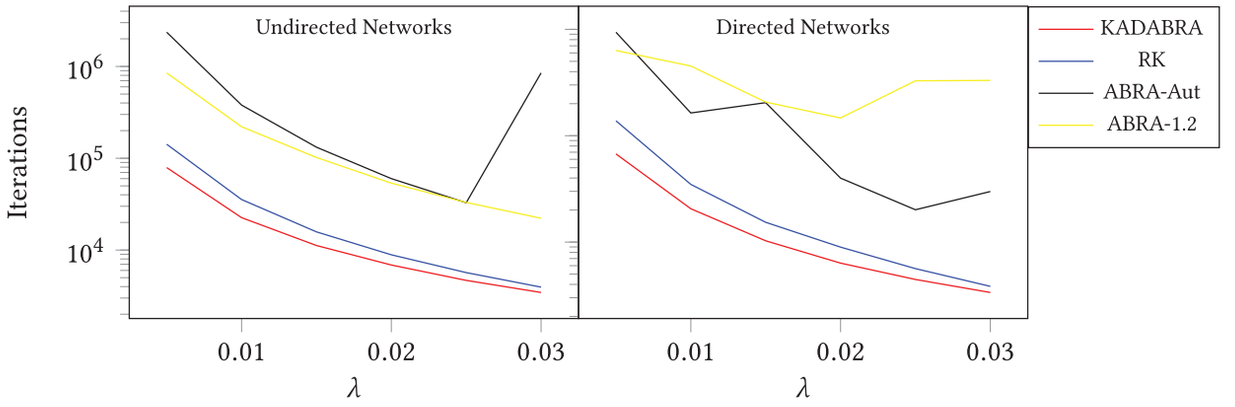


Fig. 3. The average number of samples needed by the different algorithms.

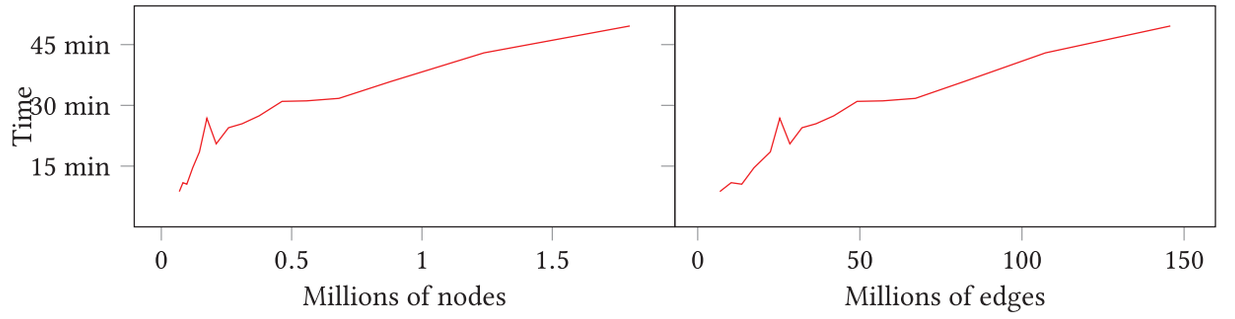


Fig. 4. The total time of computation of KADABRA on increasing snapshots of the IMDB graph.

Computing Top-k Centralities. In the second experiment, we let KADABRA compute the top-k betweenness centralities of large graphs, which were unfeasible to handle with the previous algorithms.

The first set of graphs is a series of temporal snapshots of the IMDB actor collaboration network, in which two actors are connected if they played together in a movie. The snapshots are taken every 5 years from 1940 to 2010, including a last snapshot in 2014, with 1,797,446 nodes and 145,760,312 edges. The graphs are extracted from the IMDB website (http://www.imdb.com), and they do not consider TV series, awards shows, documentaries, game shows, news, realities, and talk shows, in accordance to what was done in http://oracleofbacon.org.

The other graph considered is the Wikipedia citation network, whose nodes are Wikipedia pages, and which contains an edge from page p_1 to page p_2 if the text of page p_1 contains a link to page p_2. The graph is extracted from DBPedia 3.7 (http://wiki.dbpedia.org/), and it consists of 4,229,697 nodes and 102,165,832 edges.

We have run our algorithm with $\lambda = 0.0002$ and $\delta = 0.1$: as discussed in Section 5, this means that either two nodes are ranked correctly, or their centrality is known with precision at most λ. As a consequence, if two nodes are not ranked correctly, the difference between their real betweenness is at most 2λ. The full results are available in Section 10.2.

All the graphs were processed in less than 1 hour, apart from the Wikipedia graph, which was processed in approximately 1 hour and 38 minutes. In Figure 4, we plot the running times for the actor graphs: from the figure, it seems that the time needed by our algorithm scales slightly sublinearly with respect to the size of the graph. This result respects the results in Section 4, because the degrees in the actor collaboration network are power-law distributed with exponent $\beta \approx 2.13$ (http://konect.uni-koblenz.de/networks/actor-collaboration). Finally, we observe that the ranking is quite precise: indeed, most of the times, there are very few nodes in the top-5 with the same ranking, and the ranking rarely contains significantly more than 10 nodes.

9.1 Detailed Experimental Results

Table 1. Detailed Experimental Results (Undirected Graphs)

	Number of iterations				Time (s)				Edges
Graph	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA
$\lambda = 0.005$									
advogato	64,427	126,052	174,728	185,998	0.193	11.450	9.557	10.498	261.2
as20000102	115,797	126,052	18,329,844	4,126,626	0.231	6.990	611.584	136.764	377.6
ca-GrQc	61,611	146,052	142,982	129,165	0.126	5.574	3.500	2.839	353.4
ca-HepTh	31,735	146,052	121,587	129,165	0.222	14.921	7.389	8.168	9.9
C_elegans	69,729	146,052	204,634	185,998	0.132	6.876	5.693	5.261	270.7
com-amazon.all	40,711	166,052	69,708	74,747	0.340	122.020	12.011	11.849	21.9
dip20090126_MAX	156,552	166,052			1.374	34.595			15,354.9
D_melanogaster	51,227	126,052	144,680	154,998	0.123	19.253	15.061	16.882	520.8
email-Enron	74,745	146,052	257,989	267,838	0.280	79.296	101.529	106.278	1,408.0
HC-BIOGRID	78,804	146,052	245,780	223,198	0.177	7.751	7.534	6.951	713.2
Homo_sapiens	60,060	146,052	156,973	154,998	0.151	32.078	23.716	24.449	643.8
hprd_pp	59,125	146,052	151,499	154,998	0.127	18.323	13.425	13.458	456.4
Mus_musculus	92,081	146,052	504,669	385,688	0.168	4.058	7.723	6.083	226.6
oregon1_010526	114,829	126,052	6,798,931	2,865,712	0.228	13.281	442.370	185.711	681.6
oregon2_010526	115,764	126,052	5,714,183	2,865,712	0.236	15.823	452.554	229.234	822.2
$\lambda = 0.010$									
advogato	19,811	31,513	47,076	48,243	0.081	2.804	2.576	2.788	258.2
as20000102	29,062	31,513	2,688,614	1,070,372	0.071	1.777	88.886	35.049	377.3
ca-GrQc	18,535	36,513	37,529	33,501	0.049	1.417	0.987	0.753	350.6
ca-HepTh	13,761	36,513	31,721	33,501	0.188	3.771	2.078	2.275	10.0
C_elegans	19,888	36,513	54,327	48,243	0.048	1.803	1.586	1.483	269.4
com-amazon.all	14,641	41,513	18,007	19,386	0.312	31.004	5.196	7.623	21.5
dip20090126_MAX	39,314	41,513			0.395	8.578			1,5301.7
D_melanogaster	15,136	31,513	37,219	40,202	0.063	4.983	3.891	4.715	519.9
email-Enron	21,637	36,513	65,392	69,471	0.198	19.877	24.997	27.296	1387.2
HC-BIOGRID	22,924	36,513	62,413	57,892	0.052	1.979	1.989	1.906	712.5
Homo_sapiens	20,273	36,513	41,006	40,202	0.085	7.876	6.442	6.636	642.7
hprd_pp	18,403	36,513	39,994	40,202	0.074	4.348	4.097	3.714	456.4
Mus_musculus	25,146	36,513	130,384	100,040	0.061	1.055	1.965	1.718	223.9
oregon1_010526	30,514	31,513	1,104,167	743,313	0.087	3.254	70.383	47.740	683.3
oregon2_010526	29,117	31,513	954,515	743,313	0.088	3.983	73.942	59.103	822.1
$\lambda = 0.015$									
advogato	9570	14006	21,027	22,204	0.050	1.428	1.227	1.299	261.0
as20000102	13035	14006	705,483	492,651	0.047	0.776	22.939	16.136	377.6
ca-GrQc	8668	16228	17,419	15,419	0.031	0.637	0.493	0.361	345.8
ca-HepTh	7524	16228	15,002	15,419	0.167	1.641	0.939	1.050	11.5
C_elegans	10956	16228	25,233	22,204	0.034	0.782	0.740	0.732	267.6
com-amazon.all	8228	18451		15,419	0.301	13.814		7.785	21.9
dip20090126_MAX	17578	18451			0.203	3.851			15197.2
D_melanogaster	9350	14006	17,229	18,503	0.053	2.216	1.904	2.182	519.3
email-Enron	11209	16228	29,134	31,974	0.170	8.845	10.510	12.423	1367.4
HC-BIOGRID	12694	16228	28,805	26,645	0.043	0.858	0.946	0.947	708.6
Homo_sapiens	10142	16228	18,491	18,503	0.072	3.717	3.076	3.061	640.4
hprd_pp	10659	16228	17,969	18,503	0.056	1.919	1.719	1.752	451.5
Mus_musculus	11825	16228	59,756	46,043	0.033	0.458	0.906	0.812	222.8
oregon1_010526	13662	14006	426,845	342,118	0.056	1.522	26.420	21.871	681.4
oregon2_010526	13024	14006	333,638	342,118	0.060	1.773	26.070	27.298	833.6

(Continued)

Table 1. Continued.

	Number of iterations				Time (s)				Edges
Graph	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA
$\lambda = 0.020$									
advogato	5,874	7,879	11,993	12,915	0.054	0.710	0.665	0.765	260.3
as20000102	7,436	7,879	312,581	238,814	0.037	0.441	10.066	7.819	376.2
ca-GrQc	5,313	9,129	9,939	10,762	0.032	0.356	0.293	0.268	347.9
ca-HepTh	5,115	9,129	8,708	8,968	0.191	0.891	0.694	0.611	10.5
C_elegans	7,172	9,129	14,871	12,915	0.030	0.439	0.436	0.439	263.5
com-amazon.all	5,467	10,379	12,232	10,762	0.331	7.683	4.338	5.459	17.9
dip20090126_MAX	9,966	10,379			0.148	2.165			15,188.3
D_melanogaster	5,610	7,879	10,201	10,762	0.056	1.236	1.265	1.306	520.9
email-Enron	7,458	9,129	16,443	15,498	0.174	4.916	6.102	6.034	1371.7
HC-BIOGRID	8,459	9,129	17,406	15,498	0.026	0.505	0.602	0.582	716.6
Homo_sapiens	6,292	9,129	10,481	10,762	0.064	1.944	1.672	1.814	644.8
hprd_pp	6,611	9,129	10,501	10,762	0.050	1.089	0.930	1.050	449.8
Mus_musculus	7,227	9,129	31,634	26,782	0.026	0.255	0.507	0.532	221.0
oregon1_010526	7,733	7,879	220,948	199,011	0.051	0.863	13.584	12.989	679.2
oregon2_010526	7,381	7,879	152,242	165,842	0.059	1.031	11.676	13.290	836.0
$\lambda = 0.025$									
advogato	3,883	5,043	7,439	7,110	0.052	0.450	0.421	0.468	263.4
as20000102	4,829	5,043	130,506	157,779	0.033	0.285	4.097	5.108	373.5
ca-GrQc	3,982	5,843	6,427	5,925	0.028	0.242	0.180	0.162	342.1
ca-HepTh	3,773	5,843	6,016	5,925	0.176	0.573	0.374	0.416	11.8
C_elegans	4,477	5,843	9,557	8,532	0.025	0.292	0.293	0.293	266.6
com-amazon.all	4,059	6,643	58,995	14,745	0.338	4.744	9.644	7.217	21.3
dip20090126_MAX	6,457	6,643			0.125	1.397			15,193.8
D_melanogaster	3,993	5,043	6,279	7,110	0.056	0.793	0.827	0.870	522.6
email-Enron	4,576	5,843	11,001	12,287	0.574	3.289	3.888	4.705	1381.5
HC-BIOGRID	5,940	5,843	11,109	10,239	0.029	0.321	0.414	0.404	714.0
Homo_sapiens	4,796	5,843	7,109	7,110	0.077	1.245	1.154	1.215	647.2
hprd_pp	5,071	5,843	6,772	7,110	0.052	0.687	0.579	0.647	446.3
Mus_musculus	4,477	5,843	18,626	17,694	0.026	0.168	0.302	0.385	219.8
oregon1_010526	5,027	5,043	92,520	109,568	0.058	0.516	5.762	7.014	681.0
oregon2_010526	4,763	5,043	86,287	91,306	0.050	0.638	7.140	7.420	847.5
$\lambda = 0.030$									
advogato	3,256	3,502	5,521	5,090	0.048	0.361	0.335	0.322	260.6
as20000102	3,388	3,502	122,988	94,140	0.029	0.199	3.899	3.182	378.7
ca-GrQc	2,981	4,057	4,686	4,241	0.025	0.169	0.145	0.175	344.7
ca-HepTh	2,992	4,057	4,022	4,241	0.190	0.435	0.286	0.341	7.9
C_elegans	3,707	4,057	6,905	6,108	0.026	0.198	0.218	0.217	265.9
com-amazon.all	3,157	4,613	39,917	12,668	0.330	3.631	8.491	6.852	17.5
dip20090126_MAX	4,499	4,613	12,373,086		0.300	0.972	1,958.083		15,199.0
D_melanogaster	2,893	3,502	4,883	5,090	0.052	0.562	0.620	0.807	510.4
email-Enron	3,619	4,057	7,321	7,330	0.172	2.735	2.724	2.806	1,399.7
HC-BIOGRID	3,883	4,057	7,499	7,330	0.024	0.367	0.316	0.307	720.8
Homo_sapiens	3,322	4,057	4,982	5,090	0.066	0.897	0.842	0.877	654.2
hprd_pp	3,355	4,057	5,028	5,090	0.048	0.478	0.458	0.503	448.8
Mus_musculus	3,806	4,057	14,290	10,556	0.033	0.127	0.237	0.233	221.4
oregon1_010526	3,542	3,502	85,854	78,450	0.052	0.366	5.402	5.039	675.7
oregon2_010526	3,355	3,502	61,841	65,375	0.048	0.509	4.972	5.302	822.8

Empty values correspond to graphs on which the algorithm needed more than 1 hour.

Table 2. Detailed Experimental Results (Directed Graphs)

	Number of iterations				Time (s)				Edges
Graph	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA
$\lambda = 0.005$									
as-caida20071105	103,488	146,052	546,951	462,826	0.253	35.652	96.312	85.201	1,066.4
cfinder-google	137,313	146,052			0.820	14.190			554.4
cit-HepTh	98,054	166,052	481,476	462,826	0.579	22.651	38.339	37.720	5,773.1
ego-gplus	37,862	66,052		2,388,093	0.136	6.266		11.912	1.9
ego-twitter	37,125	66,052		154,998	0.178	6.181		4.804	2.3
freeassoc	41,602	166,052	89,424	89,697	0.116	9.384	1.036	0.997	223.5
lasagne-spanishbook	112,266	146,052	8,918,751	4,126,626	0.250	17.374	687.815	318.784	552.8
opsahl-openflights	73,744	146,052	200,164	185,998	0.179	6.191	5.165	4.849	431.1
p2p-Gnutella31	39,193	166,052	81,335	89,697	0.254	50.542	10.213	10.662	162.1
polblogs	71,423	126,052	387,278	321,406	0.174	1.165	3.522	3.017	190.3
soc-Epinions1	58,223	146,052	109,607	107,637	0.671	100.516	62.524	62.167	671.9
subelj-cora-cora	68,112	186,052	180,740	185,998	0.185	19.012	8.464	8.873	440.4
subelj-jdk-jdk	42,361	146,052	84,549	89,697	0.110	2.955	0.230	0.257	51.5
subelj-jung-j-jung-j	43,637	126,052	84,225	89,697	0.216	2.397	0.238	0.211	45.9
wiki-Vote	47,003	126,052	100,153	107,637	0.131	5.916	2.990	3.219	162.4
$\lambda = 0.010$									
as-caida20071105	30,382	36,513	132,997	120,048	0.135	8.902	22.251	20.315	1,066.1
cfinder-google	34,452	36,513			0.156	3.664			553.2
cit-HepTh	27,203	41,513	117,633	120,048	0.255	5.654	8.803	9.677	5,798.8
ego-gplus	13,123	16,513		4,602,412	0.085	1.584		22.510	2.3
ego-twitter	13,310	16,513		83,366	0.086	1.518		3.500	2.2
freeassoc	13,222	41,513	23,586	23,264	0.080	2.335	0.238	0.227	220.7
lasagne-spanishbook	32,527	36,513	1,366,576	1,070,372	0.101	4.339	104.916	83.610	553.4
opsahl-openflights	22,473	36,513	52,196	48,243	0.059	1.475	1.348	1.339	432.0
p2p-Gnutella31	13,101	41,513	21,567	23,264	0.192	12.950	2.677	2.831	162.1
polblogs	22,286	31,513	101,466	83,366	0.046	0.298	1.078	0.834	190.6
soc-Epinions1	17,061	36,513	28,493	27,917	0.320	27.194	16.516	15.974	659.5
subelj-cora-cora	23,078	46,513	47,936	48,243	0.128	4.797	1.988	2.101	432.4
subelj-jdk-jdk	14,047	36,513	22,038	23,264	0.066	0.734	0.099	0.075	52.2
subelj-jung-j-jung-j	14,894	36,513	22,266	23,264	0.064	0.696	0.113	0.083	46.4
wiki-Vote	17,380	31,513	26,352	27,917	0.088	1.446	0.792	0.870	155.7
$\lambda = 0.015$									
as-caida20071105	14,157	16,228	55,049	55,252	0.477	3.963	8.518	8.914	1,059.6
cfinder-google	15,400	16,228			0.123	1.666			558.1
cit-HepTh	13,002	18,451	47,035	46,043	0.232	2.529	3.807	3.766	5,883.0
ego-gplus	7,205	7,340		2,118,317	0.080	0.710		12.808	2.2
ego-twitter	7,403	7,340	1,958,981	114,573	0.082	0.704	14.021	5.304	2.3
freeassoc	7,095	18,451	10,956	10,707	0.297	1.072	0.115	0.110	222.0
lasagne-spanishbook	14,542	16,228	437,041	410,542	0.068	1.936	34.098	33.153	552.8
opsahl-openflights	11,550	16,228	24,433	22,204	0.034	0.649	0.643	0.648	433.9
p2p-Gnutella31	7,227	18,451	10,002	10,707	0.190	5.732	1.317	1.444	157.1
polblogs	10,296	14,006	46,648	38,369	0.029	0.136	0.516	0.435	189.5
soc-Epinions1	9,273	16,228	13,571	12,849	0.450	12.115	7.661	7.629	662.0
subelj-cora-cora	11,297	20,673	20,940	22,204	0.502	2.135	0.937	1.073	445.6
subelj-jdk-jdk	8,360	14,006	10,045	10,707	0.052	0.288	0.080	0.049	51.6
subelj-jung-j-jung-j	8,712	16,228	10,319	10,707	0.046	0.312	0.068	0.042	45.6
wiki-Vote	8,668	14,006	12,406	12,849	0.408	0.659	0.380	0.429	152.6

(Continued)

Table 2. Continued.

	Number of iterations				Time (s)				Edges
Graph	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA	RK	ABRA-Aut	ABRA-1.2	KADABRA
$\lambda = 0.020$									
as-caida20071105	9,086	9,129	31,242	32,139	0.104	2.226	4.954	5.087	1,064.2
cfinder-google	8,745	9,129			0.353	0.946			551.9
cit-HepTh	8,679	10,379	27,755	32,139	1.249	1.442	2.225	2.684	5,758.0
ego-gplus	4,785	4,129		1,478,684	0.081	0.395		9.234	2.6
ego-twitter	4,950	7,879		138,201	0.083	0.743		5.079	2.4
freeassoc	4,268	10,379	6,509	6,227	0.065	0.609	0.078	0.073	216.4
lasagne-spanishbook	8,338	9,129	294,793	286,577	0.058	1.074	22.405	22.468	555.0
opsahl-openflights	7,392	9,129	14,202	12,915	0.029	0.364	0.390	0.391	432.3
p2p-Gnutella31	4,697	10,379	5,700	6,227	0.190	3.162	0.695	0.816	156.7
polblogs	6,325	7,879	25,593	22,318	0.023	0.076	0.283	0.252	188.4
soc-Epinions1	5,489	9,129	7,686	7,473	0.457	6.738	4.506	4.335	651.8
subelj-cora-cora	6,325	11,629	12,437	12,915	0.500	1.203	0.571	0.520	450.8
subelj-jdk-jdk	5,456	9,129	6,070	6,227	0.191	0.192	0.062	0.044	52.3
subelj-jung-j-jung-j	5,643	9,129		6,227	0.217	0.176		0.045	46.6
wiki-Vote	4,939	7,879	7,125	7,473	0.075	0.368	0.221	0.259	152.2
$\lambda = 0.025$									
as-caida20071105	5,723	5,843	21,020	21,233	0.022	1.465	3.129	3.340	1,093.4
cfinder-google	6,275	5,843			0.019	0.648			758.0
cit-HepTh	5,206	6,643	15,915	21,233	0.034	0.940	1.351	1.891	6,130.5
ego-gplus	2,989	5,043		4,200,646	0.013	0.485		20.309	2.6
ego-twitter	2,958	2,643		157,779	0.012	0.248		6.291	2.4
freeassoc	2,804	6,643	4,285	4,114	0.009	0.399	0.061	0.058	261.5
lasagne-spanishbook	5,409	5,043	129,999	131,482	0.013	0.592	10.040	10.221	626.1
opsahl-openflights	4,557	5,843	10,116	8,532	0.009	0.236	0.290	0.267	561.3
p2p-Gnutella31	3,069	6,643	3,931	4,114	0.043	2.149	0.590	0.663	176.8
polblogs	3,880	5,043	15,986	14,745	0.007	0.049	0.185	0.176	241.9
soc-Epinions1	3,689	5,843	5,060	4,937	0.188	4.158	2.798	2.791	888.1
subelj-cora-cora	5,264	7,443	7,699	8,532	0.020	0.781	0.360	0.408	436.5
subelj-jdk-jdk	3,201	5,843	9,428	4,937	0.008	0.122	0.065	0.036	57.2
subelj-jung-j-jung-j	3,168	5,043	13,471	5,925	0.007	0.098	0.057	0.045	57.7
wiki-Vote	3,265	5,043	4,566	4,937	0.009	0.241	0.137	0.178	174.7
$\lambda = 0.030$									
as-caida20071105	3,956	4,057	12,696	15,202	0.017	1.029	1.973	2.434	1,285.2
cfinder-google	4,419	4,057			0.013	0.412			770.5
cit-HepTh	4,062	4,613	13,172	12,668	0.033	0.672	1.195	1.059	6,131.6
ego-gplus	2,434	1,835		4,330,990	0.009	0.188		21.395	3.1
ego-twitter	2,270	1,835	98,839	135,562	0.008	0.174	4.909	5.510	2.2
freeassoc	2,105	4,613	3,008	3,534	0.006	0.285	0.101	0.091	250.7
lasagne-spanishbook	3,820	4,057	158,028	94,140	0.010	0.487	12.564	7.855	656.8
opsahl-openflights	3,450	4,057	6,556	6,108	0.007	0.165	0.184	0.195	481.4
p2p-Gnutella31	2,367	4,613	2,874	2,945	0.036	1.412	0.422	0.445	166.5
polblogs	3,567	3,502	11,357	8,796	0.007	0.036	0.151	0.122	207.9
soc-Epinions1	2,659	4,057	3,585	3,534	0.312	3.211	2.186	2.046	918.3
subelj-cora-cora	3,790	5,169	5,681	5,090	0.016	0.564	0.272	0.265	422.6
subelj-jdk-jdk	2,425	4,057	25,575	5,090	0.006	0.097	0.100	0.064	57.4
subelj-jung-j-jung-j	2,436	3,502	43,584	5,090	0.006	0.079	0.140	0.059	57.0
wiki-Vote	2,633	3,502	3,467	3,534	0.006	0.188	0.148	0.149	188.2

Empty values correspond to graphs on which the algorithm needed more than 1 hour.

Table 3. The Top-k Betweenness Centralities of the Wikipedia Graph Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Wikipedia page	Lower bound	Estimated betweenness	Upper bound
1)	United States	0.046278	0.047173	0.048084
2)	France	0.019522	0.020103	0.020701
3)	United Kingdom	0.017983	0.018540	0.019115
4)	England	0.016348	0.016879	0.017428
5-6)	Poland	0.012092	0.012287	0.012486
5-6)	Germany	0.011930	0.012124	0.012321
7)	India	0.009683	0.010092	0.010518
8-12)	World War II	0.008870	0.009065	0.009265
8-12)	Russia	0.008660	0.008854	0.009053
8-12)	Italy	0.008650	0.008845	0.009045
8-12)	Canada	0.008624	0.008819	0.009018
8-12)	Australia	0.008620	0.008814	0.009013

Table 4. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1939 (69,011 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Meyer, Torben	0.022331	0.022702	0.023049
2)	Roulien, Raul	0.021361	0.021703	0.022071
3)	Myzet, Rudolf	0.014229	0.014525	0.014747
4)	Sten, Anna	0.013245	0.013460	0.013723
5)	Negri, Pola	0.012509	0.012768	0.012943
6-7)	Jung, Shia	0.012250	0.012379	0.012509
6-7)	Ho, Tai-Hau	0.012195	0.012324	0.012454
8)	Goetzke, Bernhard	0.010721	0.010978	0.011201
9-10)	Yamamoto, Togo	0.010095	0.010224	0.010354
9-10)	Kamiyama, Sōjin	0.010087	0.010215	0.010344

10 WIKIPEDIA AND IMDB RESULTS

In this section, we report our results on the Wikipedia citation network, and on all snapshots of the IMDB actors collaboration network. In the ranking column, we report one number if the position in the ranking is guaranteed with probability 0.9, otherwise we report a lower and an upper bound, which hold with the same probability.

We remark that, as for the IMDB database, the top-k betweenness centralities of a single snapshot of a similar graph (`hollywood-2009` in [11]) have been previously computed exactly, with 1 week of computation on a 40-core machine [50].

10.1 The Results on the IMDB Graph

In 2014, the most central actor is Ron Jeremy, who is listed in the Guinness Book of World Records for "Most Appearances in Adult Films," with more than 2,000 appearances. Among his non-adult ones, we mention *The Godfather Part III*, *Ghostbusters*, *Crank: High Voltage*, and *Family Guy*.[5] His

[5]The latter is a TV series, which are not taken into account in our data.

Table 5. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1944 (83,068 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Meyer, Torben	0.018320	0.018724	0.019136
2)	Kamiyama, Sōjin	0.012629	0.012964	0.013308
3-4)	Jung, Shia	0.010751	0.010901	0.011053
3-4)	Ho, Tai-Hau	0.010704	0.010854	0.011005
5)	Myzet, Rudolf	0.010365	0.010514	0.010666
6-7)	Sten, Anna	0.009778	0.009928	0.010080
6-7)	Goetzke, Bernhard	0.009766	0.009915	0.010066
8)	Yamamoto, Togo	0.009108	0.009327	0.009539
9)	Parìs, Manuel	0.008649	0.008859	0.009108
10)	Hayakawa, Sessue	0.007916	0.008158	0.008369

Table 6. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1949 (97,824 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Meyer, Torben	0.016139	0.016679	0.017236
2)	Kamiyama, Sōjin	0.012351	0.012822	0.013312
3)	Parìs, Manuel	0.011104	0.011552	0.011861
4)	Yamamoto, Togo	0.010342	0.010639	0.011086
5-6)	Jung, Shia	0.008926	0.009120	0.009318
5-6)	Goetzke, Bernhard	0.008567	0.008762	0.008962
7-9)	Paananen, Tuulikki	0.008147	0.008341	0.008539
7-9)	Sten, Anna	0.007969	0.008164	0.008363
7-9)	Mayer, Ruby	0.007967	0.008162	0.008362
10-12)	Ho, Tai-Hau	0.007538	0.007732	0.007930
10-12)	Hayakawa, Sessue	0.007399	0.007593	0.007792
10-12)	Haas, Hugo (I)	0.007158	0.007352	0.007552

Table 7. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1954 (120,430 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Meyer, Torben	0.013418	0.013868	0.014334
2)	Kamiyama, Sōjin	0.010331	0.010726	0.011089
3-4)	Ertugrul, Muhsin	0.009956	0.010141	0.010331
3-4)	Jung, Shia	0.009643	0.009826	0.010013
5-6)	Singh, Ram (I)	0.008657	0.008841	0.009030
5-6)	Paananen, Tuulikki	0.008383	0.008567	0.008755
7-9)	Parìs, Manuel	0.007886	0.008070	0.008257
7-10)	Goetzke, Bernhard	0.007802	0.007987	0.008176
7-10)	Yamaguchi, Shirley	0.007531	0.007716	0.007905
8-10)	Hayakawa, Sessue	0.007473	0.007657	0.007845

Table 8. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1959 (146253 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1-2)	Singh, Ram (I)	0.010683	0.010877	0.011075
1-2)	Frees, Paul	0.010372	0.010566	0.010763
3)	Meyer, Torben	0.009478	0.009821	0.010235
4-5)	Jung, Shia	0.008623	0.008816	0.009013
4-5)	Ghosh, Sachin	0.008459	0.008651	0.008847
6-7)	Myzet, Rudolf	0.007085	0.007278	0.007476
6-7)	Yamaguchi, Shirley	0.006908	0.007101	0.007299
8)	de Còrdova, Arturo	0.006391	0.006582	0.006778
9-11)	Kamiyama, Sōjin	0.005861	0.006054	0.006254
9-12)	Paananen, Tuulikki	0.005810	0.006003	0.006202
9-12)	Flowers, Bess	0.005620	0.005813	0.006012
10-12)	Parìs, Manuel	0.005442	0.005635	0.005835

Table 9. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1964 (174,826 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Frees, Paul	0.013140	0.013596	0.014067
2)	Meyer, Torben	0.007279	0.007617	0.007856
3-4)	Harris, Sam (II)	0.006813	0.006967	0.007124
3-5)	Myzet, Rudolf	0.006696	0.006849	0.007005
4-5)	Flowers, Bess	0.006422	0.006572	0.006726
6)	Kong, King (I)	0.005909	0.006104	0.006422
7)	Yuen, Siu Tin	0.005114	0.005264	0.005420
8)	Miller, Marvin (I)	0.004708	0.004859	0.005015
9-12)	de Còrdova, Arturo	0.004147	0.004299	0.004457
9-18)	Haas, Hugo (I)	0.003888	0.004039	0.004197
9-18)	Singh, Ram (I)	0.003854	0.004004	0.004160
9-18)	Kamiyama, Sōjin	0.003848	0.003999	0.004155
10-18)	Sauli, Anneli	0.003827	0.003978	0.004135
10-18)	King, Walter Woolf	0.003774	0.003923	0.004078
10-18)	Vanel, Charles	0.003716	0.003867	0.004024
10-18)	Kowall, Mitchell	0.003684	0.003834	0.003990
10-18)	Holmes, Stuart	0.003603	0.003752	0.003907
10-18)	Sten, Anna	0.003582	0.003733	0.003890

topmost centrality in the actor collaboration network has been previously observed by similar experiments on betweenness centrality [50]. Indeed, around 3 actors out of 100 in the IMDB database played in adult movies, which explains why the high number of appearances of Ron Jeremy both in the adult and non-adult film industry rises his betweenness to the top.

The second most central actor is Lloyd Kaufman, who is best known as a co-founder of *Troma Entertainment Film Studio* and as the director of many of their feature films, including the cult

Table 10. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1969 (210,527 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Frees, Paul	0.010913	0.011446	0.012005
2-3)	Yuen, Siu Tin	0.006157	0.006349	0.006547
2-3)	Tamiroff, Akim	0.006097	0.006291	0.006490
4-6)	Meyer, Torben	0.005675	0.005869	0.006069
4-7)	Harris, Sam (II)	0.005639	0.005830	0.006027
4-8)	Rubener, Sujata	0.005427	0.005618	0.005815
5-8)	Myzet, Rudolf	0.005253	0.005444	0.005641
6-8)	Flowers, Bess	0.005136	0.005328	0.005526
9-10)	Kong, King (I)	0.004354	0.004544	0.004741
9-10)	Sullivan, Elliott	0.004208	0.004398	0.004596

Table 11. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1974 (257,896 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Frees, Paul	0.008507	0.008958	0.009295
2)	Chen, Sing	0.007734	0.008056	0.008507
3)	Welles, Orson	0.006115	0.006497	0.006903
4-5)	Loren, Sophia	0.005056	0.005221	0.005392
4-7)	Rubener, Sujata	0.004767	0.004933	0.005106
5-8)	Harris, Sam (II)	0.004628	0.004795	0.004967
5-8)	Tamiroff, Akim	0.004625	0.004790	0.004962
6-10)	Meyer, Torben	0.004382	0.004548	0.004720
8-12)	Flowers, Bess	0.004259	0.004425	0.004598
8-12)	Yuen, Siu Tin	0.004229	0.004397	0.004571
9-12)	Carradine, John	0.004026	0.004192	0.004364
9-12)	Myzet, Rudolf	0.003984	0.004151	0.004325

movie *The Toxic Avenger*. His high betweenness score is likely due to his central role in the low-budget independent film industry.

The third "actor" is the historical German dictator Adolf Hitler, since his appearances in several historical footages, which were re-used in several movies (e.g., in *The Imitation Game*), are credited by IMDB as cameo roles. Indeed, he appears among the topmost actors since the 1984 snapshot, being the first one in the 1989 and 1994 ones, and during those years many movies about World War II were produced.

Observe that the betweenness centrality measure on our graph does not discriminate between important and marginal roles. For example, the actress Bess Flowers, who appears among the top actors in the snapshots from 1959 to 1979, rarely played major roles, but she appeared in over 700 movies in her 41-year career.

10.2 The Results on the Wikipedia Graph

All topmost pages in the betweenness centrality ranking, except for the World War II, are countries. This is not surprising if we consider that, for most topics (such as important people or events), the

Table 12. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1979 (310,278 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Chen, Sing	0.007737	0.008220	0.008647
2)	Frees, Paul	0.006852	0.007255	0.007737
3-5)	Welles, Orson	0.004894	0.005075	0.005263
3-6)	Carradine, John	0.004623	0.004803	0.004989
3-6)	Loren, Sophia	0.004614	0.004796	0.004985
4-6)	Rubener, Sujata	0.004284	0.004464	0.004651
7-17)	Tamiroff, Akim	0.003516	0.003696	0.003885
7-17)	Meyer, Torben	0.003479	0.003657	0.003844
7-17)	Quinn, Anthony (I)	0.003447	0.003626	0.003815
7-17)	Flowers, Bess	0.003446	0.003625	0.003815
7-17)	Mitchell, Gordon (I)	0.003417	0.003596	0.003785
7-17)	Sullivan, Elliott	0.003371	0.003551	0.003740
7-17)	Rietty, Robert	0.003368	0.003547	0.003735
7-17)	Tanba, Tetsurō	0.003360	0.003537	0.003724
7-17)	Harris, Sam (II)	0.003331	0.003510	0.003699
7-17)	Lewgoy, Josè	0.003223	0.003402	0.003590
7-17)	Dalio, Marcel	0.003185	0.003364	0.003553

Table 13. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1984 (375,322 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Chen, Sing	0.007245	0.007716	0.008218
2-4)	Welles, Orson	0.005202	0.005391	0.005587
2-4)	Frees, Paul	0.005174	0.005363	0.005559
2-5)	Hitler, Adolf	0.004906	0.005094	0.005290
4-6)	Carradine, John	0.004744	0.004932	0.005127
5-7)	Mitchell, Gordon (I)	0.004418	0.004606	0.004802
6-8)	Jürgens, Curd	0.004169	0.004356	0.004551
7-8)	Kinski, Klaus	0.003938	0.004123	0.004318
9-12)	Rubener, Sujata	0.003396	0.003585	0.003785
9-12)	Lee, Christopher (I)	0.003391	0.003576	0.003771
9-12)	Loren, Sophia	0.003357	0.003542	0.003738
9-12)	Harrison, Richard (II)	0.003230	0.003417	0.003614

corresponding Wikipedia page refers to their geographical context (since it mentions the country of origin of the given person or where a given event took place). It is also worth noting the correlation between the high centrality of the *World War II* Wikipedia page and that of Adolf Hitler in the IMDB graph.

Interestingly, a similar ranking is obtained by considering the closeness centrality measure in the inverse graph, where a link from page p_1 to page p_2 exists if a link to page p_1 appears in page p_2 [8]. However, in contrast with the results in [8] when edges are oriented in the usual way, the pages

Table 14. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1989 (463,078 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1-2)	Hitler, Adolf	0.005282	0.005467	0.005658
1-3)	Chen, Sing	0.005008	0.005192	0.005382
2-4)	Carradine, John	0.004648	0.004834	0.005027
3-4)	Harrison, Richard (II)	0.004515	0.004697	0.004887
5-6)	Welles, Orson	0.004088	0.004271	0.004462
5-9)	Mitchell, Gordon (I)	0.003766	0.003948	0.004139
6-9)	Kinski, Klaus	0.003691	0.003874	0.004065
6-11)	Lee, Christopher (I)	0.003610	0.003793	0.003984
6-11)	Frees, Paul	0.003582	0.003766	0.003960
8-13)	Jürgens, Curd	0.003306	0.003486	0.003676
8-13)	Pleasence, Donald	0.003299	0.003479	0.003670
10-13)	Mitchell, Cameron (I)	0.003105	0.003285	0.003476
10-13)	von Sydow, Max (I)	0.002982	0.003161	0.003350

Table 15. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1994 (557,373 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Hitler, Adolf	0.005227	0.005676	0.006164
2-6)	Harrison, Richard (II)	0.003978	0.004165	0.004362
2-6)	von Sydow, Max (I)	0.003884	0.004069	0.004264
2-7)	Lee, Christopher (I)	0.003718	0.003907	0.004106
2-7)	Carradine, John	0.003696	0.003883	0.004079
2-7)	Chen, Sing	0.003683	0.003871	0.004068
4-10)	Jeremy, Ron	0.003336	0.003524	0.003722
7-11)	Pleasence, Donald	0.003253	0.003439	0.003637
7-11)	Rey, Fernando (I)	0.003234	0.003420	0.003617
7-15)	Smith, William (I)	0.003012	0.003199	0.003397
8-15)	Welles, Orson	0.002885	0.003072	0.003271
10-15)	Mitchell, Gordon (I)	0.002851	0.003036	0.003232
10-15)	Kinski, Klaus	0.002705	0.002890	0.003087
10-15)	Mitchell, Cameron (I)	0.002671	0.002858	0.003058
10-15)	Quinn, Anthony (I)	0.002640	0.002826	0.003026

about specific years do not appear in the top ranking. We note that the betweenness centrality of a node in a directed graph does not change if the orientation of all edges is flipped.

Finally, the most important pages are the United States, confirming a common conjecture. Indeed, in http://wikirank.di.unimi.it/, it is shown that the United States are the center according to harmonic centrality, and many other measures. Further evidence for this conjecture comes from the Six Degree of Wikipedia game (http://thewikigame.com/6-degrees-of-wikipedia), where a player is asked to go from one page to the other following the smallest possible number of links: a hard variant of this game forces the player not to pass from the *United States* page, which is

Table 16. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 1999 (681,358 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Jeremy, Ron	0.007380	0.007913	0.008484
2)	Hitler, Adolf	0.004601	0.005021	0.005480
3-4)	Lee, Christopher (I)	0.003679	0.003849	0.004028
3-4)	von Sydow, Max (I)	0.003604	0.003775	0.003953
5-6)	Harrison, Richard (II)	0.003041	0.003211	0.003390
5-7)	Carradine, John	0.002943	0.003114	0.003296
6-11)	Chen, Sing	0.002662	0.002834	0.003018
7-14)	Rey, Fernando (I)	0.002569	0.002740	0.002922
7-14)	Smith, William (I)	0.002559	0.002729	0.002910
7-14)	Pleasence, Donald	0.002556	0.002725	0.002906
7-14)	Sutherland, Donald (I)	0.002449	0.002617	0.002796
8-14)	Quinn, Anthony (I)	0.002307	0.002476	0.002658
8-14)	Mastroianni, Marcello	0.002271	0.002440	0.002621
8-14)	Saxon, John	0.002251	0.002420	0.002602

Table 17. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 2004 (880,032 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Jeremy, Ron	0.010653	0.011370	0.012136
2)	Hitler, Adolf	0.005333	0.005840	0.006396
3-4)	von Sydow, Max (I)	0.003424	0.003608	0.003802
3-4)	Lee, Christopher (I)	0.003403	0.003587	0.003781
5-6)	Kier, Udo	0.002898	0.003081	0.003275
5-8)	Keitel, Harvey (I)	0.002646	0.002828	0.003023
6-12)	Hopper, Dennis	0.002424	0.002607	0.002804
6-16)	Smith, William (I)	0.002322	0.002504	0.002700
7-17)	Sutherland, Donald (I)	0.002241	0.002422	0.002617
7-23)	Carradine, David	0.002149	0.002329	0.002526
7-23)	Carradine, John	0.002147	0.002328	0.002524
7-23)	Harrison, Richard (II)	0.002054	0.002234	0.002430
8-23)	Sharif, Omar	0.002043	0.002222	0.002418
8-23)	Steiger, Rod	0.001988	0.002165	0.002358
8-23)	Quinn, Anthony (I)	0.001974	0.002151	0.002344
8-23)	Depardieu, Gèrard	0.001966	0.002148	0.002346
9-23)	Sheen, Martin	0.001913	0.002093	0.002291
10-23)	Rey, Fernando (I)	0.001866	0.002044	0.002238
10-23)	Kane, Sharon	0.001857	0.002038	0.002237
10-23)	Pleasence, Donald	0.001859	0.002037	0.002232
10-23)	Skarsgård, Stellan	0.001848	0.002026	0.002221
10-23)	Mueller-Stahl, Armin	0.001789	0.001969	0.002166
10-23)	Hong, James (I)	0.001780	0.001957	0.002152

Table 18. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken at the End of 2009 (1,237,879 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Jeremy, Ron	0.010531	0.011237	0.011991
2)	Hitler, Adolf	0.005500	0.006011	0.006568
3-4)	Kaufman, Lloyd	0.003620	0.003804	0.003997
3-4)	Kier, Udo	0.003472	0.003654	0.003845
5-6)	Lee, Christopher (I)	0.003056	0.003240	0.003435
5-8)	Carradine, David	0.002866	0.003050	0.003245
6-8)	Keitel, Harvey (I)	0.002659	0.002840	0.003034
6-9)	von Sydow, Max (I)	0.002532	0.002713	0.002907
8-13)	Hopper, Dennis	0.002237	0.002419	0.002616
9-15)	Skarsgård, Stellan	0.002153	0.002333	0.002529
9-15)	Depardieu, Gèrard	0.002001	0.002181	0.002377
9-15)	Hauer, Rutger	0.001894	0.002074	0.002271
9-15)	Sutherland, Donald (I)	0.001875	0.002054	0.002250
10-15)	Smith, William (I)	0.001811	0.001990	0.002186
10-15)	Dafoe, Willem	0.001805	0.001986	0.002186

Table 19. The Top-k Betweenness Centralities of a Snapshot of the IMDB Collaboration Network Taken in 2014 (1,797,446 Nodes), Computed by KADABRA with $\delta = 0.1$ and $\lambda = 0.0002$

Ranking	Actor	Lower bound	Estimated betweenness	Upper bound
1)	Jeremy, Ron	0.009360	0.010058	0.010808
2)	Kaufman, Lloyd	0.005936	0.006492	0.007100
3)	Hitler, Adolf	0.004368	0.004844	0.005373
4-6)	Kier, Udo	0.003250	0.003435	0.003631
4-6)	Roberts, Eric (I)	0.003178	0.003362	0.003557
4-6)	Madsen, Michael (I)	0.003120	0.003305	0.003501
7-9)	Trejo, Danny	0.002652	0.002835	0.003030
7-9)	Lee, Christopher (I)	0.002551	0.002734	0.002931
7-12)	Estevez, Joe	0.002350	0.002534	0.002732
9-17)	Carradine, David	0.002116	0.002296	0.002492
9-17)	von Sydow, Max (I)	0.002023	0.002206	0.002405
9-17)	Keitel, Harvey (I)	0.001974	0.002154	0.002352
10-17)	Skarsgård, Stellan	0.001945	0.002125	0.002323
10-17)	Dafoe, Willem	0.001899	0.002080	0.002279
10-17)	Hauer, Rutger	0.001891	0.002071	0.002269
10-17)	Depardieu, Gèrard	0.001763	0.001943	0.002142
10-17)	Rochon, Debbie	0.001745	0.001926	0.002126

considered to be central. Our results thus confirm that the conjecture is indeed true for the betweenness centrality measure.

ACKNOWLEDGMENTS

The authors would like to thank Matteo Riondato for several constructive comments on an earlier version of this work. We also thank Elisabetta Bergamini, Richard Lipton, and Sebastiano Vigna

for helpful discussions and Holger Dell for his help with the experiments. Finally, we thank the anonymous reviewers for their feedback on an earlier version of this article.

REFERENCES

[1] CAMVIT: Choice Routing. 2009. Retrieved on April 21, 2016 from http://www.camvit.com.
[2] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. 2009. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim (Eds.). Lecture Notes in Computer Science, Vol. 5687. Springer, Berlin.
[3] Jac M. Anthonisse. 1971. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde* BN, 9/71 (1971), 1–10.
[4] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. 2007. Approximating betweenness centrality. *The 5th Workshop on Algorithms and Models for the Web-Graph*.
[5] Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. 2011. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, Lecture Notes in Computer Science. Springer, Berlin, 21–32. DOI: https://doi.org/10.1007/978-3-642-19754-3_5
[6] Alex Bavelas. 1948. A mathematical model for group structures. *Human Organization* 7, 3 (1948), 16–30.
[7] Elisabetta Bergamini. 2016. *Private communication*.
[8] Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. 2016. Computing top-k closeness centrality faster in unweighted graphs. In *ALENEX*.
[9] Elisabetta Bergamini and Henning Meyerhenke. 2015. Fully-dynamic approximation of betweenness centrality. In *ESA*.
[10] Elisabetta Bergamini, Henning Meyerhenke, Mark Ortmann, and Arie Slobbe. 2017. Faster betweenness centrality updates in evolving networks. *arXiv preprint arXiv:1704.08592* (2017).
[11] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. 2014. BUbiNG: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*. 227–228.
[12] Béla Bollobás. 1980. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics* 1, 4 (1980), 311–316. DOI: https://doi.org/10.1016/S0195-6698(80)80030-8
[13] Béla Bollobás, Svante Janson, and Oliver Riordan. 2007. The phase transition in inhomogeneous random graphs. *Random Structures and Algorithms* 31, 1 (2007), 3–122.
[14] Michele Borassi. 2016. *Algorithms for Metric Properties of Large Real-world Networks from Theory to Practice and Back*. Electronic thesis or dissertation. http://e-theses.imtlucca.it/198/.
[15] Michele Borassi, Pierluig Crescenzi, and Michel Habib. 2015. Into the square - On the complexity of some quadratic-time solvable problems. In *Proceedings of the 16th Italian Conference on Theoretical Computer Science (ICTCS'15)*. 1–17.
[16] Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. 2016. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. *arXiv:1604.01445 [cs]* (April 2016). arXiv: 1604.01445.
[17] Stephen P. Borgatti and Martin G. Everett. 2006. A graph-theoretic perspective on centrality. *Social Networks* 28 (2006), 466–484.
[18] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology* 25, 2 (June 2001), 163–177. DOI: https://doi.org/10.1080/0022250X.2001.9990249
[19] Ulrik Brandes. 2008. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks* 30 (2008), 136–145.
[20] Ulrik Brandes and Christian Pich. 2007. Centrality estimation in large networks. *International Journal of Bifurcation and Chaos* 17, 07 (2007), 2303–2318. DOI: https://doi.org/10.1142/S0218127407018403
[21] Mostafa Haghir Chehreghani, Talel Abdessalem, et al. 2017. Metropolis-hastings algorithms for estimating betweenness centrality in large networks. *arXiv preprint arXiv:1704.07351* .
[22] Bernard S. Cohn and McKim Marriott. 1958. Networks and centres of integration in Indian civilization. *Journal of Social Research* 1, 1 (1958), 1–9.
[23] Shlomi Dolev, Yuval Elovici, and Rami Puzis. 2010. Routing betweenness centrality. *Journal of the ACM* 57 (2010).
[24] David A. Easley and Jon M. Kleinberg. 2010. Networks, crowds, and markets - Reasoning about a highly connected world. In *DAGLIB*.
[25] David Eppstein and Joseph Wang. 2001. Fast approximation of centrality. *Journal of Graph Algorithms and Applications* 8 (2001), 39–45.
[26] Dóra Erdös, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. 2015. A divide-and-conquer algorithm for betweenness centrality. *CoRR* abs/1406.4173.
[27] Daniel Fernholz and Vijaya Ramachandran. 2007. The diameter of sparse random graphs. *Random Structures and Algorithms* 31, 4 (2007), 482–516. DOI: https://doi.org/10.1002/rsa

[28] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*.
[29] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. 2001. Which problems have strongly exponential complexity? *J. Comput. System Sci.* 63, 4 (Dec. 2001), 512–530. DOI: https://doi.org/10.1006/jcss.2001.1774
[30] Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. 2004. Algorithms for centrality indices. In *DAGSTUHL*.
[31] Hermann Kaindl and Gerhard Kainz. 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research (JAIR)* 7 (1997), 283–317.
[32] Yeon-sup Lim, Daniel S. Menasché, Bruno Ribeiro, Don Towsley, and Prithwish Basu. 2011. Online estimating the k central nodes of a network. *Proceedings of IEEE NSW* (2011), 118–122.
[33] Richard J. Lipton and Jeffrey F. Naughton. 1989. Estimating the size of generalized transitive closures. In *Proceedings of the 15th erence on Very Large Data Bases*.
[34] R. J. Lipton and J. F. Naughton. 1995. Query size estimation by adaptive sampling. *Journal of Computer and System Sciences* 51, 1 (Aug. 1995), 18–25. DOI: https://doi.org/10.1006/jcss.1995.1050
[35] Linyuan Lu and Fan R. K. Chung. 2006. *Complex Graphs and Networks*. Number 107 in CBMS Regional Conference Series in Mathematics. American Mathematical Society.
[36] Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press, Oxford.
[37] Mark E. J. Newman. 2001. Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical Review E* 64, 1 (2001), 016132.
[38] Ilkka Norros and Hannu Reittu. 2006. On a conditionally Poissonian graph process. *Advances in Applied Probability* 38, 1 (2006), 59–75.
[39] Jürgen Pfeffer and Kathleen M. Carley. 2012. k-centralities: Local approximations of global measures based on shortest paths. In *Proceedings of the 21st International Conference Companion on World Wide Web*. ACM, 1043–1050.
[40] Andrea Pietracaprina, Matteo Riondato, Eli Upfal, and Fabio Vandin. 2010. Mining top-K frequent itemsets through progressive sampling. *Data Mining and Knowledge Discovery* 21, 2 (Sept. 2010), 310–326. DOI: https://doi.org/10.1007/s10618-010-0185-7
[41] Ira Pohl. 1969. *Bi-directional and Heuristic Search in Path Problems*. Ph.D. dissertation. Department of Computer Science, Stanford University.
[42] Matteo Riondato and Evgenios M. Kornaropoulos. 2015. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2015), 438–475.
[43] Matteo Riondato and Eli Upfal. 2016. ABRA: Approximating betweenness centrality in static and dynamic graphs with rademacher averages. *arXiv preprint 1602.05866* (2016), 1–27. arxiv:1602.05866
[44] Marvin E. Shaw. 1954. Group structure and the behavior of individuals in small groups. *The Journal of Psychology* 38, 1 (1954), 139–149.
[45] Alfonso Shimbel. 1953. Structural parameters of communication networks. *The Bulletin of Mathematical Biophysics* 15, 4 (1953), 501–507.
[46] Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. 2014. Networkit: An interactive tool suite for high-performance network analysis. *arXiv preprint 1403.3005* (2014), 1–25.
[47] Ümit V. Çatalyürek, Kamer Kaya, Ahmet Erdem Sariyüce, and Erik Saule. 2013. Shattering and compressing networks for betweenness centrality. In *SDM*.
[48] Remco van der Hofstad. 2014. *Random Graphs and Complex Networks*. Vol. II.
[49] Flavio Vella, Giancarlo Carbone, and Massimo Bernaschi. 2016. Algorithms and heuristics for scalable betweenness centrality computation on multi-GPU systems. *CoRR* abs/1602.00963 (2016).
[50] Sebastiano Vigna. 2016. *Private communication*.
[51] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Vol. 8. Cambridge University Press.

Received May 2017; revised June 2018; accepted September 2018

TRIÈST: Counting Local and Global Triangles in Fully Dynamic Streams with Fixed Memory Size

LORENZO DE STEFANI, Brown University
ALESSANDRO EPASTO, Google Inc.
MATTEO RIONDATO, Two Sigma Investments LP
ELI UPFAL, Brown University

"Ogni lassada xe persa."[1]– Proverb from Trieste, Italy.

We present TRIÈST, a suite of one-pass streaming algorithms to compute unbiased, low-variance, high-quality approximations of the global and local (i.e., incident to each vertex) number of triangles in a fully dynamic graph represented as an adversarial stream of edge insertions and deletions.

Our algorithms use reservoir sampling and its variants to exploit the user-specified memory space at all times. This is in contrast with previous approaches, which require hard-to-choose parameters (e.g., a fixed sampling probability) and offer no guarantees on the amount of memory they use. We analyze the variance of the estimations and show novel concentration bounds for these quantities.

Our experimental results on very large graphs demonstrate that TRIÈST outperforms state-of-the-art approaches in accuracy and exhibits a small update time.

CCS Concepts: • **Mathematics of computing** → **Graph enumeration**; **Probabilistic algorithms**; • **Information systems** → **Data stream mining**; • **Human-centered computing** → **Social networks**; • **Theory of computation** → **Dynamic graph algorithms**; **Sketching and sampling;**

Additional Key Words and Phrases: Cycle counting, reservoir sampling, subgraph counting

ACM Reference Format:
Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. TRIÈST: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Trans. Knowl. Discov. Data 11, 4, Article 43 (June 2017), 50 pages.
DOI: http://dx.doi.org/10.1145/3059194

1. INTRODUCTION

Exact computation of characteristic quantities of Web-scale networks is often impractical or even infeasible due to the humongous size of these graphs. It is natural in these

[1]Any missed chance is lost forever.

This work was supported in part by the National Science Foundation grant IIS-1247581 (https://www.nsf.gov/awardsearch/showAward?AWD_ID=1247581) and the National Institutes of Health grant R01-CA180776 (https://projectreporter.nih.gov/project_info_details.cfm?icde=0&aid=8685211).
This work is supported, in part, by funding from Two Sigma Investments, LP. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflects the views of Two Sigma Investments, LP or the National Science Foundation. A preliminary report of this work appeared in the proceedings of ACM KDD'16 as [11].
Authors' addresses: L. De Stefani and E. Upfal, Department of Computer Science, Brown University, 115 Waterman St, Providence, RI 02912; emails: {lorenzo, eli}@cs.brown.edu; A. Epasto, Google Inc., 111 8th Avenue, New York, NY 10011; email: aepasto@google.com; M. Riondato, Two Sigma Investments LP, 100 Avenue of the Americas, New York, NY 10013; email: matteo@twosigma.com.

ACM 1556-4681/2017/06-ART43 $15.00
DOI: http://dx.doi.org/10.1145/3059194

cases to resort to *efficient-to-compute approximations* of these quantities that, when of sufficiently high quality, can be used as proxies for the exact values.

In addition to being huge, many interesting networks are *fully dynamic* and can be represented as a *stream* whose elements are edges/nodes insertions and deletions that occur in an *arbitrary* (even adversarial) order. Characteristic quantities in these graphs are *intrinsically volatile*, hence there is limited added value in maintaining them exactly. The goal is rather to keep track, *at all times*, of a high-quality approximation of these quantities. For efficiency, the algorithms should *aim at exploiting the available memory space as much as possible* and they should *require only one pass over the stream*.

We introduce TRIÈST, a suite of *sampling-based, one-pass algorithms for adversarial fully dynamic streams to approximate the global number of triangles and the local number of triangles incident to each vertex*. Mining local and global triangles is a fundamental primitive with many applications (e.g., community detection [4], topic mining [13], spam/anomaly detection [3, 28], ego-networks mining [14], and protein interaction networks analysis [30]).

Many previous works on triangle estimation in streams also employ sampling (see Section 3), but they usually require the user to specify *in advance* an *edge sampling probability* p that is fixed for the entire stream. This approach presents several significant drawbacks. First, choosing a p that allows to obtain the desired approximation quality requires to know or guess a number of properties of the input (e.g., the size of the stream). Second, a fixed p implies that the sample size grows with the size of the stream, which is problematic when the stream size is not known in advance: If the user specifies a large p, the algorithm may run out of memory, while for a smaller p, it will provide a suboptimal estimation. Third, even assuming to be able to compute a p that ensures (in expectation) full use of the available space, the memory would be fully utilized only at the end of the stream, and the estimations computed throughout the execution would be suboptimal.

Contributions. We address all the above issues by taking a significant departure from the fixed-probability, independent edge sampling approach taken even by state-of-the-art methods [28]. Specifically.

- We introduce TRIÈST (*TRI*angle *E*stimation from *ST*reams), a suite of *one-pass streaming algorithms* to approximate, at each time instant, the global and local number of triangles in a *fully dynamic* graph stream (i.e., a sequence of edges additions and deletions in arbitrary order) using a *fixed amount of memory*. This is the first contribution that enjoys all these properties. TRIÈST only requires the user to specify *the amount of available memory*, an interpretable parameter that is definitively known to the user.
- Our algorithms maintain a sample of edges: They use the *reservoir sampling* [42] and *random pairing* (RP) [16] sampling schemes to exploit the available memory as much as possible. To the best of our knowledge, ours is the first application of these techniques to subgraph counting in fully dynamic, arbitrarily long, adversarially ordered streams. We present an analysis of the unbiasedness and of the variance of our estimators, and establish strong concentration results for them. The use of reservoir sampling and RP requires additional sophistication in the analysis, as the presence of an edge in the sample is *not independent* from the concurrent presence of another edge. Hence, in our proofs, we must consider the complex dependencies in events involving sets of edges. The gain is worth the effort: We prove that the variance of our algorithms is smaller than that of state-of-the-art methods [28], and this is confirmed by our experiments.

- We conduct an extensive experimental evaluation of TRIÈST on very large graphs, some with billions of edges, comparing the performances of our algorithms to those of existing state-of-the-art contributions [20, 28, 36]. *Our algorithms significantly and consistently reduce the average estimation error by up to* 90% w.r.t. the state of the art, both in the global and local estimation problems, while using the same amount of memory. Our algorithms are also extremely scalable, showing update times in the order of hundreds of microseconds for graphs with billions of edges.

Paper organization. We formally introduce the settings and the problem in Section 2. In Section 3, we discuss related works. We present and analyze TRIÈST and discuss our design choices in Section 4. The results of our experimental evaluation are presented in Section 5. We draw our conclusions in Section 6. Some of the proofs of our theoretical results are deferred to Appendix A.

2. PRELIMINARIES

We study the problem of counting global and local triangles in a fully dynamic undirected graph as an arbitrary (adversarial) stream of edge insertions and deletions.

Formally, for any (discrete) time instant $t \geq 0$, let $G^{(t)} = (V^{(t)}, E^{(t)})$ be the graph observed up to and including time t. At time $t = 0$, we have $V^{(t)} = E^{(t)} = \emptyset$. For any $t > 0$, at time $t+1$, we receive an element $e_{t+1} = (\bullet, (u, v))$ from a stream, where $\bullet \in \{+, -\}$ and u, v are two distinct vertices. The graph $G^{(t+1)} = (V^{(t+1)}, E^{(t+1)})$ is obtained by *inserting a new edge or deleting an existing edge* as follows:

$$E^{(t+1)} = \begin{cases} E^{(t)} \cup \{(u, v)\} \text{ if } \bullet = \text{“}+\text{”} \\ E^{(t)} \setminus \{(u, v)\} \text{ if } \bullet = \text{“}-\text{”} \end{cases}.$$

If u or v do not belong to $V^{(t)}$, they are added to $V^{(t+1)}$. Nodes are deleted from $V^{(t)}$ when they have degree zero.

Edges can be added and deleted in the graph in an arbitrary adversarial order, i.e., as to cause the worst outcome for the algorithm, but we assume that the adversary has *no access to the random bits* used by the algorithm. We assume that *all operations have effect*: If $e \in E^{(t)}$ (resp. $e \notin E^{(t)}$), $(+, e)$ (resp. $(-, e)$) cannot be on the stream at time $t+1$.

Given a graph $G^{(t)} = (V^{(t)}, E^{(t)})$, a *triangle* in $G^{(t)}$ is a *set* of three edges $\{(u, v), (v, w), (w, u)\} \subseteq E^{(t)}$, with u, v, and w being three distinct vertices. We refer to $\{u, v, w\} \subseteq V^{(t)}$ as the *corners* of the triangle. We denote with $\Delta^{(t)}$ the set of *all* triangles in $G^{(t)}$, and, for any vertex $u \in V^{(t)}$, with $\Delta_u^{(t)}$ the subset of $\Delta^{(t)}$ containing all and only the triangles that have u as a corner.

Problem definition. We study the *Global* (resp. *Local*) Triangle Counting Problem in Fully Dynamic Streams, which requires to compute, at each time $t \geq 0$, an estimation of $|\Delta^{(t)}|$ (resp. for each $u \in V$ an estimation of $|\Delta_u^{(t)}|$).

Multigraphs. Our approach can be further extended to count the number of global and local triangles on a *multigraph* represented as a stream of edges. Using a formalization analogous to that discussed for graphs, for any (discrete) time instant $t \geq 0$, let $G^{(t)} = (V^{(t)}, \mathcal{E}^{(t)})$ be the multigraph observed up to and including time t, where $\mathcal{E}^{(t)}$ is now a *bag* of edges between vertices of $V^{(t)}$. The multigraph evolves through a series of edge additions and deletions according to almost the same process described for graphs, with the exception that now all operations must have effect on the *bag* of edges $\mathcal{E}^{(t)}$. Thus, for example, we may have $(+, e)$ on the stream at time t and again $(+, e)$ at time $t+1$. Some additional modifications to the model are needed to handle deletions appropriately, and we outline them in Section 4.4.3. The definition of triangle in a multigraph is the same

Table I. Comparison with Previous Contributions

Work	Single pass	Fixed space	Local counts	Global counts	Fully dynamic streams
Becchetti et al. [3]	✗	✓/✗[a]	✓	✗	✗
Kolountzakis et al. [23]	✗	✗	✗	✓	✗
Pavan et al. [36]	✓	✓	✗	✓	✗
Jha et al. [20]	✓	✓	✗	✓	✗
Ahmed et al. [1]	✓	✗	✗	✓	✗
Lim and Kang [28]	✓	✗	✓	✗/✓[b]	✗
This work	✓	✓	✓	✓	✓

[a]The required space is $O(|V^{(t)}|)$, which, although not dependent on the number of triangles or on the number of edges, is not fixed, in the sense that it cannot be fixed *a priori*.
[b]Global triangle counting is not mentioned in the article, but the extension is straightforward.

as in a graph. As before we denote with $\Delta^{(t)}$ the set of *all* triangles in $G^{(t)}$, but now this set may contain multiple triangles with the same set of vertices, although each of these triangles will be a different set of three edges among those vertices, i.e., a different subset of the bag $\mathcal{E}^{(t)}$. For any vertex $u \in V^{(t)}$, we still denote with $\Delta_u^{(t)}$ the subset of $\Delta^{(t)}$ containing all and only the triangles that have u as a corner, with a similar caveat as $\Delta^{(t)}$. The problems of global and local triangle counting in multigraph edge streams are defined exactly in the same way as for graph edge streams.

3. RELATED WORK

The literature on exact and approximate triangle counting is extremely rich, including exact algorithms, graph sparsifiers [40, 41], complex-valued sketches [22, 29], and MapReduce algorithms [32–35, 38]. Here, we restrict the discussion to the works most related to ours, i.e., to those presenting algorithms for counting or approximating the number of triangles from data streams. We refer to the survey by [26] for an in-depth discussion of other works. Table I presents a summary of the comparison, in terms of desirable properties, between this work and relevant previous contributions.

Many authors presented algorithms for more restricted (i.e., less generic) settings than ours, or for which the constraints on the computation are more lax [2, 7, 21, 24]. For example, Becchetti et al. [3] and Kolountzakis et al. [23] present algorithms for approximate triangle counting from *static* graphs by performing multiple passes over the data. Pavan et al. [36] and Jha et al. [20] propose algorithms for approximating only the global number of triangles from *edge-insertion-only* streams. Bulteau et al. [6] present a one-pass algorithm for fully dynamic graphs, but the triangle count estimation is (expensively) computed only at the end of the stream and the algorithm requires, in the worst case, more memory than what is needed to store the entire graph. Ahmed et al. [1] apply the sampling-and-hold approach to insertion-only graph stream mining to obtain, only at the end of the stream and using non-constant space, an estimation of many network measures including triangles.

None of these works has *all* the features offered by TRIÈST: performs a single pass over the data, handles fully dynamic streams, uses a fixed amount of memory space, requires a single interpretable parameter, and returns an estimation at each time instant. Furthermore, our experimental results show that we outperform the algorithms from Jha et al. and Pavan et al. [20, 36] on insertion-only streams.

Lim and Kang [28] present an algorithm for insertion-only streams that is based on independent edge sampling with a fixed probability: For each edge on the stream, a coin with a user-specified fixed tails probability p is flipped, and, if the outcome is tails, the edge is added to the stored sample and the estimation of local triangles is updated. Since the memory is not fully utilized during most of the stream, the variance of the estimate is large. Our approach handles fully dynamic streams and makes better use

of the available memory space at each time instant, resulting in a better estimation, as shown by our analytical and experimental results.

Vitter [42] presents a detailed analysis of the reservoir sampling scheme and discusses methods to speed up the algorithm by reducing the number of calls to the random number generator. RP [16] is an extension of reservoir sampling to handle fully dynamic streams with insertions and deletions. Cohen et al. [9] generalize and extend the RP approach to the case where the elements on the stream are key-value pairs, where the value may be negative (and less than -1). In our settings, where the value is not less than -1 (for an edge deletion), these generalizations do not apply and the algorithm presented by Cohen et al. [9] reduces essentially to RP.

4. ALGORITHMS

We present TRIÈST, a suite of three novel algorithms for approximate global and local triangle counting from edge streams. The first two work on insertion-only streams, while the third can handle fully dynamic streams where edge deletions are allowed. We defer the discussion of the multigraph case to Section 4.4.

Parameters. Our algorithms keep an edge sample $\mathcal{S}$ containing up to M edges from the stream, where M is a positive integer parameter. For ease of presentation, we realistically assume $M \geq 6$. In Section 1, we motivated the design choice of only requiring M as a parameter and remarked on its advantages over using a fixed sampling probability p. Our algorithms are designed to use the available space as much as possible.

Counters. TRIÈST algorithms keep *counters* to compute the estimations of the global and local number of triangles. They *always* keep one global counter τ for the estimation of the global number of triangles. Only the global counter is needed to estimate the total triangle count. To estimate the local triangle counts, the algorithms keep a set of local counters τ_u for a subset of the nodes $u \in V^{(t)}$. The local counters are created on the fly as needed, and *always* destroyed as soon as they have a value of 0. Hence, our algorithms use $O(M)$ space (with one exception, see Section 4.2).

Notation. For any $t \geq 0$, let $G^{\mathcal{S}} = (V^{\mathcal{S}}, E^{\mathcal{S}})$ be the subgraph of $G^{(t)}$ containing all and only the edges in the current sample $\mathcal{S}$. We denote with $\mathcal{N}_u^{\mathcal{S}}$ the *neighborhood* of u in $G^{\mathcal{S}}$: $\mathcal{N}_u^{\mathcal{S}} = \{v \in V^{(t)} \ : \ (u, v) \in \mathcal{S}\}$ and with $\mathcal{N}_{u,v}^{\mathcal{S}} = \mathcal{N}_u^{\mathcal{S}} \cap \mathcal{N}_v^{\mathcal{S}}$ the *shared neighborhood* of u and v in $G^{\mathcal{S}}$.

Presentation. We only present the analysis of our algorithms for the problem of *global* triangle counting. For each presented result involving the estimation of the global triangle count (e.g., unbiasedness, bound on variance, concentration bound) and potentially using other global quantities (e.g., the number of pairs of triangles in $\Delta^{(t)}$ sharing an edge), it is straightforward to derive the correspondent variant for the estimation of the local triangle count, using similarly defined local quantities (e.g., the number of pairs of triangles in $\Delta_u^{(t)}$ sharing an edge.)

4.1. A First Algorithm – TRIÈST-BASE

We first present TRIÈST-BASE, which works on insertion-only streams and uses standard reservoir sampling [42] to maintain the edge sample $\mathcal{S}$:

- If $t \leq M$, then the edge $e_t = (u, v)$ on the stream at time t is deterministically inserted in $\mathcal{S}$.
- If $t > M$, TRIÈST-BASE flips a biased coin with heads probability M/t. If the outcome is heads, it chooses an edge $(w, z) \in \mathcal{S}$ uniformly at random, removes (w, z) from $\mathcal{S}$, and inserts (u, v) in $\mathcal{S}$. Otherwise, $\mathcal{S}$ is not modified.

After each insertion (resp. removal) of an edge (u, v) from $\mathcal{S}$, TRIÈST-BASE calls the procedure UPDATECOUNTERS that increments (resp. decrements) τ, τ_u and τ_v by $|\mathcal{N}^{\mathcal{S}}_{u,v}|$, and τ_c by one, for each $c \in \mathcal{N}^{\mathcal{S}}_{u,v}$.

The pseudocode for TRIÈST-BASE is presented in Algorithm 1.

ALGORITHM 1: TRIÈST-BASE

```
    Input: Insertion-only edge stream Σ, integer M ≥ 6
 1: S ← ∅, t ← 0, τ ← 0
 2: for each element (+, (u, v)) from Σ do
 3:     t ← t + 1
 4:     if SampleEdge((u, v), t) then
 5:         S ← S ∪ {(u, v)}
 6:         UpdateCounters(+, (u, v))

 7: function SampleEdge((u, v), t)
 8:     if t ≤ M then
 9:         return True
10:     else if FlipBiasedCoin(M/t) = heads then
11:         (u′, v′) ← random edge from S
12:         S ← S \ {(u′, v′)}
13:         UpdateCounters(−, (u′, v′))
14:         return True
15:     return False

16: function UpdateCounters((•, (u, v)))
17:     N^S_{u,v} ← N^S_u ∩ N^S_v
18:     for all c ∈ N^S_{u,v} do
19:         τ ← τ • 1
20:         τ_c ← τ_c • 1
21:         τ_u ← τ_u • 1
22:         τ_v ← τ_v • 1
```

4.1.1. Estimation. For any pair of positive integers a and b such that $a \leq \min\{M, b\}$, let

$$\xi_{a,b} = \begin{cases} 1 & \text{if } b \leq M \\ \binom{b}{M} \Big/ \binom{b-a}{M-a} = \prod_{i=0}^{a-1} \frac{b-i}{M-i} & \text{otherwise} \end{cases}.$$

As shown in the following lemma, $\xi^{-1}_{k,t}$ is the probability that k edges of $G^{(t)}$ are all in $\mathcal{S}$ at time t, i.e., the kth order inclusion probability of the reservoir sampling scheme. The proof can be found in Appendix A.1.

LEMMA 4.1. *For any time step t and any positive integer $k \leq t$, let B be any subset of $E^{(t)}$ of size $|B| = k \leq t$. Then, at the end of time step t,*

$$\Pr(B \subseteq \mathcal{S}) = \begin{cases} 0 & \text{if } k > M \\ \xi^{-1}_{k,t} & \text{otherwise} \end{cases}.$$

We make use of this lemma in the analysis of TRIÈST-BASE.

Let, for any $t \geq 0$, $\xi^{(t)} = \xi_{3,t}$ and let $\tau^{(t)}$ (resp. $\tau^{(t)}_u$) be the value of the counter τ at the end of time step t (i.e., after the edge on the stream at time t has been processed by TRIÈST-BASE) (resp. the value of the counter τ_u at the end of time step t if there is such

a counter, 0 otherwise). When queried at the end of time t, TRIÈST-BASE returns $\xi^{(t)}\tau^{(t)}$ (resp. $\xi^{(t)}\tau_u^{(t)}$) as the estimation for the global (resp. local for $u \in V^{(t)}$) triangle count.

4.1.2. Analysis. We now present the analysis of the estimations computed by TRIÈST-BASE. Specifically, we prove their unbiasedness (and their exactness for $t \leq M$) and then show an exact derivation of their variance and a concentration result. We show the results for the global counts, but results analogous to those in Theorems 4.2, 4.4, and 4.5 hold for the local triangle count for any $u \in V^{(t)}$, replacing the global quantities with the corresponding local ones. We also compare, theoretically, the variance of TRIÈST-BASE with that of a fixed-probability edge sampling approach [28], showing that TRIÈST-BASE has smaller variance for the vast majority of the stream.

4.1.3. Expectation. We have the following result about the estimations computed by TRIÈST-BASE.

THEOREM 4.2. *We have*

$$\xi^{(t)}\tau^{(t)} = \tau^{(t)} = |\Delta^{(t)}| \text{ if } t \leq M$$
$$\mathbb{E}\left[\xi^{(t)}\tau^{(t)}\right] = |\Delta^{(t)}| \text{ if } t > M.$$

The TRIÈST-BASE estimations are not only *unbiased in all cases*, but also actually *exact for* $t \leq M$, i.e., for $t \leq M$, they are the true global/local number of triangles in $G^{(t)}$.

To prove Theorem 4.2, we need to introduce a technical lemma. Its proof can be found in Appendix A.1. We denote with $\Delta^{\mathcal{S}}$ the set of triangles in $G^{\mathcal{S}}$.

LEMMA 4.3. *After each call to* UPDATECOUNTERS*, we have* $\tau = |\Delta^{\mathcal{S}}|$ *and* $\tau_v = |\Delta_v^{\mathcal{S}}|$ *for any* $v \in V_{\mathcal{S}}$ *s.t.* $|\Delta_v^{\mathcal{S}}| \geq 1$.

From here, the proof of Theorem 4.2 is a straightforward application of Lemma 4.3 for the case $t \leq M$ and of that lemma, the definition of expectation, and Lemma 4.1 otherwise. The complete proof can be found in Appendix A.1.

4.1.4. Variance. We now analyze the variance of the estimation returned by TRIÈST-BASE for $t > M$ (the variance is 0 for $t \leq M$).

Let $r^{(t)}$ be the *total* number of unordered pairs of distinct triangles from $\Delta^{(t)}$ sharing an edge,[2] and $w^{(t)} = \binom{|\Delta^{(t)}|}{2} - r^{(t)}$ be the number of unordered pairs of distinct triangles that do not share any edge.

THEOREM 4.4. *For any* $t > M$, *let* $f(t) = \xi^{(t)} - 1$,

$$g(t) = \xi^{(t)}\frac{(M-3)(M-4)}{(t-3)(t-4)} - 1$$

and

$$h(t) = \xi^{(t)}\frac{(M-3)(M-4)(M-5)}{(t-3)(t-4)(t-5)} - 1 \ (\leq 0).$$

We have

$$\mathrm{Var}\left[\xi(t)\tau^{(t)}\right] = |\Delta^{(t)}|\, f(t) + r^{(t)} g(t) + w^{(t)} h(t). \tag{1}$$

In our proofs, we carefully account for the fact that, as we use reservoir sampling [42], the presence of an edge a in $\mathcal{S}$ is *not independent* from the concurrent presence of

[2]Two distinct triangles can share at most one edge.

another edge b in $\mathcal{S}$. This is not the case for samples built using fixed-probability independent edge sampling, such as MASCOT [28]. When computing the variance, we must consider not only pairs of triangles that share an edge, as in the case for independent edge sampling approaches, but also pairs of triangles sharing no edge, since their respective presences in the sample are not independent events. The gain is worth the additional sophistication needed in the analysis, as the contribution to the variance by triangles no sharing edges is *non-positive* ($h(t) \leq 0$), i.e., it reduces the variance. A comparison of the variance of our estimator with that obtained with a fixed-probability independent edge sampling approach, is discussed in Section 4.1.6.

PROOF OF THEOREM 4.4. Assume $|\Delta^{(t)}| > 0$, otherwise the estimation is deterministically correct and has variance 0 and the thesis holds. Let $\lambda \in \Delta^{(t)}$ and $\delta_\lambda^{(t)}$ be as in the proof of Theorem 4.2. We have $\mathrm{Var}[\delta_\lambda^{(t)}] = \xi^{(t)} - 1$ and from this and the definition of variance and covariance, we obtain

$$\begin{aligned}
\mathrm{Var}\big[\xi^{(t)}\tau^{(t)}\big] &= \mathrm{Var}\left[\sum_{\lambda\in\Delta^{(t)}} \delta_\lambda^{(t)}\right] = \sum_{\lambda\in\Delta^{(t)}}\sum_{\gamma\in\Delta^{(t)}} \mathrm{Cov}\big[\delta_\lambda^{(t)}, \delta_\gamma^{(t)}\big] \\
&= \sum_{\lambda\in\Delta^{(t)}} \mathrm{Var}\big[\delta_\lambda^{(t)}\big] + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} \mathrm{Cov}\big[\delta_\lambda^{(t)}, \delta_\gamma^{(t)}\big] \\
&= |\Delta^{(t)}|(\xi^{(t)} - 1) + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} \mathrm{Cov}\big[\delta_\lambda^{(t)}, \delta_\gamma^{(t)}\big] \\
&= |\Delta^{(t)}|(\xi^{(t)} - 1) + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} \Big(\mathbb{E}\big[\delta_\lambda^{(t)}\delta_\gamma^{(t)}\big] - \mathbb{E}\big[\delta_\lambda^{(t)}\big]\mathbb{E}\big[\delta_\gamma^{(t)}\big]\Big) \\
&= |\Delta^{(t)}|(\xi^{(t)} - 1) + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} \Big(\mathbb{E}\big[\delta_\lambda^{(t)}\delta_\gamma^{(t)}\big] - 1\Big). \qquad (2)
\end{aligned}$$

Assume now $|\Delta^{(t)}| \geq 2$, otherwise we have $r^{(t)} = w^{(t)} = 0$ and the thesis holds as the second term on the right-hand side (r.h.s.) of Equation (2) is 0. Let λ and γ be two distinct triangles in $\Delta^{(t)}$. If λ and γ do not share an edge, we have $\delta_\lambda^{(t)}\delta_\gamma^{(t)} = \xi^{(t)}\xi^{(t)} = \xi_{3,t}^2$ if all *six* edges composing λ and γ are in $\mathcal{S}$ at the end of time step t, and $\delta_\lambda^{(t)}\delta_\gamma^{(t)} = 0$ otherwise. From Lemma 4.1, we then have that

$$\begin{aligned}
\mathbb{E}\big[\delta_\lambda^{(t)}\delta_\gamma^{(t)}\big] &= \xi_{3,t}^2 \Pr\big(\delta_\lambda^{(t)}\delta_\gamma^{(t)} = \xi_{3,t}^2\big) = \xi_{3,t}^2 \frac{1}{\xi_{6,t}} = \xi_{3,t} \prod_{j=3}^{5} \frac{M-j}{t-j} \\
&= \xi^{(t)} \frac{(M-3)(M-4)(M-5)}{(t-3)(t-4)(t-5)}. \qquad (3)
\end{aligned}$$

If instead λ and γ share exactly an edge we have $\delta_\lambda^{(t)}\delta_\gamma^{(t)} = \xi_{3,t}^2$ if all *five* edges composing λ and γ are in $\mathcal{S}$ at the end of time step t, and $\delta_\lambda^{(t)}\delta_\gamma^{(t)} = 0$ otherwise. From Lemma 4.1,

we then have that

$$\mathbb{E}\left[\delta_\lambda^{(t)}\delta_\gamma^{(t)}\right] = \xi_{3,t}^2 \Pr\left(\delta_\lambda^{(t)}\delta_\gamma^{(t)} = \xi_{3,t}^2\right) = \xi_{3,t}^2 \frac{1}{\xi_{5,t}} = \xi_{3,t} \prod_{j=3}^{4} \frac{M-j}{t-j}$$

$$= \xi^{(t)} \frac{(M-3)(M-4)}{(t-3)(t-4)}. \tag{4}$$

The thesis follows by combining Equations (2), (3), (4), recalling the definitions of $r^{(t)}$ and $w^{(t)}$, and slightly reorganizing the terms. □

4.1.5. Concentration. We have the following concentration result on the estimation returned by TRIÈST-BASE. Let $h^{(t)}$ denote the maximum number of triangles sharing a single edge in $G^{(t)}$.

THEOREM 4.5. *Let $t \geq 0$ and assume $|\Delta^{(t)}| > 0$.[3] For any $\varepsilon, \delta \in (0,1)$, let*

$$\Phi = \sqrt[3]{8\varepsilon^{-2} \frac{3h^{(t)}+1}{|\Delta^{(t)}|} \ln\left(\frac{(3h^{(t)}+1)e}{\delta}\right)}.$$

If

$$M \geq \max\left\{t\Phi\left(1 + \frac{1}{2}\ln^{2/3}(t\Phi)\right), 12\varepsilon^{-1} + e^2, 25\right\},$$

then $|\xi^{(t)}\tau^{(t)} - |\Delta^{(t)}|| < \varepsilon|\Delta^{(t)}|$ with probability $> 1-\delta$.

The roadmap to proving Theorem 4.5 is the following:

(1) We first define two simpler algorithms, named INDEP and MIX. The algorithms use, respectively, fixed-probability independent sampling of edges and reservoir sampling (but with a different estimator than the one used by TRIÈST-BASE).
(2) We then prove concentration results on the estimators of INDEP and MIX. Specifically, the concentration result for INDEP uses a result by Hajnal and Szemerédi [17] on graph coloring, while the one for MIX will depend on the concentration result for INDEP and on a Poisson-approximation-like technical result stating that probabilities of events when using reservoir sampling are close to the probabilities of those events when using fixed-probability independent sampling.
(3) We then show that the estimates returned by TRIÈST-BASE are close to the estimates returned by MIX.
(4) Finally, we combine the above results and show that, if M is large enough, then the estimation provided by MIX is likely to be close to $|\Delta^{(t)}|$ and since the estimation computed by TRIÈST-BASE is close to that of MIX, then it must also be close to $|\Delta^{(t)}|$.

Note: For ease of presentation, in the following, we use $\phi^{(t)}$ to denote the estimation returned by TRIÈST-BASE, i.e., $\phi^{(t)} = \xi^{(t)}\tau^{(t)}$.

The INDEP *algorithm*. The INDEP algorithm works as follows: It creates a sample $\mathcal{S}_{\text{IN}}$ by sampling edges in $E^{(t)}$ independently with a fixed probability p. It estimates the global number of triangles in $G^{(t)}$ as

$$\phi_{\text{IN}}^{(t)} = \frac{\tau_{\text{IN}}^{(t)}}{p^3},$$

[3] If $|\Delta^{(t)}| = 0$, our algorithms correctly estimate 0 triangles.

where $\tau_{\text{IN}}^{(t)}$ is the number of triangles in $\mathcal{S}_{\text{IN}}$. This is, for example, the approach taken by MASCOT-C [28].

The MIX *algorithm*. The MIX algorithm works as follows: It uses reservoir sampling (like TRIÈST-BASE) to create a sample $\mathcal{S}_{\text{MIX}}$ of M edges from $E^{(t)}$, but uses a different estimator than the one used by TRIÈST-BASE. Specifically, MIX uses

$$\phi_{\text{MIX}}^{(t)} = \left(\frac{t}{M}\right)^3 \tau^{(t)}$$

as an estimator for $|\Delta^{(t)}|$, where $\tau^{(t)}$ is, as in TRIÈST-BASE, the number of triangles in $G^{\mathcal{S}}$. (TRIÈST-BASE uses $\phi^{(t)} = \frac{t(t-1)(t-2)}{M(M-1)(M-2)}\tau^{(t)}$ as an estimator.)

We call this algorithm MIX because it uses reservoir sampling to create the sample, but computes the estimate as if it used fixed-probability independent sampling, hence in some sense it "mixes" the two approaches.

Concentration results for INDEP *and* MIX. We now show a concentration result for INDEP. Then, we show a technical lemma (Lemma 4.7) relating the probabilities of events when using reservoir sampling to the probabilities of those events when using fixed-probability independent sampling. Finally, we use these results to show that the estimator used by MIX is also concentrated (Lemma 4.9).

LEMMA 4.6. *Let $t \geq 0$ and assume $|\Delta^{(t)}| > 0$.*[4] *For any $\varepsilon, \delta \in (0, 1)$, if*

$$p \geq \sqrt[3]{2\varepsilon^{-2} \ln\left(\frac{3h^{(t)}+1}{\delta}\right) \frac{3h^{(t)}+1}{|\Delta^{(t)}|}}, \tag{5}$$

then

$$\Pr\left(|\phi_{\text{IN}}^{(t)} - \Delta^{(t)}|| < \varepsilon|\Delta^{(t)}|\right) > 1 - \delta.$$

PROOF. Let H be a graph built as follows: H has one node for each triangle in $G^{(t)}$ and there is an edge between two nodes in H if the corresponding triangles in $G^{(t)}$ share an edge. By this construction, the maximum degree in H is $3h^{(t)}$. Hence, by the Hajanal–Szeméredi's theorem [17], there is a proper coloring of H with at most $3h^{(t)}+1$ colors such that for each color there are at least $L = \frac{|\Delta^{(t)}|}{3h^{(t)}+1}$ nodes with that color.

Assign an arbitrary numbering to the triangles of $G^{(t)}$ (and, therefore, to the nodes of H) and let X_i be a Bernoulli random variable, indicating whether the triangle i in $G^{(t)}$ is in the sample at time t. From the properties of independent sampling of edges, we have $\Pr(X_i = 1) = p^3$ for any triangle i. For any color c of the coloring of H, let $\mathcal{X}_c$ be the set of r.v.'s X_i such that the node i in H has color c. Since the coloring of H which we are considering is proper, the r.v.'s in $\mathcal{X}_c$ are independent, as they correspond to triangles which do not share any edge and edges are sampled independent of each other. Let Y_c be the sum of the r.v.'s in $\mathcal{X}_c$. The r.v. Y_c has a binomial distribution with parameters $|\mathcal{X}_c|$ and p_i^3. By the Chernoff bound for binomial r.v.'s, we have that

$$\begin{aligned}\Pr\left(|p^{-3}Y_c - |\mathcal{X}_c|| > \varepsilon|\mathcal{X}_c|\right) &< 2\exp\left(-\varepsilon^2 p^3|\mathcal{X}_c|/2\right)\\ &< 2\exp\left(-\varepsilon^2 p^3 L/2\right)\\ &\leq \frac{\delta}{3h^{(t)}+1},\end{aligned}$$

[4]For $|\Delta^{(t)}| = 0$, INDEP correctly and deterministically returns 0 as the estimation.

where the last step comes from the requirement in Equation (5). Then, by applying the union bound over all the (at most) $3h^{(t)} + 1$ colors, we get

$$\Pr(\exists \text{ color } c \text{ s.t. } |p^{-3}Y_c - |\mathcal{X}_c|| > \varepsilon|\mathcal{X}_c|) < \delta.$$

Since $\phi_{\text{IN}}(t) = p^{-3}\sum_{\text{color } c} Y_c$, from the above equation we have that, with probability at least $1 - \delta$,

$$\begin{aligned}|\phi_{\text{IN}}^{(t)} - |\Delta^{(t)}|| &\leq \left|\sum_{\text{color } c} p^{-3}Y_c - \sum_{\text{color } c} |\mathcal{X}_c|\right| \\ &\leq \sum_{\text{color } c} |p^{-3}Y_c - |\mathcal{X}_c|| \leq \sum_{\text{color } c} \varepsilon|\mathcal{X}_c| \leq \varepsilon|\Delta^{(t)}|. \quad \square\end{aligned}$$

The above result is of independent interest and can be used, for example, to give concentration bounds to the estimation computed by MASCOT-C [28].

We remark that we cannot use the same approach from Lemma 4.6 to show a concentration result for TRIÈST-BASE because it uses reservoir sampling, hence the event of having a triangle a in $\mathcal{S}$ and the event of having another triangle b in $\mathcal{S}$ are not independent.

We can, however, show the following general result, similar in spirit to the well-known Poisson approximation of balls-and-bins processes [31]. Its proof can be found in Appendix A.1.

Fix the parameter M and a time $t > M$. Let $\mathcal{S}_{\text{MIX}}$ be a sample of M edges from $E^{(t)}$ obtained through reservoir sampling (as MIX would do), and let $\mathcal{S}_{\text{IN}}$ be a sample of the edges in $E^{(t)}$ obtained by sampling edges independently with probability M/t (as INDEP would do). We remark that the size of $\mathcal{S}_{\text{IN}}$ is in $[0, t]$ but not necessarily M.

LEMMA 4.7. *Let $f : 2^{E^{(t)}} \to \{0, 1\}$ be an arbitrary binary function from the powerset of $E^{(t)}$ to $\{0, 1\}$. We have*

$$\Pr(f(\mathcal{S}_{\text{MIX}}) = 1) \leq e\sqrt{M}\Pr(f(\mathcal{S}_{\text{IN}}) = 1).$$

We now use the above two lemmas to show that the estimator $\phi_{\text{MIX}}^{(t)}$ computed by MIX is concentrated. We will first need the following technical fact.

FACT 4.8. *For any $x \geq 5$, we have*

$$\ln\left(x(1 + \ln^{2/3} x)\right) \leq \ln^2 x.$$

LEMMA 4.9. *Let $t \geq 0$ and assume $|\Delta^{(t)}| < 0$. For any $\varepsilon, \delta \in (0, 1)$, let*

$$\Psi = 2\varepsilon^{-2}\frac{3h^{(t)} + 1}{|\Delta^{(t)}|}\ln\left(e\frac{3h^{(t)} + 1}{\delta}\right).$$

If

$$M \geq \max\left\{t\sqrt[3]{\Psi}\left(1 + \frac{1}{2}\ln^{2/3}(t\sqrt[3]{\Psi})\right), 25\right\},$$

then

$$\Pr\left(|\phi_{\text{MIX}}^{(t)} - |\Delta^{(t)}|| < \varepsilon|\Delta^{(t)}|\right) \geq 1 - \delta.$$

PROOF. For any $S \subseteq E^{(t)}$, let $\tau(S)$ be the number of triangles in S, i.e., the number of triplets of edges in S that compose a triangle in $G^{(t)}$. Define the function $g : 2^{E^{(t)}} \to \mathbb{R}$

as

$$g(S) = \left(\frac{t}{M}\right)^3 \tau(S).$$

Assume that we run INDEP with $p = M/t$, and let $\mathcal{S}_{\text{IN}} \subseteq E^{(t)}$ be the sample built by INDEP (through independent sampling with fixed probability p). Assume also that we run MIX with parameter M, and let $\mathcal{S}_{\text{MIX}}$ be the sample built by MIX (through reservoir sampling with a reservoir of size M). We have that $\phi^{(t)}_{\text{IN}} = g(\mathcal{S}_{\text{IN}})$ and $\phi^{(t)}_{\text{MIX}} = g(\mathcal{S}_{\text{MIX}})$. Define now the binary function $f : 2^{E^{(t)}} \to \{0, 1\}$ as

$$f(S) = \begin{cases} 1 \text{ if } |g(S) - |\Delta^{(t)}|| > \varepsilon|\Delta^{(t)}| \\ 0 \text{ otherwise} \end{cases}.$$

We now show that, for M as in the hypothesis, we have

$$p \geq \sqrt[3]{2\varepsilon^{-2}\frac{3h^{(t)}+1}{|\Delta^{(t)}|}\ln\left(e\sqrt{M}\frac{3h^{(t)}+1}{\delta}\right)}. \tag{6}$$

Assume for now that the above is true. From this, using Lemma 4.6 and the above fact about g we get that

$$\Pr\left(|\phi^{(t)}_{\text{IN}} - |\Delta^{(t)}|| > \varepsilon|\Delta^{(t)}|\right) = \Pr(f(\mathcal{S}_{\text{IN}}) = 1) < \frac{\delta}{e\sqrt{M}}.$$

From this and Lemma 4.7, we get that

$$\Pr(f(\mathcal{S}_{\text{MIX}}) = 1) \leq \delta$$

which, from the definition of f and the properties of g, is equivalent to

$$\Pr\left(|\phi^{(t)}_{\text{MIX}} - |\Delta^{(t)}|| > \varepsilon|\Delta^{(t)}|\right) \leq \delta$$

and the proof is complete. All that is left is to show that Equation (6) holds for M as in the hypothesis.

Since $p = M/t$, we have that Equation (6) holds for

$$\begin{aligned} M^3 &\geq t^3 2\varepsilon^{-2}\frac{3h^{(t)}+1}{|\Delta^{(t)}|}\ln\left(\sqrt{M}e\frac{3h^{(t)}+1}{\delta}\right) \\ &= t^3 2\varepsilon^{-2}\frac{3h^{(t)}+1}{|\Delta^{(t)}|}\left(\ln\left(e\frac{3h^{(t)}+1}{\delta}\right) + \frac{1}{2}\ln M\right). \end{aligned} \tag{7}$$

We now show that (7) holds.

Let $A = t\sqrt[3]{\Psi}$ and let $B = t\sqrt[3]{\Psi}\ln^{2/3}(t\sqrt[3]{\Psi})$. We now show that $A^3 + B^3$ is greater or equal to the r.h.s. of Equation (7), hence $M^3 = (A+B)^3 > A^3 + B^3$ must also be greater or equal to the r.h.s. of Equation (7), i.e., Equation (7) holds. This really reduces to show that

$$B^3 \geq t^3 2\varepsilon^{-2}\frac{3h^{(t)}+1}{|\Delta^{(t)}|}\frac{1}{2}\ln M \tag{8}$$

as the r.h.s.of Equation (7) can be written as

$$A^3 + t^3 2\varepsilon^{-2}\frac{3h^{(t)}+1}{|\Delta^{(t)}|}\frac{1}{2}\ln M.$$

We actually show that

$$B^3 \geq t^3 \Psi \frac{1}{2} \ln M, \tag{9}$$

which implies Equation (8) which, as discussed, in turn, implies Equation (7). Consider the ratio

$$\frac{B^3}{t^3 \Psi \frac{1}{2} \ln M} = \frac{\frac{1}{2} t^3 \Psi \ln^2(t\sqrt[3]{\Psi})}{t^3 \Psi \frac{1}{2} \ln M} = \frac{\ln^2(t\sqrt[3]{\Psi})}{\ln M} \geq \frac{\ln^2(t\sqrt[3]{\Psi})}{\ln(t\sqrt[3]{\Psi}(1 + \ln^{2/3}(t\sqrt[3]{\Psi})))}. \tag{10}$$

We now show that $t\sqrt[3]{\Psi} \geq 5$. By the assumptions $t > M \geq 25$ and by

$$t\sqrt[3]{\Psi} \geq \frac{t}{\sqrt[3]{|\Delta^{(t)}|}} \geq \sqrt{t},$$

which holds because $|\Delta^{(t)}| \leq t^{3/2}$ (in a graph with t edges there cannot be more than $t^{3/2}$ triangles) we have that $t\sqrt[3]{\Psi} \geq 5$. Hence, Fact 4.8 holds and we can write, from Equation (10):

$$\frac{\ln^2(t\sqrt[3]{\Psi})}{\ln(t\sqrt[3]{\Psi}(1 + \ln^{2/3}(t\sqrt[3]{\Psi})))} \geq \frac{\ln^2(t\sqrt[3]{\Psi})}{\ln^2(t\sqrt[3]{\Psi})} \geq 1,$$

which proves Equation (9), and in cascade Equations (8), (7), (6), and the thesis. □

Relationship between TRIÈST-BASE *and* MIX. When both TRIÈST-BASE and MIX use a sample of size M, their respective estimators $\phi^{(t)}$ and $\phi^{(t}_{\text{MIX}}$ are related as discussed in the following result, whose straightforward proof is deferred to Appendix A.1.

LEMMA 4.10. *For any $t > M$, we have*

$$\left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}}\right| \leq \phi^{(t)}_{\text{MIX}} \frac{4}{M-2}.$$

Tying everything together. Finally, we can use the previous lemmas to prove our concentration result for TRIÈST-BASE.

PROOF OF THEOREM 4.5. For M as in the hypothesis we have, from Lemma 4.9, that

$$\Pr\left(\phi^{(t)}_{\text{MIX}} \leq (1 + \varepsilon/2)|\Delta^{(t)}|\right) \geq 1 - \delta.$$

Suppose the event $\phi^{(t)}_{\text{MIX}} \leq (1 + \varepsilon/2)|\Delta^{(t)}|$ (i.e., $|\phi^{(t)}_{\text{MIX}} - |\Delta^{(t)}|| \leq \varepsilon|\Delta^{(t)}|/2$) is indeed verified. From this and Lemma 4.10, we have

$$\left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}}\right| \leq \left(1 + \frac{\varepsilon}{2}\right)|\Delta^{(t)}| \frac{4}{M-2} \leq |\Delta^{(t)}| \frac{6}{M-2},$$

where the last inequality follows from the fact that $\varepsilon < 1$. Hence, given that $M \geq 12\varepsilon^{-1} + e^2 \geq 12\varepsilon^{-1} + 2$, we have

$$\left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}}\right| \leq |\Delta^{(t)}| \frac{\varepsilon}{2}.$$

Using the above, we can then write

$$\begin{aligned}
\left|\phi^{(t)} - |\Delta^{(t)}|\right| &= \left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}} + \phi^{(t)}_{\text{MIX}} - |\Delta^{(t)}|\right| \\
&\leq \left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}}\right| + \left|\phi^{(t)}_{\text{MIX}} - |\Delta^{(t)}|\right| \\
&\leq \frac{\varepsilon}{2}|\Delta^{(t)}| + \frac{\varepsilon}{2}|\Delta^{(t)}| = \varepsilon|\Delta^{(t)}|,
\end{aligned}$$

which completes the proof. □

4.1.6. Comparison with Fixed-probability Approaches. We now compare the variance of TRIÈST-BASE to the variance of the fixed-probability sampling approach MASCOT-C [28], which samples edges *independently* with a fixed probability p and uses $p^{-3}|\Delta_{\mathcal{S}}|$ as the estimate for the global number of triangles at time t. As shown by Lim and Kang [28, Lemma 2], the variance of this estimator is

$$\mathrm{Var}[p^{-3}|\Delta_{\mathcal{S}}|] = |\Delta^{(t)}|\,\bar{f}(p) + r^{(t)}\bar{g}(p),$$

where $\bar{f}(p) = p^{-3} - 1$ and $\bar{g}(p) = p^{-1} - 1$.

Assume that we give MASCOT-C the additional information that the stream has finite length T, and assume we run MASCOT-C with $p = M/T$ so that the expected sample size at the end of the stream is M.[5] Let $\mathbb{V}^{(t)}_{\mathrm{fix}}$ be the resulting variance of the MASCOT-C estimator at time t, and let $\mathbb{V}^{(t)}$ be the variance of our estimator at time t (see Equation (1)). For $t \leq M$, $\mathbb{V}^{(t)} = 0$, hence $\mathbb{V}^{(t)} \leq \mathbb{V}^{(t)}_{\mathrm{fix}}$.

For $M < t < T$, we can show the following result. Its proof is more tedious than interesting so we defer it to Appendix A.1.

LEMMA 4.11. *Let $0 < \alpha < 1$ be a constant. For any constant $M > \max(\frac{8\alpha}{1-\alpha}, 42)$ and any $t \leq \alpha T$, we have $\mathbb{V}^{(t)} < \mathbb{V}^{(t)}_{\mathrm{fix}}$.*

For example, if we set $\alpha = 0.99$ and run TRIÈST-BASE with $M \geq 400$ and MASCOT-C with $p = M/T$, we have that TRIÈST-BASE has strictly smaller variance than MASCOT-C for 99% of the stream.

What about $t = T$? The difference between the definitions of $\mathbb{V}^{(t)}_{\mathrm{fix}}$ and $\mathbb{V}^{(t)}$ is in the presence of $\bar{f}(M/T)$ instead of $f(t)$ (resp. $\bar{g}(M/T)$ instead of $g(t)$) as well as the additional term $w^{(t)}h(M,t) \leq 0$ in our $\mathbb{V}^{(t)}$. Let $M(T)$ be an arbitrary slowly increasing function of T. For $T \to \infty$, we can show that $\lim_{T\to\infty} \frac{\bar{f}(M(T)/T)}{f(T)} = \lim_{T\to\infty} \frac{\bar{g}(M(T)/T)}{g(T)} = 1$, hence, *informally*, $\mathbb{V}^{(T)} \to \mathbb{V}^{(T)}_{\mathrm{fix}}$, for $T \to \infty$.

A similar discussion also holds for the method we present in Section 4.2, and explains the results of our experimental evaluations, which shows that our algorithms have strictly lower (empirical) variance than fixed-probability approaches for most of the stream.

4.1.7. Update Time. The time to process an element of the stream is dominated by the computation of the shared neighborhood $\mathcal{N}_{u,v}$ in UPDATECOUNTERS. A MERGESORT-based algorithm for the intersection requires $O(\deg(u) + \deg(v))$ time, where the degrees are w.r.t. the graph $G_{\mathcal{S}}$. By storing the neighborhood of each vertex in a Hash Table (resp. an AVL tree), the update time can be reduced to $O(\min\{\deg(v), \deg(u)\})$ (resp. amortized time $O(\min\{\deg(v), \deg(u)\} + \log\max\{\deg(v), \deg(u)\})$).

4.2. Improved Insertion Algorithm – TRIÈST-IMPR

TRIÈST-IMPR is a variant of TRIÈST-BASE with small modifications that result in higher quality (i.e., lower variance) estimations. The changes are as follows:

(1) UPDATECOUNTERS is called *unconditionally for each element on the stream*, before the algorithm decides whether or not to insert the edge into $\mathcal{S}$. W.r.t. the pseudocode in Algorithm 1, this change corresponds to moving the call to UPDATECOUNTERS on line 6 to *before* the *if* block. MASCOT [28] uses a similar idea, but TRIÈST-IMPR is

[5] We are giving MASCOT-C a significant advantage: If only space M were available, we should run MASCOT-C with a sufficiently smaller $p' < p$, otherwise there would be a constant probability that MASCOT-C would run out of memory.

significantly different as TRIÈST-IMPR allows edges to be removed from the sample, since it uses reservoir sampling.

(2) TRIÈST-IMPR *never* decrements the counters when an edge is removed from $\mathcal{S}$. W.r.t. the pseudocode in Algorithm 1, we remove the call to UPDATECOUNTERS on line 13.

(3) UPDATECOUNTERS performs a *weighted* increase of the counters using $\eta^{(t)} = \max\{1, (t-1)(t-2)/(M(M-1))\}$ as weight. W.r.t. the pseudocode in Algorithm 1, we replace "1" with $\eta^{(t)}$ on lines 19–22 (given change 2 above, all the calls to UPDATECOUNTERS have $\bullet = +$).

The resulting pseudocode for TRIÈST-IMPR is presented in Algorithm 2.

ALGORITHM 2: TRIÈST-IMPR

Input: Insertion-only edge stream Σ, integer $M \geq 6$

```
 1: S ← ∅, t ← 0, τ ← 0
 2: for each element (+, (u, v)) from Σ do
 3:     t ← t + 1
 4:     UpdateCounters(u, v)
 5:     if SampleEdge((u, v), t) then
 6:         S ← S ∪ {(u, v)}

 7: function SampleEdge((u, v), t)
 8:     if t ≤ M then
 9:         return True
10:     else if FlipBiasedCoin(M/t) = heads then
11:         (u', v') ← random edge from S
12:         S ← S \ {(u', v')}
13:         return True
14:     return False

15: function UpdateCounters(u, v)
16:     N^S_{u,v} ← N^S_u ∩ N^S_v
17:     η = max{1, (t − 1)(t − 2)/(M(M − 1))}
18:     for all c ∈ N^S_{u,v} do
19:         τ ← τ + η
20:         τ_c ← τ_c + η
21:         τ_u ← τ_u + η
22:         τ_v ← τ_v + η
```

Counters. If we are interested only in estimating the global number of triangles in $G^{(t)}$, TRIÈST-IMPR needs to maintain only the counter τ and the edge sample $\mathcal{S}$ of size M, so it still requires space $O(M)$. If instead we are interested in estimating the local triangle counts, at any time t TRIÈST-IMPR maintains (non-zero) local counters *only* for the nodes u such that at least one triangle with a corner u has been detected by the algorithm up until time t. The number of such nodes may be greater than $O(M)$, but this is the price to pay to obtain estimations with lower variance (Theorem 4.13).

4.2.1. Estimation. When queried for an estimation, TRIÈST-IMPR returns the value of the corresponding counter, unmodified.

4.2.2. Analysis. We now present the analysis of the estimations computed by TRIÈST-IMPR, showing results involving their unbiasedness, their variance, and their concentration around their expectation. Results analogous to those in Theorems 4.12, 4.13,

and 4.15 hold for the local triangle count for any $u \in V^{(t)}$, replacing the global quantities with the corresponding local ones.

4.2.3. Expectation. As in TRIÈST-BASE, the estimations by TRIÈST-IMPR are *exact* at time $t \leq M$ and unbiased for $t > M$. The proof of the following theorem follows the same steps as the one for Theorem 4.2, and can be found in Appendix A.2.

THEOREM 4.12. *We have* $\tau^{(t)} = |\Delta^{(t)}|$ *if* $t \leq M$ *and* $\mathbb{E}[\tau^{(t)}] = |\Delta^{(t)}|$ *if* $t > M$.

4.2.4. Variance. We now show an *upper bound* to the variance of the TRIÈST-IMPR estimations for $t > M$. The proof relies on a very careful analysis of the covariance of two triangles that depends on the order of arrival of the edges in the stream (which we assume to be adversarial). For any $\lambda \in \Delta^{(t)}$, we denote as t_λ the time at which the last edge of λ is observed on the stream. Let $z^{(t)}$ be the number of unordered pairs (λ, γ) of distinct triangles in $G^{(t)}$ that share an edge g and are such that

(1) g is neither the last edge of λ nor γ on the stream; and
(2) $\min\{t_\lambda, t_\gamma\} > M + 1$.

THEOREM 4.13. *Then, for any time* $t > M$, *we have*

$$\mathrm{Var}\left[\tau^{(t)}\right] \leq |\Delta^{(t)}|(\eta^{(t)} - 1) + z^{(t)}\frac{t-1-M}{M}. \tag{11}$$

The bound to the variance presented in Equation (11) is extremely pessimistic and loose. Specifically, it does not contain the *negative* contribution to the variance given by the $\binom{|\Delta^{(t)}|}{2} - z^{(t)}$ triangles that do not satisfy the requirements in the definition of $z^{(t)}$. Among these pairs there are not only, for example, all pairs of triangles not sharing any edge, but also many pairs of triangles that share an edge, as the definition of $z^{(t)}$ consider only a subsets of these. All these pairs would give a negative contribution to the variance, i.e., decrease the r.h.s. of Equation (11), whose more correct form would be

$$|\Delta^{(t)}|(\eta^{(t)} - 1) + z^{(t)}\frac{t-1-M}{M} + \left(\binom{|\Delta^{(t)}|}{2} - z^{(t)}\right)\omega_{M,t},$$

where $\omega_{M,t}$ is (an upper bound to) the minimum *negative* contribution of a pair of triangles that do not satisfy the requirements in the definition of $z^{(t)}$. Sadly, computing informative upper bounds to $\omega_{M,t}$ is not straightforward, even in the restricted setting where only pairs of triangles not sharing any edge are considered.

To prove Theorem 4.13, we first need Lemma 4.14, whose proof is deferred to Appendix A.2.

For any time step t and any edge $e \in E^{(t)}$, we denote with t_e the time step at which e is on the stream. For any $\lambda \in \Delta^{(t)}$, let $\lambda = (\ell_1, \ell_2, \ell_3)$, where the edges are numbered in order of appearance on the stream. We define the event D_λ as the event that ℓ_1 and ℓ_2 are both in the edge sample $\mathcal{S}$ at the end of time step $t_\lambda - 1$.

LEMMA 4.14. *Let* $\lambda = (\ell_1, \ell_2, \ell_3)$ *and* $\gamma = (g_1, g_2, g_3)$ *be two* disjoint *triangles, where the edges are numbered in order of appearance on the stream, and assume, w.l.o.g., that the last edge of* λ *is on the stream* before *the last edge of* γ. *Then,*

$$\Pr(D_\gamma \mid D_\lambda) \leq \Pr(D_\gamma).$$

We can now prove Theorem 4.13.

PROOF OF THEOREM 4.13. Assume $|\Delta^{(t)}| > 0$, otherwise TRIÈST-IMPR estimation is deterministically correct and has variance 0 and the thesis holds. Let $\lambda \in \Delta^{(t)}$ and let δ_λ be

a random variable that takes value $\xi_{2,t_\lambda-1}$ if both ℓ_1 and ℓ_2 are in $\mathcal{S}$ at the end of time step $t_\lambda - 1$, and 0 otherwise. Since

$$\text{Var}\left[\delta_\lambda\right] = \xi_{2,t_\lambda-1} - 1 \leq \xi_{2,t-1},$$

we have

$$\begin{aligned}
\text{Var}\left[\tau^{(t)}\right] &= \text{Var}\left[\sum_{\lambda\in\Delta^{(t)}} \delta_\lambda\right] = \sum_{\lambda\in\Delta^{(t)}}\sum_{\gamma\in\Delta^{(t)}} \text{Cov}[\delta_\lambda, \delta_\gamma] \\
&= \sum_{\lambda\in\Delta^{(t)}} \text{Var}\left[\delta_\lambda\right] + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} \text{Cov}[\delta_\lambda, \delta_\gamma] \\
&\leq |\Delta^{(t)}|(\xi_{2,t-1} - 1) + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} (\mathbb{E}[\delta_\lambda\delta_\gamma] - \mathbb{E}[\delta_\lambda]\mathbb{E}[\delta_\gamma]) \\
&\leq |\Delta^{(t)}|(\xi_{2,t-1} - 1) + \sum_{\substack{\lambda,\gamma\in\Delta^{(t)}\\ \lambda\neq\gamma}} (\mathbb{E}[\delta_\lambda\delta_\gamma] - 1). \qquad (12)
\end{aligned}$$

For any $\lambda \in \Delta^{(t)}$ define $q_\lambda = \xi_{2,t_\lambda-1}$. Assume now $|\Delta^{(t)}| \geq 2$, otherwise we have $r^{(t)} = w^{(t)} = 0$ and the thesis holds as the second term on the r.h.s. of Equation (12) is 0. Let now λ and γ be two distinct triangles in $\Delta^{(t)}$ (hence $t \geq 5$). We have

$$\mathbb{E}[\delta_\lambda\delta_\gamma] = q_\lambda q_\gamma \Pr(\delta_\lambda\delta_\gamma = q_\lambda q_\gamma).$$

The event "$\delta_\lambda\delta_\gamma = q_\lambda q_\gamma$" is the intersection of events $D_\lambda \cap D_\gamma$, where D_λ is the event that the first two edges of λ are in $\mathcal{S}$ at the end of time step $t_\lambda - 1$, and similarly for D_γ. We now look at $\Pr(D_\lambda \cap D_\gamma)$ in the various possible cases.

Assume that λ and γ do not share any edge, and, w.l.o.g., that the third (and last) edge of λ appears on the stream before the third (and last) edge of γ, i.e., $t_\lambda < t_\gamma$. From Lemma 4.14 and Lemma 4.1, we then have

$$\Pr(D_\lambda \cap D_\gamma) = \Pr(D_\gamma|D_\lambda)\Pr(D_\lambda) \leq \Pr(D_\gamma)\Pr(D_\lambda) \leq \frac{1}{q_\lambda q_\gamma}.$$

Consider now the case where λ and γ share an edge g. W.l.o.g., let us assume that $t_\lambda \leq t_\gamma$ (since the shared edge may be the last on the stream both for λ and for γ, we may have $t_\lambda = t_\gamma$). There are the following possible sub-cases:

g is the last on the stream among all the edges of λ and γ: In this case, we have $t_\lambda = t_\gamma$. The event "$D_\lambda \cap D_\gamma$" happens if and only if the *four* edges that, together with g, compose λ and γ are all in $\mathcal{S}$ at the end of time step $t_\lambda - 1$. Then, from Lemma 4.1, we have

$$\Pr(D_\lambda \cap D_\gamma) = \frac{1}{\xi_{4,t_\lambda-1}} \leq \frac{1}{q_\lambda}\frac{(M-2)(M-3)}{(t_\lambda-3)(t_\lambda-4)} \leq \frac{1}{q_\lambda}\frac{M(M-1)}{(t_\lambda-1)(t_\lambda-2)} \leq \frac{1}{q_\lambda q_\gamma}.$$

g is the last on the stream among all the edges of λ and the first among all the edges of γ: In this case, we have that D_λ and D_γ are independent. Indeed, the fact that the first two edges of λ (neither of which is g) are in $\mathcal{S}$ when g arrives on the stream has no influence on the probability that g and the second edge of γ are inserted in $\mathcal{S}$ and are not evicted until the third edge of γ is on the stream. Hence, we have

$$\Pr(D_\lambda \cap D_\gamma) = \Pr(D_\gamma)\Pr(D_\lambda) = \frac{1}{q_\lambda q_\gamma}.$$

g is the last on the stream among all the edges of λ and the second among all the edges of γ: In this case, we can follow an approach similar to the one in the proof for Lemma 4.14 and have that

$$\Pr(D_\lambda \cap D_\gamma) \leq \Pr(D_\gamma)\Pr(D_\lambda) \leq \frac{1}{q_\lambda q_\gamma}.$$

The intuition behind this is that if the first two edges of λ are in $\mathcal{S}$ when g is on the stream, their presence lowers the probability that the first edge of γ is in $\mathcal{S}$ at the same time, and hence that the first edge of γ and g are in $\mathcal{S}$ when the last edge of γ is on the stream.

g is not the last on the stream among all the edges of λ: There are two situations to consider, or better, one situation and all other possibilities. The situation we consider is that

(1) g is the first edge of γ on the stream; and
(2) the second edge of γ to be on the stream is on the stream at time $t_2 > t_\lambda$.

Suppose this is the case. Recall that if D_λ is verified, than we know that g is in $\mathcal{S}$ at the beginning of time step t_λ. Define the following events:

- E_1: the set of edges evicted from $\mathcal{S}$ between the beginning of time step t_λ and the beginning of time step t_2 does not contain g.
- E_2: the second edge of γ, which is on the stream at time t_2, is inserted in $\mathcal{S}$ and the edge that is evicted is not g.
- E_3: the set of edges evicted from $\mathcal{S}$ between the beginning of time step t_2+1 and the beginning of time step t_γ does not contain either g or the second edge of γ.

We can then write

$$\Pr(D_\gamma \mid D_\lambda) = \Pr(E_1 \mid D_\lambda)\Pr(E_2 \mid E_1 \cap D_\lambda)\Pr(E_3 \mid E_2 \cap E_1 \cap D_\lambda).$$

We now compute the probabilities on the r.h.s., where we denote with $\mathbb{1}_{t_2>M}(1)$ the function that has value 1 if $t_2 > M$, and value 0 otherwise

$$\begin{aligned}
\Pr(E_1 \mid D_\lambda) &= \prod_{j=\max\{t_\lambda, M+1\}}^{t_2-1} \left(\left(1-\frac{M}{j}\right) + \frac{M}{j}\left(\frac{M-1}{M}\right)\right) \\
&= \prod_{j=\max\{t_\lambda, M+1\}}^{t_2-1} \frac{j-1}{j} = \frac{\max\{t_\lambda - 1, M\}}{\max\{M, t_2-1\}}; \\
\Pr(E_2 \mid E_1 \cap D_\lambda) &= \frac{M}{\max\{t_2, M\}} \frac{M - \mathbb{1}_{t_2>M}(1)}{M} = \frac{M - \mathbb{1}_{t_2>M}(1)}{\max\{t_2, M\}}; \\
\Pr(E_3 \mid E_2 \cap E_1 \cap D_\lambda) &= \prod_{j=\max\{t_2+1, M+1\}}^{t_\gamma-1} \left(\left(1-\frac{M}{j}\right) + \frac{M}{j}\left(\frac{M-2}{M}\right)\right) \\
&= \prod_{j=\max\{t_2+1, M+1\}}^{t_\gamma-1} \frac{j-2}{j} = \frac{\max\{t_2, M\}\max\{t_2-1, M-1\}}{\max\{t_\gamma-2, M-1\}\max\{t_\gamma-1, M\}}.
\end{aligned}$$

Hence, we have

$$\Pr(D_\gamma \mid D_\lambda) = \frac{\max\{t_\lambda - 1, M\}(M - \mathbb{1}_{t_2>M}(1))\max\{t_2-1, M-1\}}{\max\{M, t_2-1\}\max\{(t_\gamma-2)(t_\gamma-1), M(M-1)\}}.$$

With a (somewhat tedious) case analysis, we can verify that

$$\Pr(D_\gamma \mid D_\lambda) \leq \frac{1}{q_\gamma} \frac{\max\{M, t_\lambda - 1\}}{M}.$$

Consider now the complement of the situation we just analyzed. In this case, two edges of γ, that is, g and another edge h are on the stream before time t_λ, in some non-relevant order (i.e., g could be the first or the second edge of γ on the stream). Define now the following events:

- E_1: h and g are both in $\mathcal{S}$ at the beginning of time step t_λ.
- E_2: the set of edges evicted from $\mathcal{S}$ between the beginning of time step t_λ and the beginning of time step t_γ does not contain either g or h.

We can then write

$$\Pr(D_\gamma \mid D_\lambda) = \Pr(E_1 \mid D_\lambda) \Pr(E_2 \mid E_1 \cap D_\lambda).$$

If $t_\lambda \leq M + 1$, we have that $\Pr(E_1 \mid D_\lambda) = 1$. Consider instead the case $t_\lambda > M + 1$. If D_λ is verified, then both g and the other edge of λ are in $\mathcal{S}$ at the beginning of time step t_λ. At this time, all subsets of $E^{(t_\lambda - 1)}$ of size M and containing both g and the other edge of λ have an equal probability of being $\mathcal{S}$, from Lemma A.1. There are $\binom{t_\lambda - 3}{M-2}$ such sets. Among these, there are $\binom{t_\lambda - 4}{M-3}$ sets that also contain h. Therefore, if $t_\lambda > M + 1$, we have

$$\Pr(E_1 \mid D_\lambda) = \frac{\binom{t_\lambda - 4}{M-3}}{\binom{t_\lambda - 3}{M-2}} = \frac{M-2}{t_\lambda - 3}.$$

Considering what we said before for the case $t_\lambda \leq M + 1$, we then have

$$\Pr(E_1 \mid D_\lambda) = \min\left\{1, \frac{M-2}{t_\lambda - 3}\right\}.$$

We also have

$$\Pr(E_2 \mid E_1 \cap D_\lambda) = \prod_{j=\max\{t_\lambda, M+1\}}^{t_\gamma - 1} \frac{j-2}{j} = \frac{\max\{(t_\lambda - 2)(t_\lambda - 1), M(M-1)\}}{\max\{(t_\gamma - 2)(t_\gamma - 1), M(M-1)\}}.$$

Therefore,

$$\Pr(D_\gamma \mid D_\lambda) = \min\left\{1, \frac{M-2}{t_\lambda - 3}\right\} \frac{\max\{(t_\lambda - 2)(t_\lambda - 1), M(M-1)\}}{\max\{(t_\gamma - 2)(t_\gamma - 1), M(M-1)\}}.$$

With a case analysis, one can show that

$$\Pr(D_\gamma \mid D_\lambda) \leq \frac{1}{q_\gamma} \frac{\max\{M, t_\lambda - 1\}}{M}.$$

To recap, we have the following two scenarios when considering two distinct triangles γ and λ:

(1) If λ and γ share an edge and, assuming w.l.o.g. that the third edge of λ is on the stream no later than the third edge of γ, and the shared edge is neither the last among all edges of λ to appear on the stream nor the last among all edges of γ to

appear on the stream, then we have

$$\begin{aligned}\mathrm{Cov}[\delta_\lambda, \delta_\gamma] &\leq \mathbb{E}[\delta_\lambda \delta_\gamma] - 1 = q_\lambda q_\gamma \Pr(\delta_\lambda \delta_\gamma = q_\lambda q_\gamma) - 1 \\ &\leq q_\lambda q_\gamma \frac{1}{q_\lambda q_\gamma} \frac{\max\{M, t_\lambda - 1\}}{M} - 1 \leq \frac{\max\{M, t_\lambda - 1\}}{M} - 1 \leq \frac{t - 1 - M}{M};\end{aligned}$$

where the last inequality follows from the fact that $t_\lambda \leq t$ and $t - 1 \geq M$.

For the pairs (λ, γ) such that $t_\lambda \leq M + 1$, we have $\max\{M, t_\lambda - 1\}/M = 1$ and therefore $\mathrm{Cov}[\delta_\lambda, \delta_\gamma] \leq 0$. We should therefore only consider the pairs for which $t_\lambda > M + 1$. Their number is given by $z^{(t)}$.

(2) In all other cases, including when γ and λ do not share an edge, we have $\mathbb{E}[\delta_\lambda \delta_\gamma] \leq 1$, and since $\mathbb{E}[\delta_\lambda]\mathbb{E}[\delta_\gamma] = 1$, we have

$$\mathrm{Cov}[\delta_\lambda, \delta_\gamma] \leq 0.$$

Hence, we can bound

$$\sum_{\substack{\lambda, \gamma \in \Delta^{(t)} \\ \lambda \neq \gamma}} \mathrm{Cov}[\delta_\lambda, \delta_\gamma] \leq z^{(t)} \frac{t - 1 - M}{M}$$

and the thesis follows by combining this into Equation (12). □

4.2.5. Concentration. We now show a concentration result on the estimation of TRIÈST-IMPR, which relies on Chebyshev's inequality [31, Theorem 3.6] and Theorem 4.13.

THEOREM 4.15. *Let $t \geq 0$ and assume $|\Delta^{(t)}| > 0$. For any $\varepsilon, \delta \in (0, 1)$, if*

$$M > \max\left\{ \sqrt{\frac{2(t-1)(t-2)}{\delta\varepsilon^2|\Delta^{(t)}| + 2} + \frac{1}{4}} + \frac{1}{2}, \frac{2z^{(t)}(t-1)}{\delta\varepsilon^2|\Delta^{(t)}|^2 + 2z^{(t)}} \right\},$$

then $|\tau^{(t)} - |\Delta^{(t)}|| < \varepsilon|\Delta^{(t)}|$ with probability $> 1 - \delta$.

PROOF. By Chebyshev's inequality, it is sufficient to prove that

$$\frac{\mathrm{Var}[\tau^{(t)}]}{\varepsilon^2|\Delta^{(t)}|^2} < \delta.$$

We can write

$$\frac{\mathrm{Var}[\tau^{(t)}]}{\varepsilon^2|\Delta^{(t)}|^2} \leq \frac{1}{\varepsilon^2|\Delta^{(t)}|} \left((\eta(t) - 1) + z^{(t)} \frac{t - 1 - M}{M|\Delta^{(t)}|} \right).$$

Hence, it is sufficient to impose the following two conditions:

Condition 1

$$\begin{aligned}\frac{\delta}{2} &> \frac{\eta(t) - 1}{\varepsilon^2|\Delta^{(t)}|} \\ &> \frac{1}{\varepsilon^2|\Delta^{(t)}|} \frac{(t-1)(t-2) - M(M-1)}{M(M-1)},\end{aligned} \tag{13}$$

which is verified for

$$M(M-1) > \frac{2(t-1)(t-2)}{\delta\varepsilon^2|\Delta^{(t)}| + 2}.$$

As $t > M$, we have $\frac{2(t-1)(t-2)}{\delta\varepsilon^2|\Delta^{(t)}|+2} > 0$. The condition in Equation (13) is thus verified for

$$M > \frac{1}{2}\left(\sqrt{4\frac{2(t-1)(t-2)}{\delta\varepsilon^2|\Delta^{(t)}|+2}+1}+1\right).$$

Condition 2

$$\frac{\delta}{2} > z^{(t)}\frac{t-1-M}{M\varepsilon^2|\Delta^{(t)}|^2},$$

which is verified for

$$M > \frac{2z^{(t)}(t-1)}{\delta\varepsilon^2|\Delta^{(t)}|^2+2z^{(t)}}.$$

The theorem follows. □

In Theorems 4.13 and 4.15, it is possible to replace the value $z^{(t)}$ with the more interpretable $r^{(t)}$, which is agnostic to the order of the edges on the stream but gives a looser upper bound to the variance.

4.3. Fully Dynamic Algorithm – TRIÈST-FD

TRIÈST-FD computes unbiased estimates of the global and local triangle counts in a *fully dynamic stream where edges are inserted/deleted in any arbitrary, adversarial order*. It is based on RP [16], a sampling scheme that extends reservoir sampling and can handle deletions. The idea behind the RP scheme is that edge deletions seen on the stream will be "compensated" by future edge insertions. Following RP, TRIÈST-FD keeps a counter d_i (resp. d_o) to keep track of the number of uncompensated edge deletions involving an edge e that was (resp. was *not*) in $\mathcal{S}$ at the time the deletion for e was on the stream.

When an edge deletion for an edge $e \in E^{(t-1)}$ is on the stream at the beginning of time step t, then, if $e \in \mathcal{S}$ at this time, TRIÈST-FD removes e from $\mathcal{S}$ (effectively decreasing the number of edges stored in the sample by one) and increases d_i by one. Otherwise, it just increases d_o by one. When an edge insertion for an edge $e \notin E^{(t-1)}$ is on the stream at the beginning of time step t, if $d_\text{i}+d_\text{o}=0$, then TRIÈST-FD follows the standard reservoir sampling scheme. If $|\mathcal{S}| < M$, then e is deterministically inserted in $\mathcal{S}$ without removing any edge from $\mathcal{S}$ already in $\mathcal{S}$, otherwise it is inserted in $\mathcal{S}$ with probability M/t, replacing an uniformly chosen edge already in $\mathcal{S}$. If instead $d_\text{i}+d_\text{o} > 0$, then e is inserted in $\mathcal{S}$ with probability $d_\text{i}/(d_\text{i}+d_\text{o})$; since it must be $d_\text{i} > 0$, then it must be $|\mathcal{S}| < M$ and no edge already in $\mathcal{S}$ needs to be removed. In any case, after having handled the eventual insertion of e into $\mathcal{S}$, the algorithm decreases d_i by 1 if e was inserted in $\mathcal{S}$, otherwise it decreases d_o by 1. TRIÈST-FD also keeps track of $s^{(t)} = |E^{(t)}|$ by appropriately incrementing or decrementing a counter by 1 depending on whether the element on the stream is an edge insertion or deletion. The pseudocode for TRIÈST-FD is presented in Algorithm 3, where the UPDATECOUNTERS procedure is the one from Algorithm 1.

4.3.1. Estimation. We denote as $M^{(t)}$ the size of $\mathcal{S}$ at the end of time t (we always have $M^{(t)} \leq M$). For any time t, let $d_\text{i}^{(t)}$ and $d_\text{o}^{(t)}$ be the value of the counters d_i and d_o at the end of time t, respectively, and let $\omega^{(t)} = \min\{M, s^{(t)} + d_\text{i}^{(t)} + d_\text{o}^{(t)}\}$. Define

$$\kappa^{(t)} = 1 - \sum_{j=0}^{2}\binom{s^{(t)}}{j}\binom{d_\text{i}^{(t)}+d_\text{o}^{(t)}}{\omega^{(t)}-j}\Big/\binom{s^{(t)}+d_\text{i}^{(t)}+d_\text{o}^{(t)}}{\omega^{(t)}}. \tag{14}$$

ALGORITHM 3: TRIÈST-FD

Input: Fully dynamic edge stream Σ, integer $M \geq 6$
$\mathcal{S} \leftarrow \emptyset$, $d_{\text{i}} \leftarrow 0$, $d_{\text{o}} \leftarrow 0$, $t \leftarrow 0$, $s \leftarrow 0$
for each element $(\bullet, (u, v))$ from Σ **do**
  $t \leftarrow t + 1$
  $s \leftarrow s \bullet 1$
  if $\bullet = +$ **then**
    if SAMPLEEDGE(u, v) **then**
      UPDATECOUNTERS$(+, (u, v))$ ▷ UPDATECOUNTERS is defined as in Algorithm 1.
  else if $(u, v) \in \mathcal{S}$ **then**
    UPDATECOUNTERS$(-, (u, v))$
    $\mathcal{S} \leftarrow \mathcal{S} \setminus \{(u, v)\}$
    $d_{\text{i}} \leftarrow d_{\text{i}} + 1$
  else $d_{\text{o}} \leftarrow d_{\text{o}} + 1$

function SAMPLEEDGE(u, v)
  if $d_{\text{o}} + d_{\text{i}} = 0$ **then**
    if $|\mathcal{S}| < M$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{(u, v)\}$
      return True
    else if FLIPBIASEDCOIN$(\frac{M}{t}) =$ heads **then**
      Select (z, w) uniformly at random from $\mathcal{S}$
      UPDATECOUNTERS$\big(-, (z, w)\big)$
      $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{(z, w)\}) \cup \{(u, v)\}$
      return True
  else if FLIPBIASEDCOIN$\left(\frac{d_{\text{i}}}{d_{\text{i}}+d_{\text{o}}}\right) =$ heads **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(u, v)\}$
    $d_{\text{i}} \leftarrow d_{\text{i}} - 1$
    return True
  else
    $d_{\text{o}} \leftarrow d_{\text{o}} - 1$
    return False

For any three positive integers a, b, c s.t. $a \leq b \leq c$, define[6]

$$\psi_{a,b,c} = \binom{c}{b} \Big/ \binom{c-a}{b-a} = \prod_{i=0}^{a-1} \frac{c-i}{b-i}.$$

When queried at the end of time t, for an estimation of the global number of triangles, TRIÈST-FD returns

$$\rho^{(t)} = \begin{cases} 0 \text{ if } M^{(t)} < 3 \\ \frac{\tau^{(t)}}{\kappa^{(t)}} \psi_{3,M^{(t)},s^{(t)}} = \frac{\tau^{(t)}}{\kappa^{(t)}} \frac{s^{(t)}(s^{(t)}-1)(s^{(t)}-2)}{M^{(t)}(M^{(t)}-1)(M^{(t)}-2)} \text{ othw.} \end{cases}$$

TRIÈST-FD can keep track of $\kappa^{(t)}$ during the execution, each update of $\kappa^{(t)}$ taking time $O(1)$. Hence, the time to return the estimations is still $O(1)$.

4.3.2. Analysis. We now present the analysis of the estimations computed by TRIÈST-IMPR, showing results involving their unbiasedness, their variance, and their concentration around their expectation. Results analogous to those in Theorems 4.16, 4.17, and 4.18 hold for the local triangle count for any $u \in V^{(t)}$, replacing the global quantities with the corresponding local ones.

[6]We follow the convention that $\binom{0}{0} = 1$.

4.3.3. Expectation. Let t^* be the first $t \geq M+1$ such that $|E^{(t)}| = M+1$, if such a time step exists (otherwise $t^* = +\infty$).

THEOREM 4.16. *We have $\rho^{(t)} = |\Delta^{(t)}|$ for all $t < t^*$, and $\mathbb{E}[\rho^{(t)}] = |\Delta^{(t)}|$ for $t \geq t^*$.*

The proof, deferred to Appendix A.3, relies on properties of RP and on the definitions of $\kappa^{(t)}$ and $\rho^{(t)}$. Specifically, it uses Lemma A.6, which is the correspondent of Lemma 4.1 but for RP, and some additional technical lemmas (including an equivalent of Lemma 4.3 but for RP) and combine them using the law of total expectation by conditioning on the value of $M^{(t)}$.

4.3.4. Variance.

THEOREM 4.17. *Let $t > t^*$ s.t. $|\Delta^{(t)}| > 0$ and $s^{(t)} \geq M$. Suppose we have $d^{(t)} = d_o^{(t)} + d_i^{(t)} \leq \alpha s^{(t)}$ total unpaired deletions at time t, with $0 \leq \alpha < 1$. If $M \geq \frac{1}{2\sqrt{\alpha'-\alpha}} 7 \ln s^{(t)}$ for some $\alpha < \alpha' < 1$, we have*

$$\begin{aligned}\mathrm{Var}\left[\rho^{(t)}\right] \leq (\kappa^{(t)})^{-2}|\Delta^{(t)}|\left(\psi_{3,M(1-\alpha'),s^{(t)}} - 1\right) + (\kappa^{(t)})^{-2}2 \\ + (\kappa^{(t)})^{-2} r^{(t)}\left(\psi^2_{3,M(1-\alpha'),s^{(t)}}\psi^{-1}_{5,M(1-\alpha'),s^{(t)}} - 1\right).\end{aligned}$$

The proof of Theorem 4.17 is deferred to Appendix A.3. It uses two results on the variance of $\rho^{(t)}$ conditioned on a specific value of $M^{(t)}$ (Lemmas A.9 and A.10), and an analysis of the probability distribution of $M^{(t)}$ (Lemma A.11 and Corollary A.12). These results are then combined using the law of total variance.

4.3.5. Concentration. The following result relies on Chebyshev's inequality and on Theorem 4.17, and the proof (reported in Appendix A.3) follows the steps similar to those in the proof for Theorem 4.13.

THEOREM 4.18. *Let $t \geq t^*$ s.t. $|\Delta^{(t)}| > 0$ and $s^{(t)} \geq M$. Let $d^{(t)} = d_o^{(t)} + d_i^{(t)} \leq \alpha s^{(t)}$ for some $0 \leq \alpha < 1$. For any $\varepsilon, \delta \in (0,1)$, if for some $\alpha < \alpha' < 1$*

$$\begin{aligned}M > \max\Bigg\{ &\frac{1}{\sqrt{\alpha'-\alpha}} 7 \ln s^{(t)}, \\ &(1-\alpha')^{-1}\left(\sqrt[3]{\frac{2s^{(t)}(s^{(t)}-1)(s^{(t)}-2)}{\delta\varepsilon^2|\Delta^{(t)}|(\kappa^{(t)})^2 + 2\frac{|\Delta^{(t)}|-2}{|\Delta^{(t)}|}}} + 2\right), \\ &\frac{(1-\alpha')^{-1}}{3}\left(\frac{r^{(t)}s^{(t)}}{\delta\varepsilon^2|\Delta^{(t)}|^2(\kappa^{(t)})^{-2} + 2r^{(t)}}\right)\Bigg\},\end{aligned}$$

then $|\rho^{(t)} - |\Delta^{(t)}|| < \varepsilon|\Delta^{(t)}|$ with probability $> 1-\delta$.

4.4. Counting Global and Local Triangles in Multigraphs

We now discuss how to extend TRIÈST to approximate the local and global triangle counts in multigraphs.

4.4.1. TRIÈST-BASE on Multigraphs. TRIÈST-BASE can be adapted to work on multigraphs as follows. First of all, the sample $\mathcal{S}$ should be considered a *bag*, i.e., it may contain multiple copies of the same edge. Second, the function UPDATECOUNTERS must be changed as presented in Algorithm 4, to take into account the fact that inserting or removing an edge (u, v) from $\mathcal{S}$, respectively, increases or decreases the global number of triangles in $G^{\mathcal{S}}$ by a quantity that depends on the product of the number of edges $(c, u) \in \mathcal{S}$ and

$(c, v) \in \mathcal{S}$, for c in the shared neighborhood (in $G^{\mathcal{S}}$) of u and v (and similarly for the local number of triangles incidents to c).

ALGORITHM 4: UPDATECOUNTERS Function for TRIÈST-BASE on Multigraphs

```
1: function UpdateCounters((•, (u, v)))
2:     N^S_{u,v} ← N^S_u ∩ N^S_v
3:     for all c ∈ N^S_{u,v} do
4:         y_{c,u} ← number of edges between c and u in S
5:         y_{c,v} ← number of edges between c and v in S
6:         y_c ← y_{c,u} · y_{c,v}
7:         τ ← τ • y_c
8:         τ_c ← τ_c • y_c
9:         τ_u ← τ_u • y_c
10:        τ_v ← τ_v • y_c
```

For this modified version of TRIÈST-BASE, that we call TRIÈST-BASE-M, an equivalent version of Lemma 4.3 holds. Therefore, we can prove a result on the unbiasedness of TRIÈST-BASE-M equivalent (i.e., with the same statement) as Theorem 4.2. The proof of such result is also the same as the one for Theorem 4.2.

To analyze the variance of TRIÈST-BASE-M, we need to take into consideration the fact that, in a multigraph, a pair of triangles may share *two* edges, and the variance depends (also) on the number of such pairs. Let $r_1^{(t)}$ be the number of unordered pairs of distinct triangles from $\Delta^{(t)}$ sharing an edge and let $r_2^{(t)}$ be the number of unordered pairs of distinct triangles from $\Delta^{(t)}$ sharing *two* edges (such pairs may exist in a multigraph, but not in a simple graph). Let $q^{(t)} = \binom{|\Delta^{(t)}|}{2} - r_1^{(t)} - r_2^{(t)}$ be the number of unordered pairs of distinct triangles that do not share any edge.

THEOREM 4.19. *For any $t > M$, let $f(t) = \xi^{(t)} - 1$,*

$$g(t) = \xi^{(t)} \frac{(M-3)(M-4)}{(t-3)(t-4)} - 1$$

and

$$h(t) = \xi^{(t)} \frac{(M-3)(M-4)(M-5)}{(t-3)(t-4)(t-5)} - 1 \ (\leq 0),$$

and

$$j(t) = \xi^{(t)} \frac{M-3}{t-3} - 1.$$

We have

$$\mathrm{Var}\left[\xi(t)\tau^{(t)}\right] = |\Delta^{(t)}| f(t) + r_1^{(t)} g(t) + r_2^{(t)} j(t) + q^{(t)} h(t).$$

The proof follows the same lines as the one for Theorem 4.4, with the additional steps needed to take into account the contribution of the $r_2^{(t)}$ pairs of triangles in $G^{(t)}$ sharing two edges.

4.4.2. TRIÈST-IMPR on Multigraphs. A variant TRIÈST-IMPR-M of TRIÈST-IMPR for multigraphs can be obtained by using the function UPDATECOUNTERS defined in Algorithm 4, modified to increment[7] the counters by $\eta^{(t)} y_c^{(t)}$, rather than $y_c^{(t)}$, where

[7]As in TRIÈST-IMPR, all calls to UPDATECOUNTERS in TRIÈST-IMPR-M have $\bullet = +$. See also Algorithm 2.

$\eta^{(t)} = \max\{1, (t-1)(t-2)/(M(M-1))\}$. The result stated in Theorem 4.12 holds also for the estimations computed by TRIÈST-IMPR-M. An upper bound to the variance of the estimations, similar to the one presented in Theorem 4.13 for TRIÈST-IMPR, could potentially be obtained, but its derivation would involve a high number of special cases, as we have to take into consideration the order of the edges in the stream.

4.4.3. TRIÈST-FD on Multigraphs. TRIÈST-FD can be modified in order to provide an approximation of the number of global and local triangles on multigraphs observed as a stream of edge deletions and deletions. It is, however, necessary to clearly state the data model. We assume that for all pairs of vertices $u, v \in V^{(t)}$, each edge connecting u and v is assigned a label that is unique among the edges connecting u and v. An edge is therefore uniquely identified by its endpoints and its label as $((u, v), label)$. Elements of the stream are now in the form $(\bullet, (u, v), label)$, where $\bullet$ is either $+$ or $-$. This assumption, somewhat strong, is necessary in order to apply the *RP* sampling scheme [16] to fully dynamic multigraph edge streams.

Within this model, we can obtain an algorithm TRIÈST-FD-M for multigraphs by adapting TRIÈST-FD as follows. The sample $\mathcal{S}$ is a *set* of elements $((u, v), label)$. When a deletion $(-, (u, v), label)$ is on the stream, the sample $\mathcal{S}$ is modified if and only if $((u, v), label)$ belongs to $\mathcal{S}$. This change can be implemented in the pseudocode from Algorithm 3 by modifying line 8 to be

"**else if** $((u, v), label) \in \mathcal{S}$ **then.**"

Additionally, the function UPDATECOUNTERS to be used is the one presented in Algorithm 4.

We can prove a result on the unbiasedness of TRIÈST-FD-M equivalent (i.e., with the same statement) as Theorem 4.16. The proof of such result is also the same as the one for Theorem 4.16. An upper bound to the variance of the estimations, similar to the one presented in Theorem 4.17 for TRIÈST-FD, could be obtained by considering the fact that in a multigraph two triangles can share two edges, in a fashion similar to what we discussed in Theorem 4.19.

4.5. Discussion

We now briefly discuss over the algorithms we just presented, the techniques they use, and the theoretical results we obtained for TRIÈST, in order to highlight advantages, disadvantages, and limitations of our approach.

On reservoir sampling. Our approach of using reservoir sampling to keep a random sample of edges can be extended to many other graph mining problems, including approximate counting of other subgraphs more or less complex than triangles (e.g., squares, trees with a specific structure, wedges, cliques, and so on). The estimations of such counts would still be unbiased, but as the number of edges composing the subgraph(s) of interest increases, the variance of the estimators also increases, because the probability that all edges composing a subgraph are in the sample (or all but the last one when the last one arrives, as in the case of TRIÈST-IMPR), decreases as their number increases. Other works in the triangle counting literature [20, 36] use samples of wedges, rather than edges. They perform worse than TRIÈST in both accuracy and runtime (see Section 5), but the idea of sampling and storing more complex structures rather than simple edges could be a potential direction for approximate counting of larger subgraphs.

On the analysis of the variance. We showed an exact analysis of the variance of TRIÈST-BASE but for the other algorithms we presented *upper bounds* to the variance of the estimates. These bounds can still be improved as they are not currently tight. For

example, we already commented on the fact that the bound in Equation (11) does not include a number of negative terms that would tighten it (i.e., decrease the bound), and that could potentially be no smaller than the term depending on $z^{(t)}$. The absence of such terms is due to the fact that it seems very challenging to obtain non-trivial *upper bounds* to them that are valid for every $t > M$. Our proof for this bound uses a careful case-by-case analysis, considering the different situations for pair of triangles (e.g., sharing or not sharing an edge, and considering the order of edges on the stream). It may be possible to obtain tighter bounds to the variance by following a more holistic approach that takes into account the fact that the sizes of the different classes of triangle pairs are highly dependent on each other.

Another issue with the bound to the variance from Equation (11) is that the quantity $z^{(t)}$ depends on the order of edges on the stream. As already discussed, the bound can be made independent of the order by loosening it even more. Very recent developments in the sampling theory literature [12] presented sampling schemes and estimators whose second-order sampling probabilities do not depend on the order of the stream, so it should be possible to obtain such bounds also for the triangle counting problem, but a sampling scheme different than reservoir sampling would have to be used, and a careful analysis is needed to establish its net advantages in terms of performances and scalability to billion-edges graphs.

On the tradeoff between speed and accuracy. We concluded both previous paragraphs in this subsection by mentioning techniques different than reservoir sampling of edges as potential directions to improve and extend our results. In both the cases, these techniques are more complex not only in their analysis but also *computationally*. Given that the main goal of algorithms like TRIÈST is to make it possible to analyze graphs with billions (and possibly more) nodes, the gain in accuracy need to be weighted against expected slowdowns in execution. As we show in our experimental evaluation in the next section, TRIÈST, especially in the TRIÈST-IMPR variant, actually seems to strike the right balance between accuracy and tradeoff, when compared with existing contributions.

5. EXPERIMENTAL EVALUATION

We evaluated TRIÈST on several real-world graphs with up to a billion edges. The algorithms were implemented in C++, and ran on the Brown University CS department cluster.[8] Each run employed a single core and used at most 4GB of RAM. The code is available from http://bigdata.cs.brown.edu/triangles.html. Most of this section is related to experiments on graphs, while results for multigraphs are described in Section 5.3.

Datasets. We created the streams from the following publicly available graphs (properties in Table II).

- Patent (Co-Aut.) and Patent (Cit.). The *Patent (Co-Aut.)* and *Patent (Cit.)* graphs are obtained from a dataset of ≈2 million U.S. patents granted between '75 and '99 [18]. In *Patent (Co-Aut.)*, the nodes represent inventors and there is an edge with timestamp t between two co-inventors of a patent if the patent was granted in year t. In *Patent (Cit.)*, nodes are patents and there is an edge (a, b) with timestamp t if patent a cites b and a was granted in year t.
- LastFm. The LastFm graph is based on a dataset [8, 39] of ≈20 million last.fm song listenings, ≈1 million songs, and ≈1,000 users. There is a node for each song and an edge between two songs if ≥3 users listened to both on day t.

[8]https://cs.brown.edu/about/system/services/hpc/grid/.

Table II. Properties of the Dynamic Graph Streams Analyzed. $|V|$, $|E|$, $|E_u|$, $|\Delta|$ Refer, Respectively, to the Number of Nodes in the Graph, the Number of Edge Addition Events, the Number of Distinct Edges Additions, and the Maximum Number of Triangles in the Graph (For Yahoo! Answers and Twitter Estimated with TRIÈST-IMPR with $M = 1{,}000{,}000$, Otherwise Computed Exactly with the Naïve Algorithm)

Graph	$\|V\|$	$\|E\|$	$\|E_u\|$	$\|\Delta\|$
Patent (Co-Aut.)	1,162,227	3,660,945	2,724,036	3.53×10^{06}
Patent (Cit.)	2,745,762	13,965,410	13,965,132	6.91×10^{06}
LastFm	681,387	43,518,693	30,311,117	1.13×10^{09}
Yahoo! Answers	2,432,573	1.21×10^{9}	1.08×10^{9}	7.86×10^{10}
Twitter	41,652,230	1.47×10^{9}	1.20×10^{9}	3.46×10^{10}

Yahoo!-Answers. The Yahoo! Answers graph is obtained from a sample of ≈160 million answers to ≈25 millions questions posted on Yahoo! Answers [10]. An edge connects two users at time $\max(t_1, t_2)$ if they both answered the same question at times t_1, t_2, respectively. We removed six outliers questions with more than 5,000 answers.

Twitter. This is a snapshot [5, 25] of the Twitter followers/following network with ≈41 million nodes and ≈1.5 billions edges. We do not have time information for the edges, hence we assign a random timestamp to the edges (of which we ignore the direction).

Ground truth. To evaluate the accuracy of our algorithms, we computed the *ground truth* for our smaller graphs (i.e., the exact number of global and local triangles for each time step), using an exact algorithm. The entire current graph is stored in memory and when an edge u, v is inserted (or deleted) we update the current count of local and global triangles by checking how many triangles are completed (or broken). As exact algorithms are not scalable, computing the exact triangle count is feasible only for small graphs such as Patent (Co-Aut.), Patent (Cit.), and LastFm. Table II reports the exact total number of triangles at the end of the stream for those graphs (and an estimate for the larger ones using TRIÈST-IMPR with $M = 1{,}000{,}000$).

5.1. Insertion-only Case

We now evaluate TRIÈST on insertion-only streams and compare its performances with those of state-of-the-art approaches [20, 28, 36], showing that TRIÈST has an average estimation error significantly smaller than these methods both for the global and local estimation problems, while using the same amount of memory.

Estimation of the global number of triangles. Starting from an empty graph we add one edge at a time, in timestamp order. Figure 1 illustrates the evolution, over time, of the estimation computed by TRIÈST-IMPR with $M = 1{,}000{,}000$. For smaller graphs for which the ground truth can be computed exactly, the curve of the exact count is practically indistinguishable from TRIÈST-IMPR estimation, showing the precision of the method. The estimations have very small variance even on the very large Yahoo! Answers and Twitter graphs (point-wise max/min estimation over ten runs is almost coincident with the average estimation). These results show that TRIÈST-IMPR is very accurate even when storing less than a 0.001 fraction of the total edges of the graph.

Comparison with the state of the art. We compare quantitatively with three state-of-the-art methods: MASCOT [28], Jha et al. [20], and Pavan et al. [36]. MASCOT is a suite of local triangle counting methods (but provides also a global estimation). The other two are global triangle counting approaches. None of these can handle fully dynamic streams, in contrast with TRIÈST-FD. We first compare the three methods to TRIÈST for the

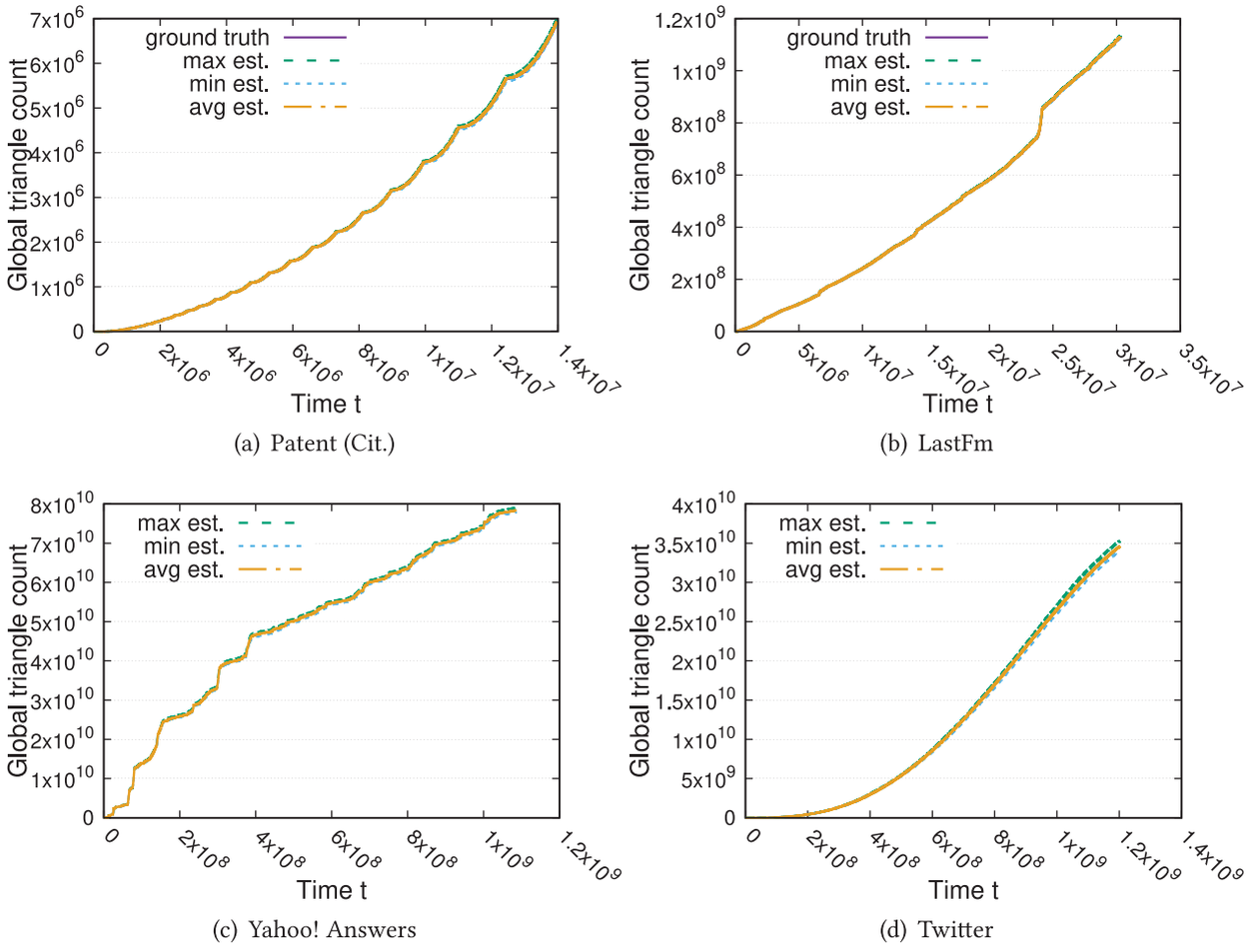
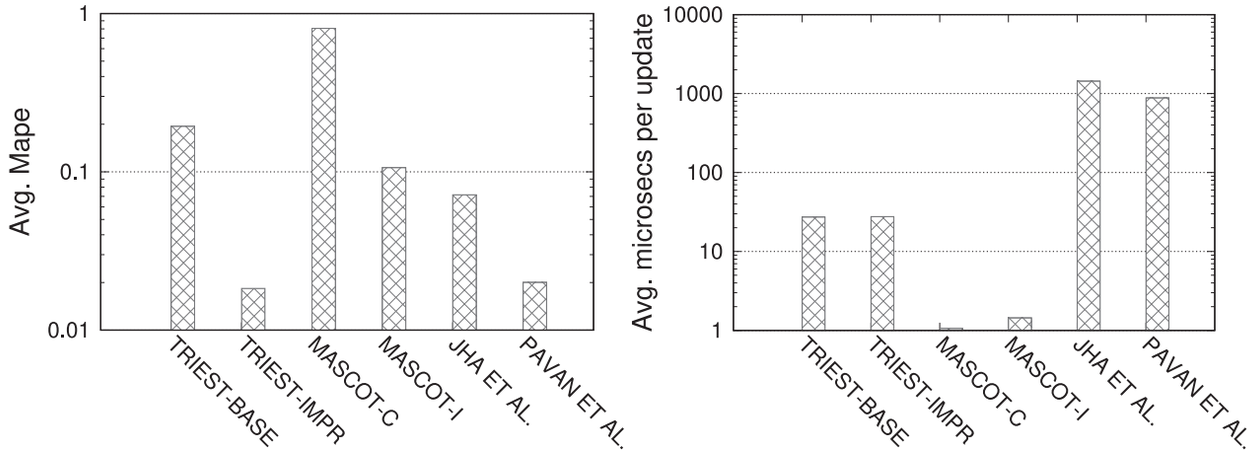


Fig. 1. Estimation by TRIÈST-IMPR of the global number of triangles over time (intended as number of elements seen on the stream). The max, min, and avg are taken over 10 runs. The curves are *indistinguishable on purpose*, to highlight the fact that TRIÈST-IMPR estimations have very small error and variance. For example, the ground truth (for graphs for which it is available) is indistinguishable even from the max/min point-wise estimations over ten runs. For graphs for which the ground truth is not available, the small deviations from the avg suggest that the estimations are also close to the true value, given that our algorithms gives unbiased estimations.

global triangle counting estimation. MASCOT comes in two memory efficient variants: the basic MASCOT-C variant and an improved MASCOT-I variant.[9] Both variants sample edges with fixed probability p, so there is no guarantee on the amount of memory used during the execution. To ensure fairness of comparison, we devised the following experiment. First, we run both MASCOT-C and MASCOT-I for $\ell = 10$ times with a fixed p using the same random bits for the two algorithms run-by-run (i.e., the same coin tosses used to select the edges) measuring each time the number of edges M_i' stored in the sample at the end of the stream (by construction this is same for the two variants run-by-run). Then, we run our algorithms using $M = M_i'$ (for $i \in [\ell]$). We do the same to fix the size of the edge memory for Jha et al. [20] and Pavan et al. [36].[10] This way, all algorithms use the same amount of memory for storing edges (run-by-run).

[9]In the original work [28], this variant had no suffix and was simply called MASCOT. We add the -I suffix to avoid confusion. Another variant MASCOT-A can be forced to store the entire graph with probability 1 by appropriately selecting the edge order (which we assume to be adversarial) so we do not consider it here.
[10]More precisely, we use $M_i'/2$ estimators in Pavan et al. as each estimator stores two edges. For Jha et al., we set the two reservoirs in the algorithm to have each size $M_i'/2$. This way, all algorithms use M_i' cells for storing (w)edges.



(a) MAPE (b) Update Time

Fig. 2. Average MAPE and average update time of the various methods on the Patent (Co-Aut.) graph with $p = 0.01$ (for MASCOT, see the main text for how we computed the space used by the other algorithms) – insertion only. TRIÈST-IMPR has the lowest error. Both Pavan et al. and Jha et al. have very high update times compared to our method and the two MASCOT variants.

We use the *MAPE* (Mean Average Percentage Error) to assess the accuracy of the global triangle estimators over time. The MAPE measures the average percentage of the prediction error with respect to the ground truth, and is widely used in the prediction literature [19]. For $t = 1, \ldots, T$, let $\overline{\Delta}^{(t)}$ be the estimator of the number of triangles at time t, the MAPE is defined as $\frac{1}{T}\sum_{t=1}^{T} |\frac{|\Delta^{(t)}| - \overline{\Delta}^{(t)}}{|\Delta^{(t)}|}|$.[11]

In Figure 2(a), we compare the average MAPE of TRIÈST-BASE and TRIÈST-IMPR as well as the two MASCOT variants and the other two streaming algorithms for the Patent (Co-Aut.) graph, fixing $p = 0.01$. TRIÈST-IMPR has the smallest error of all the algorithms compared.

We now turn our attention to the efficiency of the methods. Whenever we refer to one operation, we mean handling one element on the stream, either one edge addition or one edge deletion. The average update time per operation is obtained by dividing the total time required to process the entire stream by the number of operations (i.e., elements on the streams).

Figure 2(b) shows the average update time per operation in Patent (Co-Aut.) graph, fixing $p = 0.01$. Both Jha et al. [20] and Pavan et al. [36] are up to $\approx$3 orders of magnitude slower than the MASCOT variants and TRIÈST. This is expected as both algorithms have an update complexity of $\Omega(M)$ (they have to go through the entire reservoir graph at each step), while both MASCOT algorithms and TRIÈST need only to access the neighborhood of the nodes involved in the edge addition.[12] This allows both algorithms to efficiently exploit larger memory sizes. We can use efficiently M up to 1 million edges in our experiments, which only requires few megabytes of RAM.[13] MASCOT is one order of magnitude faster than TRIÈST (which runs in $\approx$28 micros/op), because it does not have to handle edge removal from the sample, as it offers no guarantees on the used memory.

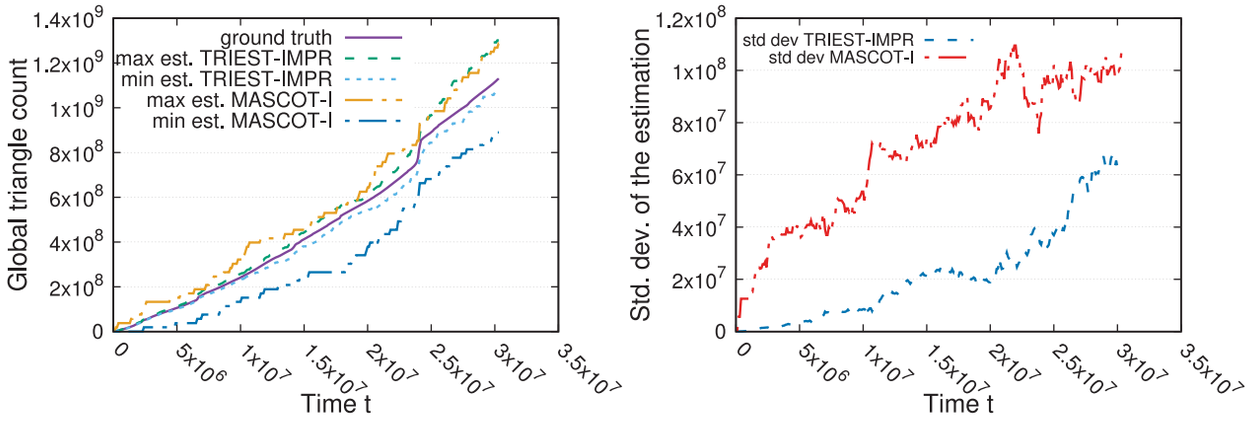
[11]The MAPE is not defined for t s.t. $\Delta^{(t)} = 0$ so we compute it only for t s.t. $|\Delta^{(t)}| > 0$. All algorithms we consider are guaranteed to output the correct answer for t s.t. $|\Delta^{(t)}| = 0$.

[12]We observe that Pavan et al. [36] would be more efficient with batch updates. However, we want to estimate the triangles continuously at each update. In their experiments they use batch sizes of million of updates for efficiency.

[13]The experiments by Jha et al. [20] use M in the order of 10^3, and in those by Pavan et al. [36], large M values require large batches for efficiency.

Table III. Global Triangle Estimation MAPE for TRIÈST and MASCOT. The Rightmost Column Shows the Reduction in Terms of the Avg. MAPE Obtained by Using TRIÈST. Rows with *Y* in Column "Impr." Refer to Improved Algorithms (TRIÈST-IMPR and MASCOT-I) While Those with *N* to Basic Algorithms (TRIÈST-BASE and MASCOT-C)

Graph	Impr.	p	Max. MAPE		Avg. MAPE		
			MASCOT	TRIÈST	MASCOT	TRIÈST	Change
Patent (Cit.)	N	0.01	0.9231	0.2583	0.6517	0.1811	−72.2%
	Y	0.01	0.1907	0.0363	0.1149	0.0213	−81.4%
	N	0.1	0.0839	0.0124	0.0605	0.0070	−88.5%
	Y	0.1	0.0317	0.0037	0.0245	0.0022	−91.1%
Patent (Co-aut.)	N	0.01	2.3017	0.3029	0.8055	0.1820	−77.4%
	Y	0.01	0.1741	0.0261	0.1063	0.0177	−83.4%
	N	0.1	0.0648	0.0175	0.0390	0.0079	−79.8%
	Y	0.1	0.0225	0.0034	0.0174	0.0022	−87.2%
LastFm	N	0.01	0.1525	0.0185	0.0627	0.0118	−81.2%
	Y	0.01	0.0273	0.0046	0.0141	0.0034	−76.2%
	N	0.1	0.0075	0.0028	0.0047	0.0015	−68.1%
	Y	0.1	0.0048	0.0013	0.0031	0.0009	−72.1%



(a) Ground truth, max, and min

(b) Standard deviation

Fig. 3. Accuracy and stability of the estimation of TRIÈST-IMPR with $M = 10{,}000$ and of MASCOT-I with same expected memory, on LastFM, over 10 runs. TRIÈST-IMPR has a smaller standard deviation and moreover the max/min estimation lines are closer to the ground truth. Average estimations not shown as they are qualitatively similar.

As we will show, TRIÈST has much higher precision and scales well on billion-edges graphs.

Given the slow execution of the other algorithms on the larger datasets, we compare in details TRIÈST only with MASCOT.[14] Table III shows the average MAPE of the two approaches. The results confirm the pattern observed in Figure 2(a): TRIÈST-BASE and TRIÈST-IMPR both have an average error significantly smaller than that of the basic MASCOT-C and improved MASCOT variant, respectively. We achieve up to a 91% (i.e., 9-fold) reduction in the MAPE while using the same amount of memory. This experiment confirms the theory: Reservoir sampling has overall lower or equal variance in all steps for the same expected total number of sampled edges.

To further validate this observation, we run TRIÈST-IMPR and the improved MASCOT-I variant using the same (expected memory) $M = 10,000$. Figure 3 shows the max–min estimation over 10 runs and the standard deviation of the estimation over those runs. TRIÈST-IMPR shows significantly lower standard deviation (hence variance) over the evolution of the stream, and the max and min lines are also closer to the ground truth.

[14]We attempted to run the other two algorithms but they did not complete after 12 hours for the larger datasets in Table III with the prescribed p parameter setting.

Table IV. Comparison of the Quality of the Local Triangle Estimations between Our Algorithms and the State-of-the-Art Approach in Lim and Kang [28]. Rows with *Y* in Column "Impr." Refer to Improved Algorithms (TRIÈST-IMPR and MASCOT-I) While Those with *N* to Basic Algorithms (TRIÈST-BASE and MASCOT-C). In Virtually All Cases we Significantly Outperform MASCOT Using the Same Amount of Memory

Graph	Impr.	p	Avg. Pearson			Avg. ε Err.		
			MASCOT	TRIÈST	Change	MASCOT	TRIÈST	Change
LastFm	*Y*	0.1	0.99	1.00	+1.18%	0.79	0.30	−62.02%
		0.05	0.97	1.00	+2.48%	0.99	0.47	−52.79%
		0.01	0.85	0.98	+14.28%	1.35	0.89	−34.24%
	N	0.1	0.97	0.99	+2.04%	1.08	0.70	−35.65%
		0.05	0.92	0.98	+6.61%	1.32	0.97	−26.53%
		0.01	0.32	0.70	+117.74%	1.48	1.34	−9.16%
Patent (Cit.)	*Y*	0.1	0.41	0.82	+99.09%	0.62	0.37	−39.15%
		0.05	0.24	0.61	+156.30%	0.65	0.51	−20.78%
		0.01	0.05	0.18	+233.05%	0.65	0.64	−1.68%
	N	0.1	0.16	0.48	+191.85%	0.66	0.60	−8.22%
		0.05	0.06	0.24	+300.46%	0.67	0.65	−3.21%
		0.01	0.00	0.003	+922.02%	0.86	0.68	−21.02%
Patent (Co-aut.)	*Y*	0.1	0.55	0.87	+58.40%	0.86	0.45	−47.91%
		0.05	0.34	0.71	+108.80%	0.91	0.63	−31.12%
		0.01	0.08	0.26	+222.84%	0.96	0.88	−8.31%
	N	0.1	0.25	0.52	+112.40%	0.92	0.83	−10.18%
		0.05	0.09	0.28	+204.98%	0.92	0.92	0.10%
		0.01	0.01	0.03	+191.46%	0.70	0.84	20.06%

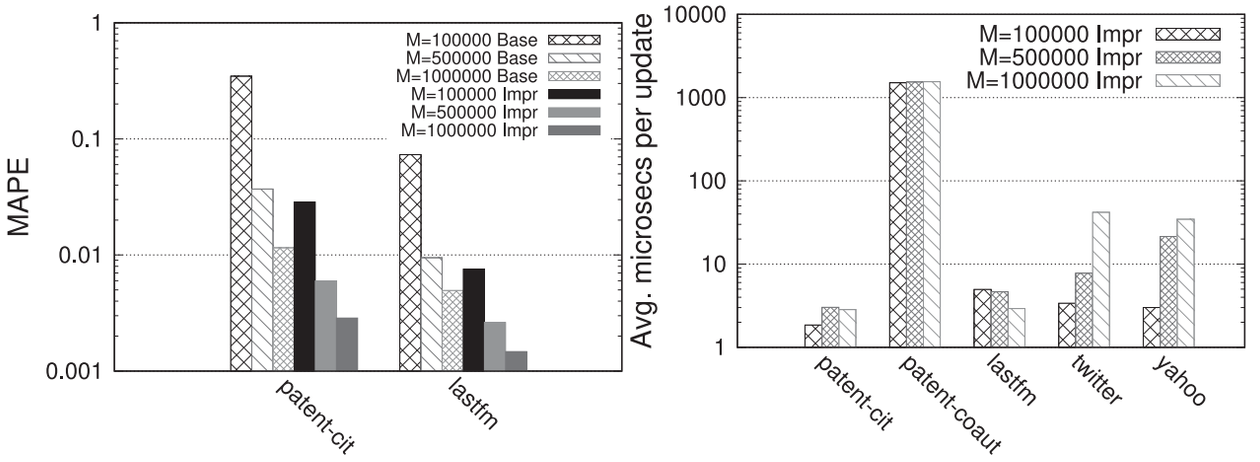
This confirms our theoretical observations in the previous sections. Even with very low M (about 2/10,000 of the size of the graph) TRIÈST gives high-quality estimations.

Local triangle counting. We compare the precision in local triangle count estimation of TRIÈST with that of MASCOT [28] using the same approach of the previous experiment. We can not compare with Jha et al. and Pavan et al. algorithms as they provide only global estimation. As in Lim and Kang [28], we measure the Pearson coefficient and the average ε error (see Lim and Kang [28] for definitions). In Table IV, we report the Pearson coefficient and average ε error over all timestamps for the smaller graphs.[15] TRIÈST (significantly) improves (i.e., has higher correlation and lower error) over the state-of-the-art MASCOT, using the same amount of memory.

Tradeoffs between memory and accuracy. We study the tradeoffs between the sample size M, the running time, and the accuracy of the estimators. Figure 4(a) shows the tradeoffs between the accuracy of the estimation (as MAPE) and the size M for the smaller graphs for which the ground truth number of triangles can be computed exactly using the naive algorithm. Even with small M, TRIÈST-IMPR achieves very low MAPE value. As expected, larger M corresponds to higher accuracy and for the same M TRIÈST-IMPR outperforms TRIÈST-BASE.

Figure 4(b) shows the average time per update in microseconds (μs) for TRIÈST-IMPR as function of M. Some considerations on the running time are in order. First, a larger edge sample (larger M) generally requires longer average update times per operation. This is expected as a larger sample corresponds to a larger sample graph on which to count triangles. Second, on average a few hundreds microseconds are sufficient for handling any update even in very large graphs with billions of edges. Our algorithms can handle hundreds of thousands of edge updates (stream elements) per second, with very small

[15]For efficiency, in this test, we evaluate the local number of triangles of all nodes every 1,000 edge updates.



(a) Trade-off between M and MAPE

(b) Trade-off bewtween M and average time per update (TRIÈST-IMPR)

Fig. 4. Tradeoffs between M and MAPE and average time per update in μs – edge insertion only. Larger M implies lower errors but generally higher update times.

error (Figure 4(a)), and therefore TRIÈST can be used efficiently and effectively in high-velocity contexts. The larger average time per update for Patent (Co-Auth.) can be explained by the fact that the graph is relatively dense and has a small size (compared to the larger Yahoo! and Twitter graphs). More precisely, the average time per update (for a fixed M) depends on two main factors: the average degree and the length of the stream. The denser the graph is, the higher the update time as more operations are needed to update the triangle count every time the sample is modified. On the other hand, the longer the stream, for a fixed M, the lower is the frequency of updates to the reservoir (it can be show that the expected number of updates to the reservoir is $O(M(1 + \log(\frac{t}{M})))$, which grows sublinearly in the size of the stream t). This explains why the average update time for the large and dense Yahoo! and Twitter graphs is so small, allowing the algorithm to scale to billions of updates.

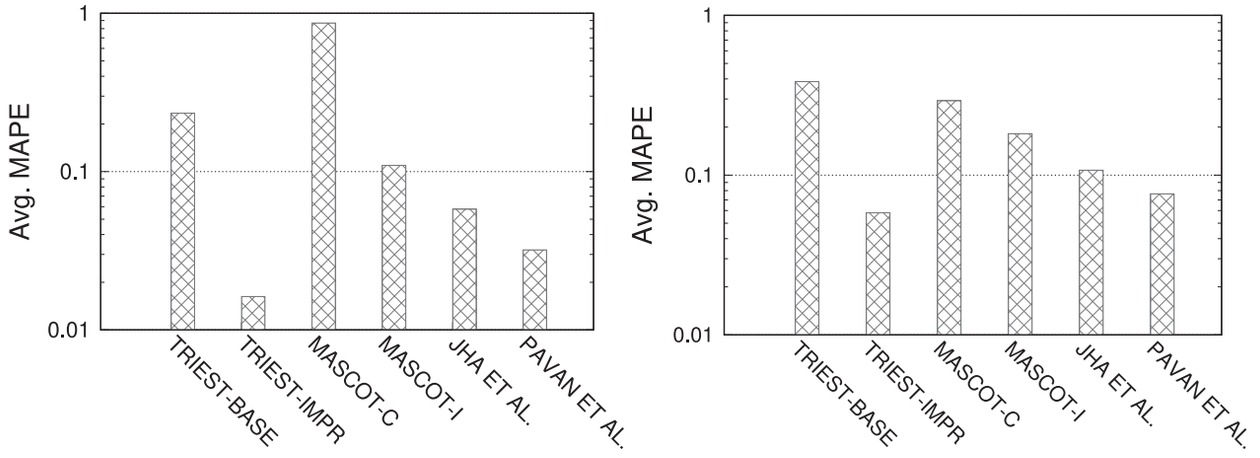
Alternative edge orders. In all previous experiments, the edges are added in their natural order (i.e., in order of their appearance).[16] While the natural order is the most important use case, we have assessed the impact of other ordering on the accuracy of the algorithms. We experiment with both the uniform-at-random (u.a.r.) order of the edges and the random BFS order: Until all the graph is explored, a BFS is started from a u.a.r. unvisited node and edges are added in order of their visit (neighbors are explored in u.a.r. order). The results for the random BFS order and u.a.r. order (Figure 5) confirm that TRIÈST has the lowest error and is very scalable in every tested ordering.

5.2. Fully Dynamic Case

We evaluate TRIÈST-FD on fully dynamic streams. We cannot compare TRIÈST-FD with the algorithms previously used [20, 28, 36] as they only handle insertion-only streams.

In the first set of experiments, we model deletions using the widely used *sliding window model*, where a sliding window of the most recent edges defines the current graph. The sliding window model is of practical interest as it allows to observe recent trends in the stream. For Patent (Co-Aut.) & (Cit.), we keep in the sliding window the edges generated in the last 5 years, while for LastFm we keep the edges generated

[16]Excluding Twitter for which we used the random order, given the lack of timestamps.



(a) BFS order (b) Uniform-at-random order

Fig. 5. Average MAPE on Patent (Co-Aut.), with $p = 0.01$ (for MASCOT, see the main text for how we computed the space used by the other algorithms) – insertion only in Random BFS order and in uniform-at-random order. TRIÈST-IMPR has the lowest error.

in the last 30 days. For Yahoo! Answers, we keep the last 100 millions edges in the window.[17]

Figure 6 shows the evolution of the global number of triangles in the sliding window model using TRIÈST-FD using $M = 200{,}000$ ($M = 1{,}000{,}000$ for Yahoo! Answers). The sliding window scenario is significantly more challenging than the addition-only case (very often the entire sample of edges is flushed away) but TRIÈST-FD maintains good variance and scalability even when, as for LastFm and Yahoo! Answers, the global number of triangles varies quickly.

Continuous monitoring of triangle counts with TRIÈST-FD allows to detect patterns that would otherwise be difficult to notice. For LastFm (Figure 6(c)), we observe a sudden spike of several order of magnitudes. The dataset is anonymized so we cannot establish which songs are responsible for this spike. In Yahoo! Answers (Figure 6(d)), a popular topic can create a sudden (and shortly lived) increase in the number of triangles, while the evolution of the Patent co-authorship and co-citation networks is slower, as the creation of an edge requires filing a patent (Figure 6(a) and (b)). The almost constant increase over time[18] of the number of triangles in Patent graphs is consistent with previous observations of *densification* in collaboration networks as in the case of nodes' degrees [27] and the observations on the density of the densest subgraph [15].

Table V shows the results for both the local and global triangle counting estimation provided by TRIÈST-FD. In this case, we cannot compare with previous works, as they only handle insertions. It is evident that precision improves with M values, and even relatively small M values result in a low MAPE (global estimation), high Pearson correlation, and low ε error (local estimation). Figure 7 shows the tradeoffs between memory (i.e., accuracy) and time. In all cases, our algorithm is very fast and it presents update times in the order of hundreds of microseconds for datasets with billions of updates (Yahoo! Answers).

Alternative models for deletion. We evaluate TRIÈST-FD using other models for deletions than the sliding window model. To assess the resilience of the algorithm to massive deletions, we run the following experiments. We added edges in their natural order

[17]The sliding window model is not interesting for the Twitter dataset as edges have random timestamps. We omit the results for Twitter but TRIÈST-FD is fast and has low variance.

[18]The decline at the end is due to the removal of the last edges from the sliding window after there are no more edge additions.

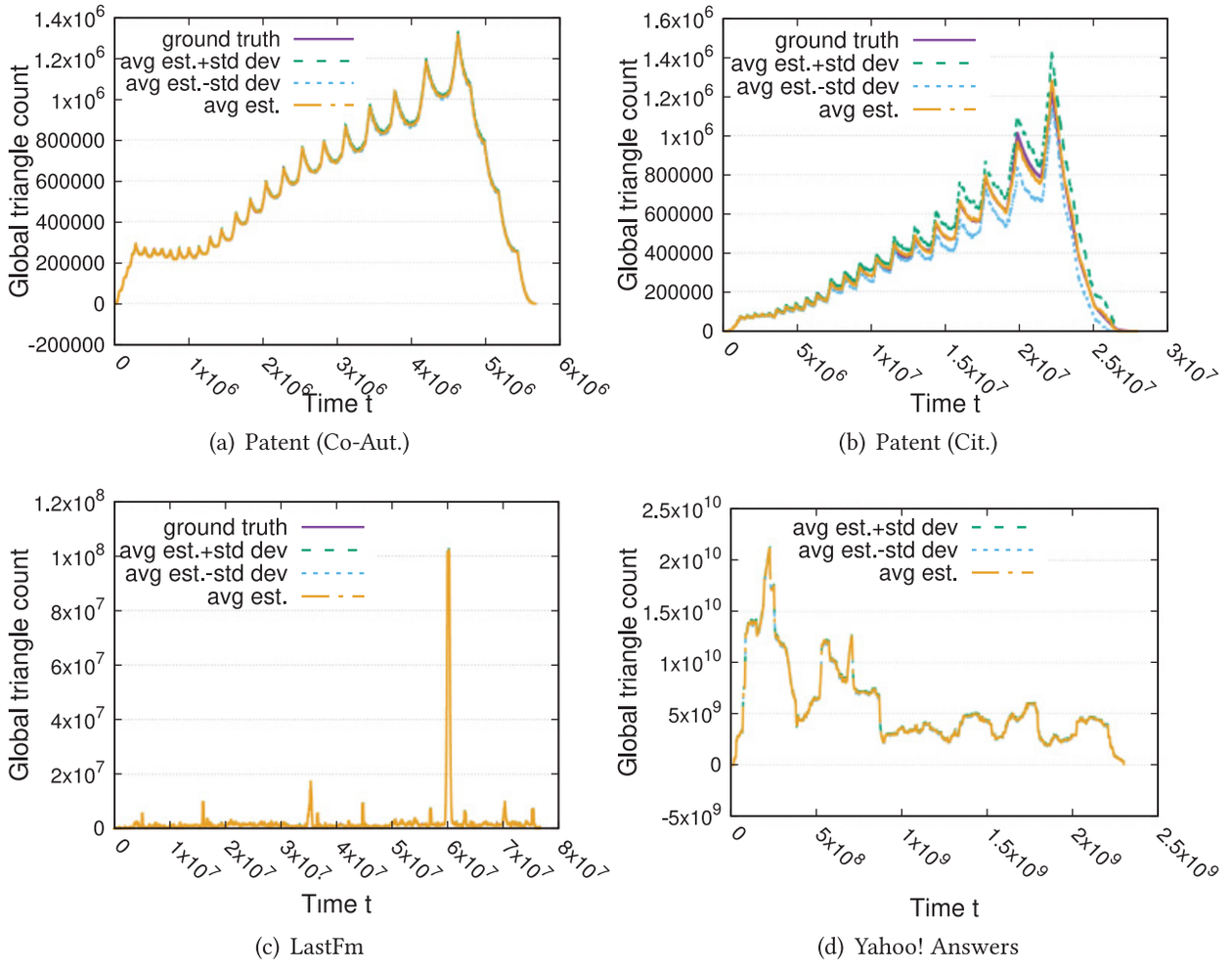


Fig. 6. Evolution of the global number of triangles in the fully dynamic case (sliding window model for edge deletion). The curves are *indistinguishable on purpose*, to remark the fact that TRIÈST-FD estimations are extremely accurate and consistent. We comment on the observed patterns in the text.

Table V. Estimation Errors for TRIÈST-FD

		Avg. Global	Avg. Local	
Graph	M	MAPE	Pearson	ε Err.
LastFM	200,000	0.005	0.980	0.020
	1,000,000	0.002	0.999	0.001
Patent (Co-Aut.)	200,000	0.010	0.660	0.300
	1,000,000	0.001	0.990	0.006
Patent (Cit.)	200,000	0.170	0.090	0.160
	1,000,000	0.040	0.600	0.130

but each edge addition is followed with probability q by a mass deletion event where each edge currently in the graph is deleted with probability d independently. We run experiments with $q = 3{,}000{,}000^{-1}$ (i.e., a mass deletion expected every 3 millions edges) and $d = 0.80$ (in expectation 80% of edges are deleted). The results are shown in Table VI.

We observe that TRIÈST-FD maintains a good accuracy and scalability even in face of a massive (and unlikely) deletions of the vast majority of the edges: For example, for LastFM with $M = 200{,}000$ (resp. $M = 1{,}000{,}000$), we observe 0.04 (resp. 0.006) Avg. MAPE.

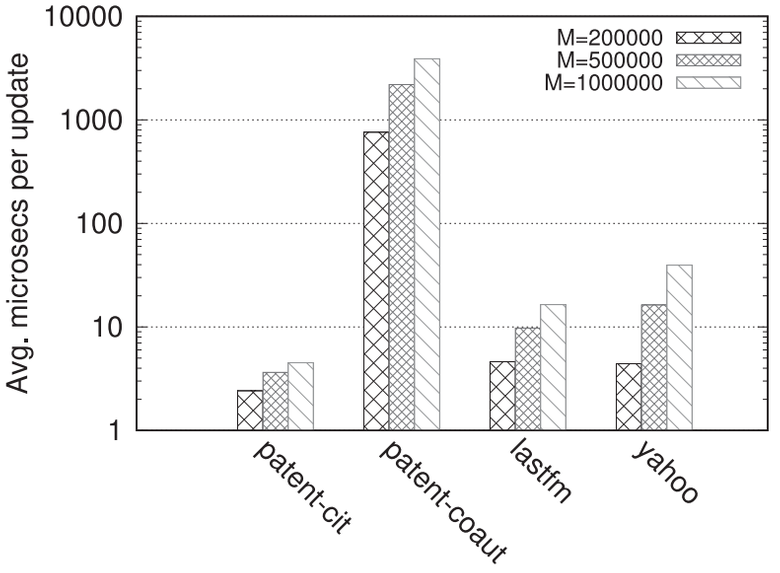


Fig. 7. Tradeoffs between the avg. update time (μs) and M for TRIÈST-FD.

Table VI. Estimation Errors for TRIÈST-FD – Mass Deletion Experiment, $q = 3{,}000{,}000^{-1}$ and $d = 0.80$

Graph	M	Avg. Global MAPE	Avg. Local Pearson	Avg. Local ε Err.
LastFM	200,000	0.040	0.620	0.53
	1,000,000	0.006	0.950	0.33
Patent (Co-Aut.)	200,000	0.060	0.278	0.50
	1,000,000	0.006	0.790	0.21
Patent (Cit.)	200,000	0.280	0.068	0.06
	1,000,000	0.026	0.510	0.04

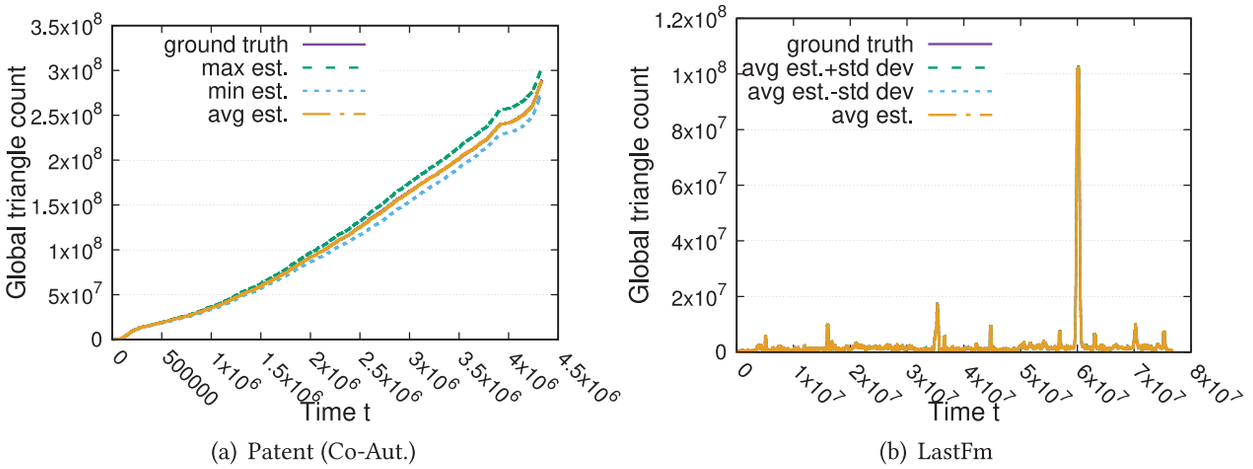
5.3. Multigraphs

We now evaluate our algorithms designed for multigraphs. We obtained multigraph versions of Patent (Co-Auth.) (resp. LastFM) by allowing multiple edges to be placed between pairs of authors (resp. songs) at multiple time steps (i.e., edges with different timestamps) if the two authors co-author multiple papers (resp. the songs are co-listened on different dates). We ran our insertion-only algorithms on these multigraphs and report the results in the next paragraphs.

Figure 8 shows the evolution of the number of triangles in the two datasets as estimated by our TRIÈST-IMPR-M algorithm using $M = 100{,}000$. For these smaller datasets, we are able to compute the exact number of triangles. Our algorithm is very precise with average, min and max estimations close to the ground truth. The overall observations made for the simple graph case also hold for the multigraph case: Our suite of algorithms allows precise and efficient estimation of the number of triangles with limited memory.

Figure 9 shows the average update time in microseconds using TRIÈST-IMPR-M algorithm in our multigraph datasets: Few microseconds are sufficient on average to update the triangle estimation, which is consistent with the results of the previous sections.

Finally, we evaluate the accuracy of the estimation using our TRIÈST-BASE-M and TRIÈST-IMPR-M algorithms. The results are shown in Table VII. We observe that TRIÈST-BASE-M and TRIÈST-IMPR-M maintain a good accuracy with performance comparable to the one observed for the simple graph case.



(a) Patent (Co-Aut.) (b) LastFm

Fig. 8. Evolution of the global number of triangles in the insertion-only case on multigraphs using TRIÈST-IMPR-M and $M = 100{,}000$. The algorithm estimations are consistently very accurate, and the curves are shown as almost undistinguishable on purpose to highlight this fact.

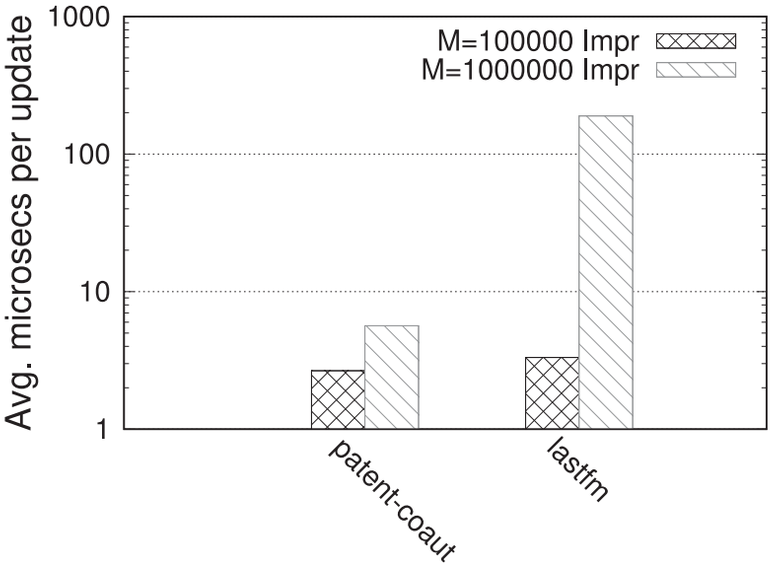


Fig. 9. Tradeoffs between the avg. update time (μs) and M for TRIÈST-IMPR-M – multigraphs.

Table VII. Estimation Errors for TRIÈST-BASE-M and TRIÈST-IMPR-M – Multigraphs

Graph	M	Global Error TRIÈST-BASE-M		Global Error TRIÈST-IMPR-M	
		Avg. MAPE	Max MAPE	Avg. MAPE	Max MAPE
LastFM	100,000	0.015	0.024	0.008	0.015
	1,000,000	0.006	0.012	0.003	0.008
Paten (Co-Aut.)	100,000	0.068	0.141	0.023	0.049
	1,000,000	0.011	0.017	0.003	0.006

6. CONCLUSIONS

We presented TRIÈST, the first suite of algorithms that use reservoir sampling and its variants to continuously maintain unbiased, low-variance estimates of the local and global number of triangles in fully dynamic graphs streams of arbitrary edge/vertex insertions and deletions using a fixed, user-specified amount of space. Our experimental evaluation shows that TRIÈST outperforms state-of-the-art approaches and achieves high accuracy on real-world datasets with more than one billion of edges, with update times of hundreds of microseconds.

APPENDIX: ADDITIONAL THEORETICAL RESULTS

In this section, we present the theoretical results (statements and proofs) not included in the main body.

A.1. Theoretical Results for TRIÈST-BASE

Before proving Lemma 4.1, we need to introduce the following lemma, which states a well-known property of the reservoir sampling scheme.

LEMMA A.1 ([42, SECTION 2]). *For any $t > M$, let A be any subset of $E^{(t)}$ of size $|A| = M$. Then, at the end of time step t,*

$$\Pr(\mathcal{S} = A) = \frac{1}{\binom{|E^{(t)}|}{M}} = \frac{1}{\binom{t}{M}},$$

i.e., the set of edges in $\mathcal{S}$ at the end of time t is a subset of $E^{(t)}$ of size M chosen uniformly *at random from all subsets of $E^{(t)}$ of the same size.*

PROOF OF LEMMA 4.1. If $k > \min\{M, t\}$, we have $\Pr(B \subseteq \mathcal{S}) = 0$ because it is impossible for B to be equal to $\mathcal{S}$ in these cases. From now on, we then assume $k \leq \min\{M, t\}$.

If $t \leq M$, then $E^{(t)} \subseteq \mathcal{S}$ and $\Pr(B \subseteq \mathcal{S}) = 1 = \xi_{k,t}^{-1}$.

Assume instead that $t > M$, and let $\mathcal{B}$ be the family of subsets of $E^{(t)}$ that *1.* have size M, and *2.* contain B:

$$\mathcal{B} = \{C \subset E^{(t)} \ : \ |C| = M, B \subseteq C\}.$$

We have

$$|\mathcal{B}| = \binom{|E^{(t)}| - k}{M - k} = \binom{t - k}{M - k}. \tag{A.1}$$

From this and Lemma A.1, we then have

$$\begin{aligned}\Pr(B \subseteq \mathcal{S}) = \Pr(\mathcal{S} \in \mathcal{B}) &= \sum_{C \in \mathcal{B}} \Pr(\mathcal{S} = C) \\ &= \frac{\binom{t-k}{M-k}}{\binom{t}{M}} = \frac{\binom{t-k}{M-k}}{\binom{t-k}{M-k}\prod_{i=0}^{k-1}\frac{t-i}{M-i}} = \prod_{i=0}^{k-1}\frac{M-i}{t-i} = \xi_{k,t}^{-1}. \quad \square\end{aligned}$$

A.1.1. Expectation.

PROOF OF LEMMA 4.3. We only show the proof for τ, as the proof for the local counters follows the same steps.

The proof proceeds by induction. The thesis is true after the first call to UPDATECOUNTERS at time $t = 1$. Since only one edge is in $\mathcal{S}$ at this point, we have $\Delta^{\mathcal{S}} = 0$, and $\mathcal{N}_{u,v}^{\mathcal{S}} = \emptyset$, so UPDATECOUNTERS does not modify τ, which was initialized to 0. Hence, $\tau = 0 = \Delta^{\mathcal{S}}$.

Assume now that the thesis is true for any subsequent call to UPDATECOUNTERS up to some point in the execution of the algorithm where an edge (u, v) is inserted or removed from $\mathcal{S}$. We now show that the thesis is still true after the call to UPDATECOUNTERS that follows this change in $\mathcal{S}$. Assume that (u, v) was *inserted* in $\mathcal{S}$ (the proof for the case of an edge being removed from $\mathcal{S}$ follows the same steps). Let $\mathcal{S}^{\mathrm{b}} = \mathcal{S} \setminus \{(u, v)\}$ and τ^{b} be the value of τ before the call to UPDATECOUNTERS and, for any $w \in V_{\mathcal{S}^{\mathrm{b}}}$, let τ_w^{b} be the value of τ_w before the call to UPDATECOUNTERS. Let $\Delta_{u,v}^{\mathcal{S}}$ be the set of triangles in $G_{\mathcal{S}}$ that have u and v as corners. We need to show that, after the call, $\tau = |\Delta^{\mathcal{S}}|$. Clearly we have

$\Delta^{\mathcal{S}} = \Delta^{\mathcal{S}^{\mathrm{b}}} \cup \Delta^{\mathcal{S}}_{u,v}$ and $\Delta^{\mathcal{S}^{\mathrm{b}}} \cap \Delta^{\mathcal{S}}_{u,v} = \emptyset$, so

$$|\Delta^{\mathcal{S}}| = |\Delta^{\mathcal{S}^{\mathrm{b}}}| + |\Delta^{\mathcal{S}}_{u,v}|.$$

We have $|\Delta^{\mathcal{S}}_{u,v}| = |\mathcal{N}^{\mathcal{S}}_{u,v,}|$ and, by the inductive hypothesis, we have that $\tau^{\mathrm{b}} = |\Delta^{\mathcal{S}^{\mathrm{b}}}|$. Since UPDATECOUNTERS increments τ by $|\mathcal{N}^{\mathcal{S}}_{u,v,}|$, the value of τ after UPDATECOUNTERS has completed is exactly $|\Delta^{\mathcal{S}}|$. □

We can now prove Theorem 4.2 on the unbiasedness of the estimation computed by TRIÈST-BASE (and on their exactness for $t \leq M$).

PROOF OF THEOREM 4.2. We prove the statement for the estimation of global triangle count. The proof for the local triangle counts follows the same steps.

If $t \leq M$, we have $G_{\mathcal{S}} = G^{(t)}$ and from Lemma 4.3, we have $\tau^{(t)} = |\Delta^{\mathcal{S}}| = |\Delta^{(t)}|$, hence the thesis holds.

Assume now that $t > M$, and assume that $|\Delta^{(t)}| > 0$, otherwise, from Lemma 4.3, we have $\tau^{(t)} = |\Delta^{\mathcal{S}}| = 0$ and TRIÈST-BASE estimation is deterministically correct. Let $\lambda = (a, b, c) \in \Delta^{(t)}$, (where a, b, c are edges in $E^{(t)}$), and let $\delta^{(t)}_{\lambda}$ be a random variable that takes value $\xi^{(t)}$ if $\lambda \in \Delta_{\mathcal{S}}$ (i.e., $\{a, b, c\} \subseteq \mathcal{S}$) at the end of the step instant t, and 0 otherwise. From Lemma 4.1, we have that

$$\mathbb{E}\left[\delta^{(t)}_{\lambda}\right] = \xi^{(t)} \Pr(\{a, b, c\} \subseteq \mathcal{S}) = \xi^{(t)} \frac{1}{\xi_{3,t}} = \xi^{(t)} \frac{1}{\xi^{(t)}} = 1. \tag{A.2}$$

We can write

$$\xi^{(t)} \tau^{(t)} = \sum_{\lambda \in \Delta^{(t)}} \delta^{(t)}_{\lambda}$$

and from this, Equation (A.2), and linearity of expectation, we have

$$\mathbb{E}\left[\xi^{(t)} \tau^{(t)}\right] = \sum_{\lambda \in \Delta^{(t)}} \mathbb{E}\left[\delta^{(t)}_{\lambda}\right] = |\Delta^{(t)}|. \quad \square$$

A.1.2. Concentration.

PROOF OF LEMMA 4.7. Using the law of total probability, we have

$$\begin{aligned}
\Pr(f(\mathcal{S}_{\mathrm{IN}}) = 1) &= \sum_{k=0}^{t} \Pr(f(\mathcal{S}_{\mathrm{IN}}) = 1 \mid |\mathcal{S}_{\mathrm{IN}}| = k) \Pr(|\mathcal{S}_{\mathrm{IN}}| = k) \\
&\geq \Pr(f(\mathcal{S}_{\mathrm{IN}}) = 1 \mid |\mathcal{S}_{\mathrm{IN}}| = M) \Pr(|\mathcal{S}_{\mathrm{IN}}| = M) \\
&\geq \Pr(f(\mathcal{S}_{\mathrm{MIX}}) = 1) \Pr(|\mathcal{S}_{\mathrm{IN}}| = M), \qquad (A.3)
\end{aligned}$$

where the last inequality comes from Lemma A.1: The set of edges included in $\mathcal{S}_{\mathrm{MIX}}$ is a uniformly-at-random subset of M edges from $E^{(t)}$, and the same holds for $\mathcal{S}_{\mathrm{IN}}$ when conditioning its size being M.

Using the Stirling approximation $\sqrt{2\pi n}(\frac{n}{e})^n \leq n! \leq e\sqrt{n}(\frac{n}{e})^n$ for any positive integer n, we have

$$\begin{aligned}\Pr(|\mathcal{S}_{\text{IN}}| = M) &= \binom{t}{M}\left(\frac{M}{t}\right)^M\left(\frac{t-M}{t}\right)^{t-M}\\ &\geq \frac{t^t\sqrt{t}\sqrt{2\pi}e^{-t}}{e^2\sqrt{M}\sqrt{t-M}e^{-t}M^M(t-M)^{t-M}}\frac{M^M(t-M)^{t-M}}{t^t}\\ &\geq \frac{1}{e\sqrt{M}}.\end{aligned}$$

Plugging this into Equation (A.3) concludes the proof. □

FACT A.2. *For any $x > 2$, we have*

$$\frac{x^2}{(x-1)(x-2)} \leq 1 + \frac{4}{x-2}.$$

PROOF OF LEMMA 4.10. We start by looking at the ratio between $\frac{t(t-1)(t-2)}{M(M-1)(M-2)}$ and $(t/M)^3$. We have

$$\begin{aligned}1 \leq \frac{t(t-1)(t-2)}{M(M-1)(M-2)}\left(\frac{M}{t}\right)^3 &= \frac{M^2}{(M-1)(M-2)}\frac{(t-1)(t-2)}{t^2}\\ &\leq \frac{M^2}{(M-1)(M-2)}\\ &\leq 1 + \frac{4}{M-2},\end{aligned}$$

where the last step follows from Fact A.2. Using this, we obtain

$$\begin{aligned}\left|\phi^{(t)} - \phi^{(t)}_{\text{MIX}}\right| &= \left|\tau^{(t)}\frac{t(t-1)(t-2)}{M(M-1)(M-2)} - \tau^{(t)}\left(\frac{t}{M}\right)^3\right|\\ &= \left|\tau^{(t)}\left(\frac{t}{M}\right)^3\left(\frac{t(t-1)(t-2)}{M(M-1)(M-2)}\left(\frac{M}{t}\right)^3 - 1\right)\right|\\ &\leq \tau^{(t)}\left(\frac{t}{M}\right)^3\frac{4}{M-2}\\ &= \phi^{(t)}_{\text{MIX}}\frac{4}{M-2}.\ \square\end{aligned}$$

A.1.3. Variance Comparison. We now prove Lemma 4.11, about the fact that the variance of the estimations computed by TRIÈST-BASE is smaller, for most of the stream, than the variance of the estimations computed by MASCOT-C [28]. We first need the following technical fact.

FACT A.3. *For any $x > 42$, we have*

$$\frac{x^2}{(x-3)(x-4)} \leq 1 + \frac{8}{x}.$$

PROOF OF LEMMA 4.11. We focus on $t > M > 42$ otherwise the theorem is immediate. We show that for such conditions $f(M,t) < \bar{f}(M/T)$ and $g(M,t) < \bar{g}(M/T)$. Using the

fact that $t \leq \alpha T$ and Fact A.2, we have

$$\begin{aligned} f(M,t) - \bar{f}(M/T) &= \frac{t(t-1)(t-2)}{M(M-1)(M-2)} - \frac{T^3}{M^3} \\ &< \frac{\alpha^3 T^3}{M^3} \frac{M^2}{(M-1)(M-2)} - \frac{T^3}{M^3} \\ &\leq \frac{\alpha^3 T^3}{M^3} \left(1 + \frac{4}{M-2}\right) - \frac{T^3}{M^3} \\ &\leq \frac{T^3}{M^3} \left(\alpha^3 + \frac{4\alpha^3}{M-2} - 1\right). \end{aligned} \tag{A.4}$$

Given that T and M are ≥ 42, the r.h.s. of (A.4) is non-positive iff

$$\alpha^3 + \frac{4\alpha^3}{M-2} - 1 \leq 0.$$

Solving for M we have that the above is verified when $M \geq \frac{4\alpha^3}{1-\alpha^3} + 2$. This is always true given our assumption that $M > \max(\frac{8\alpha}{1-\alpha}, 42)$: For any $0 < \alpha < 0.6$, we have $\frac{4\alpha^3}{1-\alpha^3} + 2 < 42 \leq M$ and for any $0.6 \leq \alpha < 1$, we have $\frac{4\alpha^3}{1-\alpha^3} + 2 < \frac{8\alpha}{1-\alpha} \leq M$. Hence, the r.h.s. of Equation (A.4) is ≤ 0 and $f(M,t) < \bar{f}(M/T)$.

We also have

$$\begin{aligned} g(M,t) - \bar{g}(M/T) &= \frac{t(t-1)(t-2)(M-3)(M-4)}{(t-3)(t-4)M(M-1)(M-2)} - \frac{T}{M} \\ &< \frac{t}{M} \frac{t^2}{(t-3)(t-4)} - \frac{T}{M} \\ &\leq \frac{t}{M} \left(1 + \frac{8}{t}\right) - \frac{T}{M}, \end{aligned} \tag{A.5}$$

where the last inequality follow from Fact A.3, since $t > M > 42$. Now, from Equation (A.5) since $t \leq \alpha T$ and $t > M$, we can write

$$g(M,t) - \bar{g}(M/T) < \frac{T}{M} \left(\alpha + \frac{8\alpha}{M} - 1\right).$$

The r.h.s. of this equation is non-positive given the assumption $M > \frac{8\alpha}{1-\alpha}$, hence $g(M,t) < \bar{g}(M/T)$. □

A.2. Theoretical Results for TRIÈST-IMPR

A.2.1. Expectation.

Proof of Theorem 4.12. If $t \leq M$, TRIÈST-IMPR behaves exactly like TRIÈST-BASE, and the statement follows from Lemma 4.2.

Assume now $t > M$ and assume that $|\Delta^{(t)}| > 0$, otherwise, the algorithm deterministically returns 0 as an estimation and the thesis follows. Let $\lambda \in \Delta^{(t)}$ and denote with a, b, and c the edges of λ and assume, w.l.o.g., that they appear in this order (not necessarily consecutively) on the stream. Let t_λ be the time step at which c is on the stream. Let δ_λ be a random variable that takes value $\xi_{2,t_\lambda-1}$ if a and b are in $\mathcal{S}$ at the end of time step $t_\lambda - 1$, and 0 otherwise. Since it must be $t_\lambda - 1 \geq 2$, from Lemma 4.1, we have that

$$\Pr(\delta_\lambda = \xi_{2,t_\lambda-1}) = \frac{1}{\xi_{2,t_\lambda-1}}. \tag{A.6}$$

When $c = (u, v)$ is on the stream, i.e., at time t_λ, TRIÈST-IMPR calls UPDATECOUNTERS and increments the counter τ by $|\mathcal{N}^{\mathcal{S}}_{u,v}|\xi_{2,t_\lambda-1}$, where $|\mathcal{N}^{\mathcal{S}}_{u,v}|$ is the number of triangles with (u, v) as an edge in $\Delta^{\mathcal{S}\cup\{c\}}$. All these triangles have the corresponding random variables taking the same value $\xi_{2,t_\lambda-1}$. This means that the random variable $\tau^{(t)}$ can be expressed as

$$\tau^{(t)} = \sum_{\lambda\in\Delta^{(t)}} \delta_\lambda.$$

From this, linearity of expectation, and Equation (A.6), we get

$$\mathbb{E}\left[\tau^{(t)}\right] = \sum_{\lambda\in\Delta^{(t)}} \mathbb{E}[\delta_\lambda] = \sum_{\lambda\in\Delta^{(t)}} \xi_{2,t_\lambda-1}\Pr(\delta_\lambda = \xi_{2,t_\lambda-1}) = \sum_{\lambda\in\Delta^{(t)}} \xi_{2,t_\lambda-1}\frac{1}{\xi_{2,t_\lambda-1}} = |\Delta^{(t)}|. \quad \square$$

A.2.2. Variance.

PROOF OF LEMMA 4.14. Consider first the case where *all* edges of λ appear on the stream before *any* edge of γ, i.e.,

$$t_{\ell_1} < t_{\ell_2} < t_{\ell_3} < t_{g_1} < t_{g_2} < t_{g_3}.$$

The presence or absence of either or both ℓ_1 and ℓ_2 in $\mathcal{S}$ at the beginning of time step t_{ℓ_3} (i.e., whether D_λ happens or not) has no effect whatsoever on the probability that g_1 and g_2 are in the sample $\mathcal{S}$ at the beginning of time step t_{g_3}. Hence, in this case,

$$\Pr(D_\gamma \mid D_\lambda) = \Pr(D_\gamma).$$

Consider now the case where, for any $i \in \{1, 2\}$, the edges $g_1, \ldots, g_i$ appear on the stream before ℓ_3 does. Define now the events

- A_i: The edges $g_1, \ldots, g_i$ are in the sample $\mathcal{S}$ at the beginning of time step t_{ℓ_3}.
- B_i: If $i = 1$, this is the event "the edge g_2 is inserted in the sample $\mathcal{S}$ during time step t_{g_2}." If $i = 2$, this event is the whole event space, i.e., the event that happens with probability 1.
- C: Neither g_1 nor g_2 was among the edges removed from $\mathcal{S}$ between the beginning of time step t_{ℓ_3} and the beginning of time step t_{g_3}.

We can rewrite D_γ as

$$D_\gamma = A_i \cap B_i \cap C.$$

Hence,

$$\begin{aligned}\Pr(D_\gamma \mid D_\lambda) &= \Pr(A_i \cap B_i \cap C \mid D_\lambda)\\ &= \Pr(A_i \mid D_\lambda)\Pr(B_i \cap C \mid A_i \cap D_\lambda).\end{aligned} \tag{A.7}$$

We now show that

$$\Pr(A_i \mid D_\lambda) \le \Pr(A_i).$$

If we assume that $t_{\ell_3} \le M + 1$, then all the edges that appeared on the stream up until the beginning of t_{ℓ_3} are in $\mathcal{S}$. Therefore,

$$\Pr(A_i \mid D_\lambda) = \Pr(A_i) = 1.$$

Assume instead that $t_{\ell_3} > M + 1$. Among the $\binom{t_{\ell_3}-1}{M}$ subsets of $E^{(t_{\ell_3}-1)}$ of size M, there are $\binom{t_{\ell_3}-3}{M-2}$ that contain ℓ_1 and ℓ_2. From Lemma A.1, we have that at the beginning of time t_{ℓ_3}, $\mathcal{S}$ is a subset of size M of $E^{(t_{\ell_3}-1)}$ chosen uniformly at random. Hence, if we condition on the fact that $\{\ell_1, \ell_2\} \subset \mathcal{S}$, we have that $\mathcal{S}$ is chosen uniformly at random

from the $\binom{t_{\ell_3}-3}{M-2}$ subsets of $E^{(t_{\ell_3}-1)}$ of size M that contain ℓ_1 and ℓ_2. Among these, there are $\binom{t_{\ell_3}-3-i}{M-2-i}$ that also contain $g_1, \ldots, g_i$. Therefore,

$$\Pr(A_i \mid D_\lambda) = \frac{\binom{t_{\ell_3}-3-i}{M-2-i}}{\binom{t_{\ell_3}-3}{M-2}} = \prod_{j=0}^{i-1} \frac{M-2-j}{t_{\ell_3}-3-j}.$$

From Lemma 4.1, we have

$$\Pr(A_i) = \frac{1}{\xi_{i,t_{\ell_3}-1}} = \prod_{j=0}^{i-1} \frac{M-j}{t_{\ell_3}-1-j},$$

where the last equality comes from the assumption $t_{\ell_3} > M+1$. From the same assumption and from the fact that for any $j \geq 0$ and any $y \geq x > j$ it holds $\frac{x-j}{y-j} \leq \frac{x}{y}$, then we have

$$\Pr(A_i \mid D_\lambda) \leq \Pr(A_i).$$

This implies, from Equation (A.7), that

$$\Pr(D_\gamma \mid D_\lambda) \leq \Pr(A_i)\Pr(B_i \cap C \mid A_i \cap D_\lambda). \tag{A.8}$$

Consider now the events B_i and C. When conditioned on A_i, these event are *both independent* from D_λ: If the edges $g_1, \ldots, g_i$ are in $\mathcal{S}$ at the beginning of time t_{ℓ_3}, the fact that the edges ℓ_1 and ℓ_2 were also in $\mathcal{S}$ at the beginning of time t_{ℓ_3} has no influence whatsoever on the actions of the algorithm (i.e., whether an edge is inserted in or removed from $\mathcal{S}$). Thus,

$$\Pr(A_i)\Pr(B_i \cap C \mid A_i \cap D_\lambda) = \Pr(A_i)\Pr(B_i \cap C \mid A_i).$$

Putting this together with Equation (A.8), we obtain

$$\Pr(D_\gamma \mid D_\lambda) \leq \Pr(A_i)\Pr(B_i \cap C \mid A_i) \leq \Pr(A_i \cap B_i \cap C) \leq \Pr(D_\gamma),$$

where the last inequality follows from the fact that $D_\gamma = A_i \cap B_i \cap C$ by definition. □

A.3. Theoretical Results for TRIÈST-IMPR

A.3.1. Expectation. Before proving Theorem 4.16, we need the following technical lemmas.

The following is a corollary of Gemulla et al. [16, Theorem 1].

LEMMA A.4. *For any $t > 0$, and any j, $0 \leq j \leq s^{(t)}$, let $\mathcal{B}^{(t)}$ be the collection of subsets of $E^{(t)}$ of size j. For any $B \in \mathcal{B}^{(t)}$, it holds*

$$\Pr\left(\mathcal{S} = B \mid M^{(t)} = j\right) = \frac{1}{\binom{|E^{(t)}|}{j}}.$$

That is, conditioned on its size at the end of time step t, $\mathcal{S}$ is equally likely to be, at the end of time step t, any of the subsets of $E^{(t)}$ of that size.

The next lemma is an immediate corollary of Gemulla et al. [16, Theorem 2].

LEMMA A.5. *Recall the definition of $\kappa^{(t)}$ from Equation* (14)*. We have*

$$\kappa^{(t)} = \Pr(M^{(t)} \geq 3).$$

The next lemma follows from Lemma A.4 in the same way as Lemma 4.1 follows from Lemma A.1.

LEMMA A.6. *For any time step t and any j, $0 \leq j \leq s^{(t)}$, let B be any subset of $E^{(t)}$ of size $|B| = k \leq s^{(t)}$. Then, at the end of time step t,*

$$\Pr\left(B \subseteq \mathcal{S} \mid M^{(t)} = j\right) = \begin{cases} 0 & \text{if } k > j \\ \dfrac{1}{\psi_{k,j,s^{(t)}}} & \text{otherwise} \end{cases}.$$

The next two lemmas discuss properties of TRIÈST-FD for $t < t^*$, where t^* is the first time that $|E^{(t)}|$ had size $M + 1$ ($t^* \geq M + 1$).

LEMMA A.7. *For all $t < t^*$, we have*

(1) $d_o^{(t)} = 0$*; and*
(2) $\mathcal{S} = E^{(t)}$*; and*
(3) $M^{(t)} = s^{(t)}$.

PROOF. Since the third point in the thesis follows immediately from the second, we focus on the first two points.

The proof is by induction on t. In the base for $t = 1$: the element on the stream must be an insertion, and the algorithm deterministically inserts the edge in $\mathcal{S}$. Assume now that it is true for all time steps up to (but excluding) some $t \leq t^* - 1$. We now show that it is also true for t.

Assume the element on the stream at time t is a deletion. The corresponding edge must be in $\mathcal{S}$, from the inductive hypothesis. Hence, TRIÈST-FD removes it from $\mathcal{S}$ and increments the counter d_i by 1. Thus, it is still true that $\mathcal{S} = E^{(t)}$ and $d_o^{(t)} = 0$, and the thesis holds.

Assume now that the element on the stream at time t is an insertion. From the inductive hypothesis, we have that the current value of the counter d_o is 0.

If the counter d_i has currently value 0 as well, then, because of the hypothesis that $t < t^*$, it must be that $|\mathcal{S}| = M^{(t-1)} = s^{(t-1)} < M$. Therefore, TRIÈST-FD always inserts the edge in $\mathcal{S}$. Thus, it is still true that $\mathcal{S} = E^{(t)}$ and $d_o^{(t)} = 0$, and the thesis holds.

If otherwise $d_i > 0$, then TRIÈST-FD flips a biased coin with probability of heads equal to

$$\frac{d_i}{d_i + d_o} = \frac{d_i}{d_i} = 1,$$

therefore TRIÈST-FD always inserts the edge in $\mathcal{S}$ and decrements d_i by one. Thus, it is still true that $\mathcal{S} = E^{(t)}$ and $d_o^{(t)} = 0$, and the thesis holds. □

The following result is an immediate consequence of Lemma A.5 and Lemma A.7.

LEMMA A.8. *For all $t < t^*$ such that $s^{(t)} \geq 3$, we have $\kappa^{(t)} = 1$.*

We can now prove Theorem 4.16.

PROOF OF THEOREM 4.16. Assume for now that $t < t^*$. From Lemma A.7, we have that $s^{(t)} = M^{(t)}$. If $M^{(t)} < 3$, then it must be $s^{(t)} < 3$, hence $|\Delta^{(t)}| = 0$ and indeed the algorithm returns $\rho^{(t)} = 0$ in this case. If instead $M^{(t)} = s^{(t)} \geq 3$, then we have

$$\rho^{(t)} = \frac{\tau^{(t)}}{\kappa^{(t)}}.$$

From Lemma A.8, we have that $\kappa^{(t)} = 1$ for all $t < t^*$, hence $\rho^{(t)} = \tau^{(t)}$ in these cases. Since (an identical version of) Lemma 4.3 also holds for TRIÈST-FD, we have $\tau^{(t)} = |\Delta^{\mathcal{S}}| = |\Delta^{(t)}|$, where the last equality comes from the fact that $\mathcal{S} = E^{(t)}$ (Lemma A.7). Hence, $\rho^{(t)} = |\Delta^{(t)}|$ for any $t \leq t^*$, as in the thesis.

Assume now that $t \geq t^*$. Using the law of total expectation, we can write

$$\mathbb{E}\big[\rho^{(t)}\big] = \sum_{j=0}^{\min\{s^{(t)},M\}} \mathbb{E}\big[\rho^{(t)} \mid M^{(t)} = j\big] \Pr\big(M^{(t)} = j\big). \tag{A.9}$$

Assume that $|\Delta^{(t)}| > 0$, otherwise, the algorithm deterministically returns 0 as an estimation and the thesis follows. Let λ be a triangle in $\Delta^{(t)}$, and let $\delta_\lambda^{(t)}$ be a random variable that takes value

$$\frac{\psi_{3,M^{(t)},s^{(t)}}}{\kappa^{(t)}} = \frac{s^{(t)}(s^{(t)}-2)(s^{(t)}-2)}{M^{(t)}(M^{(t)}-1)(M^{(t)}-2)} \frac{1}{\kappa^{(t)}}$$

if all edges of λ are in $\mathcal{S}$ at the end of the time instant t, and 0 otherwise. Since (an identical version of) Lemma 4.3 also holds for TRIÈST-FD, we can write

$$\rho^{(t)} = \sum_{\lambda \in \Delta^{(t)}} \delta_\lambda^{(t)}.$$

Then, using Lemma A.5 and Lemma A.6, we have, for $3 \leq j \leq \min\{M, s^{(t)}\}$,

$$\begin{aligned}\mathbb{E}\big[\rho^{(t)} \mid M^{(t)} = j\big] &= \sum_{\lambda \in \Delta^{(t)}} \frac{\psi_{3,j,s^{(t)}}}{\kappa^{(t)}} \Pr\left(\delta_\lambda^{(t)} = \frac{\psi_{3,j,s^{(t)}}}{\kappa^{(t)}} \mid M^{(t)} = j\right) \\ &= |\Delta^{(t)}| \frac{\psi_{3,j,s^{(t)}}}{\kappa^{(t)}} \frac{1}{\psi_{3,j,s^{(t)}}} = \frac{1}{\kappa^{(t)}} |\Delta^{(t)}|,\end{aligned} \tag{A.10}$$

and

$$\mathbb{E}\big[\rho^{(t)} \mid M^{(t)} = j\big] = 0, \text{if } 0 \leq j \leq 2. \tag{A.11}$$

Plugging this into Equation (A.9), and using Lemma A.5, we have

$$\mathbb{E}\big[\rho^{(t)}\big] = |\Delta^{(t)}| \frac{1}{\kappa^{(t)}} \sum_{j=3}^{\min\{s^{(t)},M\}} \Pr(M^{(t)} = j) = |\Delta^{(t)}|. \quad \square$$

A.3.2. Variance. We now move to prove Theorem 4.17 about the variance of the TRIÈST-FD estimator. We first need some technical lemmas.

LEMMA A.9. *For any time $t \geq t^*$, and any j, $3 \leq j \leq \min\{s^{(t)}, M\}$, we have*

$$\begin{aligned}\mathrm{Var}\big[\rho^{(t)} | M^{(t)} = j\big] = (\kappa^{(t)})^{-2} \big(&|\Delta^{(t)}| \left(\psi_{3,j,s^{(t)}} - 1\right) + r^{(t)} \left(\psi_{3,j,s^{(t)}}^2 \psi_{5,j,s^{(t)}}^{-1} - 1\right) \\ &+ w^{(t)} \left(\psi_{3,j,s^{(t)}}^2 \psi_{6,j,s^{(t)}}^{-1} - 1\right)\big).\end{aligned} \tag{A.12}$$

An analogous result holds for any $u \in V^{(t)}$, replacing the global quantities with the corresponding local ones.

PROOF. The proof is analogous to that of Theorem 4.4, using j in place of M, $s^{(t)}$ in place of t, $\psi_{a,j,s^{(t)}}$ in place of $\xi_{a,t}$, and using Lemma A.6 instead of Lemma 4.1. The additional $(k^{(t)})^{-2}$ multiplicative term comes from the $(k^{(t)})^{-1}$ term used in the definition of $\rho^{(t)}$. $\square$

The term $w^{(t)}(\psi_{3,j,s^{(t)}}^2 \psi_{6,j,s^{(t)}}^{-1} - 1)$ is non-positive.

LEMMA A.10. *For any time $t \geq t^*$, and any j, $6 < j \leq \min\{s^{(t)}, M\}$, if $s^{(t)} \geq M$ we have*

$$\mathrm{Var}\big[\rho^{(t)} | M^{(t)} = i\big] \leq (\kappa^{(t)})^{-2}\big(|\Delta^{(t)}|\big(\psi_{3,j,s^{(t)}} - 1\big) + r^{(t)}\big(\psi^2_{3,j,s^{(t)}}\psi^{-1}_{5,j,s^{(t)}} - 1\big)\big), \text{ for } i \geq j$$
$$\mathrm{Var}\big[\rho^{(t)} | M^{(t)} = i\big] \leq (\kappa^{(t)})^{-2}\big(|\Delta^{(t)}|\big(\psi_{3,3,s^{(t)}} - 1\big) + r^{(t)}\big(\psi^2_{3,5,s^{(t)}}\psi^{-1}_{5,5,s^{(t)}} - 1\big)\big), \text{ for } i < j$$

An analogous result holds for any $u \in V^{(t)}$, replacing the global quantities with the corresponding local ones.

PROOF. The proof follows by observing that the term $w^{(t)}(\psi^2_{3,j,s^{(t)}}\psi^{-1}_{6,j,s^{(t)}} - 1)$ is non-positive, and that Equation (A.12) is a non-increasing function of the sample size. □

The following lemma deals with properties of the r.v. $M^{(t)}$.

LEMMA A.11. *Let $t > t^*$, with $s^{(t)} \geq M$. Let $d^{(t)} = d_o^{(t)} + d_i^{(t)}$ denote the total number of unpaired deletions at time t.*[19] *The sample size $M^{(t)}$ follows the hypergeometric distribution:*[20]

$$\Pr\big(M^{(t)} = j\big) = \begin{cases} \binom{s^{(t)}}{j}\binom{d^{(t)}}{M-j}\Big/\binom{s^{(t)}+d^{(t)}}{M} & \text{for } \max\{M - d^{(t)}, 0\} \leq j \leq M \\ 0 & \text{otherwise} \end{cases}. \tag{A.13}$$

We have

$$\mathbb{E}\big[M^{(t)}\big] = M\frac{s^{(t)}}{s^{(t)} + d^{(t)}}, \tag{A.14}$$

and for any $0 < c < 1$

$$\Pr\big(M^{(t)} > \mathbb{E}\big[M^{(t)}\big] - cM\big) \geq 1 - \frac{1}{e^{2c^2M}}. \tag{A.15}$$

PROOF. Since $t > t^*$, from the definition of t^* we have that the $M^{(t)}$ has reached size M at least once (at t^*). From this and the definition of $d^{(t)}$ (number of uncompensated deletion), we have that $M^{(t)}$ cannot be less than $M - d^{(t)}$. The rest of the proof for Equation (A.13) and for Equation (A.14) follows from Gemulla et al. [16, Theorem 2].

The concentration bound in Equation (A.15) follows from the properties of the hypergeometric distribution discussed by Skala [37]. □

The following is an immediate corollary from Lemma A.11.

COROLLARY A.12. *Consider the execution of* TRIÈST-FD *at time $t > t^*$. Suppose we have $d^{(t)} \leq \alpha s^{(t)}$, with $0 \leq \alpha < 1$ and $s^{(t)} \geq M$. If $M \geq \frac{1}{2\sqrt{\alpha'-\alpha}}c' \ln s^{(t)}$ for $\alpha < \alpha' < 1$, we have*

$$\Pr\big(M^{(t)} \geq M(1-\alpha')\big) > 1 - \frac{1}{\big(s^{(t)}\big)^{c'}}.$$

We can now prove Theorem 4.17.

[19] While both $d_o^{(t)}$ and $d_i^{(t)}$ are r.v.s, their sum is not.
[20] We use here the convention that $\binom{0}{0} = 1$, and $\binom{k}{0} = 1$.

Proof of Theorem 4.17. From the law of total variance, we have

$$\begin{aligned}\mathrm{Var}\big[\rho^{(t)}\big] = &\sum_{j=0}^{M} \mathrm{Var}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big)\\ &+\sum_{j=0}^{M}\mathbb{E}\big[\rho^{(t)}|M^{(t)}=j\big]^2\big(1-\Pr\big(M^{(t)}=j\big)\big)\Pr\big(M^{(t)}=j\big)\\ &-2\sum_{j=1}^{M}\sum_{i=0}^{j-1}\mathbb{E}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big)\mathbb{E}\big[\rho^{(t)}|M^{(t)}=i\big]\Pr\big(M^{(t)}=i\big).\end{aligned}$$

As shown in Equations (A.10) and (A.11), for any $j = 0, 1, \ldots, M$, we have $\mathbb{E}[\rho^{(t)}|M^{(t)} = j] \geq 0$. This, in turn, implies

$$\begin{aligned}\mathrm{Var}\big[\rho^{(t)}\big] \leq &\sum_{j=0}^{M} \mathrm{Var}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big)\\ &+\sum_{j=0}^{M}\mathbb{E}\big[\rho^{(t)}|M^{(t)}=j\big]^2\big(1-\Pr\big(M^{(t)}=j\big)\big)\Pr\big(M^{(t)}=j\big).\end{aligned} \tag{A.16}$$

Let us consider separately the two main components of Equation (A.16). From Lemma A.10, we have

$$\sum_{j=0}^{M}\mathrm{Var}\left[\rho^{(t)}|M^{(t)}=j\right]\Pr\left(M^{(t)}=j\right) \tag{A.17}$$

$$\begin{aligned}&= \sum_{j\geq M(1-\alpha')}^{M}\mathrm{Var}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big)\big) + \sum_{j=0}^{M(1-\alpha')}\mathrm{Var}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big)\\ &\leq (\kappa^{(t)})^{-2}\big(|\Delta^{(t)}|\big(\psi_{3,j,s^{(t)}}-1\big)+r^{(t)}\big(\psi^2_{3,j,s^{(t)}}\psi^{-1}_{5,j,s^{(t)}}-1\big)\big)\times\Pr\big(M^{(t)}>M(1-\alpha')\big)\\ &\leq (\kappa^{(t)})^{-2}\left(|\Delta^{(t)}|\frac{\big(s^{(t)}\big)^3}{6}+r^{(t)}\frac{s^{(t)}}{6}\right)\Pr\big(M^{(t)}\leq M(1-\alpha')\big).\end{aligned} \tag{A.18}$$

According to our hypothesis $M \geq \frac{1}{2\sqrt{\alpha'-\alpha}}7\ln s^{(t)}$, thus we have, from Corollary A.12

$$\Pr\big(M^{(t)}\leq M(1-\alpha'))\big)\leq\frac{1}{(s^{(t)})^7}.$$

As $r^{(t)} < |\Delta^{(t)}|^2$ and $|\Delta^{(t)}| \leq (s^{(t)})^3$, we have

$$(\kappa^{(t)})^{-2}\left(|\Delta^{(t)}|\frac{\big(s^{(t)}\big)^3}{6}+r^{(t)}\frac{s^{(t)}}{6}\right)\Pr\big(M^{(t)}\leq M(1-\alpha')\big)\leq(\kappa^{(t)})^{-2}.$$

We can therefore rewrite Equation (A.18) as

$$\begin{aligned}\sum_{j=0}^{M}\mathrm{Var}\big[\rho^{(t)}|M^{(t)}=j\big]\Pr\big(M^{(t)}=j\big) \leq &(\kappa^{(t)})^{-2}\big(|\Delta^{(t)}|\left(\psi_{3,M(1-\alpha'),s^{(t)}}-1\right)\big)\\ &+(\kappa^{(t)})^{-2}\big(r^{(t)}\big(\psi^2_{3,M(1-\alpha'),s^{(t)}}\psi^{-1}_{5,M(1-\alpha'),s^{(t)}}-1\big)+1\big).\end{aligned} \tag{A.19}$$

Let us now consider the term $\sum_{j=0}^{M} \mathbb{E}[\rho^{(t)}|M^{(t)}=j]^2(1-\Pr(M^{(t)}=j))\Pr(M^{(t)}=j)$. Recall that, from Equations (A.10) and (A.11), we have $\mathbb{E}[\rho^{(t)}|M^{(t)}=j] = |\Delta^{(t)}|(\kappa^{(t)})^{-1}$ for $j=3,\ldots,M$, and $\mathbb{E}\left[\rho^{(t)}|M^{(t)}=j\right]=0$ for $j=0,1,2$. From Corollary A.12, we have that for $j \leq (1-\alpha')M$ and $M \geq \frac{1}{2\sqrt{\alpha'-\alpha}}7\ln s^{(t)}$

$$\Pr\left(M^{(t)}=j\right) \leq \Pr\left(M^{(t)} \leq (1-\alpha')M\right) \leq \frac{1}{\left(s^{(t)}\right)^7},$$

and thus

$$\sum_{j=0}^{(1-\alpha')M} \mathbb{E}\left[\rho^{(t)}|M^{(t)}=j\right]^2\left(1-\Pr\left(M^{(t)}=j\right)\right)\Pr\left(M^{(t)}=j\right) \leq \frac{(1-\alpha')M|\Delta^{(t)}|^2(\kappa^{(t)})^{-2}}{\left(s^{(t)}\right)^7}$$
$$\leq (1-\alpha')(\kappa^{(t)})^{-2}, \tag{A.20}$$

where the last passage follows since, by hypothesis, $M \leq s^{(t)}$.

Let us now consider the values $j > (1-\alpha')M$, we have

$$\sum_{j>(1-\alpha')M}^{M} \mathbb{E}\left[\rho^{(t)}|M^{(t)}=j\right]^2\left(1-\Pr\left(M^{(t)}=j\right)\right)\Pr\left(M^{(t)}=j\right)$$
$$\leq \alpha' M|\Delta^{(t)}|^2(\kappa^{(t)})^{-2}\left(1-\sum_{j>(1-\alpha')M}^{M}\Pr\left(M^{(t)}=j\right)\right)$$
$$\leq \alpha' M|\Delta^{(t)}|^2(\kappa^{(t)})^{-2}\left(1-\Pr\left(M^{(t)}>(1-\alpha')M\right)\right)$$
$$\leq \frac{\alpha' M|\Delta^{(t)}|^2(\kappa^{(t)})^{-2}}{\left(s^{(t)}\right)^7}$$
$$\leq \alpha'(\kappa^{(t)})^{-2}, \tag{A.21}$$

where the last passage follows since, by hypothesis, $M \leq s^{(t)}$.

The theorem follows from composing the upper bounds obtained in Equations (A.19), (A.20), and (A.21) according to Equation (A.16). □

A.3.3. Concentration. We now prove Theorem 4.18 about TRIÈST-FD.

PROOF OF THEOREM 4.18. By Chebyshev's inequality, it is sufficient to prove that

$$\frac{\mathrm{Var}[\rho^{(t)}]}{\varepsilon^2|\Delta^{(t)}|^2} < \delta.$$

From Lemma 4.17, for $M \geq \frac{1}{\sqrt{\alpha'-\alpha}}7\ln s^{(t)}$, we have

$$\mathrm{Var}\left[\rho^{(t)}\right] \leq (\kappa^{(t)})^{-2}|\Delta^{(t)}|\left(\psi_{3,M(1-\alpha'),s^{(t)}}-1\right)$$
$$+(\kappa^{(t)})^{-2}r^{(t)}\left(\psi^2_{3,M(1-\alpha'),s^{(t)}}\psi^{-1}_{5,M(1-\alpha'),s^{(t)}}-1\right)$$
$$+(\kappa^{(t)})^{-2}2.$$

Let $M'=(1-\alpha')M$. In order to verify that the lemma holds, it is sufficient to impose the following two conditions:

Condition (1)

$$\frac{\delta}{2} > \frac{(\kappa^{(t)})^{-2}\left(|\Delta^{(t)}|\left(\psi_{3,M(1-\alpha'),s^{(t)}}-1\right)+2\right)}{\varepsilon^2|\Delta^{(t)}|^2}.$$

As by hypothesis $|\Delta^{(t)}| > 0$, we can rewrite this condition as

$$\frac{\delta}{2} > \frac{(\kappa^{(t)})^{-2}\left(\psi_{3,M(1-\alpha'),s^{(t)}}-\left(\frac{|\Delta^{(t)}|-2}{|\Delta^{(t)}|}\right)\right.}{\varepsilon^2|\Delta^{(t)}|},$$

which is verified for

$$M'(M'-1)(M'-2) > \frac{2s^{(t)}(s^{(t)}-1)(s^{(t)}-2)}{\delta\varepsilon^2|\Delta^{(t)}|(\kappa^{(t)})^2+2\frac{|\Delta^{(t)}|-2}{|\Delta^{(t)}|}},$$

$$M' > \sqrt[3]{\frac{2s^{(t)}(s^{(t)}-1)(s^{(t)}-2)}{\delta\varepsilon^2|\Delta^{(t)}|(\kappa^{(t)})^2+2\frac{|\Delta^{(t)}|-2}{|\Delta^{(t)}|}}}+2,$$

$$M > (1-\alpha')^{-1}\left(\sqrt[3]{\frac{2s^{(t)}(s^{(t)}-1)(s^{(t)}-2)}{\delta\varepsilon^2|\Delta^{(t)}|(\kappa^{(t)})^2+2\frac{|\Delta^{(t)}|-2}{|\Delta^{(t)}|}}}+2\right).$$

Condition (2)

$$\frac{\delta}{2} > \frac{(\kappa^{(t)})^{-2}}{\varepsilon^2|\Delta^{(t)}|^2}r^{(t)}\left(\psi^2_{3,M(1-\alpha'),s^{(t)}}\psi^{-1}_{5,M(1-\alpha'),s^{(t)}}-1\right). \tag{A.22}$$

As we have

$$(\kappa^{(t)})^{-2}r^{(t)}\left(\psi^2_{3,M(1-\alpha'),s^{(t)}}\psi^{-1}_{5,M(1-\alpha'),s^{(t)}}-1\right) \le (\kappa^{(t)})^{-2}r^{(t)}\left(\frac{s^{(t)}}{6M(1-\alpha')}-1\right).$$

The condition (A.22) is verified for

$$M > \frac{(1-\alpha')^{-1}}{3}\left(\frac{r^{(t)}s^{(t)}}{\delta\varepsilon^2|\Delta^{(t)}|^2(\kappa^{(t)})^{-2}+2r^{(t)}}\right).$$

The theorem follows. □

REFERENCES

[1] Nesreen K. Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. 2014. Graph sample and hold: A framework for big-graph analytics. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. ACM, 1446–1455.

[2] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*. SIAM, 623–632.

[3] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data* 4, 3 (2010), 13:1–13:28.

[4] Jonathan W. Berry, Bruce Hendrickson, Randall A. LaViolette, and Cynthia A. Phillips. 2011. Tolerating the community detection resolution limit with edge weighting. *Physical Review E* 83, 5 (2011), 056119.

[5] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. ACM, 587–596.

[6] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. 2016. Triangle counting in dynamic graph streams. *Algorithmica* 76, 1 (Sep. 2016), 259–278.

[7] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. 2006. Counting triangles in data streams. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'06)*. ACM, 253–262.

[8] Òscar Celma Herrada. 2009. *Music Recommendation and Discovery in the Long Tail*. Technical Report. Universitat Pompeu Fabra.

[9] Edith Cohen, Graham Cormode, and Nick Duffield. 2012. Don't let the negatives bring you down: Sampling from streams of signed updates. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (2012), 343–354.

[10] Yahoo! Research Webscope Datasets. 2016. Yahoo! Answers browsing behavior version 1.0. http://webscope.sandbox.yahoo.com. ((Accessed on) September 2016).

[11] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2016. TRIÈST: Counting local and global triangles in fully-dynamic streams with fixed memory size. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 825–834.

[12] Jean-Claude Deville and Yves Tillé. 2004. Efficient balanced sampling: The cube method. *Biometrika* 91, 4 (2004), 893–912. https://doi.org/10.1093/biomet/91.4.893

[13] Jean-Pierre Eckmann and Elisha Moses. 2002. Curvature of co-links uncovers hidden thematic layers in the World Wide Web. *Proceedings of the National Academy of Sciences* 99, 9 (2002), 5825–5829.

[14] Alessandro Epasto, Silvio Lattanzi, Vahab Mirrokni, Ismail Oner Sebe, Ahmed Taei, and Sunita Verma. 2015b. Ego-net community mining applied to friend suggestion. *Proceedings of the VLDB Endowment* 9, 4 (2015), 324–335.

[15] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015a. Efficient densest subgraph computation in evolving graph. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. ACM, 300–310.

[16] Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. 2008. Maintaining bounded-size sample synopses of evolving datasets. *The VLDB Journal* 17, 2 (2008), 173–201.

[17] A. Hajnal and E. Szemerédi. 1970. Proof of a conjecture of P. Erdős.d. In *Combinatorial Theory and Its Applications, II (Proc. Colloq., Balatonfüred, 1969)*, P. Erdős, A. Rényi, and Vera T. Sós (Eds.). 601–623.

[18] Bronwyn H. Hall, Adam B. Jaffe, and Manuel Trajtenberg. 2001. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. Technical Report. National Bureau of Economic Research.

[19] Rob J. Hyndman and Anne B. Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* 22, 4 (2006), 679–688.

[20] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Transactions on Knowledge Discovery from Data* 9, 3 (2015), 15:1–15:21.

[21] Hossein Jowhari and Mohammad Ghodsi. 2005. New streaming algorithms for counting triangles in graphs. In *Proceeding of the 11th Annual International Conference on Computing and Combinatorics: (COCOON'05)*. Springer, 710–716.

[22] Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *Automata, Languages, and Programming*, Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer (Eds.). Lecture Notes in Computer Science, Vol. 7392. Springer, 598–609.

[23] Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. 2012. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics* 8, 1–2 (2012), 161–185.

[24] Konstantin Kutzkov and Rasmus Pagh. 2013. On the streaming complexity of computing local clustering coefficients. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. ACM, 677–686.

[25] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. ACM, 591–600.

[26] Matthieu Latapy. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science* 407, 1 (2008), 458–473.

[27] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (2007), 2.

[28] Yongsub Lim and U. Kang. 2015. MASCOT: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, 685–694.

[29] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. 2011. Approximate counting of cycles in streams. In *European Symposium on Algorithms (ESA'11)*. Springer, 677–688.

[30] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[31] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis* (2nd ed.). Cambridge University Press.

[32] Rasmus Pagh and Charalampos E. Tsourakakis. 2012. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters* 112, 7 (Mar. 2012), 277–281.

[33] Ha-Myung Park, Francesco Silvestri, U. Kang, and Rasmus Pagh. 2014. MapReduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM'14)*. ACM, 1739–1748. https://doi.org/10.1145/2661829.2662017

[34] Ha-Myung Park and Chin-Wan Chung. 2013. An efficient MapReduce algorithm for counting triangles in a very large graph. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management (CIKM'13)*. ACM, 539–548.

[35] Ha-Myung Park, Sung-Hyon Myaeng, and U. Kang. 2016. PTE: Enumerating trillion triangles on distributed systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 1115–1124.

[36] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1870–1881.

[37] Matthew Skala. 2013. Hypergeometric tail inequalities: Ending the insanity. *arXiv preprint* 1311.5939 (2013).

[38] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. ACM, 607–614.

[39] The Koblenz Network Collection (KONECT). 2016. Last.fm song network dataset. Retrieved from http://konect.uni-koblenz.de/networks/lastfm_song.

[40] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. 2009. Doulion: Counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, 837–846.

[41] Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. 2011. Triangle sparsifiers. *Journal of Graph Algorithms and Applications* 15, 6 (2011), 703–726.

[42] Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software* 11, 1 (1985), 37–57.

Received June 2016; revised January 2017; accepted March 2017