Approximate 2-hop neighborhoods on incremental graphs: An efficient lazy approach

Luca Becchetti
Sapienza University of Rome
Italy
becchetti@diag.uniroma1.it

Andrea Clementi
Tor Vergata University of Rome
Italy
clementi@mat.uniroma2.it

Luciano Gualà
Tor Vergata University of Rome
Italy
guala@mat.uniroma2.it

Luca Pepé Sciarria
Tor Vergata University of Rome
Italy
luca.pepesciarria@gmail.com

Alessandro Straziota
Tor Vergata University of Rome
Italy
alessandro.straziota@uniroma2.it

Matteo Stromieri
Tor Vergata University of Rome
Italy
matteo.stromieri@students.uniroma2.eu

ABSTRACT

In this work, we propose, analyze and empirically validate a lazy-update approach to maintain accurate approximations of the 2-hop neighborhoods of dynamic graphs resulting from sequences of edge insertions.

We first show that under random input sequences, our algorithm exhibits an optimal trade-off between accuracy and insertion cost: it only performs $O(\frac{1}{\varepsilon})$ (amortized) updates per edge insertion, while the estimated size of any vertex's 2-hop neighborhood is at most a factor ε away from its true value in most cases, *regardless* of the underlying graph topology and for any $\varepsilon > 0$.

As a further theoretical contribution, we explore adversarial scenarios that can force our approach into a worst-case behavior at any given time t of interest. We show that while worst-case input sequences do exist, a necessary condition for them to occur is that the *girth* of the graph released up to time t be at most 4.

Finally, we conduct extensive experiments on a collection of real, incremental social networks of different sizes, which typically have low girth. Empirical results are consistent with and typically better than our theoretical analysis anticipates. This further supports the robustness of our theoretical findings: forcing our algorithm into a worst-case behavior not only requires topologies characterized by a low girth, but also carefully crafted input sequences that are unlikely to occur in practice.

Combined with standard sketching techniques, our lazy approach proves an effective and efficient tool to support key neighborhood queries on large, incremental graphs, including neighborhood size, Jaccard similarity between neighborhoods and, in general, functions of the union and/or intersection of 2-hop neighborhoods.

1 INTRODUCTION

In this paper, we consider the task of processing a possibly large, dynamic graph $G(V, E)$, incrementally provided as a stream of edge insertions, so that at any point of the stream it is possible to efficiently evaluate different queries that involve functions of the *h-hop neighborhoods* of its vertices. For a vertex $v \in V$, its *h*-hop neighborhood is simply the *set* of vertices that are within h hops from v. In the remainder, *h*-hop neighborhoods are called *h-balls* for brevity. As concrete examples of query types we consider, one might want to estimate the size of the 2-ball at a given vertex, or the Jaccard similarity between the 2-balls centered at any given two vertices, or other indices of a similar flavor that depend on the intersection or union between 1-balls and/or 2-balls, just to mention a few.

Neighborhood-based indices are common in key mining tasks, such as link prediction in social [33] and biological networks [43] or to describe statistical properties of large social graphs [6]. For example, 2-hop neighborhoods are important in social network analysis and similarity-based link prediction [4, 47, 49], while accurate approximations of *h*-balls' sizes are used to estimate key statistical properties of (very) large social networks [3, 8], or as link-based features in classifiers for Web spam detection [7].

When the graph is static, an effective approach to this general task is to treat *h*-balls as subsets of the vertices of the graph, suitably represented using approximate summaries or sketches [1]. This line of attack has proved successful, for example in the efficient and scalable evaluation of important neighborhood-based queries on massive graphs that in part or mostly reside on secondary storage [6, 8, 24, 34].

Nowadays, standard applications in social network analysis often entail dynamic scenarios in which input graphs *evolve over time*, under a sequence of edge insertions and possibly deletions [2].

With respect to a static scenario, the dynamic case poses new and significant challenges even in the incremental setting, as soon as $h > 1$.[1] To see this, it may be useful to briefly sketch the cost of maintaining 1- and 2-balls exactly under a sequence of edge

---

[1]The case $h = 1$ is considerably simpler and it *barely relates to graphs*: adding or removing one edge (u, v) simply requires updating the 1-balls of u and v accordingly,

insertions, as we discuss in more detail in Section 2. When $h = 2$, each $\texttt{Insert}(u, v)$ operation entails (see Algorithm 1 and Figure 1): i) updating the 2-ball of u to its union with the 1-ball of v and viceversa (what we call a *heavy* update); ii) adding v to the 2-ball of *each* neighbor of u and viceversa (what we call a *light* update). Both heavy and light updates can result in high computational costs per edge insertion: a heavy update can be expensive if at least one of the neighborhoods to merge is large; on the other hand, light updates are relatively inexpensive, but they can be numerous when large neighborhoods are involved, again resulting in a high overall cost per edge insertion. Unfortunately, h-balls can grow extremely fast with h in many social networks, already as one switches from $h = 1$ to $h = 2$ [3, 7]. For the same reason, maintaining lossless representations of 2-balls for each vertex of such networks might require considerable memory resources and might negatively impact the cost of serving neighborhood-based queries that involve moderately or highly central vertices.

To address the aforementioned issues for graphs that reside in main memory, one might want to trade some degree of accuracy for the following broad goals: 1) designing algorithms with low update costs, possibly $O(1)$ amortized per edge insertion; 2) minimizing memory footprint beyond what is needed to store the graph; 3) maintaining 1- and 2-balls using data structures that afford efficient, real-time computation of queries as the ones mentioned earlier with minimal memory footprint.

Heavy updates are natural and well-known candidates for efficient (albeit approximate) implementation using compact, sketch-based data structures [1, 5, 10, 11, 26]. However, sketches alone are of no avail in handling light updates, whose sheer potential number requires a novel approach. The literature on efficient data structures that handle insertions and often deletions over dynamic graphs is rich. However, efficient solutions to implement neighborhood-based queries on dynamic edge streams are only known for 1-balls [13, 17, 29, 45], nor do approaches devised for other dynamic problems adapt to our setting in any obvious way, something we elaborate more upon in Section 1.2.

1.1 Our Contribution

In this paper, we propose an approach that trades some degree of accuracy for a substantial improvement in the average number of light updates. In a nutshell, upon an edge insertion, our algorithm performs the (two) corresponding heavy updates, but in general only a subset of the required light updates, according to a scheme that combines a threshold-based mechanism and a randomized, batch-update policy. Hence, for every vertex u, we only keep an approximation (a subset to be specific) of u's 2-ball. If 1- and 2-balls are represented with suitable data sketches, our approach affords constant average update cost per edge insertion.[2] While the behavior and accuracy guarantees of most sketching techniques are well understood, the estimation error induced by lazy updates can be

arbitrarily high in some cases. The main focus of this paper is on the latter aspect, which is absent in the static case but critical in the dynamic setting. Accordingly, we assume lossless representations of 1-balls and approximate 2-balls in our theoretical analyses in Sections 2 and 4, while we use standard sketching techniques to represent 1- and 2-balls in the actual implementations of the algorithms and baselines we consider in the experimental analysis discussed in Section 5.

Almost-optimal performance on random sequences. We prove in Section 3 that even a simplified, deterministic variant of our lazy-update Algorithm 3 achieves asymptotically optimal expected performance when the sequence of edge insertions is a random, uniform permutation over an *arbitrary* set of edges. In other words, our lazy approach is robust to adversarial topologies as long as the edge sequence follows a random order. Formally, we prove that, for any desired $0 < \varepsilon < 1$, our algorithm only performs $O(\frac{1}{\varepsilon})$ (amortized) updates per edge insertion, while at any time t and for every vertex v, the estimated size of v's 2-ball is, in expectation, at most a factor ε away from its true value. We further prove that this approximation result holds with a probability that exponentially increases with the true size of the 2-ball itself (Theorem 1). Thanks to this analysis in concentration, our results can be extended to other functions of 2-balls, including union, intersection and Jaccard similarity (see Corollary 2 for this less obvious case).

As positive as this result may sound, it begs the following questions from a careful reader: 1) Are the results above robust to adversarial sequences? 2) Is a performance analysis under random sequences representative of practical scenarios? More generally, does our lazy scheme offer significant practical advantages?

Performance analysis on adversarial inputs. While our results for random sequences are optimal regardless of the underlying graph's topology, one might wonder about the ability of an adversary to design *worst-case, adaptive sequences* that force our approach to behave poorly and, in this case, whether any conditions on the graph topology are *necessary* for this to happen. We investigate this issues in Section 4, where we first show that it is possible to design worst-case sequences of edge insertions that force our algorithm to perform arbitrarily worse than the random setting (Theorem 3). However, as a further contribution, we also prove that worst-case input sequences exist *only if* the *girth* [19] of the final graph is at most 4. More precisely, we show that a randomized, special case of Algorithm 3 achieves asymptotically optimal performance on a class of graphs that contains all graphs with girth at least 5, [3] even when the input sequence is chosen by an adaptive adversary.

Experimental analysis. As for the second question above, an analysis under random permutation sequences as the one in Section 3 is relatively common in the literature on dynamic edge streams and data streams [12, 28, 30, 36]. Yet, one might rightly wonder about its practical significance for the task considered in this paper. We investigate this question in Section 5, where we conduct experiments on small, medium and large-sized, incremental graphs (whose main properties are summarized in Table 1). At least on the diverse sample of real networks we consider, experimental results

---

i.e., updating two corresponding set sketches by adding or removing one item. This has been the focus of extensive work in the recent past that we discuss in Section 1.2.

[2]The particular sketch used depends on the neighborhood queries we want to be able to serve. When sketches are used, the cost of merging two neighborhoods corresponds to the cost of combining the corresponding sketches, which is typically a constant that depends on the desired approximation guarantees. For example, if we are interested in the Jaccard similarity between pairs of 1- and/or 2-balls, this cost will be proportional to the (constant) number of minhash values we use to represent each neighborhood.

[3]The class is in fact more general since it also includes graphs with a "bounded" number of cycles of length at most 4. See Definition 4.1, for a formal definition of this class.

on the estimation of key queries such as size and Jaccard similarity are consistent with the theoretical findings from Section 3. The main take-away is that, when using sketches to represent the 1- and 2-balls, the errors obtained with our lazy update policy are similar and fully comparable to those of the baseline, which performs all necessary light updates. At the same time, our algorithm proves to be significantly faster than the baseline, sometimes achieving a speedup of up to 90×.

We finally remark that the datasets we consider are samples of real social networks. As such, they have relatively large local and global clustering coefficients[4] and thus low girth. Hence, our experimental analysis further supports the robustness of our theoretical findings: forcing our algorithm(s) into a worst-case behavior not only requires topologies characterized by a low girth, but also carefully crafted input sequences that are unlikely to occur in practice.

Remark. We finally stress that although we present them for the undirected case for ease of presentation and for the sake of brevity, our algorithms apply to directed graphs as well,[5] while our analysis extends to the directed case with minor modifications.

1.2 Further related work

Efficient data structures for queries that involve h-balls of a dynamic graph turn out to be useful in different network applications. Besides those we mentioned earlier [4, 7, 47], we cite here the work [14], where the notion of *2-hop Neighbor Class Similarity* (2NCS) is proposed: this is a new quantitative graph structural property that correlates with *Graph Neural Networks* (GNN) [40, 44] performance more strongly and consistently than alternative metrics. 2NCS considers two-hop neighborhoods as a theoretically derived consequence of the two-step label propagation process governing GCN's training-inference process.

As remarked in the introduction, efficient solutions for Jaccard similarity queries on 1-balls have been proposed for different dynamic graph models: all of them share the use of suitable data sketches to manage insertion and deletion of elements from sets. In particular, [13, 29, 45] proposes and compares different approaches that work in the fully-dynamic streaming model, while an efficient solution, based on a buffered version of the k-min-hashing scheme is proposed in [17]. This works in the fully-dynamic streaming model and allows recovery actions when certain "worst-case" edge deletion events occur. A further algorithm is presented in [48], where *bottom-k sketches* [18] are used to perform dynamic graph clustering based on Jaccard similarity among vertices' neighborhoods. We remark that none of these previous approaches include ideas or tools that can be adapted to efficiently manage the 2-ball update-operations we need to implement in this work.

As for other queries that might be "related" or "useful" in our setting, a considerable amount of work on data structures that support edge insertions and deletions exists for several queries, such as connectivity or reachability, (exact or approximate) distances, minimum spanning tree, (approximate) *betweenness centrality*, and so on. We refer the reader to [27] for a nice survey on experimental

and theoretical results on the topic. To the best of our knowledge however, none of these approaches can be obviously adapted to handle the types of queries we consider in this work. For example, a natural idea would be using an incremental data structure to dynamically maintain the first h levels of a BFS tree, such as [23, 38], that achieve $O(h)$ amortized update time. However, let alone effectiveness in efficiently serving queries as the ones we consider here, the data structure uses $\Omega(n)$ space per BFS. This is prohibitive in our setting, where we would need to instantiate one such data structure for each vertex, with total space $\Omega(n^2)$. Moreover, since in a degree-Δ graph $\Theta(\Delta)$ BFS trees can change following a single edge insertion, the corresponding amortized time per edge insertion could be as high as $\Theta(\Delta)$, which is basically the same cost of the baseline solution we discuss at the beginning of Section 2.

2 LAZY-UPDATE ALGORITHMS

After giving some preliminaries we will use through all this paper, in Section 2.1 we describe the lazy-update algorithmic scheme, while in Section 2.2, we provide a general bound on its amortized update cost that holds for arbitrary sequences of edge insertions.

Preliminaries and notations. The dynamic (incremental) graph model we study can be defined as a sequence $\mathcal{G} = \{G^{(0)}(V, E^{(0)}), \ldots, G^{(t)}(V, E^{(t)}), \ldots G^{(T)}(V, E^{(T)})\}$, where: (i) the set of vertices $V = \{1, \ldots, n\}$ is fixed, (ii) $T \leq \binom{n}{2}$ is the final graph, while (iii) $E^{(t)}$ is the subset of edges at time t. Note that this changes in every time step $t \geq 1$, as a new edge $e^{(t)}$ is inserted, so that $E^{(t+1)} = E^{(t)} \cup \{e^{(t)}\}$. We remark that our analysis and all our results can be easily adapted to a more general model that includes any combination of the following variants: (i) growing vertex sets, (ii) multiple insertions of the same edge, and (iii) directed edges (thus yielding directed graphs). However, the corresponding adaptations of our analysis would require significantly heavier notation and some technicalities that we decided to avoid for the sake of clarity and space.

Our goal is to design algorithms that, at every time step $t \geq 1$, are able to efficiently compute queries over the current 2-balls of $G^{(t)}$. As mentioned in the introduction, our focus is on queries that are typical in graph mining such as: (i) given a vertex u, estimate the size of $B_2(u)$, and (ii) given two vertices $u, v \in V$, estimate the Jaccard similarity of the corresponding 2-balls:

$$\mathbf{J}(B_2(u), B_2(v)) = \frac{|B_2(u) \cap B_2(u)|}{|B_2(u) \cup B_2(u)|}.$$

Both the theoretical and experimental analysis of our lazy-update algorithms consider the following key performance measures: the *amortized update time* per edge insertion and the *approximation ratio* of our algorithms on the quantities $|B_2(u)|$ and $\mathbf{J}(B_2(u), B_2(v))$, for any choice of the input vertices. Intuitively, the amortized update time is the average time it takes to process a new edge, a more formal definition is deferred to Section 2, after a detailed description of the algorithms we consider.

We next summarize notation that is extensively used in the remainder of the paper. For a vertex $v \in V$ of a graph $G(V, E)$, we define:

$\mathcal{N}(v)$: the set of neighborhoods of the vertex v.

$\deg_v$: the degree of v. Notice that $\deg_v = |\mathcal{N}(v)|$;

$L_h(v)$: set of vertices at distance exactly h from v;

---

[4]At least the undirected ones.

[5]Of course, in this case we have directed h-balls, i.e., sets of a vertices that can be reached in at most h hops from a given vertex, or from which it is possible to reach the vertex under consideration in at most h hops.

$B_h(v)$: set of vertices at distance at most h from v.

The reader may have noticed that, in our notation above, the term t does not appear: this is due to the fact that our analysis holds at any (arbitrarily-fixed) time step, which is always clear from context.

2.1 Algorithm description

Consider the addition of a new edge (u, v) to G. Clearly, the only 2-balls that are affected are those centered at u, v, and at every vertex $w \in \mathcal{N}(u) \cup \mathcal{N}(v)$. A *baseline* strategy, given as Algorithm 1 for the sake of reference, tracks changes exactly and thus updates all 2-balls that are affected by an edge insertion.

Algorithm 1: Baseline algorithm.

1 **Function** Insert(u, v):
2   **foreach** $x \in \mathcal{N}(u) \cup \{u\}$ **do**
3         $B_2(v) \leftarrow B_2(v) \cup \{x\}$
4         $B_2(x) \leftarrow B_2(x) \cup \{v\}$
5   **end**
6   **foreach** $x \in \mathcal{N}(v) \cup \{v\}$ **do**
7         $B_2(u) \leftarrow B_2(u) \cup \{x\}$
8         $B_2(x) \leftarrow B_2(x) \cup \{u\}$
9   **end**
10 **end**

The magnitude of the changes (and the associated computational costs) induced by Insert(u, v) vary. In particular, $B_2(u)$ can change significantly, as all vertices in $B_1(v)$ will be included in $B_2(u)$ (we refer to this as a *heavy* update). Instead, for any vertex $w \in \mathcal{N}(u) \setminus \{v\}$, $B_2(w)$ will grow by at most one element, namely v (this is referred to as a *light* update). Symmetrically, the same holds for v and for every $w \in \mathcal{N}(v) \setminus \{u\}$.

A key observation at this point is that, while heavy updates can be addressed using (possibly, approximate) data structures that allow efficient merging of 1- and 2-balls, this line of attack fails with light updates, whose cost derives from their potential number, which can be large in many real cases, as we noted in the introduction.

A first idea to reduce the average number of updates per edge insertion is to perform heavy updates immediately, instead processing light updates in batches that are performed occasionally. More precisely, when a new edge (u, v) arrives, it is initially marked as a *red edge*. Whenever the number of red edges incident to a vertex u exceeds a certain threshold, all the corresponding light updates are processed, and the state of red edges is updated to *black*. See Figure 1 for an example.

The idea behind the threshold-based approach is to maintain a balance between the number of black and red edges for every vertex. While useful when edge insertions appear in a random order, this approach may fail when red edges considerably expand the original size of the 2-ball of some vertex u. In order to mitigate this problem, our algorithm uses a second ingredient: upon each edge insertion (u, v), the algorithm selects k vertices from $\mathcal{N}(u)$ and k from $\mathcal{N}(v)$ uniformly at random and performs a batch of light updates for the selected vertices, even if the threshold has not been reached yet.
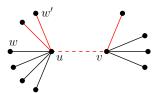

Figure 1: Example of insertion of a new edge (u, v). The algorithm merges the 1-ball of v with the 2-ball of u (heavy update), while it does not immediately add vertex v to the 2-ball of vertex w or any other of u's neighbors.

These ideas are formalized in Algorithm 3. For each vertex v, our algorithm maintains two sets $\hat{B}_1(v)$ and $\hat{B}_2(v)$, as well as the *black degree* Δ_v and *red degree* δ_v of v. Our algorithm guarantees that $\hat{B}_1(v)$ is exactly $B_1(v)$, while $\hat{B}_2(v)$ is in general a subset of $B_2(v)$. The algorithm uses two global parameters, namely a *threshold* $\varphi \in [0, 1]$, and an integer k. The role of the parameter φ can be understood as follows: when φ is set to 0, the algorithm performs all heavy and light updates for every edge insertion, ensuring that $\hat{B}_2(v)$ always matches $B_2(v)$. As φ increases, the update function becomes lazier: light updates are not always executed, and $\hat{B}_2(v)$ is typically a proper subset of $B_2(v)$. For instance, when $\varphi = 1$, light updates are performed in batches every time the degree of a vertex doubles. Parameter k specifies the number of neighbors of v that are randomly selected for update of their 2-balls whenever an edge insertion involving v occurs. This mechanism corresponds to Lines 14 to 16 of Algorithm 3.

We call Lazy-Alg(φ, k) the algorithm that runs Algorithm 2 on an initial graph $G^{(0)}$ and then processes a sequence S of edge insertions by running Algorithm 3 on each edge of S.

Algorithm 2: Init operation

Data: An undirected graph $G = (V, E)$, a threshold parameter $0 \leq \varphi \leq 1$, and an integer $k \geq 0$.
1 set φ and k as global parameters
2 **foreach** *vertex* $u \in V$ **do**
3     $\delta_u \leftarrow 0$
4     $\Delta_u \leftarrow \deg_u$
5     $\hat{B}_1(u) \leftarrow B_1(u)$
6     $\hat{B}_2(u) \leftarrow B_2(u)$
7 **end**

A note on neighborhood representation. As we mentioned in the introduction, we treat $\hat{B}_1(v)$ and $\hat{B}_2(v)$ as sets of vertices in this section and in Section 4. We remark that this only serves the purpose of analyzing the error introduced by our lazy update policies: lossless representations of 1- and 2-balls may be unfeasible for medium or large graphs and compact data sketches are typically used to represent them in such cases. The choice of the actual sketch strongly depends on the type of query (or queries) one wants to support, such as 1- or 2-ball sizes [8, 25] or Jaccard similarity between 2-balls [6, 10, 18]. All sketches used for typical neighborhood queries are well-understood and come with strong accuracy guarantees. Moreover, they allow to perform the union of 1- and 2-balls

Algorithm 3: INSERT

```
 1 Function Insert((u, v)):
 2     for x ∈ {u, v} do
 3         let y ∈ {u, v} \ {x}
 4         B̂₁(x) ← B̂₁(x) ∪ {y}
           // heavy update
 5         B̂₂(x) ← B̂₂(x) ∪ B̂₁(y)
 6         δₓ ← δₓ + 1
 7         if δₓ ≥ φ · Δₓ then
 8             Δₓ ← Δₓ + δₓ
 9             δₓ ← 0
10             foreach z ∈ N(x) do
                  // batch of light updates
11                 B̂₂(z) ← B̂₂(z) ∪ B̂₁(x)
12             end
13         else
14             select k vertices w₁, . . . , wₖ ∈ N(x) u.a.r.
15             for i = 1, . . . , k do
                  // batch of light updates
16                 B̂₂(wᵢ) ← B̂₂(wᵢ) ∪ B̂₁(x)
17             end
18         end
19     end
20 end
```

we are interested in time proportional to the sketch size, which is independent of the sizes of the balls to merge [1].

2.2 Cost analysis for arbitrary sequences

Consistently to what we remarked above, our cost analysis focuses on the number of *set-union* operations: This performance measure in fact dominates the computational cost of Algorithm 3. More in detail, given any sequence S of edge insertions, starting from an initial graph $G^{(0)}$, we denote by $T(S)$ the overall number of union operations performed in Lines 4, 5, 11 and 16 of Algorithm 3 on the input sequence S.

We observe that a trivial upper bound to $T(S)$ is $O(\Delta|S|)$, since each insertion can cost $O(\Delta)$ union operations where Δ is the maximum degree of the current graph. However, this trivial argument turns out to be too pessimistic: in what follows, we provide a more refined analysis of the amortized cost[6] per edge insertion. We say that an algorithm has *amortized cost* $\hat{c}$ per edge insertion if, for any sequence S of edge insertions, we have $T(S) \leq \hat{c}|S|$.

LEMMA 1. *Given any initial graph $G^{(0)}$ and any sequence S of edge insertions, the amortized update cost of Algorithm 3 is $O(\frac{1}{\varphi} + k)$ per edge insertion.*

PROOF. Let us first consider the case $k = 0$, i.e., when the random selection and the consequent instructions in Lines 14 to 16 are never

performed. Our amortized analysis makes use of the *accounting method* [41]. The idea is paying the cost of any batch of light updates by charging it to previous edge insertions. More precisely, we assign *credits* to each edge insertion that we will use to pay the cost of subsequent batches of light updates. Formally, the *amortized* cost of an edge insertion is defined as the *actual* cost of the operation, plus the credits we assign to it, minus the credits (accumulated from previous operations) we spend for it. We need to carefully define such credits in order to guarantee that the sum of the amortized costs is an upper bound to the sum of the actual costs, i.e. we always have enough credits to pay for costly batch light updates.

We proceed as follows. When we insert the edge (u, v), we put $2/\varphi$ credits on u and $2/\varphi$ credits on v. Now we bound the actual and amortized cost of each insertion.

First, consider an edge insertion (u, v) that does not trigger a batch of light updates. Its actual cost is 4 union operations (those in Lines 4 and 5, 2 for each endpoint of (u, v)). Then its amortized cost is upper-bounded by $4 + 4/\varphi = O(1/\varphi)$. Now consider the case in which the insertion causes a batch of light updates for u, or v, or both. We show that the credits accumulated by previous insertions are sufficient to pay for its cost. To see this, consider a batch of light updates involving vertex $x \in \{u, v\}$. And let Δ_x and δ_x be the current black and red degrees of x at that time (just before Line 7 is evaluated). It is clear that for vertex x we have accumulated $\delta_x \cdot 2/\varphi$ credits that now we use to pay for the cost of Lines 10 and 11. This cost equals to $\deg_x$ union operations, where $\deg_x$ is the current degree of vertex x. Since the batch of light updates has just been triggered, we have that $\delta_x \geq \varphi\Delta_x$, and hence we have at least $\delta_x \cdot 2/\varphi \geq \varphi\Delta_x \cdot 2/\varphi = 2\Delta_x$ credits to pay for the $\deg_x = \Delta_x + \delta_x = \Delta_x + \varphi\Delta_x \leq 2\Delta_x$ union operations. This concludes the proof.

Finally, to obtain the claim when $k > 0$, we notice that, in this case, every edge insertion causes $O(k)$ additional union operations. □

3 RANDOM EDGE SEQUENCES

In this section, we analyze the accuracy of our lazy-update algorithm(s) over an arbitrary dynamic graph, whose edges are given in input as a uniformly sampled, random permutation over its edge set. Dynamic graphs resulting from random sequences of edge insertions have been an effective tool to provide theoretical insights that have often proved robust to empirical validation in various dynamic scenarios [15, 30, 34–36]. In more detail, assume $G = (V, E)$, with $|E| = t$, is the graph observed up to some time t of interest. Following [35, 36], we assume that the sequence of edges up to time t is chosen uniformly at random from the set of all permutations over E.[7] The following fact is an immediate consequence of well-known and intuitive properties of random permutations. We state it informally for the sake of completeness, avoiding any further, unnecessary notation.

---

[6]The *amortized analysis* is a well-known method originally introduced in [41] that allows to compute tight bounds on the cost of a *sequence* of operations, rather than the worst-case cost of an individual operation. In more detail, we average the cost of a *worst case* sequence of operations to obtain a more meaningful cost per operation.

[7]It should be noted that this includes the general case in which t is any intermediate point of a longer stream that possibly extends well beyond t. In this case, it is well-known and easy to see that, conditioned on the subset E of the edges released up to time t, their sequence is just a permutation of E.

FACT 1. *Consider a dynamic graph $G = (V, E)$, whose edges are observed sequentially according to a permutation over E chosen uniformly at random. Then, for every $E' \subseteq E$, the sequence in which edges in E' are observed is itself a uniformly chosen, random permutation over E'.*

In the remainder, for an arbitrary vertex v, we analyze how well the output $\hat{B}_2(v)$ of Algorithm 3 approximates $B_2(v)$ at any round t in terms of its *coverage*:

DEFINITION 3.1. *We say that the output $\hat{B}_2(v)$ of LAZY-ALG(φ, k) is a $(1 - \varepsilon)$-covering of $B_2(v)$ if the following holds: i) $\hat{B}_2(v) \subseteq B_2(v)$; ii) $\mathbb{E}\left[|\hat{B}_2(v)|\right] \geq (1 - \varepsilon)|B_2(v)|$, where expectation is taken over the randomness of the algorithm and/or the input sequence. When the algorithm produces a $(1 - \varepsilon)$-covering $\hat{B}_2(v)$ of $B_2(v)$ for every v, we say it has* approximation ratio $\frac{1}{(1-\varepsilon)}$.

Our main result in this section is formalized in the following

THEOREM 1. *Let $\varepsilon \in (0, 1)$, and fix parameters $k = 0$ and $\varphi = \frac{\varepsilon}{1-\varepsilon}$. Consider any graph $G(V, E)$ submitted as a uniform random permutation of its edge set E to LAZY-ALG(φ, k). Then, at every time step $t \leq |E|$, the algorithm has approximation ratio $\frac{1}{1-\varepsilon}$. Moreover, for every $\alpha > 0$ and every vertex $v \in V$, we have:*

$$\mathbb{P}\left(|\hat{B}_2(v)| < \frac{1 - \alpha}{1 + \varphi}|L_2(v)|\right) \leq e^{-\frac{2\alpha^2|L_2(v)|}{(1+\varphi)^2}}.$$

PROOF. Fix a vertex $v \in V$ and a round $t \geq 1$. In the remainder of this proof, all quantities are taken at time t. We are interested in how close $|\hat{B}_2(v)|$ is to $|B_2(v)|$. To begin, we note that the following relationship holds deterministically:

$$|\hat{B}_2(v)| = 1 + |L_1(v)| + |\hat{B}_2(v) \cap L_2(v)|, \tag{1}$$

where the only random variable on the right hand side is the last term. We next define a partition $C = \{C_u : u \in L_1(v)\}$ of $L_2(v)$ as
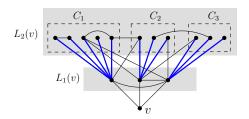

Figure 2: Example of a partition of $L_2(v)$ into three sets C_1, C_2, C_3. Edges connecting vertices $w \in L_2(v)$ to their respective partitions are thicker.

follows: for each $w \in L_2(v)$, we choose a vertex $u \in L_1(v) \cap \mathcal{N}(w)$ and assign w to C_u. This way, each vertex $w \in L_2(v)$ is associated to exactly *one* edge connecting one vertex in $L_1(v)$ to w (see Figure 2, where the edges in question are thick in the picture). Let E_v denote the set of such edges and note that i) E_v is a subset of the edges connecting vertices in $L_1(v)$ to those in $L_2(v)$, ii) $|E_v| = |L_2(v)|$ by definition and iii) $|C_u| \leq \deg_u -1$ for every $u \in L_1(v)$, given that C_u contains a subset of u's neighbors and (v, u) is always present. Moreover, Algorithm 3 guarantees that $|\hat{B}_2(v) \cap L_2(v)|$ is at least the

number of edges in E_v that are black. These considerations allow us to conclude that

$$|\hat{B}_2(v) \cap L_2(v)| \geq |\{e \in E_v : e \text{ is black}\}|.$$

A key observation at this point is that Algorithm 3 implies that for every $x \in V$, $\delta_x \leq \left\lfloor \frac{\varphi}{1+\varphi} \deg_x \right\rfloor$. As a consequence, if some $e = (u, w) \in E_v$ was not among the last $\left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor$ edges incident in u that were released within time t, it is necessarily black. For $e = (u, w) \in E_v$, with $u \in L_1(v)$ and $w \in L_2(v)$, let $X_e = 1$ if e was among the first $\deg_u - \left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor$ edges incident in u that were released up to time t and let $X_e = 0$ otherwise. Following the argument above, the event $(X_e = 1)$ implies the event "e is black", whence:

$$|\hat{B}_2(v) \cap L_2(v)| \geq |\{e \in E_v : e \text{ is black}\}| \geq \sum_{e \in E_v} X_e. \tag{2}$$

Next, we are interested in bounds on $\mathbb{P}(X_e = 1)$. Assume e is incident in u and let S be the set of edges incident in u observed up to time t. Then, from Fact 1, the sequence in which these edges are observed is just a random permutation of S. This immediately implies that, if e is incident to vertex $u \in L_1(v)$, then

$$\mathbb{P}(X_e = 1) = \frac{\deg_u - \left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor}{\deg_u} \geq \frac{1}{1 + \varphi}.$$

Together with (2) this yields:

$$\mathbb{E}\left[|\hat{B}_2(v)|\right] \geq 1 + |L_1(v)| + \frac{1}{1 + \varphi}|L_2(v)| \geq \frac{1}{1 + \varphi}|B_2(v)|.$$

We next show that $\sum_{e \in E_v} X_e$ is concentrated around its expectation when $|L_2(v)|$ is large enough, which implies that $|\hat{B}_2(v)|$ is concentrated around a value close to $|B_2(v)|$ in this case. The main technical hurdle here is that the X_e's correlated (albeit mildly, as we shall see). To prove concentration, we resort to Martingale properties of random edge sequences to apply the method of (Average) Bounded Differences [20]. In order to do this, we need bounds on $\mathbb{P}\left(X_e = 1|X_f = 1\right)$ and $\mathbb{P}\left(X_e = 1|X_f = 0\right)$, for $e, f \in E_v$. Assume again that e is incident in $u \in L_1(v)$ and that S is the set of edges incident in u observed up to time t. Assume first that f is also incident in u and that, without loss of generality, f is the i-th edge to appear among those in S. $X_f = 1$ implies $i \leq \deg_u - \left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor$. On the other hand, for any such choice for f's position in the sequence, Fact 1 implies that e will appear in a position j that is sampled uniformly at random from the remaining ones, so that $\mathbb{P}\left(X_e = 1|X_f = 1\right) = \frac{\deg_u - \left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor - 1}{\deg_u - 1}$ in this case. With a similar argument, it can be seen that $\mathbb{P}\left(X_e = 1|X_f = 0\right) = \frac{\deg_u - \left\lfloor \frac{\varphi}{1+\varphi} \deg_u \right\rfloor}{\deg_u - 1}$. Intuitively and unsurprisingly, the events $(X_e = 1)$ and $(X_f = 1)$ are negatively correlated, while $(X_e = 1)$ and $(X_f = 0)$ are positively correlated. This allows us to conclude that $\mathbb{P}\left(X_e = 1|X_f = 1\right) \leq \mathbb{P}\left(X_e = 1|X_f = 0\right)$ and

$$\mathbb{P}\left(X_e = 1|X_f = 0\right) - \mathbb{P}\left(X_e = 1|X_f = 1\right) \leq \frac{1}{\deg_u - 1}.$$

Assume next that f is not incident in u. Again from Fact 1, in this case f has no bearing on the relative order in which edges incident in u appear, so that $\mathbb{P}\left(X_e = 1 | X_f = 0\right) = \mathbb{P}\left(X_e = 1 | X_f = 1\right) = \mathbb{P}\left(X_e = 1\right)$. Now, without loss of generality, suppose that $f = (z, w)$, with $z \in L_1(v)$, so that $w \in C_z$. Denote by $E_v(z)$ the subset of edges in E_v with one end point in C_z. Moving to conditional expectations we have

$$
\begin{aligned}
&\mathbb{E}\left[\sum_{e \in E_v} X_e \mid X_f = 0\right] - \mathbb{E}\left[\sum_{e \in E_v} X_e \mid X_f = 1\right] \\
&= \sum_{e \in E_v} \left(\mathbb{P}\left(X_e = 1 \mid X_f = 0\right) - \mathbb{P}\left(X_e = 1 \mid X_f = 1\right)\right) \\
&= \sum_{e \in E_v \setminus E_v(z)} \left(\mathbb{P}\left(X_e = 1 \mid X_f = 0\right) - \mathbb{P}\left(X_e = 1 \mid X_f = 1\right)\right) \\
&\quad + \sum_{e \in E_v(z)} \left(\mathbb{P}\left(X_e = 1 \mid X_f = 0\right) - \mathbb{P}\left(X_e = 1 \mid X_f = 1\right)\right) \\
&\leq \frac{|C_z|}{\deg_z - 1} \leq 1,
\end{aligned}
$$

where the third inequality follows from the definition of C_z, since f is incident in z, while the last inequality follows since $|C_z| \leq \deg_z - 1$ for every $z \in L_1(v)$, because one of the edges incident in z is by definition the one shared with v.

We can therefore apply [20, Definition 5.5 and Corollary 5.1], with $c \leq |L_2(v)|$ to obtain, for every $\alpha > 0$:

$$
\mathbb{P}\left(\mathbb{E}\left[\sum_{e \in E_v} X_e\right] - \sum_{e \in E_v} X_e > \alpha \mathbb{E}\left[\sum_{e \in E_v} X_e\right]\right) \leq e^{-\frac{2\alpha^2 |L_2(v)|}{(1+\varphi)^2}},
$$

where in the right hand side we also used the bound $\mathbb{E}\left[\sum_{e \in E_v} X_e\right] \geq \frac{1}{1+\varphi}|L_2(v)|$ we showed earlier. Finally, we recall (1) and (2) to conclude that $|\hat{B}_2(v)| \geq \frac{1-\alpha}{1+\varphi}|L_2(v)|$ with (at least) the same probability. □

Theorem 1 easily implies approximation bounds on indices that depend on the union and/or intersection of 2-balls. For example, we immediately have the following approximation bound on the Jaccard similarity between any pair of 2-balls.

COROLLARY 2. *Under the same assumptions as Theorem 1, at any time step $t \geq 1$ and for any pair of vertices $u, v \in V$, LAZY-ALG(φ, k) guarantees the following approximation of the Jaccard similarity between $B_2(u)$ and $B_2(v)$ with probability at least $1 - e^{-\frac{2\alpha^2 |L_2(u)|}{(1+\varphi)^2}} - e^{-\frac{2\alpha^2 |L_2(v)|}{(1+\varphi)^2}}$:*

$$
\frac{\mathbf{J}(B_2(u), B_2(v))}{1 - 2\varepsilon'} \geq \mathbf{J}(\hat{B}_2(u), \hat{B}_2(v)) \geq (1 - \varepsilon')\mathbf{J}(B_2(u), B_2(v)) - \varepsilon',
\tag{3}
$$

where $\varepsilon' = \frac{\varphi + \alpha}{1 + \varphi}$.

PROOF. It is easy to see that $|\hat{B}_2(u)| \geq (1 - \varepsilon')|B_2(u)|$ and $|\hat{B}_2(v)| \geq (1 - \varepsilon')|B_2(v)|$ together imply (3) deterministically. The result then immediately follows from Theorem 1 and a union bound on the events $(|\hat{B}_2(u)| < (1 - \varepsilon')|L_2(u)|)$ and $(|\hat{B}_2(v)| < (1 - \varepsilon')|L_2(v)|)$. □

4 ADVERSARIAL EDGE SEQUENCES

We next study our lazy-update algorithm in an adversarial framework. We show in Section 4.1 that if the adversary can *both*: i) choose a worst-case graph G *and* ii) submit G according to an *adaptive* sequence of edge insertions, then it is possible to prove a strong lower bound on the achievable update-time/approximation trade-off of the whole parameterized scheme LAZY-ALG(φ, k).

On the other hand, in Section 4.2 we provide a *necessary* condition for the adversarial, worst-case framework above: the *girth* [19] of G must be smaller than 5. More precisely, G must contain an unbounded number of triangles and cycles of length 4. We do this by showing that for a suitable parameter setting, algorithm LAZY-ALG(φ, k) achieves almost-optimal trade-offs even on adversarial edge insertion sequences, for every graph that has a bounded number of such small cycles (see Definition 4.1 for a formal definition of this class of graphs).

4.1 A lower bound for adversarial sequences

The lower bound for the adversarial framework described above is formalized in the following result on the approximation ratio (see Def. 3.1)

THEOREM 3. *For every $\varphi \in [0, 1]$, and integer $k \geq 0$, if LAZY-ALG(φ, k) has approximation ratio $\rho \geq 1$, then it must have an amortized update cost of $\Omega(\Delta/\rho^3)$, where Δ is the maximum degree of the graph.*

Figure 3: Black edges are present at $t = 0$, while red ones are inserted in the interval $\{1, 2, \ldots, \Delta\rho^2\}$. At time $t > 0$, an edge with one endpoint in $u_{1t \mod \Delta}$ and the other in a distinct 0-degree vertex in S_2 is added.

PROOF. Fix $\rho \geq 1$, and assume that LAZY-ALG(φ, k) has an approximation ratio of at most ρ. We will show that there exist an initial graph $G^{(0)}$ with degree Δ and a sequence of edge insertions against which LAZY-ALG(φ, k) must incur an amortized update time of $\Omega(\Delta/\rho^3)$.

Note that for $k > 0$, the algorithm is randomized. In order to address this, we prove our lower bound for every possible realization of the randomness used by the algorithm. Therefore, we assume the values of the random bits used by LAZY-ALG(φ, k) are fixed arbitrarily (and optimally) and we assume henceforth that the behavior of the algorithm is completely deterministic.

The initial graph $G^{(0)}$ consists on 2Δ vertices forming a complete bipartite graph with sides $S_0 = \{u_{01}, \ldots, u_{0\Delta}\}$ and $S_1 = \{u_{11}, \ldots, u_{1\Delta}\}$, along with an additional set S_2 of $\Delta\rho^2$ isolated vertices (see Figure 3). The sequence of edge insertions is defined as

follows: for each vertex in S_1, we insert ρ^2 new edges. Each of these $\Delta\rho^2$ edges connects a vertex in S_1 to a previously isolated vertex in S_2.

Consider the time instant right after all edge insertions. Since we assumed that the algorithm guarantees an approximation ratio of ρ, it holds that for every vertex $u \in S_0$, $|\hat{B}_2(u)| \geq \frac{1}{\rho}|B_2(u)| = \frac{1}{\rho}(2\Delta + \Delta\rho^2) = \Delta\rho + 2\Delta/\rho$. This implies that after all edge insertions, u must be aware of at least $\Delta\rho + 2\Delta/\rho - 2\Delta = \Delta\rho - 2\Delta(1 - 1/\rho)$ vertices from S_2.

We say that there is a *message* from v to u if vertex v performs a union operation of the form $\hat{B}_2(u) \leftarrow \hat{B}_2(u) \cup \hat{B}_1(v)$.

Since, at any time, every vertex $v \in S_1$ is adjacent to at most ρ^2 vertices in S_2, it must be that each $u \in S_0$ must have received at least $\frac{\Delta\rho - 2\Delta(1-1/\rho)}{\rho^2} = \Omega(\Delta/\rho)$ messages from vertices in S_1. As a consequence, the total number of messages are at least $\Omega(\Delta^2/\rho)$. As the number of insertions is $\Delta\rho^2$, the amortized update cost per insertion is at least $\frac{\Omega(\Delta^2/\rho)}{\Delta\rho^2} = \Omega(\Delta/\rho^3)$. □

We have special cases as corollaries. We need amortized update cost $\Omega(\Delta)$ if we want $\rho = O(1)$, $\Omega(\sqrt[4]{\Delta})$ if we want $\rho = O(\sqrt[4]{\Delta})$ and so on.

REMARK 1. *The lower bound in Section 4.1 in fact holds for a wider class of algorithms. Informally speaking, this class includes any local algorithm that limits its online updates to the 2-hop neighbors of u and v only. Making this claim more formal requires addressing several technical issues that are outside the scope of the present work.*

4.2 Locally γ-sparse graphs

In this section, we provide the characterization of a class of graphs for which our lazy-update approach always guarantees good amortized cost/approximation trade-offs, even under the assumption of adversarial edge insertion sequences.

Given a graph $G(V, E)$ and a subset $V' \subseteq V$, we denote by $G[V']$ the subgraph induced by V'. Informally, a graph is locally γ-sparse if every node in $B_2(u) \setminus \{u\}$ has roughly at most γ neighbors in $L_1(u)$. This notion can be formalized as follows.

DEFINITION 4.1 (LOCALLY γ-SPARSE GRAPHS). *Let $\gamma \in \{0, 1 \ldots, n-1\}$. A graph $G(V, E)$ is said locally γ-sparse if for each vertex $u \in V$: (i) $\forall v \in L_1(u)$ the degree of v in $G[L_1(u)]$ is at most γ, and (ii) $\forall w \in L_2(u)$ the degree of w in $G[L_1(v) \cup \{w\}]$ is at most $\gamma + 1$.*

Observe that the class of locally γ-sparse graphs grows monotonically with γ, including all possible graphs for $\gamma = n - 1$, while the most restricted class is obtained for $\gamma = 0$. It is interesting to note that locally γ-sparse graphs are not necessarily sparse in absolute terms. For example, for $\gamma = 0$, the class coincides with the well-known class of graphs with *girth* at least 5: these graphs can have up to $\Theta(n^{\frac{3}{2}})$ edges assuming Erdös' Girth Conjecture [22] (the proof of such equivalence is given in Appendix A.2).

A first, preliminary analysis of our lazy-update approach considers the deterministic version of Algorithm 3, i.e., when $k = 0$. It turns out that this version achieves an approximation ratio of $\frac{\gamma+1}{1-\varepsilon}$ and amortized cost $O(1/\varepsilon)$ (see Appendix A.2). So, the approximation accuracy decreases linearly in the parameter γ. Interestingly enough, we instead show that a suitable number of random light

updates allows Algorithm 3 to perform much better than its deterministic version. This is the main result of this section and it is formalized in the next

THEOREM 4. *Let $\varepsilon \in (0, 1)$, and let $G^{(0)}$ be an initial graph. Consider any sequence of edge insertions that yields a final graph G. If G is locally γ-sparse* LAZY-ALG$\left(\varphi = 1, nk = \frac{4(\gamma+1)}{\varepsilon}\right)$ *has approximation ratio of $\frac{1}{1-\varepsilon}$ and amortized cost $O\left(\frac{\gamma+1}{\varepsilon}\right)$.*

Proof of Theorem 4. For the remainder of this proof, we define the notion of *quasi-black* edge. A red edge (v, w), with $v \in L_1(u)$ and $w \in L_2(u)$, is said to be *quasi-black* for u if u has been randomly selected by v at Line 14 of Algorithm 3 *at least once* during or after the insertion of (v, w), ensuring that $w \in \hat{B}_2(u)$.

In this section, we use $\hat{\ell}_v$ and ℓ_v to denote the number of *quasi-black* and *red* edges, respectively, that connect v to vertices in $L_2(u)$. Similarly, we use b_v to represent the number of *black* edges of v having the other endpoint in $L_2(u)$. Notice that $\hat{\ell}_v$ is a random variable that counts how many vertices out of ℓ_v are in $\hat{B}_2(u)$. We now proceed by first stating a property whose proof can be found in Appendix A.3.

LEMMA 2. *For each $v \in L_1(u)$, we have $\mathbb{E}\left[\hat{\ell}_v\right] \geq \ell_v - \frac{2(b_v + \gamma + 1)}{k}$.*

Now, let β denote the number of vertices in $L_2(u)$ that have at least one black edge from $L_1(u)$. Consequently, these vertices are included in $\hat{B}_2(u)$. We have the following:

LEMMA 3. *Let $G = (V, E)$ be locally γ-sparse, and $u \in V$. Then*

$$\beta \geq \sum_{v \in L_1(u)} \frac{b_v}{\gamma + 1}.$$

PROOF. The inequality follows from the fact that every node in $L_2(u)$ can have at most $\gamma + 1$ neighbors in $L_1(u)$. □

We are now ready to prove Theorem 4.

The amortized update cost follows directly from Lemma 1.

For the approximation quality, let us consider any vertex $u \in V$. For technical convenience, we will define a subgraph $\widetilde{G}$ of G by removing suitable edges from G, and we establish the following two properties: (i) if $k \geq \frac{2(\gamma+1)}{\varepsilon}$, the LAZY-ALG guarantees a $(1-\varepsilon)$-covering of $B_2(u)$ when the sequence of edge insertion is restricted to edges in $\widetilde{G}$; (ii) property (i) implies that the LAZY-ALG also guarantees a $(1-\varepsilon)$-covering of $B_2(u)$ for G, provided that $k \geq \frac{4(\gamma+1)}{\varepsilon}$.

The subgraph $\widetilde{G}$ is obtained from G through the following process. For each vertex $w \in L_2(u)$, if there exists a black edge (v, w) with $v \in L_1(u)$, then we remove all the red edges incident to w that originate from $L_1(u)$. Otherwise, if all edges from $L_1(u)$ to w are red, we retain only one and remove the rest (see Figure 4).

We now prove property (i). We analyze the process at a generic time $t > 0$. We want to prove that $\mathbb{E}\left[|\hat{B}_2(u)|\right] \geq (1-\varepsilon)|B_2(u)|$, for any vertex $u \in V$. Since $L_1(u)$ is always included in $\hat{B}_2(u)$, it is sufficient to prove that $|\hat{B}_2(u) \cap L_2(u)| \geq (1-\varepsilon)|L_2(u)|$ in expectation.

By construction of $\widetilde{G}$, we have that $|\hat{B}_2(u) \cap L_2(u)| = \beta + \sum_{v \in L_1(v)} \hat{\ell}_v$, while $|L_2(u)| = \beta + \sum_{v \in L_1(v)} \ell_v$. Thus, we want to
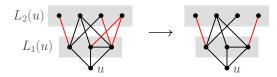
Figure 4: The 2-hop neighborhood of a vertex u (left), and its corresponding structure in the subgraph $\widetilde{G}$ (right).

show that

$$\sum_{v \in L_1(v)} \hat{\ell}_v \geq (1-\varepsilon) \sum_{v \in L_1(v)} \ell_v - \varepsilon\beta. \tag{4}$$

By Lemma 3, (4) is true when

$$\sum_{v \in L_1(u)} \hat{\ell}_v \geq (1-\varepsilon) \sum_{v \in L_1(u)} \ell_v - \frac{\varepsilon}{\gamma+1} \sum_{v \in L_1(u)} b_v$$

$$= \sum_{v \in L_1(u)} \left((1-\varepsilon)\ell_v - \frac{\varepsilon}{\gamma+1} b_v \right). \tag{5}$$

In turn, the inequality in (5) holds in expectation if it holds term-by-term, i.e., when

$$\mathbb{E}\left[\hat{\ell}_v\right] \geq (1-\varepsilon)\ell_v - \frac{\varepsilon}{1+\gamma} b_v, \quad \forall v \in L_1(u).$$

Clearly, if $\ell_v = 0$ then $\hat{\ell}_v = 0$ and the inequality holds; thus we focus on the case $\ell_v > 0$. From Lemma 2 we know that $\mathbb{E}\left[\hat{\ell}_v\right] \geq \ell_v - \frac{2(b_v+\gamma+1)}{k}$. By setting $k \geq \frac{2(\gamma+1)}{\varepsilon}$ we have

$$\ell_v - \frac{2(b_v + \gamma + 1)}{k} \geq \ell_v - \frac{\varepsilon}{\gamma+1} b_v \geq (1-\varepsilon)\ell_v - \frac{\varepsilon}{\gamma+1} b_v,$$

therefore proving property (i).

Finally, we prove (ii). Notice that we build $\widetilde{G}$ by removing *only* red edges from G. Since $\varphi = 1$, the degrees of the vertices $v \in L_1(u)$ in $\widetilde{G}$ are *at most halved* compared to G. As a consequence, the probability that v selects u at Line 14 of Algorithm 3 in G is at most half of the probability we have in $\widetilde{G}$. Therefore, doubling the value of k to $\frac{4(\gamma+1)}{\varepsilon}$ ensures that the analysis we conducted on $\widetilde{G}$ also holds on G, guaranteeing a $(1-\varepsilon)$-covering of $B_2(u)$ in expectation on G as well.

5 EXPERIMENTAL ANALYSIS

The purpose of this experimental analysis, conducted on real datasets, is the validation of our lazy approach along three main axes: i) the qualitative consistency between the theoretical findings from Sections 2 to 4 and the actual behavior of our lazy approach on real, incremental datasets; ii) the accuracy of our approach on two key neighborhood-based queries, namely, size and Jaccard similarity; iii) its computational savings with respect to non-lazy baselines on medium and large incremental graphs.

5.1 Experimental setup

Platform. Our experiments were performed on a machine with 2.3 GHz Intel Xeon Gold 5118 CPU with 24 cores, 192 GB of RAM, cache L1 32KB, shared L3 of 16MB and UMA architecture. The whole code is written in C++, compiled with GCC 10 and with the following compilation flags: -DARCH_X86_64 -Wall -Wextra -g -pg -O3 -lm.

Algorithms. We compared Lazy-Alg(φ, k) with various combinations (φ, k) with the following baselines:
i) the exact *algorithm* 1, which is not scalable and is only used in the first round of experiments on smaller datasets, in order to isolate the error introduced by lazy updates;
ii) the naive *sketch-based baseline*, which adopts sketch-based representations of 1- and 2-balls, but performs all necessary updates. It corresponds to Algorithm 3 with $\varphi = 0$ and $k = 0$, but where 1- and/or 2-ball unions correspond to merging the corresponding sketches.[8]

With the exception of the first set of experiments (i.e. item (i) above), we compared our algorithm to the sketch-based baseline. For some results, we needed to compute the true value of the parameter of interest for 2-balls (e.g., size) by executing a suitably optimized BFS. For our Lazy-Alg, we used combinations of the following values: $\varphi = 0.1, 0.25, 0.5, 0.75, 1$ and $k = 0, 2, 4, 8$.

Implementation details. To best assess the performance of algorithms, it would be ideal to minimize the overhead deriving from the management of edge insertions in the graph. We observe that this overhead is the same for all algorithms we tested. Hence, we represented graphs using the *compressed sparse row* format [21] and, since we knew the edge insertion sequence in advance, we pre-allocated the memory needed to accommodate them, so as to minimize overhead. For experiments that required hash functions, we used *tabulation hashing* [37].

Datasets. We considered real incremental graphs of different sizes, both directed and undirected, available from NetworkRepository [39]. In our time analysis, we also extracted a large incremental dataset from a large static graph, namely soc-friendster [46], available from SNAP [32]. Following previous work on dynamic graphs [16, 28], we generated an incremental graph by adding edges sequentially and in random order, starting from an empty graph. The main features of our datasets are summarized in Table 1.

5.2 Results

As we remarked in the introduction, we focused on the case of undirected graphs for ease of exposition and for the sake of space, but our approach extends seamlessly to directed graphs, such as some of the real examples we consider in this section. In such cases, the only caveat to keep in mind is that we define the h-ball of a vertex u as the subset of vertices that are reachable from u over a directed path traversing at most h edges.[9]

Impact of lazy updates. The goal of our first experiment is twofold: i) assessing the impact of lazy updates on the estimation of 2-balls; ii) assessing the degree of consistency between the theoretical findings of Sections 3 and 4 and the actual behavior of our algorithms on real datasets. In order to isolate the specific contribution of lazy updates in the estimation error, we implemented (true and

---

[8]Again, the specific sketch (or sketches) used depends on the neighborhood queries one wants to support.
[9]One could alternatively define the h-ball of u as the subset of vertices from which it is possible to reach u traversing at most h directed edges. We stick to the former definition, which is more frequent in social network analysis.

Dataset	$|V|$	$|E|$	Directed	Dynamic
comm-linux-kernel-reply	27,927	242,976	✔	✔
fb-wosn-friends [42]	63,731	817,090	✘	✔
ia-enron-email-all	87,273	321,918	✔	✔
soc-flickr-growth	2,302,925	33,140,017	✔	✔
soc-youtube-growth	3,223,589	9,376,594	✔	✔
soc-friendster [46]	65,608,366	1,806,067,135	✘	✘

Table 1: Summary table of real networks used in the experiments.

approximate) 1-balls and 2-balls losslessly, as dictionaries. This way, the error in 2-ball size estimation is only determined by our lazy update policy. Since, as we argued elsewhere in the paper, lossless representations of 2-balls quickly becomes unfeasible for larger datasets, this first experiment was run on 3 small-medium datasets, namely, `comm-linux-kernel-reply`, `fb-wosn-friends` and `ia-enron-email-all` (results for `comm-linux-kernel-reply` are reported in Figure 8c in Appendix B).

For each of the above graphs, we selected as a sample 5000 vertices whose 2-balls are the largest at the end of the edge insertion sequence. For every pair (φ, k) of parameter values for LAZY-ALG(φ, k), we performed 10 independent runs. Each run is organized into the following steps: 1) the initial graph $G^{(0)}(V, E^{(0)})$ corresponds to the first 20% edge insertions; 2) we measure the coverage (see Definition 3.1) of each of the 5000 2-balls above by LAZY-ALG(φ, k) at each of the 100 equally spaced timestamps, the same in each run. For the generic timestamp t, we measure the average coverage

$$C_t = \frac{1}{5000} \sum_{i=1}^{5000} \frac{\hat{B}_i}{B_i},$$

where B_i and $\hat{B}_i$ respectively denote the true and estimated sizes of the i-th 2-ball from the sample. Finally, for each timestamp t, we plot the average of the 10 values of C_t computed in every run. The results, summarized in Figure 5, are fully consistent with our theoretical findings from Section 3. At least for the diverse dataset sample considered here, uniform random permutations are a reasonable theoretical proxy of real sequences. More in general, real sequences seem to be rather far from the pathological worst-cases analyzed in Section 4.1, so that the actual behavior of our algorithm is not only in line, but better than our analysis predicts. We also have an initial insight into the effects of the parameters φ and k, which will be examined more thoroughly in the subsequent subsections.

Ball size estimation via sketches. The goal of the next round of experiments was assessing the accuracy of our algorithms in 2-ball size estimation, in the realistic setting in which approximate 1- and 2-balls are represented via state-of-art sketches that are based on probabilistic counters [5]. In this case, we have a compound estimation error, arising from both our lazy update policy and the use of probabilistic counters. We ran experiments on the three small-medium sized datasets considered previously, plus two large ones, namely, `soc-youtube-growth` and `soc-flickr-growth`. Again, we considered the top-5000 largest 2-balls as sample. The experiments were executed as in the previous case, with the following differences: i) we considered 3 timestamps, respectively corresponding to 50%,

(a) fb-wosn-friends

(b) ia-enron-email-all

Figure 5: Average coverage C_t from a network with 20% of its edges to the end of the insertion sequence. Dashed lines show the theoretical expected coverage for random sequences (see Theorem 1).

75% and 100% of all edge insertions of each dataset; ii) in this case, 1- and 2-balls are not explicitly represented as sets (not even by the baseline). As a result, ball sizes might be overestimated and coverage has no clear meaning. We therefore use Mean Absolute Percentage Error (MAPE) to measure accuracy, defined as follows:

$$\text{MAPE} = \frac{1}{5000} \sum_{i=1}^{5000} \frac{|B_i - \hat{B}_i|}{B_i},$$

where i refers to the 2-ball size of the i-th sampled vertex. For each dataset, this index is computed for each of 10 independent runs at each of the 3 timestamps we consider. Results are summarized for all datasets and combinations (φ, k) we consider in Table 2 for the last timestamp (100% of edge insertions). Results for other timestamps (50% and 75%) are similar and are omitted for the sake of space (see Appendix B for complete results). The main takeaways here are that i) the additional error introduced by the use of sketches (which can be controlled by varying the size of the sketch) is relatively modest; ii) even relatively large values of φ and/or small values of k result in performances that are close to those of the baseline that uses sketches to represent balls, but naively performs all light updates. The effect of k is more pronounced when φ is large, contributing to a reduction in both error and, to a lesser extent, variance. Finally, we note that although the effect of increasing k can be achieved by decreasing φ, our analysis in Section 4 suggests that the parameter k provides robustness against worst-case scenarios. Even though the analysis of the impact of sketches on error is out of the scope of this paper, the results in the previous paragraph and the small difference with the baseline (Table 2) when using the sketches further suggest that the analysis of random sequences may also apply to real sequences.

Table 2: Mean and standard deviation of absolute percentage errors for 2-hop neighborhood size estimation. Size estimates were made using the KMV probabilistic counter [5], with size 32. Queries were made at the end of the insertion sequence.

	k	$\varphi = 0.1$	$\varphi = 0.5$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.19 ± 0.14	0.17 ± 0.11	
	2	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.11	0.12 ± 0.10
	4	0.14 ± 0.09	0.17 ± 0.12	0.14 ± 0.10	
	8	0.14 ± 0.11	0.14 ± 0.09	0.13 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.11	0.21 ± 0.12	
	2	0.13 ± 0.09	0.15 ± 0.10	0.17 ± 0.11	0.14 ± 0.11
	4	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.11	
	8	0.16 ± 0.13	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.16 ± 0.11	0.20 ± 0.14	
	2	0.14 ± 0.11	0.16 ± 0.12	0.16 ± 0.12	0.13 ± 0.11
	4	0.13 ± 0.12	0.15 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.11	0.14 ± 0.10	0.16 ± 0.13	
flickr	0	0.17 ± 0.14	0.18 ± 0.12	0.17 ± 0.11	
	2	0.16 ± 0.12	0.12 ± 0.09	0.14 ± 0.10	0.17 ± 0.14
	4	0.14 ± 0.09	0.13 ± 0.10	0.16 ± 0.10	
	8	0.14 ± 0.11	0.13 ± 0.10	0.14 ± 0.09	
youtube	0	0.15 ± 0.11	0.15 ± 0.10	0.24 ± 0.11	
	2	0.16 ± 0.13	0.14 ± 0.10	0.19 ± 0.11	0.14 ± 0.11
	4	0.13 ± 0.11	0.13 ± 0.10	0.15 ± 0.11	
	8	0.13 ± 0.10	0.12 ± 0.09	0.15 ± 0.11	

Accuracy in Jaccard similarity estimation. With the same goal as in the previous experiment, we now evaluate the quality of our lazy approach policy combined with the use of sketches in another graph mining task: the Jaccard similarity estimation for 2-hop neighborhoods. The sketch used to represent the 1- and 2-balls is the well-known h-minhash signature [9], with $h = 100$ hash functions.

As before, 10 independent runs were performed for each combination of parameters φ, k, and the baseline, on each of the previously used datasets. Errors were measured at 3 different timestamps, corresponding to 50%, 75%, and 100% of the edge insertion sequence.

Considering all vertex pairs in the entire graph would be computationally prohibitive, as their number is too large. Moreover, many of these pairs would have an extremely low Jaccard similarity, making it difficult to estimate them with a reasonably small error [10, 11]. Therefore, to better evaluate our algorithm's and baseline's quality, we need vertex pairs with a sufficiently high similarity in their 2-hop neighborhoods. To address this, we adopted a similar sampling process as before: we selected the 5000 vertices with the largest 2-hop neighborhoods at the end of the edge insertion sequence and randomly chose 1000 pairs whose similarity is at least 0.2. On that sample, we evaluated the MAPE of the Jaccard similarity estimate computed using the h-minhash signatures. Table 3 reports the results at the end of the insertion sequence, while results for the other timestamps (50% and 75%) are similar and are omitted for sake of space (see Appendix B for complete results).

This experiment further confirms the observations from the previous one: when using sketches to represent the 2-balls, the errors obtained with our lazy update policy (with appropriate choices of parameters φ, k) are similar and fully comparable to those of the baseline, which performs all necessary updates.

Run time analysis. Finally, we measured the running times of our algorithm and the corresponding speed-up with respect to the baseline. In the following, we report and discuss the results for the task of size estimation, using probabilistic counters. The results for the task of Jaccard similarity estimation are analogous, and thus reported in Table 12 in Appendix B.

Table 4 reports the average speed-up of our algorithm with respect to the naive baseline for the same combinations (φ, k) considered previously, for all datasets except com-friendster. Speed-ups are computed in terms of *total update time*, i.e., the total time it takes to process the whole insertion sequence. In Table 5 reports the overall processing times on com-friendster, for a subset of the combinations of φ and k. It should be noted that the baseline did not complete within a reasonable amount of time in this case. Finally, Figure 6 illustrates the average time cost per operation for the baseline, as well as the slowest and fastest parameter settings of φ and k in our algorithm. These experiments clearly highlight that the number of lazy updates is the crucial factor affecting performance and that our approach is very effective at addressing this problem, resulting in considerable speed-ups that grow with the size of the graph. These results regarding processing times, in conjunction with previous findings, underscore the significant advantage in terms of speed at the expense of a modest and acceptable reduction in query quality which, due to the necessary use of sketches, is never totally accurate.

Table 3: Mean and standard deviation of absolute percentage errors for Jaccard similarity estimation, with 100 hash functions. Queries were made at the end of the insertion sequence.

	k	$\varphi = 0.1$	$\varphi = 0.5$	$\varphi = 1$	baseline
linux	0	0.11 ± 0.09	0.13 ± 0.09	0.11 ± 0.09	
	2	0.09 ± 0.07	0.12 ± 0.09	0.11 ± 0.09	0.10 ± 0.08
	4	0.10 ± 0.08	0.12 ± 0.09	0.10 ± 0.08	
	8	0.09 ± 0.07	0.12 ± 0.09	0.10 ± 0.08	
fb-wosn	0	0.70 ± 0.25	0.72 ± 0.24	0.74 ± 0.23	
	2	0.70 ± 0.24	0.70 ± 0.25	0.72 ± 0.24	0.70 ± 0.24
	4	0.70 ± 0.24	0.70 ± 0.24	0.72 ± 0.24	
	8	0.70 ± 0.24	0.70 ± 0.24	0.71 ± 0.25	
enron	0	0.09 ± 0.09	0.14 ± 0.12	0.15 ± 0.12	
	2	0.10 ± 0.09	0.12 ± 0.10	0.14 ± 0.12	0.09 ± 0.09
	4	0.09 ± 0.09	0.11 ± 0.10	0.13 ± 0.11	
	8	0.10 ± 0.09	0.10 ± 0.09	0.11 ± 0.10	
flickr	0	0.12 ± 0.09	0.11 ± 0.09	0.11 ± 0.09	
	2	0.11 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.11 ± 0.09
	4	0.11 ± 0.08	0.10 ± 0.08	0.10 ± 0.08	
	8	0.11 ± 0.08	0.10 ± 0.08	0.11 ± 0.08	
youtube	0	0.12 ± 0.09	0.11 ± 0.09	0.17 ± 0.13	
	2	0.11 ± 0.09	0.11 ± 0.09	0.15 ± 0.11	0.11 ± 0.09
	4	0.11 ± 0.09	0.13 ± 0.10	0.16 ± 0.11	
	8	0.11 ± 0.09	0.12 ± 0.09	0.14 ± 0.10	

Table 4: Speed up with respect to the baseline, using KMV probabilistic counters [5], with size 32.

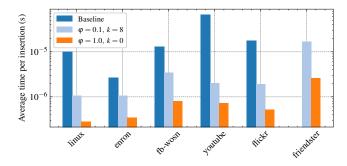|	k			φ		
		0.1	0.25	0.5	0.75	1
linux	0	14.39x	22.73x	30.27x	32.36x	35.65x
	2	12.00x	18.22x	22.03x	21.49x	23.18x
	4	11.52x	15.34x	17.63x	16.93x	17.54x
	8	9.57x	11.86x	12.80x	12.03x	12.22x
fb-wosn	0	5.48x	9.40x	11.15x	15.23x	16.22x
	2	5.22x	7.35x	7.16x	9.21x	9.67x
	4	4.32x	6.14x	6.05x	7.18x	7.23x
	8	3.79x	4.38x	4.91x	5.31x	5.22x
enron	0	4.17x	5.38x	6.64x	7.12x	7.70x
	2	3.61x	4.11x	4.68x	4.82x	5.27x
	4	3.05x	3.42x	3.78x	3.96x	4.21x
	8	2.51x	2.73x	2.85x	3.01x	3.12x
flickr	0	13.52x	19.97x	26.54x	31.51x	34.20x
	2	12.16x	16.41x	19.52x	21.93x	22.61x
	4	10.76x	13.96x	16.35x	17.17x	17.79x
	8	9.10x	11.19x	12.36x	12.99x	13.28x
youtube	0	41.61x	61.80x	77.08x	86.50x	93.73x
	2	38.16x	51.64x	60.39x	64.93x	65.51x
	4	36.08x	47.43x	52.26x	55.06x	55.90x
	8	33.33x	39.60x	42.73x	43.61x	43.82x

Figure 6: Average time per insertion operation (in seconds). The baseline time for `soc-friendster` dataset is not reported, since it exceed a time limit of 36 hrs.

Table 5: Total running time for ball size estimation using probabilistic counters, for `com-friendster` dataset. The time for the baseline algorithm is not reported since it exceed a time limit of 36 hrs.

k			φ		
	0.1	0.25	0.5	0.75	1
0	$5h\ 2'$	$2h\ 55'$	$1h\ 53'$	$1h\ 27'$	$1h\ 18'$
2	$6h\ 3'$	$3h\ 48'$	$2h\ 59'$	$2h\ 42'$	$2h\ 34'$
4	$6h\ 50'$	$4h\ 42'$	$3h\ 56'$	$3h\ 39'$	$3h\ 32'$
8	$8h\ 23'$	$6h\ 53'$	$5h\ 54'$	$5h\ 30'$	$5h\ 19'$

6 DISCUSSION AND OUTLOOK

In this work, we showed that relatively simple, lazy update algorithms can play a key role to efficiently support queries over multi-hop vertex neighborhoods on large, incremental graphs. At the same time, this work leaves a number of open questions that might deserve further investigation. A first, obvious direction is extending our approach to handle edge deletions. While our approach is potentially useful in the general case, the main problem here is handling deletions when compact, sketch-based data structures

are used to represent 1- and 2-balls. Some recent contributions address similar issues in dynamic data streams [13, 17], but extending our analyses to this general case does not seem straightforward. Another interesting direction is investigating strategies to handle queries over h-balls when $h > 2$, for example maintaining their sizes under dynamic updates. In this case, each edge addition/deletion potentially has cascading effects over h-hops. Optimizing (amortized) update costs in this general setting does not seem trivial and we conjecture that a dependence on h might be necessary. Finally, we remark that our algorithms are inherently local, i.e., whether or not to perform updates involving any vertex v only depends on v's immediate neighborhood. As a consequence, a potentially interesting avenue for further research is to investigate distributed variants

of our approach, possibly with massive parallel architectures in mind [31].

REFERENCES

[1] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):1–28, 2013.

[2] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):1–36, 2014.

[3] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th annual ACM Web science conference*, pages 33–42, 2012.

[4] Sachin U. Balvir, Mukesh M. Raghuwanshi, and Purushottam D. Shobhane. An overview of similarity-based methods in predicting social network links: A comparative analysis. *IEEE Access*, 12:120913–120934, 2024.

[5] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, RANDOM '02, page 1–10, Berlin, Heidelberg, 2002. Springer-Verlag.

[6] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–24, 2008.

[7] Luca Becchetti, Carlos Castillo, Debora Donato, Ricardo Baeza-Yates, and Stefano Leonardi. Link analysis for web spam detection. *ACM Transactions on the Web (TWEB)*, 2(1):1–42, 2008.

[8] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. Hyperanf: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th international conference on World Wide Web*, pages 625–634, 2011.

[9] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[10] Andrei Z Broder. Identifying and filtering near-duplicate documents. In *Annual symposium on combinatorial pattern matching*, pages 1–10. Springer, 2000.

[11] Andrei Z Broder and Michael Mitzenmacher. Completeness and robustness properties of min-wise independent permutations. *Random Structures & Algorithms*, 18(1):18–30, 2001.

[12] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 253–262, 2006.

[13] Marc Bury, Chris Schwiegelshohn, and Mara Sorella. Similarity search for dynamic data streams. *IEEE Transactions on Knowledge and Data Engineering*, 32:2241–2253, 2020.

[14] Andrea Cavallo, Grohnfeldt Claas, Russo Michele, Lovisotto Giulio, Luca Vassio, et al. 2-hop neighbor class similarity (2ncs): A graph structural metric indicative of graph neural network performance. In *3rd Workshop on Graphs and more Complex structures for Learning and Reasoning (GCLR) at AAAI 2023*. Arxiv, 2023.

[15] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 641–650, 2008.

[16] Qing Chen, Oded Lachish, Sven Helmer, and Michael H. Böhlen. Dynamic spanning trees for connectivity queries on fully-dynamic undirected graphs. *Proc. VLDB Endow.*, 15(11):3263–3276, July 2022.

[17] Andrea Clementi, Luciano Gualà, Luca Pepè Sciarria, and Alessandro Straziota. Maintaining k-minhash signatures over fully-dynamic data streams with recovery. *CoRR*, abs/2407.21614, 2024 (to appear on ACM WSDM'25).

[18] Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 225–234, 2007.

[19] Reinhard Diestel. *Graph theory*. Springer (print edition); Reinhard Diestel (eBooks), 2024.

[20] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[21] Stanley C. Eisenstat, M. C. Gursky, Martin H. Schultz, and Andrew H. Sherman. Yale sparse matrix package i: The symmetric codes. *International Journal for Numerical Methods in Engineering*, 18:1145–1151, 1982.

[22] Paul Erdös. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3:113–116, 1965.

[23] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981.

[24] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[25] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[26] Phillip B Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 281–291, 2001.

[27] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, 2022.

[28] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms – a quick reference guide. *ACM J. Exp. Algorithmics*, 27, December 2022.

[29] Peng Jia, Pinghui Wang, Jing Tao, and Xiaohong Guan. A fast sketch method for mining user similarities over fully dynamic graph streams. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1682–1685, 2019.

[30] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 734–751. SIAM, 2014.

[31] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *AcM sIGMoD record*, 40(4):11–20, 2012.

[32] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[33] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

[34] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

[35] Morteza Monemizadeh, S Muthukrishnan, Pan Peng, and Christian Sohler. Testable bounded degree graph properties are random order streamable. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017.

[36] Pan Peng and Christian Sohler. Estimating graph parameters from random order streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2449–2466. SIAM, 2018.

[37] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3), June 2012.

[38] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.

[39] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[40] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[41] Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985.

[42] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, pages 37–42, 2009.

[43] Xu-Wen Wang, Lorenzo Madeddu, Kerstin Spirohn, Leonardo Martini, Adriano Fazzone, Luca Becchetti, Thomas P Wytock, István A Kovács, Olivér M Balogh, Bettina Benczik, et al. Assessment of community efforts to advance network-based prediction of protein–protein interactions. *Nature communications*, 14(1):1582, 2023.

[44] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Xiaojie Guo. Graph neural networks: foundation, frontiers and applications. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4840–4841, 2022.

[45] Qingjun Xiao, Shiwei Yang, Panpan Li, Kangying Li, and Lin Wen. Multi-resolution odd sketch for mining extended jaccard similarity of dynamic streaming sets. *IEEE Transactions on Network Science and Engineering*, pages 1–15, 2023.

[46] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA, 2012. Association for Computing Machinery.

[47] Ahmad Zareie and Rizos Sakellariou. Similarity-based link prediction in social networks using latent relationships between the users. *Scientific reports*, 10(1):20137, 2020.

[48] Fangyuan Zhang and Sibo Wang. Effective indexing for dynamic structural graph clustering. *Proceedings of the VLDB Endowment*, 15(11):2908–2920, 2022.

[49] Tao Zhou, Yan-Li Lee, and Guannan Wang. Experimental analyses on 2-hop-based and 3-hop-based link prediction algorithms. *Physica A: Statistical Mechanics and its Applications*, 564:125532, 2021.

A PROOFS FROM SECTION 4

A.1 On the girth of locally γ-sparse graphs

LEMMA 4. *Let $G = (V, E)$ be an undirected graph with girth $g(G)$. Then G is locally 0-sparse if and only if $g(G) \geq 5$.*

PROOF. We first prove that if G is locally 0-sparse then $g(G)$ must be at least 5. In order to prove that, we simply negate the statement and prove that if G has girth < 5 then G can not be locally 0-sparse. Without loss of generality, assume that $g(G) = 4$ (the case $g(G) = 3$ is similar). Then there must exist a cycle $C = (u_1, u_2, u_3, u_4)$ of 4 vertices. It is simple to see that $u_2, u_4 \in L_1(u_1)$ and $u_3 \in L_2(u_1)$. Since u_3 is a neighbor of both u_2 and u_4, the degree of u_3 in the subgraph $G[L_1(u_1) \cup \{u_4\}]$ is at least 2, hence G is not locally 0-sparse (see Figure 7a).

We now prove that if $g(G) \geq 5$ then G must be locally 0-sparse. Again, we negate this statement and prove that if G is not locally 0-sparse then the girth of G must be less then 5. Let us assume that G is locally γ-sparse, for any $\gamma > 0$, thus it is not locally 0-sparse. Since G is not locally 0-sparse there exists a vertex $v \in V$ such that at least one of the following properties holds (see Figure 7b):

(1) $\exists u \in L_1(v)$ such that the degree of u in $G[L_1(v)]$ is greater then 0, or;

(2) $\exists w \in L_2(v)$ such that the degree of w in $G[L_1(v) \cup \{w\}]$ is greater then 1.

In the first case, we have a cycle of 3 vertices, then $g(G) = 3$. In the second case, we have a cycle of 4 vertices, then $g(G) = 4$. In both cases $g(G) < 5$. □
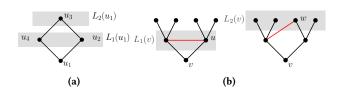

(a) **(b)**

Figure 7

A.2 Deterministic lazy-update on γ-sparse graphs

THEOREM 5. *Let $\varepsilon \in (0, 1)$, and let $G^{(0)}$ be an initial graph. Consider any sequence of edge insertions that yields a final graph G. If G is locally γ-sparse, LAZY-ALG($\varphi = \frac{\varepsilon}{1-\varepsilon}, k = 0$) has an approximation ratio of $\frac{\gamma+1}{1-\varepsilon}$ and amortized update cost $O(1/\varepsilon)$.*

PROOF. Recall that Δ_u denotes the black degree of u, and that Algorithm 3 guarantees that $\deg_u$ is at most $(1 + \varphi)\Delta_u$. Then, it is simple to give an upper bound to the size of $B_2(u)$, that is $|B_2(u)| \leq 1 + \sum_{v \in L_1(u)} (1+\varphi)\Delta_v$. Consider a vertex $v \in L_1(u)$. Since G is locally γ-sparse, the number of neighbors of v belonging to $L_2(u)$ is at lest $\deg_v - (\gamma + 1)$ of which $\Delta_v - (\gamma + 1)$ must belong to $\hat{B}_2(u)$. Moreover,

a vertex in $L_2(u)$ has at most $\gamma + 1$ neighbors in $L_1(u)$. Therefore:

$$
|\hat{B}_2(u)| \geq \Delta_u + 1 + \frac{1}{\gamma + 1} \sum_{v \in L_1(u)} (\Delta_v - (\gamma + 1))
$$

$$
= \Delta_u + 1 + \frac{1}{\gamma + 1} \sum_{v \in L_1(u)} \Delta_v - \underbrace{\frac{1}{\gamma + 1} \sum_{v \in L_1(u)} (\gamma + 1)}_{= \Delta_u}
$$

$$
= 1 + \frac{1}{\gamma + 1} \sum_{v \in L_1(u)} \Delta_v.
$$

As a consequence, $|\hat{B}_2(u)|/|B_2(u)| \geq \frac{1}{(1+\varphi)(\gamma+1)}$. By setting $\varphi = \frac{\varepsilon}{1-\varepsilon}$, and by using Lemma 1, the claim follows. □

A.3 Proof of Lemma 2

PROOF. Let $e_1, \ldots, e_{\ell_v}$ be the *red edges* between v and $L_2(u)$, and define the binary random variable $\hat{\ell}_v(i)$ that is equal to 1 if e_i is a *quasi-black edge* for u, 0 otherwise, for $i = 1, \ldots, \ell_v$. Thus we can express $\hat{\ell}_v = \sum_{i=1}^{\ell_v} \hat{\ell}_v(i)$, with expectation

$$
\mathbb{E}\left[\hat{\ell}_v\right] = \sum_{i=1}^{\ell_v} \mathbb{P}\left(\hat{\ell}_v(i) = 1\right) = \ell_v - \sum_{i=1}^{\ell_v} \mathbb{P}\left(\hat{\ell}_v(i) = 0\right). \tag{6}
$$

Without loss of generality, assume that the edges $e_1, \ldots, e_{\ell_v}$ have been inserted at times $t_1 < \cdots < t_{\ell_v}$, respectively. If e_i is not a quasi-black edge for u, then it must be that u is not selected by v at Line 14 of Algorithm 3, at times $t_i, t_{i+1}, \ldots, t_{\ell_v}$. This holds with probability

$$
\mathbb{P}\left(\hat{\ell}_v(i) = 0\right) \leq \prod_{j=i}^{\ell_v} \left(1 - \frac{k}{\deg_v^{(t_j)}}\right) \leq \prod_{j=i}^{\ell_v} \left(1 - \frac{k}{\deg_v^{(t_{\ell_v})}}\right)
$$

$$
\leq \left(1 - \frac{k}{b_v + \ell_v + \gamma + 1}\right)^{\ell_v - i + 1} \leq \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)^{\ell_v - i}. \tag{7}
$$

The third inequality holds since the edges incident to v having endpoints in $L_1(u)$ are at most γ, while those having endpoints in $L_2(u)$ are exactly $b_v + \ell_v$. Moreover, the last inequality holds because $\ell_v \leq \delta_v \leq \Delta_v \leq b_v + \gamma + 1$, given the assumption $\varphi = 1$.

By plugging in (7) into (6) and we obtain

$$
\mathbb{E}\left[\hat{\ell}_v\right] \geq \ell_v - \sum_{i=1}^{\ell_v} \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)^{\ell_v - i}
$$

$$
= \ell_v - \sum_{i=0}^{\ell_v - 1} \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)^i \leq \ell_v - \frac{1 - \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)^{\ell_v}}{1 - \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)}
$$

$$
\geq \ell_v - \frac{1}{1 - \left(1 - \frac{k}{2(b_v + \gamma + 1)}\right)} \geq \ell_v - \frac{2(b_v + \gamma + 1)}{k}.
$$

□

B FURTHER EXPERIMENTAL RESULTS

In this section, we present the comprehensive set of results, providing an exhaustive overview and offering a detailed picture that enriches the information already illustrated in Section 5.

Table 6: Mean and standard deviation of absolute percentage errors for 2-hop neighborhood size estimation. Size estimates were made using the KMV counter [5], with size 32. Each result is done on 10 independent runs, over a sample of the first 5000 vertices with the largest 2-hop neighborhood, after the insertion of 50% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.13 ± 0.09	0.16 ± 0.11	0.21 ± 0.13	0.23 ± 0.13	
	2	0.13 ± 0.09	0.15 ± 0.11	0.14 ± 0.10	0.17 ± 0.11	0.20 ± 0.12	0.12 ± 0.10
	4	0.13 ± 0.09	0.15 ± 0.11	0.15 ± 0.10	0.15 ± 0.11	0.17 ± 0.11	
	8	0.14 ± 0.11	0.13 ± 0.10	0.12 ± 0.09	0.20 ± 0.13	0.17 ± 0.11	
fb-wosn	0	0.15 ± 0.12	0.16 ± 0.10	0.19 ± 0.11	0.21 ± 0.11	0.27 ± 0.13	
	2	0.14 ± 0.10	0.14 ± 0.10	0.16 ± 0.11	0.20 ± 0.12	0.20 ± 0.11	0.13 ± 0.11
	4	0.14 ± 0.11	0.14 ± 0.10	0.15 ± 0.10	0.17 ± 0.12	0.17 ± 0.11	
	8	0.14 ± 0.12	0.14 ± 0.11	0.15 ± 0.11	0.15 ± 0.11	0.15 ± 0.10	
enron	0	0.13 ± 0.10	0.16 ± 0.11	0.15 ± 0.10	0.18 ± 0.12	0.19 ± 0.13	
	2	0.14 ± 0.11	0.14 ± 0.11	0.16 ± 0.12	0.15 ± 0.11	0.16 ± 0.11	0.14 ± 0.12
	4	0.14 ± 0.12	0.14 ± 0.11	0.15 ± 0.10	0.15 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.10	0.13 ± 0.11	0.14 ± 0.10	0.15 ± 0.13	0.16 ± 0.13	
flickr	0	0.19 ± 0.15	0.16 ± 0.12	0.18 ± 0.12	0.16 ± 0.12	0.17 ± 0.12	
	2	0.15 ± 0.11	0.15 ± 0.11	0.12 ± 0.10	0.16 ± 0.13	0.12 ± 0.09	0.14 ± 0.12
	4	0.13 ± 0.09	0.15 ± 0.12	0.14 ± 0.11	0.15 ± 0.11	0.15 ± 0.11	
	8	0.13 ± 0.11	0.19 ± 0.15	0.14 ± 0.12	0.13 ± 0.11	0.13 ± 0.09	
youtube	0	0.14 ± 0.11	0.16 ± 0.12	0.17 ± 0.11	0.22 ± 0.12	0.24 ± 0.13	
	2	0.14 ± 0.10	0.15 ± 0.11	0.15 ± 0.11	0.18 ± 0.12	0.18 ± 0.11	0.15 ± 0.12
	4	0.15 ± 0.12	0.15 ± 0.11	0.15 ± 0.11	0.16 ± 0.11	0.17 ± 0.12	
	8	0.15 ± 0.13	0.14 ± 0.12	0.14 ± 0.11	0.16 ± 0.11	0.16 ± 0.11	

Table 7: Mean and standard deviation of absolute percentage errors for 2-hop neighborhood size estimation. Size estimates were made using the KMV counter [5], with size 32. Each result is done on 10 independent runs, over a sample of the first 5000 vertices with the largest 2-hop neighborhood, after the insertion of 75% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.13 ± 0.11	0.14 ± 0.10	0.19 ± 0.15	0.18 ± 0.12	0.18 ± 0.12	
	2	0.14 ± 0.11	0.15 ± 0.10	0.14 ± 0.10	0.16 ± 0.11	0.17 ± 0.11	0.11 ± 0.09
	4	0.14 ± 0.09	0.16 ± 0.11	0.16 ± 0.12	0.14 ± 0.11	0.15 ± 0.10	
	8	0.15 ± 0.13	0.14 ± 0.10	0.13 ± 0.09	0.19 ± 0.15	0.14 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.10	0.16 ± 0.11	0.17 ± 0.11	0.22 ± 0.12	
	2	0.13 ± 0.10	0.14 ± 0.10	0.15 ± 0.10	0.17 ± 0.11	0.18 ± 0.11	0.14 ± 0.11
	4	0.14 ± 0.11	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.13	0.16 ± 0.11	
	8	0.15 ± 0.13	0.13 ± 0.11	0.14 ± 0.11	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.16 ± 0.12	0.15 ± 0.11	0.18 ± 0.12	0.20 ± 0.13	
	2	0.14 ± 0.11	0.14 ± 0.10	0.17 ± 0.12	0.15 ± 0.11	0.16 ± 0.12	0.13 ± 0.11
	4	0.14 ± 0.12	0.14 ± 0.10	0.15 ± 0.11	0.16 ± 0.12	0.16 ± 0.12	
	8	0.13 ± 0.10	0.15 ± 0.11	0.14 ± 0.10	0.14 ± 0.11	0.16 ± 0.13	
flickr	0	0.18 ± 0.15	0.16 ± 0.11	0.19 ± 0.12	0.16 ± 0.11	0.18 ± 0.11	
	2	0.14 ± 0.11	0.16 ± 0.12	0.12 ± 0.09	0.16 ± 0.11	0.14 ± 0.09	0.17 ± 0.14
	4	0.14 ± 0.09	0.17 ± 0.13	0.13 ± 0.10	0.15 ± 0.11	0.16 ± 0.10	
	8	0.13 ± 0.10	0.17 ± 0.13	0.13 ± 0.10	0.13 ± 0.10	0.14 ± 0.10	
youtube	0	0.14 ± 0.11	0.15 ± 0.10	0.19 ± 0.11	0.21 ± 0.11	0.24 ± 0.12	
	2	0.15 ± 0.13	0.14 ± 0.09	0.15 ± 0.11	0.15 ± 0.11	0.17 ± 0.10	0.13 ± 0.10
	4	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.12	0.14 ± 0.09	0.16 ± 0.11	
	8	0.15 ± 0.12	0.14 ± 0.11	0.13 ± 0.09	0.14 ± 0.10	0.16 ± 0.10	

Table 8: Mean and standard deviation of absolute percentage errors for 2-hop neighborhood size estimation. Size estimates were made using the KMV probabilistic counter [5], with size 32. Each result is done on 10 independent runs, over a sample of the first 5000 vertices with the largest 2-hop neighborhood, after the insertion of 100% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.14 ± 0.12	0.14 ± 0.10	0.19 ± 0.14	0.16 ± 0.11	0.17 ± 0.11	
	2	0.13 ± 0.11	0.16 ± 0.10	0.14 ± 0.10	0.15 ± 0.11	0.16 ± 0.11	0.12 ± 0.10
	4	0.14 ± 0.09	0.16 ± 0.11	0.17 ± 0.12	0.15 ± 0.14	0.14 ± 0.10	
	8	0.14 ± 0.11	0.13 ± 0.10	0.14 ± 0.09	0.21 ± 0.16	0.13 ± 0.10	
fb-wosn	0	0.16 ± 0.12	0.15 ± 0.10	0.15 ± 0.11	0.17 ± 0.11	0.21 ± 0.12	
	2	0.13 ± 0.09	0.14 ± 0.10	0.15 ± 0.10	0.17 ± 0.11	0.17 ± 0.11	0.14 ± 0.11
	4	0.14 ± 0.11	0.14 ± 0.11	0.15 ± 0.10	0.16 ± 0.12	0.16 ± 0.11	
	8	0.16 ± 0.13	0.13 ± 0.11	0.14 ± 0.10	0.14 ± 0.10	0.15 ± 0.11	
enron	0	0.13 ± 0.10	0.15 ± 0.12	0.16 ± 0.11	0.18 ± 0.12	0.20 ± 0.14	
	2	0.14 ± 0.11	0.14 ± 0.10	0.16 ± 0.12	0.15 ± 0.10	0.16 ± 0.12	0.13 ± 0.11
	4	0.13 ± 0.12	0.14 ± 0.10	0.15 ± 0.12	0.16 ± 0.12	0.15 ± 0.12	
	8	0.13 ± 0.11	0.15 ± 0.11	0.14 ± 0.10	0.14 ± 0.11	0.16 ± 0.13	
flickr	0	0.17 ± 0.14	0.15 ± 0.11	0.18 ± 0.12	0.17 ± 0.11	0.17 ± 0.11	
	2	0.16 ± 0.12	0.17 ± 0.12	0.12 ± 0.09	0.16 ± 0.11	0.14 ± 0.10	0.17 ± 0.14
	4	0.14 ± 0.09	0.16 ± 0.13	0.13 ± 0.10	0.14 ± 0.11	0.16 ± 0.10	
	8	0.14 ± 0.11	0.16 ± 0.12	0.13 ± 0.10	0.13 ± 0.10	0.14 ± 0.09	
youtube	0	0.15 ± 0.11	0.13 ± 0.10	0.15 ± 0.10	0.23 ± 0.13	0.24 ± 0.11	
	2	0.16 ± 0.13	0.14 ± 0.10	0.14 ± 0.10	0.15 ± 0.11	0.19 ± 0.11	0.14 ± 0.11
	4	0.13 ± 0.11	0.16 ± 0.10	0.13 ± 0.10	0.15 ± 0.09	0.15 ± 0.11	
	8	0.13 ± 0.10	0.15 ± 0.11	0.12 ± 0.09	0.14 ± 0.10	0.15 ± 0.11	

Table 9: Mean and standard deviation of absolute percentage errors for Jaccard similarity estimation, with 100 hash functions. Queries were made after inserting the first 50% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.11 ± 0.09	0.11 ± 0.09	0.09 ± 0.08	0.12 ± 0.09	0.14 ± 0.10	
	2	0.09 ± 0.07	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.15 ± 0.11	0.08 ± 0.07
	4	0.09 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.11 ± 0.09	
	8	0.09 ± 0.07	0.11 ± 0.08	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	
fb-wosn	0	0.80 ± 0.25	0.80 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.81 ± 0.24	
	2	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.80 ± 0.24
	4	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	0.80 ± 0.24	
	8	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	
enron	0	0.09 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.14 ± 0.11	
	2	0.09 ± 0.09	0.11 ± 0.10	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.09 ± 0.09
	4	0.09 ± 0.08	0.10 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.12 ± 0.10	
	8	0.09 ± 0.09	0.10 ± 0.09	0.10 ± 0.09	0.11 ± 0.09	0.10 ± 0.09	
flickr	0	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.12 ± 0.10	
	2	0.10 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	0.11 ± 0.09
	4	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.08	
	8	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	
youtube	0	0.10 ± 0.09	0.10 ± 0.09	0.10 ± 0.08	0.11 ± 0.09	0.13 ± 0.11	
	2	0.11 ± 0.09	0.11 ± 0.10	0.11 ± 0.09	0.16 ± 0.12	0.12 ± 0.09	0.12 ± 0.10
	4	0.10 ± 0.08	0.10 ± 0.08	0.13 ± 0.11	0.18 ± 0.13	0.13 ± 0.09	
	8	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.15 ± 0.11	0.16 ± 0.11	

Table 10: Mean and standard deviation of absolute percentage errors for Jaccard similarity estimation, with 100 hash functions. Queries were made after inserting the first 75% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.11 ± 0.09	0.11 ± 0.09	0.09 ± 0.08	0.12 ± 0.09	0.14 ± 0.10	
	2	0.09 ± 0.07	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.15 ± 0.11	0.08 ± 0.07
	4	0.09 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.11 ± 0.09	
	8	0.09 ± 0.07	0.11 ± 0.08	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	
fb-wosn	0	0.80 ± 0.25	0.80 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.81 ± 0.24	
	2	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.80 ± 0.24
	4	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	0.80 ± 0.24	
	8	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	
enron	0	0.09 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.14 ± 0.11	
	2	0.09 ± 0.09	0.11 ± 0.10	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.09 ± 0.09
	4	0.09 ± 0.08	0.10 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.12 ± 0.10	
	8	0.09 ± 0.09	0.10 ± 0.09	0.10 ± 0.09	0.11 ± 0.09	0.10 ± 0.09	
flickr	0	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.12 ± 0.10	
	2	0.10 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	0.11 ± 0.09
	4	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.08	
	8	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	
youtube	0	0.10 ± 0.09	0.10 ± 0.09	0.10 ± 0.08	0.11 ± 0.09	0.13 ± 0.11	
	2	0.11 ± 0.09	0.11 ± 0.10	0.11 ± 0.09	0.16 ± 0.12	0.12 ± 0.09	0.12 ± 0.10
	4	0.10 ± 0.08	0.10 ± 0.08	0.13 ± 0.11	0.18 ± 0.13	0.13 ± 0.09	
	8	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.15 ± 0.11	0.16 ± 0.11	

Table 11: Mean and standard deviation of absolute percentage errors for Jaccard similarity estimation, with 100 hash functions. Queries were made after inserting the first 100% of the edges.

	k	$\varphi = 0.1$	$\varphi = 0.25$	$\varphi = 0.5$	$\varphi = 0.75$	$\varphi = 1$	baseline
linux	0	0.11 ± 0.09	0.11 ± 0.09	0.09 ± 0.08	0.12 ± 0.09	0.14 ± 0.10	
	2	0.09 ± 0.07	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.15 ± 0.11	0.08 ± 0.07
	4	0.09 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.11 ± 0.09	
	8	0.09 ± 0.07	0.11 ± 0.08	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.09	
fb-wosn	0	0.80 ± 0.25	0.80 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.81 ± 0.24	
	2	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.81 ± 0.24	0.80 ± 0.24
	4	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	0.80 ± 0.24	
	8	0.80 ± 0.24	0.79 ± 0.25	0.79 ± 0.25	0.80 ± 0.24	0.80 ± 0.24	
enron	0	0.09 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.14 ± 0.11	
	2	0.09 ± 0.09	0.11 ± 0.10	0.11 ± 0.10	0.12 ± 0.10	0.13 ± 0.11	0.09 ± 0.09
	4	0.09 ± 0.08	0.10 ± 0.09	0.11 ± 0.10	0.12 ± 0.10	0.12 ± 0.10	
	8	0.09 ± 0.09	0.10 ± 0.09	0.10 ± 0.09	0.11 ± 0.09	0.10 ± 0.09	
flickr	0	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.11 ± 0.09	0.12 ± 0.10	
	2	0.10 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	0.11 ± 0.09
	4	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.11 ± 0.08	0.11 ± 0.08	
	8	0.11 ± 0.08	0.11 ± 0.09	0.10 ± 0.08	0.12 ± 0.09	0.11 ± 0.09	
youtube	0	0.10 ± 0.09	0.10 ± 0.09	0.10 ± 0.08	0.11 ± 0.09	0.13 ± 0.11	
	2	0.11 ± 0.09	0.11 ± 0.10	0.11 ± 0.09	0.16 ± 0.12	0.12 ± 0.09	0.12 ± 0.10
	4	0.10 ± 0.08	0.10 ± 0.08	0.13 ± 0.11	0.18 ± 0.13	0.13 ± 0.09	
	8	0.10 ± 0.08	0.10 ± 0.08	0.11 ± 0.09	0.15 ± 0.11	0.16 ± 0.11	

Table 12: Speed up with respect to the baseline, using min-hash sketches, with 100 hash functions.

	k	φ 0.1	0.25	0.5	0.75	1
linux	0	6.73x	8.01x	8.67x	8.96x	9.08x
	2	6.35x	7.36x	7.85x	8.03x	8.11x
	4	6.04x	6.90x	7.26x	7.38x	7.45x
	8	5.52x	6.13x	6.35x	6.40x	6.45x
fb-wosn	0	2.55x	3.13x	3.47x	3.63x	3.70x
	2	2.38x	2.80x	3.04x	3.13x	3.18x
	4	2.27x	2.60x	2.76x	2.81x	2.86x
	8	2.07x	2.28x	2.37x	2.38x	2.41x
enron	0	1.85x	1.99x	2.07x	2.10x	2.12x
	2	1.77x	1.88x	1.94x	1.96x	1.97x
	4	1.70x	1.80x	1.84x	1.86x	1.86x
	8	1.59x	1.65x	1.67x	1.68x	1.68x
flickr	0	12.36x	17.15x	20.52x	22.13x	23.10x
	2	11.18x	14.69x	16.81x	17.66x	18.19x
	4	10.36x	13.11x	14.63x	15.22x	15.59x
	8	9.08x	10.90x	11.76x	12.10x	12.32x
youtube	0	41.70x	53.71x	62.06x	66.81x	69.02x
	2	38.74x	47.56x	52.71x	55.24x	56.28x
	4	36.69x	43.71x	47.37x	49.06x	49.79x
	8	33.38x	38.10x	40.17x	40.92x	41.30x

(a) fb-wosn-friends

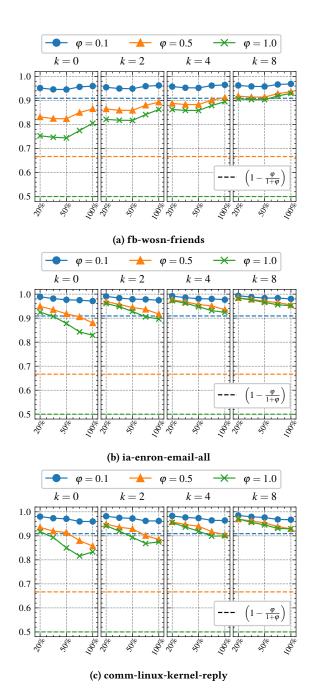(b) ia-enron-email-all

(c) comm-linux-kernel-reply

Figure 8: Average coverage C_t from a network with 20% of its edges to the end of the insertion sequence. Dashed lines show the theoretical expected coverage for random sequences (see Theorem 1).