Link Prediction in Graph Streams

Peixiang Zhao
Florida State University
Tallahassee, Florida 32306
zhao@cs.fsu.edu

Charu Aggarwal
IBM Watson Research Center
Yorktown Heights, New York 10598
charu@us.ibm.com

Gewen He
Florida State University
Tallahassee, Florida 32306
he@cs.fsu.edu

***Abstract*—Link prediction is a fundamental problem that aims to estimate the likelihood of the existence of edges (links) based on the current observed structure of a graph, and has found numerous applications in social networks, bioinformatics, E-commerce, and the Web. In many real-world scenarios, however, graphs are massive in size and dynamically evolving in a fast rate, which, without loss of generality, are often modeled and interpreted as *graph streams*. Existing link prediction methods fail to generalize in the graph stream setting because graph snapshots where link prediction is performed are no longer readily available in memory, or even on disks, for effective graph computation and analysis. It is therefore highly desirable, albeit challenging, to support link prediction *online* and in a *dynamic* way, which, in this paper, is referred to as the *streaming link prediction* problem in graph streams. In this paper, we consider three fundamental, neighborhood-based link prediction target measures, *Jaccard coefficient*, *common neighbor*, and *Adamic-Adar*, and provide accurate estimation to them in order to address the streaming link prediction problem in graph streams. Our main idea is to design cost-effective graph sketches (constant space per vertex) based on MinHash and vertex-biased sampling techniques, and to propose efficient sketch based algorithms (constant time per edge) with both theoretical accuracy guarantee and robust estimation results. We carry out experimental studies in a series of real-world graph streams. The results demonstrate that our graph sketch based methods are accurate, efficient, cost-effective, and thus can be practically employed for link prediction in real-world graph streams.**

I. Introduction

Link prediction is a long-standing, fundamental problem in data mining, machine learning, and network sciences, due in particular to its significance and wide applications in social networks [1], [2], [3], biology [4], [5], and the Web [6], [7]. The basic instance of the link prediction problem is to make use of the current structure of a graph to predict unseen edges arising in the forseeable future [3]. Formally, given a snapshot $G(t)$ of a graph at time t (or a series of graph snapshots till t), link prediction is to determine which new edges (links) will arise in the future graph $G(t')$ within the time interval $[t, t']$, where $t' > t$. Existing link prediction solutions, regardless of supervised [8], [9], [10] or unsupervised [11], [3], typically build on a key, *snapshot-based* assumption, where graph snapshots should fit either in memory or on disks such that they are always readily available for graph computation in link prediction.

However, in many real-world scenarios, graphs where link prediction is performed are often too massive to be safely maintained in main memory or on disks. Furthermore, real-world graphs are dynamically evolving in extremely fast speed, thus making themselves unavailable in their entirety at any particular moment of time. Without loss of generality, such graphs can naturally be modeled and interpreted as *graph streams* [12], [13], [14], [15], [16], [17], [18], where a potentially infinite sequence $(e_0, e_1, \ldots, e_t, \ldots)$ of edges $e_i = (u, v)$, $i \geq 0, u, v \in V$, representing high-speed interactions between entities of underlying graphs, are received and updated dynamically. In the graph stream context, link prediction turns out to be a new and more challenging *streaming link prediction* problem, which, interestingly enough, has found a wide range of real-world applications:

- In social networks, interactions between participants including friend requests, document/photo sharing, and message exchanges are fast, transient, and of very large scale. It is desirable to support online link prediction in social graph streams for user/commodity recommendation, market analysis, and business intelligence [19], [20], [21], [22];
- Studies in network evolution have been widely applied in trend analysis and topology discovery in Internet and communication networks [23], [24], [25]. In principle, each network evolution model corresponds to a link prediction instance in graph streams;
- In intrusion monitoring and counter-terrorism applications, continuous communication between IP-pairs or individuals constitute basic interactive patterns of underlying dynamic graphs. It is of special interest to employ intrusion history to predict in real time malicious communication that could happen in the future in order to circumvent large-scale outbreaks of cyber-attacks and crime [26], [27], [15].

Unfortunately, state-of-the-art link prediction solutions fail to generalize in graph streams due in particular to two critical reasons. First, existing link prediction methods rely primarily on the computation and exploration of *static* graph snapshots. In fast evolving graph streams, however, such methods become clearly inaccurate in detecting and capturing seasonal changes of linkage patterns in real time. Streaming link prediction is inherently a dynamic problem, which is supposed to be supported *online* and *in a dynamic way* in parallel with the rapid changes and evolutions of underlying graph streams. Second, a typical graph stream contains a potentially endless sequence of edges that are too massive to be explicitly materialized in memory or on hard disks [13]. It is therefore hard to support multi-pass traversals of the underlying graph for any nontrivial graph-based computation [17], [15], [16]. Consider, for instance, a typical communication network containing 10^8 vertices. The number of possible vertex-pairs, which constitute

the problem space to be explored for link prediction, is of the order of 10^{15} (in petabyte-scale). Although the number of real edges occurring in the graph stream turns out to be significantly smaller due to graph sparsity, it remains challenging to compute and maintain on such a large graph crucial graph measures and structure-aware properties, such as common neighbor, shortest path, random walk, and personalized PageRank, which have been widely adopted in existing link prediction solutions [3]. As a result, the snapshot-based assumption for existing link prediction solutions simply does not hold in the graph stream setting, whereas new techniques are in an urgent need to address the challenging streaming link prediction problem in graph streams.

In this paper, we consider three elementary, neighborhood-based link prediction measures, *Jaccard coefficient*, *common neighbor*, and *Adamic-Adar* [7], and design efficient and cost-effective graph sketches for accurate estimation to them in graph streams. Although simple in their formulations, these target measures are effective means for link prediction and have proven to outperform many complicated, learning-based link prediction methods in terms of prediction accuracy [11], [9] with theoretical justification [28]. The broad idea of our work is to identify structural correlations of different target measures from within the neighborhood of vertices in a graph stream, and encode such salient, structure-aware correlations into different space-efficient graph sketches (constant space complexity per vertex) for streaming link prediction. Specifically, we design MinHash [29] based graph sketches to estimate the Jaccard coefficient measure, and vertex-biased reservoir sampling [30], [31] based graph sketches to estimate the common neighbor and Adamic-Adar measures. For Adamic-Adar, an alternative, *truncated* Adamic-Adar measure is further proposed, which offers almost identically accurate link prediction results as the original Adamic-Adar measure, while it can be computed *exactly* based on our graph sketches. Efficient approximate, graph sketch based algorithms (constant time complexity per edge) are further proposed to estimate the target measures with both theoretical guarantee and empirically robust and accurate estimation results.

The main contributions of this work are briefly summarized as follows,

1) We formalize the streaming link prediction problem in graph streams and identify a series of fundamental, neighborhood-based link prediction measures, including Jaccard coefficient, common neighbor, and Adamic-Adar, as target measures for streaming link prediction. To the best of our knowledge, this is the first work addressing the link prediction problem in the graph stream setting;
2) We design MinHash based graph sketches to estimate Jaccard coefficient and vertex-biased reservoir sampling based graph sketches to estimate common neighbor and Adamic-Adar with both theoretical guarantee and empirically robust and accurate estimation. The proposed graph sketches and corresponding estimation algorithms are efficient, cost-effective, and of theoretical and practical interest in their own right in graph streams;
3) We carry out extensive experimental studies on three real-world graph streams. Experimental results clearly demonstrate the effectiveness and efficiency of the proposed graph sketches and corresponding estimation algorithms, in comparison with a series of existing snapshot-based link prediction methods, for streaming link prediction in graph streams.

The remainder of this paper is organized as follows. We will first brief the related work in Section II. In Section III, we will formally define the streaming link prediction problem in graph streams, and review the neighborhood-based target measures for streaming link prediction. In Section IV, we will discuss in detail the MinHash based graph sketches for Jaccard coefficient estimation. In Section V and Section VI, we will examine the vertex-biased sampling based graph sketches to estimate common neighbor and Adamic-Adar, respectively, in graph streams. We will present our experimental studies and results in Section VII. The concluding remarks will be further summarized in Section VIII.

II. Related Work

Link prediction. The problem of link prediction has been extensively studied in data mining, machine learning, recommendation systems, and network sciences [32], [11], [33]. Many unsupervised link prediction solutions are based on the computation of graph proximity measures or structure-aware heuristics [9], [34], [3]. It was further verified that none of these measures or heuristics is particularly dominant in different situations [9]. However, the simple neighborhood-based measures, such as common neighbor, Jaccard coefficient, and Adamic-Ada, often outperform more complicated link prediction methods in terms of prediction performance [9], [3] with theoretical justification [28]. Link prediction has also been explored more generally in the context of classification [10], [35], as it can be treated as a special case of supervised learning in which features and class labels (corresponding to existence or absence of edges) are associated with the edges of a graph. This opened the door to a variety of machine learning and data mining techniques for link prediction [6], [22], [9], [36]. Research on link prediction across multiple networks has also been discussed [5], [20], [30]. It is worth mentioning that, state-of-the-art link prediction methods, regardless of supervised or unsupervised, are primarily designed for dynamic graphs with an underlying snapshot-based assumption, where each graph snapshot is static and can be explicitly materialized either in memory or on disks before link prediction is performed. This assumption however does not hold in real-world graph streams that are of excessively large scale and with a fast evolving rate.

Graph streams. Over the last decade, there has been considerable interest in processing massive graphs in the data stream model motivated in particular by the fact that real-world dynamic graphs are often too large to be stored in main memory or disks of a single machine [13]. A typical graph stream consists of edges of the underlying graph as a fast input stream, and algorithms in this model process the stream in the order it arrives with short response time and limited space. Noteworthy examples of computation in graph streams include, but are not limited to, graph connectivity and sparsification [12], maximal matching and vertex cover [37], triangle counting and sampling [38], [39], event pattern matching [40], query processing [15], outlier detection [17], and PageRank

estimation [18]. A common practice for graph stream processing is to build and maintain cost-effective data summaries, termed as *graph sketches* or *graph synopses* [41], for efficient and accurate estimation of graph-based functions or metrics. Technical challenges stem primarily from the intrinsic trade-off between time/space cost of graph sketches and accuracy attained for online approximate estimation [13]. Graph streams have proven to have deep connections with a variety of research areas including graph data management, big data, communication complexity, and approximation algorithms.

There exist a number of important graph-theoretic problems, including link prediction, left unsolved in the graph stream setting. The streaming link prediction problem in graph streams is considered particularly difficult, as there is no way to materialize snapshots of underlying graphs in memory or on disks for multi-pass graph traversal and computation. Meanwhile, no existing work has been proposed to estimate critical graph proximity measures that have been extensively used for link prediction. A scalable method [34] was proposed to estimate path-based proximity measures in social networks, including *Katz*, *rooted PageRank*, and *escape probability*. However, this method is based on the multi-pass traversal of underlying graphs and a series of costly, matrix-based operations, thus making it impossible to be employed in graph streams. Therefore, it is crucial to design efficient, cost-effective, and highly accurate graph sketch based methods that can encode the core structural information and intrinsic properties of dynamic graphs to address the streaming link prediction problem in graph streams.

III. Streaming Link Prediction: Definitions and Target Measures

In this paper, we consider the streaming link prediction problem in a graph stream that receives a sequence $(e_0, e_1, \ldots, e_t, \ldots)$ of edges, each of which is in the form of $e_i = (u, v)$ at the time point i, where $u, v \in V$ are incident vertices of the edge e_i. For simplicity of exposition, we assume the underlying graph is *undirected*, where the ordering of u and v of the edge is insignificant. The proposed graph sketches and corresponding estimation algorithms can be generalized to *directed* graphs with minor revision.

At any given moment of time t, the edges seen thus far from the graph stream imply a conceptual graph $G(t) = (V(t), E(t))$, where $V(t)$ is the set of vertices, and $E(t)$ is the set of *distinct* edges up to time t. We use $\tau(u, t)$ to represent the set of adjacent vertices of the vertex u in the graph $G(t)$. That is, $\tau(u, t)$ contains the *distinct* vertices adjacent to u in the graph stream till t, and the degree of u is denoted as $d(u, t) = |\tau(u, t)|$. To explicitly store a graph stream, we potentially need $O(|V(t)|^2)$ space, which is prohibitive. To circumvent this challenge, most of existing work focused on the *semi-streaming model* [13], [42], where graph stream algorithms are permitted $O(|V(t)| \times \text{poly}(\log|V(t)|))$ space, as most graph-theoretic problems turn out to be provably intractable if the available space is sub-linear in $|V(t)|$, whereas they become feasible if the memory is roughly proportional to the number of vertices in the graph. In this paper, we adopt this well-accepted semi-streaming model for graph streams.

The streaming link prediction problem in graph streams can be formalized as follows,

Definition 1 (Streaming Link Prediction). *Given a graph stream $G(t)$ at time t, the streaming link prediction problem is to predict whether there is or will be an edge $e = (u, v)$ for any pair of vertices $u, v \in V(t)$ and $e \notin E(t)$.*

The streaming link prediction problem is challenging because the potential problem space is $(V(t) \times V(t)) \setminus E(t)$, and we cannot materialize the graph stream as snapshots for link prediction. Instead, we have only one chance to examine each edge in the graph stream and make predictions online and in a dynamic way.

To this end, we will examine a series of elementary, neighborhood-based graph proximity measures and provide accurate estimation to them in order to address the streaming link prediction problem in graph streams. These measures lie at the heart of link prediction, and represent the ideal goals we aim to compute, although they are of course impossible to be derived *exactly* from the fast, massive graph streams. We refer to these graph measures as *target measures* for streaming link prediction. The most well-known neighborhood-based measures in link prediction were first discussed in the seminal paper [3], including *preferential attachment*, *common neighbor*, *Jaccard coefficient*, and *Adamic-Adar*:

Definition 2 (Target Measures). *In a graph stream $G(t)$ and for any $u, v \in V(t)$, the target measures for streaming link prediction are defined as follows,*

1) ***Preferential attachment:*** $|\tau(u, t)| \times |\tau(v, t)|$;
2) ***Common neighbor:*** $|\tau(u, t) \cap \tau(v, t)|$;
3) ***Jaccard coefficient:*** $\frac{|\tau(u,t) \cap \tau(v,t)|}{|\tau(u,t) \cup \tau(v,t)|}$;
4) ***Adamic-Adar:*** $\sum_{w \in \tau(u,t) \cap \tau(v,t)} \frac{1}{\log(|\tau(w,t)|)}$

Although many path or random-walk based graph proximity measures including *Katz*, *hitting-time*, *rooted PageRank*, and *SimRank* have also been employed for link prediction [33], [9], [3], neighborhood-based measures have proven to perform competitively well, or even better, in comparison with those path and random-walk based ones in existing link prediction solutions [28], [3] (Our experimental studies in Section VII further verifies this point). Meanwhile, it is straightforward to generalize the neighborhood-based measures as a first step in the challenging graph stream scenario.

For preferential attachment, it requires an accurate estimation of the number of distinct edges incident on *each vertex*, *i.e.*, $|\tau(u, t)|, u \in V(t)$. This is not as easy as it sounds because the distinct edges cannot be explicitly maintained and updated for exact counting in a graph stream. Nevertheless, existing methods such as Count-Min [43] and gSketch [15] can be directly employed to estimate $|\tau(u, t)|$ for the computation of preferential attachment in graph streams.

The other three neighborhood based measures, however, are even harder to estimate in graph streams, because they require *pairwise* vertex tracking rather than a clean product of vertex-centric distinct element cardinalities, as formulated in preferential attachment. Common neighbor, as the name literally dictates, counts the number of common incident vertices of both u and v in a graph stream. Jaccard coefficient normalizes for the common neighbors of u and v between which the

link is to be predicted. Adamic-Adar [7] penalizes the "spam" vertices having high vertex degrees with low weights. That is, if two vertices u and v share a common neighbor w with a larger degree, w is then weighted less in Adamic-Adar because of the term-weighting $1/\log(|\tau(w,t)|)$ of w. In the sequel, we will consider **Jaccard coefficient**, **common neighbor**, and **Adamic-Adar** as our main target measures, and discuss how each of them can be estimated accurately in graph streams by efficient and cost-effective graph sketches. We will provide both theoretical guarantees and empirical analysis on the accuracy of our estimation algorithms. Note that these target measures are so representative that many neighborhood-based link prediction measures, such as *Salton index*, *Sørensen index*, *hub promoted index*, and *hub depressed index* [33], are just their variations or special cases. As a result, the proposed graph sketches and approximate algorithms can be extended effortlessly to estimate a family of neighborhood-based link prediction measures in graph streams.

IV. Jaccard Coefficient Estimation

In this section, we design MinHash based graph sketches to estimate the first target measure, Jaccard coefficient, in graph streams. MinHash [29] has been employed in a series of data mining tasks, such as community detection, itemset counting, and classification, to approximate Jaccard coefficient [44], [16]. We generalize the idea of MinHash in the graph stream setting for streaming link prediction.

Our algorithm adapts a hash function $\mathcal{H} : u \in V \rightarrow [0, 1]$ with the vertex identifier u as the argument, and a hashed value in the range $[0, 1]$ as output. Given a graph stream, we maintain a MinHash based graph sketch for each vertex u with two key values: the *minimum adjacent hash value*, $H(u)$, and the *minimum adjacent hash index*, $I(u)$. The former maps on to the minimum hash value $\mathcal{H}(v)$ at some adjacent vertex v of u, $v \in \tau(u, t)$, and the latter maps on to the index v:

$$H(u) = \min \mathcal{H}(v), (u, v) \in E(t) \tag{1}$$

$$I(u) = \operatorname{argmin} \mathcal{H}(v), (u, v) \in E(t) \tag{2}$$

Note these two values are rather easy to maintain and update, as shown in Algorithm 1. This algorithm starts off by initializing $H(\cdot)$ to ∞, and $I(\cdot)$ to NULL for every vertex in the graph stream (Lines 1-2). For each new incoming edge (u, v) in the graph stream, we update MinHash-based graph sketches for vertices u and v, respectively (Lines 4-7). An immediate observation is the following:

Lemma 1. *The probability that $I(u) = I(v)$ is exactly equal to Jaccard coefficient between vertices u and v.*

Proof: The number of possible values of $I(u)$ (or $I(v)$) is equal to the size of the adjacent set $\tau(u, t)$ (or $\tau(v, t)$). So the number of possible values for either $I(u)$ or $I(v)$ is equal to $|\tau(u, t) \cup \tau(v, t)|$, as both $I(u)$ and $I(v)$ follow the same sorting order with the use of an identical hash function, $\mathcal{H}(\cdot)$, and all such values appear with equal probability. The number of possibilities for which $I(u) = I(v)$ is $|\tau(u, t) \cap \tau(v, t)|$. Therefore, the overall probability for $I(u) = I(v)$ is

$$\Pr(I(u) = I(v)) = \frac{|\tau(u, t) \cap \tau(v, t)|}{|\tau(u, t) \cup \tau(v, t)|} \tag{3}$$

Algorithm 1: Basic MinHash Based Graph Sketches

Input : A graph stream $G(t)$
Output: MinHash based graph sketches
// Sketch initialization
1 **foreach** *vertex* $u \in G(t)$ **do**
2 $H(u) \leftarrow \infty$; $I(u) \leftarrow$ NULL
// Sketch maintenance and update
3 **while** *an edge* $(u, v) \in G(t)$ **do**
4 **if** $\mathcal{H}(v) < H(u)$ **then**
5 $H(u) \leftarrow \mathcal{H}(v)$; $I(u) \leftarrow v$
6 **if** $\mathcal{H}(u) < H(v)$ **then**
7 $H(v) \leftarrow \mathcal{H}(u)$; $I(v) \leftarrow u$

which is exactly equal to Jaccard coefficient. ■

Furthermore, the above result provides an immediate, but stronger, way to estimating Jaccard coefficient as a probability by considering K ($K > 1$) independent, minwise hash functions $\mathcal{H}_1(\cdot), \ldots, \mathcal{H}_K(\cdot)$. Consequently, multiple minimum hash values $H_1(\cdot), \ldots, H_K(\cdot)$ and the corresponding indices $I_1(\cdot), \ldots, I_K(\cdot)$ are maintained accordingly in MinHash based graph sketches for each vertex. These key values are analogously initialized and updated for *each* $k \in \{1 \ldots K\}$:

if $\mathcal{H}_k(v) < H_k(u)$, **then** $H_k(u) = \mathcal{H}_k(v)$ and $I_k(u) = v$

if $\mathcal{H}_k(u) < H_k(v)$, **then** $H_k(v) = \mathcal{H}_k(u)$ and $I_k(v) = u$

Therefore, K operations need to be performed for every vertex u (and v) of each incoming edge $e_t = (u, v)$ in the graph stream. The overall time and space complexity of this algorithm is $O(K)$ per edge, which is constant. It is also easy to generalize Lemma 1 for the following theorem:

Theorem 1. *Let $r \leq K$ be the number of minwise hash functions for which $I_k(u) = I_k(v)$, $1 \leq k \leq K$. Jaccard coefficient can be estimated by r/K.*

This theorem follows immediately from Lemma 1. It remains to establish theoretical bounds on the *quality* of the estimated Jaccard coefficient, as follows,

Theorem 2. *Let the estimate $J'(u, v)$ of Jaccard coefficient of a pair of vertices (u, v) based on Theorem 1 have the true expected value $\bar{J}(u, v)$. It can be shown that, for any $\delta \in (0, 1)$,*

$$\Pr(J'(u, v) - \bar{J}(u, v) < (1 - \delta)\bar{J}(u, v)) \leq e^{-K \cdot \bar{J}(u,v) \cdot \delta^2/2} \tag{4}$$

and for any $\delta' \in (0, 2e - 1)$,

$$\Pr(J'(u, v) - \bar{J}(u, v) > (1 + \delta')\bar{J}(u, v)) \leq e^{-K \cdot \bar{J}(u,v) \cdot \delta'^2/4} \tag{5}$$

Proof: The estimate $J'(u, v)$ can be expressed as an average of K *i.i.d.* Bernoulli random variables $X_1 \ldots X_K$,

$$J'(u, v) = \sum_{k=1}^{K} X_k / K \tag{6}$$

The value of X_k $(1 \leq k \leq K)$ is 1, if the k-th MinHash indexes $I_k(u)$ and $I_k(v)$ are equal for vertices u and v. Each of these K Bernoulli random variables takes on the value of 1 with probability of $\bar{J}(u,v)$ according to Lemma 1. Therefore, we can apply the lower- and upper-tail Chernoff bounds to derive the aforementioned results [45]. ■

It is important to note that because K, the number of minwise hash functions, occurs in the exponent of Equation 4 and Equation 5, the probability of the bounds being violated falls off *exponentially*. This implies that, for relatively modest values of K, one can obtain tight bounds on the accuracy of the estimated Jaccard coefficient based on the MinHash based graph sketches.

V. Common Neighbor Estimation

To estimate common neighbors of two vertices in a graph stream, a natural question arises: *whether a sample of edges from the graph stream are good enough to estimate this target measure accurately?* Unfortunately, unbiased sampling provides a negative answer to this question mainly due to the power-law degree distribution of real-world graphs [46]. For example, consider a graph in which the top 1% of the high-degree vertices (degrees greater than 10) contain 99% of the edges. A down-sampling with a rate of 10% will capture the neighborhood information of such top 1% high-degree vertices robustly, but may not capture even a single edge for the remaining 99% low-degree vertices, which, however, are very important in streaming link prediction because many new edges occur between these low-degree vertices.

To address this issue, we design the *vertex-biased* sampling based graph sketches, where for each vertex $u \in G(t)$, a **reservoir** $S(u)$ of budget L is associated to dynamically sample L incident edges of u (Note if an edge (u,v) is sampled, the vertex v, not the edge itself, is maintained and updated in u's reservoir, $S(u)$). This sampling approach is biased, because the number of sampled edges, L, is fixed for all vertices in the graph stream, and is therefore disproportionately higher for low-degree vertices than high-degree ones. While sampling L incident edges of each vertex can be implemented by the traditional reservoir sampling method [31], another problem arises for high-degree vertices instead: for two vertices with degrees larger than L, the number of common neighbors is hard to estimate accurately from their *independent* samples. For example, consider the case where the budget is $L = 10$, and the two vertices u and v have a degree $1,000$ each with all such $1,000$ incident vertices being common neighbors of u and v. Then, if 10 adjacent neighbors of u and v are sampled independently, the expected number of common neighbors of u and v is $10 * 10/1000 = 0.1$, which definitely is not an accurate estimation.

To this end, we adopt this constant budget-based sampling approach with an important caveat: *random samples in different reservoirs of vertices are forced to be **dependent** on each other*. To model this dependency, we consider an implicit sorting order of vertices to impose a priority order on the vertex set of the graph stream. Such a priority order is enforced with the use of a hash function $\mathcal{G} : u \in V \rightarrow (0,1)$, where the argument u is a vertex identifier, and the output is a real number in the range $(0,1)$ with lower values indicating higher priority. Specifically, vertices retained in the reservoir $S(u)$ are those having the highest L priority orders among all incident neighbors of u. In order to dynamically maintain and update $S(u)$ when the graph stream evolves, it is important to note that, if the degree of the vertex u satisfies $d(u,t) > L$, only a fraction $\eta(u,t) = L/d(u,t)$ of all the incident vertices of u can be retained in the sketch $S(u)$. To account for this, we define a *threshold* of the priority value $\mathcal{G}(\cdot)$ for each incident neighbor of u that can survive in the reservoir $S(u)$ as

$$\eta(u,t) = \min\{1, L/d(u,t)\} \tag{7}$$

Unfortunately, it is impossible to online pick L incident vertices whose priority values are less than $\eta(u,t)$ (L incident vertices with the highest-L priority), because the degree $d(u,t)$ is unknown in advance[1]. Alternatively, we choose to dynamically maintain a lower estimate, $\underline{\eta}(u)$, and an upper estimate, $\overline{\eta}(u)$, $(\underline{\eta}(u) \leq \overline{\eta}(u))$ of the priority threshold, $\eta(u,t)$, for each vertex u in order to regulate the selection of prioritized incident vertices in the reservoir $S(u)$.

Algorithm 2: Vertex-biased Sampling Graph Sketches

Input : A graph stream $G(t)$
Output: Vertex-biased sampling based graph sketches

```
// Sketch initialization
foreach vertex u ∈ G(t) do
    η̲(u) = η̄(u) = 1, S(u) ← ∅
// Sketch maintenance and update
while a new edge (u, v) ∈ G(t) do
    if v ∉ S(u) then
        if |S(u)| < L then
            S(u) ← S(u) ∪ {v}
        else
            if G(v) ≤ η̲(u) then
                k ← argmax G(w), w ∈ S(u)
                S(u) ← S(u) \ {k}
                S(u) ← S(u) ∪ {v}
                η̄(u) ← G(k)
                k* ← argmax G(w), w ∈ S(u)
                η̲(u) ← G(k*)
    if u ∉ S(v) then
        if |S(v)| < L then
            S(v) ← S(v) ∪ {u}
        else
            if G(u) ≤ η̲(v) then
                k ← argmax G(w), w ∈ S(v)
                S(v) ← S(v) \ {k}
                S(v) ← S(v) ∪ {u}
                η̄(v) ← G(k)
                k* ← argmax G(w), w ∈ S(v)
                η̲(v) ← G(k*)
```

Algorithm 2 illustrates the construction and maintenance of vertex-biased sampling based graph sketches for a graph

[1] After all, streaming link prediction can be executed at any moment in the lifetime of a graph stream, and we cannot stop the stream and calculate key statistics, such as $d(u,t)$, offline.

stream $G(t)$. It starts with an initialization for every vertex u in the graph stream, $\underline{\eta}(u) = \overline{\eta}(u) = 1$, because all incident neighbors of u are eligible candidates to be potentially included in the empty reservoir, $S(u)$ at the beginning. To account for this case, we set the priority thresholds equal to the maximum possible value, 1, representing the lowest priority (Lines 1-2). For each new edge (u, v) in the graph stream, $S(u)$ and $S(v)$ are updated accordingly. In the following, only the update for the sketch $S(u)$ is described (Lines 4-14), though the procedure for updating the sketch $S(v)$ is very similar (Lines 15-25).

When an edge (u, v) is received from the graph stream, it is first checked whether v has already been in the reservoir $S(u)$ of u (Line 4). If so, nothing needs to be done. Otherwise, we further check if $S(u)$ is full to its capacity L (Line 5). If not, v can be safely added to $S(u)$ (Line 6), and both $\underline{\eta}(u)$ and $\overline{\eta}(u)$ remain unchanged. Otherwise, if $S(u)$ is indeed full, it is further checked if $\mathcal{G}(v) \leq \underline{\eta}(u)$, meaning the edge (u, v) is with the highest L priority and well qualified to be maintained in $S(u)$ (Line 8). However, another vertex $k \in S(u)$ with the largest value of $\mathcal{G}(k)$ (*i.e.*, lowest priority) has to be ejected first from $S(u)$ before v is added (Lines 9-11). The value of $\underline{\eta}(u)$ is correspondingly reduced to the current largest hash value of the vertex k^* remaining in the modified reservoir $S(u)$, and the value of $\overline{\eta}(u)$ is set to the hash value $\mathcal{G}(k)$ of the ejected vertex k (Lines 12-14).

We remark that the use of a universal hash function $\mathcal{G}(\cdot)$ in biased sampling ensures that a common incident vertex w of both u and v will be given the same priority in the reservoirs, $S(u)$ and $S(v)$. For example, in the previous example of two vertices with degree 1000 each (all of which are common neighbors), both vertices will have *exactly* the same reservoirs, which provide an accurate indication that 100% of the incident edges of each vertex respectively will contribute to the same set of common neighbors. This is a crucial property of our vertex-biased sampling based graph sketches.

The space complexity of Algorithm 2 is $O(L)$ per vertex, and its time complexity is $O(1)$ for each edge in the graph stream, which is constant. Theoretically, if the number of common neighbors in the reservoirs $S(u)$ and $S(v)$ is $|S(u) \cap S(v)|$, the real number of common neighbors of u and v can be estimated based on the following theorem:

Theorem 3. *Let $\eta(u, t)$ and $\eta(v, t)$ be the fraction (threshold) of incident neighbors of u and v, respectively, which are sampled. The number of common neighbors, C_{uv}, of u and v is*

$$C_{uv} = \frac{|S(u) \cap S(v)|}{\max\{\eta(u, t), \eta(v, t)\}} \tag{8}$$

Proof: because vertices in the two reservoirs $S(u)$ and $S(v)$ are selected in the same priority order, sampling these two reservoirs at the rates of $\eta(u, t)$ and $\eta(v, t)$ respectively will produce the same expected result as sampling each at the rate of $\max\{\eta(u, t), \eta(v, t)\}$. The $\max\{\eta(u, t), \eta(v, t)\}$ fraction of vertices are expected to be common from each reservoir. It means that the expected value of $|S(u) \cap S(v)|$ is $C_{uv} \cdot \max\{\eta(u, t), \eta(v, t)\}$, so the above result holds. ■

The last unsolved issue is about the determination of the hypothetical sampling parameter $\eta(u, t)$, which cannot be computed exactly, but can be estimated by the lower and upper estimates, $\underline{\eta}(u)$ and $\overline{\eta}(u)$, as follows,

Theorem 4. *Given a graph stream, the reservoir $S(u)$ of the vertex u provides a biased sample of the edges incident on u with the sampling parameter $\eta(u, t)$ that can be estimated as follows,*

$$\eta(u, t) = (\underline{\eta}(u) + \overline{\eta}(u))/2 \tag{9}$$

Proof: The biased nature of the samples follows naturally from the hash function $\mathcal{G}(\cdot)$ used for selection of incident edges. All edges with hash values less than $\underline{\eta}(u)$ are maintained in $S(u)$. The value of $\underline{\eta}(u)$ underestimates the sampling probability $\eta(u, t)$. A symmetric argument can be constructed demonstrating that $\overline{\eta}(u)$ is an upper-bound estimate by examining the conceptual reservoir consisting of edges with hash values greater than $\overline{\eta}(u)$ only. This corresponds to a reservoir with the sampling probability $(1 - \eta(u, t))$. Because of the symmetry of the argument, the true estimated value of $\eta(u, t)$ is therefore $(\underline{\eta}(u) + \overline{\eta}(u))/2$. ■

As a result, once the value of $\eta(u, t)$ is determined, the number of common neighbors between a pair of vertices can be estimated accurately, according to Theorem 3, in a graph stream.

VI. Adamic-Adar Estimation

An important observation of Adamic-Adar is that the common neighbor w of vertices u and v with a higher vertex-degree is weighted less, because of its proclivity to be a noisy vertex, as accounted for in the term $1/\log(|\tau(w, t)|)$ [7]. Such a weighting strategy makes it very challenging to estimate Adamic-Adar in graph streams. In this section, we consider an approximate variation of Adamic-Adar, which *truncates* the insignificant high-degree common neighbors of u and v for streaming link prediction, because they contribute little in the computation of Adamic-Adar. It turns out that such an approximation still preserves the accuracy of Adamic-Adar, but can be computed *exactly* based on the vertex-biased sampling based graph sketches, as proposed in Section V.

Definition 3 (Truncated Adamic-Adar). *The truncated Adamic-Adar, denoted as $\overline{AA}(u, v)$, between two vertices u and v, is defined in the same way as Adamic-Adar, except that the components contributed by common neighbors with degrees greater than d_{max} are eliminated:*

$$\overline{AA}(u, v) = \sum_{\{w \in \tau(u,t) \cap \tau(v,t) : |\tau(w,t)| \leq d_{max}\}} \frac{1}{log(|\tau(w, t)|)} \tag{10}$$

The truncated Adamic-Adar can be computed exactly by vertex-biased sampling based graph sketches, as presented in Algorithm 3. The core idea is that each of the components in Equation 10 can be directly computed if both vertices u and v are present in the graph sketch $S(w)$ of their common neighbor w, whose degree is no larger than d_{max}. At the beginning, we initialize the capacity of the reservoirs, $L = d_{max}$, for all vertices (Lines 1-2). This is because vertices whose degrees are larger than d_{max} will not be considered in the computation of truncated Adamic-Adar . We then examine all the graph sketches one by one. For each vertex w, both vertices u and v

Algorithm 3: Truncated Adamic-Adar Estimation

Input : An edge (u, v) in the graph stream $G(t)$, the degree threshold d_{max}
Output: Truncated Adamic-Adar, $\overline{AA}(u, v)$

```
// Sketch initialization
1 foreach vertex w ∈ G(t) do
2     L ← d_max
// Truncated Adamic-Adar estimation
3 AA(u, v) ← 0
4 foreach vertex w ∈ G(t) do
5     if u ∈ S(w) and v ∈ S(w) then
6         if η(w) = 1 and η̄(w) = 1 then
7             AA(u, v) ← AA(u, v) + 1/log|S(w)|
```

need be present in $S(w)$ (Line 5). It is further checked whether w has a degree at most d_{max} by examining if the values of $\eta(w)$ and $\overline{\eta}(w)$ equal to 1, indicating that the graph sketch $\overline{S}(w)$ is not full yet (Line 6). If so, $(1/\log(|\tau(w,t)|))$ is added to $\overline{AA}(u, v)$ (Line 7). Note that the degree value, $|\tau(w,t)|$, is equal to the current reservoir size, $|S(w)|$.

The main computational bottleneck of Algorithm 3 is that the graph sketches of all vertices need to be scanned once for each incoming new edge (u, v) in the graph stream. To alleviate this problem, we consider adopting inverted indexes to facilitate the computation of truncated Adamic-Ada. Specifically, for each vertex u in the graph stream, an auxiliary inverted index structure, $L(u)$, is built to maintain vertex identifiers of v whose graph sketch $S(v)$ contains u, *i.e.*, $v \in L(u)$ if and only if $u \in S(v)$. Note that such inverted indexes have drastically different sizes, but the summation of their sizes is exactly equal to the summation of the sizes for all graph sketches:

$$\sum_{u \in G(t)} |L(u)| = \sum_{u \in G(t)} |S(u)| \leq \sum_{u \in G(t)} d_{max} = |V(t)| * d_{max} \tag{11}$$

The inverted index structures can be maintained accordingly when graph sketches are updated in graph streams: (1) when the vertex v is added to the sketch $S(u)$, the vertex u is maintained into the inverted index $L(v)$ simultaneously; (2) when v is removed from the reservoir $S(u)$, u should be removed from the corresponding inverted index $L(v)$ as well. Based on these inverted indexes, we need not scan all vertices of the graph stream to compute truncated Adamic-Adar. Instead, the common neighbor w of u and v can be efficiently retrieved from inverted indexes of u and v, $w \in L(u) \cap L(v)$. As a result, the time complexity of this index-based method turns out to be $O(|L(u)| + |L(v)|)$, and its space complexity is $O(|L(u)| + |L(v)| + 2 * d_{max})$, while the average-case time and space complexity is simply $O(d_{max})$.

VII. Experiments

In this section, we present our experimental studies for the streaming link prediction problem in a series of real-world graph streams. We demonstrate the accuracy, efficiency, and cost of our graph sketch based estimation methods in comparison with a variety of snapshot-based link prediction techniques. All our experiments were carried out on a desktop PC with Intel Quad Core 3.20GHz CPU and 8GB memory running Windows 7 operating system. All the methods were implemented in C++.

A. Datasets

We chose three real-world, publicly available graph datasets in our experimental studies. Note that their edges are attached with timestamps indicating when they are first created in the graphs, and thus can be ordered, formulated, and processed in a form of graph streams:

1. DBLP. We extracted all conference papers with publication dates ranging from 1956 to 2008 in DBLP database[2]. There were $595,406$ authors and $602,684$ papers in total. For each paper, the authors are listed in a specific order as $a_1, a_2, \ldots, a_k$, and unordered author-pairs (a_i, a_j) are generated as collaboration edges $(1 \leq i < j \leq k)$. There are a total of $1,954,776$ author-pairs, thus forming a graph stream on the underlying co-authorship graph;

2. Amazon Product Co-purchasing Network. This dataset was collected from Amazon based on the *customers-who-bought-this-item-also-bought* feature[3]. Namely, if an item i is purchased together with another item j, the co-purchase graph will contain an edge between i and j, and the time of this co-purchase transaction is imposed on the edge, thus forming a large graph stream comprising $410,271$ vertices and $11,179,587$ edges;

3. Wikipedia. This dataset models the growth and evolution of the online knowledge base, Wikipedia[4]. Vertices are Wikipedia articles, and there exists an edge if article i references article j, affiliated with a time-stamp for this reference. A graph stream thus contains a large number of reference edges between Wikipedia articles with $1,870,709$ vertices and $39,953,145$ edges in total.

B. Evaluation Methods

Given a graph stream $G(t)$, we consider two specific timestamps t_0 and t_1, $t_0 \leq t_1$, and apply different streaming link prediction methods upon $G(t_0)$ to generate a list of predicted edges not present in $G(t_0)$ but expected to appear in $G(t_1)$. We refer to the time interval $[0, t_0]$ as the *training interval* and $[t_0, t_1]$ as the *test interval*. For a graph stream, we further denote ρ as the *ratio of the training interval w.r.t.* the whole lifetime of the graph stream, and ρ is set 50% if not specified otherwise, meaning that the training interval and the test interval is temporally partitioned evenly for the whole graph stream: $|t_0| = |t_1 - t_0|$. We choose a set $\mathsf{Core} \subseteq V(t_0)$ of vertices that show up in both the training interval and the test interval for streaming link prediction, because for new vertices arising in the test interval (during the time period of $[t_0, t_1]$), it is impossible to predict the edges incident to them at the training interval as they have not even shown up yet (before the timestamp t_0). Moreover, we denote E_{old} as the set of edges $(u, v) \in E(t_0)$, $u, v \in \mathsf{Core}$, and E_{new} as the set of edges $(u', v') \in E(t_1) \setminus E(t_0)$, $u', v' \in \mathsf{Core}$. So E_{new} contains all the edges that arise in the test interval but *not* in

[2] http://www.informatik.uni-trier.de/~ley/db/
[3] https://snap.stanford.edu/data/index.html
[4] http://socialnetworks.mpi-sws.org/data-wosn2008.html

TABLE I: Link prediction accuracy of 12 methods on the DBLP graph stream

Methods	Ext-Ja	App-Ja	Ext-CN	App-CN	Ext-Adar	App-Adar
Accuracy	120.57	104.25	112.66	108.91	93.47	92.57
Methods	Katz	Preferential Attachment	PropFlow	Rooted PageRank	Shortest-Path Count	SimRank
Accuracy	100.48	114.97	96.76	59.82	107.49	74.85

TABLE II: Link prediction accuracy of 12 methods on the Amazon co-purchasing graph stream

Methods	Ext-Ja	App-Ja	Ext-CN	App-CN	Ext-Adar	App-Adar
Accuracy	116.15	106.74	121.07	109.70	116.57	116.15
Methods	Katz	Preferential Attachment	PropFlow	Rooted PageRank	Shortest-Path Count	SimRank
Accuracy	88.38	116.23	98.75	71.48	147.70	110.95

TABLE III: Link prediction accuracy of 6 methods on the Wikipedia graph stream

Methods	Ext-Ja	App-Ja	Ext-CN	App-CN	Ext-Adar	App-Adar
Accuracy	8.19	8.04	9.27	9.08	10.55	11.90

the training interval, and thus is the target set of edges we seek to predict in the graph streams.

For any link prediction method p, its output is a ranked list L_p of vertex-pairs in the set $(\mathsf{CORE} \times \mathsf{CORE}) \setminus E_{old}$, which contains the predicted (but not verified) edges in a decreasing order of predictive values. The *accuracy* of p is then determined from the resultant ranked list L_p: we take the first $|E_{new}|$ vertex-pairs from L_p, and consider the *percentage* of vertex-pairs that truly arise as edges in E_{new}. In order to compare different methods for streaming link prediction, we adopt the evaluation metric in the classical paper [3], which considers the prediction accuracy in a relative term to a *random predictor*. This predictor chooses vertex pairs (u, v) from CORE that have no edges in the training interval, and predicts at random there will be (or there will not be) an edge between u and v in the test interval. In other words, the probability of the random predictor being correct for the prediction of an edge (u, v) in the test interval is:

$$\Pr(u, v) = \frac{1}{\binom{|Core|}{2} - |E_{old}|} \tag{12}$$

As a result, the *link prediction accuracy* is expressed in terms of *relative improvement* to this random predictor. This way, different streaming link prediction methods can be compared on a fair basis.

We extensively examine a series of methods for streaming link prediction, including the *exact* versions of our approximate methods. Specifically, Ext-CN denotes the *exact* common neighbor method, and App-CN is the vertex-biased sampling based graph sketch method (in Section V, $L = 80$ by default); Ext-Ja denotes the *exact* Jaccard coefficient method, and App-Ja is the MinHash based graph sketch method (in Section IV, $K = 50$ by default); Ext-Adar denotes the *exact* Adamic-Adar method, and App-Adar is the truncated, inverted index based Adamic-Adar method (in Section VI, $L = 80$ by default). We also consider a series of existing link prediction methods[5] [8], [9], including Katz (with default parameters of the maximum shortest path distance $l = 3$ and the damping factor $\beta = 0.05$. Note shorter-distance paths usually offer better link prediction accuracy results [9]), Preferential Attachment, PropFlow [9] (with a default parameter of the maximum distance $l = 3$), Rooted PageRank (with a default random walk restart parameter $\alpha = 0.5$), ShortestPath Count (with a default parameter of the maximum path distance $l = 3$) and SimRank (with a default parameter of the damping factor $C = 0.8$). It is worth noting that, except for the graph sketch based methods as proposed in this paper, including App-CN, App-Ja, and App-Adar, all the other nine methods are snapshot-based, so we have to explicitly materialize graph streams in the training interval, $G(t_0)$, before link prediction is performed. These snapshot-based link prediction methods have a "built-in" advantage to examine the global graph structure many times without consideration of streaming constraints during link prediction. Nevertheless, we will demonstrate that the prediction accuracy of our graph sketch based methods is as competitive as, or sometimes even better than that of snapshot-based link prediction methods. Moreover, our methods are extremely fast, space-efficient, and can be practically employed as high-quality solutions to the streaming link prediction problem in graph streams.

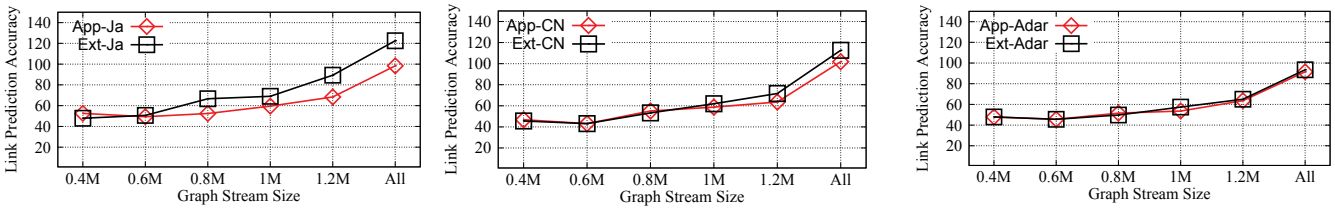
[5] https://github.com/rlichtenwalter/LPmade

C. Experimental Results

In our experimental studies, we mainly focus on the following evaluation metrics: (1) *link prediction accuracy* as defined in Section VII-B, (2) *runtime cost* for streaming link prediction, and (3) *space cost* of graph sketches or graph snapshots consumed by approximate or exact link prediction methods, respectively. We also evaluate these performance metrics in terms of key parameters in different link prediction methods, including the graph stream size S, the training interval ratio ρ, the number of minwise hash functions, K, in MinHash based graph sketches, and the reservoir budget size, L, in vertex-biased sampling based graph sketches.

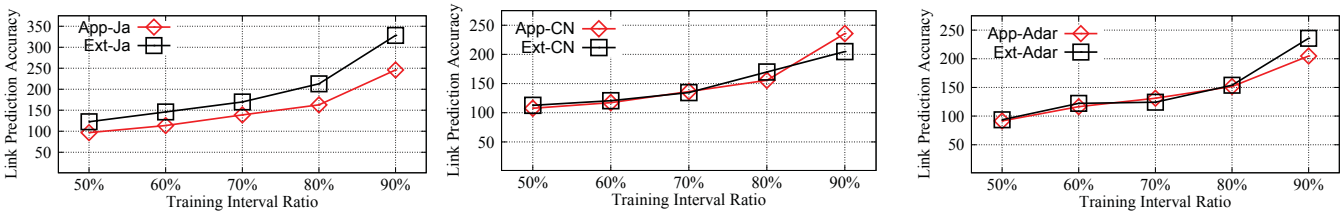
C.1 Link Prediction Accuracy

We first evaluate the link prediction accuracy of different methods in the three graph streams. The results are illustrated



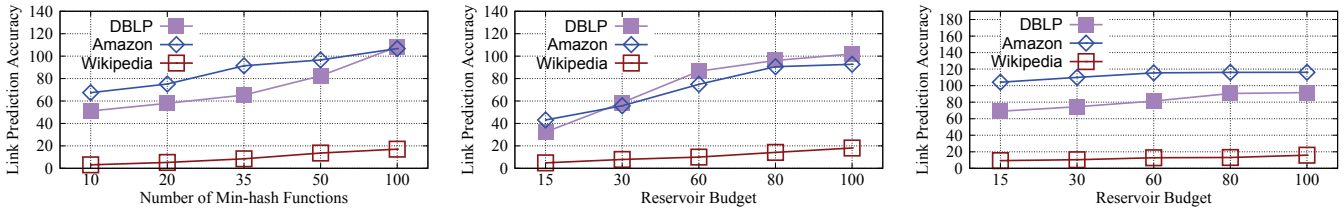
(a) App-Ja vs. Ext-Ja (b) App-CN vs. Ext-CN (c) App-Adar vs. Ext-Adar

Fig. 1: Accuracy of exact and approximate methods *w.r.t.* the graph stream size S on DBLP graph stream



(a) App-Ja vs. Ext-Ja (b) App-CN vs. Ext-CN (c) App-Adar vs. Ext-Adar

Fig. 2: Accuracy of exact and approximate methods *w.r.t.* the training interval ratio ρ on DBLP graph stream



(a) App-Ja: accuracy in terms of K (b) App-CN: accuracy in terms of L (c) App-Adar: accuracy in terms of L

Fig. 3: Accuracy of approximate methods *w.r.t.* algorithm parameters

in Tables I, II, III, respectively. Note for the largest Wikipedia stream (Table III), path-based link prediction methods, including Katz, PropFlow, Rooted PageRank,, ShortestPath Count, and SimRank, cannot finish within 4 hours, so we only list the experimental results for neighborhood based target measures. Our first observation is that neighborhood-based graph proximity measures (listed as the first row of each table), including common neighbor, Jaccard coefficient, and Adamic-Adar, can make prediction with high accuracy, compared with advanced, non-neighborhood based methods. Although there is no method that can outperform all the others in all different graph streams, exact neighborhood based methods and their approximate counterparts perform consistently well for link prediction. Independent of the main results in this paper, this observation is interesting because it suggests that simple neighborhood-based methods can be as competitive as, or even better than the advanced methods, which is in accordance with previous studies [3].

The second significant observation is that the approximate, graph sketch based methods can make prediction with very close accuracy to their corresponding exact methods in all the three graph streams. For instance, in the DBLP graph stream, the exact Jaccard coefficient method, Ext-Ja, offers the highest accuracy with 120.57. Its approximate version, App-Ja, provides a similar accuracy result with 104.25, based on the estimation from the MinHash based graph sketches. Similar evidences are witnessed for Ext-CN versus App-CN, and Ext-Adar versus App-Adar, which validate the effectiveness of our graph sketch-based methods. Interestingly, on the Wikipedia graph stream, App-Adar (11.90) yields higher accuracy results than Ext-Adar (10.55). This is mainly because the truncated Adamic-Adar method can effectively preclude high-degree vertices, which are noisy and may bring negative effects for streaming link prediction, especially in large graph streams.

Henceforth, we will mainly focus on the experimental studies of the exact and approximate methods for the target measures, *i.e.*, common neighbor, Jaccard coefficient, and Adamic-Adar.

We further examine the link prediction accuracy of exact methods, including Ext-Ja, Ext-CN, and Ext-Adar, and their approximate, graph sketch based counterparts, including App-Ja, App-CN, and App-Adar, with respect to different algorithm parameters. We mainly report experimental results in the DBLP graph for brevity of exposition, while we have drawn similar conclusions from the other two graph streams. First, we generate a series of graph streams with varied lengths by tuning the number of edges ranging from $400,000$ to *All* including all 1.95 million edges of the stream. We test link prediction methods in graph streams of varied size S, as shown in Figure 1. Specifically, We compare App-Ja with

TABLE IV: Runtime cost *w.r.t.* graph stream size S on DBLP

Runtime Cost (in seconds)	Graph Stream Size S			
	0.4M	0.8M	1.2M	All
App-Ja	0.004	0.04	0.09	0.12
Ext-Ja	21.68	135.16	833.50	2748.40
App-CN	0.007	0.048	0.09	0.395
Ext-CN	18.0	194.31	801.72	2935.22
App-Adar	0.29	4.45	5.05	19.86
Ext-Adar	16.44	260.13	1193.77	3308.26

TABLE V: Runtime cost on Amazon

Runtime Cost			
App-Ja	0.223 sec.	Ext-Ja	>2 hours
App-CN	0.139 sec.	Ext-CN	>2 hours
App-Adar	22.644 sec.	Ext-Adar	>2 hours

Ext-Ja in Figure-1a, App-CN with Ext-CN in Figure-1b, and App-Adar with Ext-Adar in Figure-1c, respectively. It is interesting to note that when the graph stream scales up, the link prediction accuracy of different methods gets improved accordingly, because more graph structural information encoded in the stream can be leveraged for streaming link prediction. More importantly, the prediction accuracy of approximate methods is consistently close to that of their corresponding exact methods. This observation further verifies that our graph sketch based methods can achieve very close high-accuracy link prediction results in comparison with their corresponding exact counterparts defined in the snapshot-based scenario.

We then examine how the training interval ratio, ρ, will affect the link prediction accuracy in graph streams. We choose the DBLP graph stream and tune the value of ρ from 50% up to 90% for different link prediction methods, and the link prediction accuracy results are shown in Figure 2. We note for Jaccard coefficient (Figure-2a), common neighbor (Figure-2b), and Adamic-Adar (Figure-2c), when ρ increases, both exact and approximate methods can yield link prediction results with higher accuracy. The main reason is that more graph structural information in ever-growing training intervals can be leveraged for streaming link prediction. When $\rho = 90\%$, the relative improvement of link prediction accuracy can be at least 200x for all methods. Meanwhile, for different values of ρ, the differences of link prediction accuracy between exact methods and their corresponding approximate methods are very small, especially for the common neighbor and Adamic-Adar measures. This again suggests that the graph sketch based, approximate methods are high-accuracy surrogates of the corresponding exact methods for streaming link prediction in graph streams.

We further examine the link prediction accuracy *w.r.t.* core parameters of the approximate, graph sketch based methods on three graph streams. In Figure-3a, we regulate the number of minwise hash functions, K, for App-Ja. In Figure-3b and Figure-3c, we modify the reservoir size budget, L, for App-CN and App-Adar. We observe that the link prediction accuracy of all three methods is enhanced with larger values of K and L, especially in DBLP and Amazon graph streams. In Wikipedia graph stream, however, the approximate methods are not sensitive to the changes of these parameters, partially

TABLE VI: Runtime cost on Wikipedia

Runtime Cost			
App-Ja	0.297 sec.	Ext-Ja	>2 hours
App-CN	0.385 sec.	Ext-CN	>2 hours
App-Adar	109.35 sec.	Ext-Adar	>2 hours

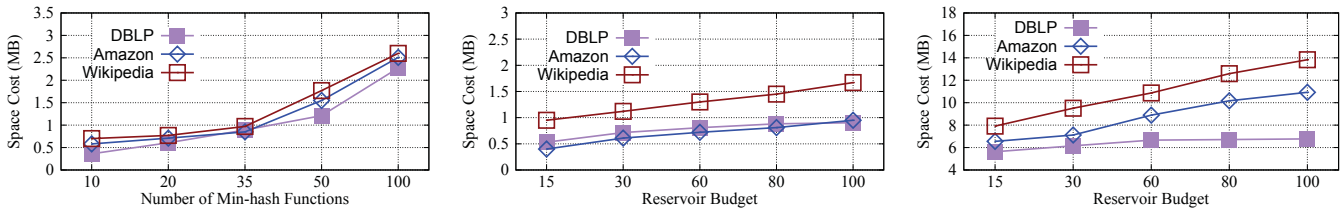
because of the extreme sparsity of the underlying graph, in which little neighborhood information can be used for streaming link prediction for most vertices. It also suggests that the choices of parameters depend on the characteristics of graph streams to be examined.

C.2 Link Prediction Efficiency

An important factor for streaming link prediction is the efficiency of the graph sketch based algorithms. Because real-world graph streams are usually massive and fast evolving all the time, we expect streaming link prediction can be performed efficiently without explicitly storing dynamic graphs in advance. All the existing link prediction methods, however, are snapshot based and not designed specifically for the streaming scenario. Therefore, in order to enable a fair comparison of different link prediction methods, we explicitly allow the storage of the graph stream snapshots for the exact methods. For our sketch-based estimation methods, however, streaming link prediction can be performed *online* without materializing the graph streams. Note that other advanced link prediction methods, such as Katz, PropFlow, Rooted PageRank, ShortestPath Count, and SimRank, have to traverse the underlying graph snapshots, thus rendering them extremely time-consuming (often in multiple hours) and impractical for streaming link prediction. As a result, we mainly focus on the runtime cost based on the target measures (including both exact methods and approximate methods) in different graph streams: Jaccard coefficient, common neighbor, and Adamic-Adar, and the runtime cost reported below is the *overall* time to predict all edges in the Core set of graph streams.

In Table IV, we report the runtime cost of both exact and approximate, graph sketch based link prediction methods by increasing the graph stream size, S, of the DBLP graph stream. All approximate link prediction methods are significantly faster than their exact counterparts, and the speedup can be at least two orders of magnitude. The differences of streaming link prediction performance between approximate and exact methods are especially more significant for larger streams. This is one of the most crucial benefits of our graph sketch based methods. We remark that the differences of streaming link prediction accuracy are small between approximate and exact methods. However, approximate methods are much more efficient than their exact counterparts, and thus are more suitable for streaming link prediction in graph streams.

Table V and VI illustrate the runtime cost of graph sketch based methods and their exact counterparts in Amazon and Wikipedia graph streams, respectively. Note all exact methods cannot accomplish the streaming link prediction task within two hours. However, the approximate methods finish all the link prediction workload almost in real time. One exception is App-Ja, which takes slightly more time on the retrieval of inverted indexes for the computation of truncated Adamic-Adar. As a result, in very large graph streams, the graph sketch



(a) App-Ja: space cost in terms of K (b) App-CN: space cost in terms of L (c) App-Adar: space cost in terms of L

Fig. 4: Space cost of sketch based methods *w.r.t.* algorithm parameters

TABLE VII: Space cost on different graph streams

Space Cost (in MB)	DBLP	Amazon	Wikipeida
App-Ja	2.27	2.51	2.60
App-CN	0.90	0.95	1.67
App-Adar	6.77	10.93	13.83

based methods become the only feasible choice for streaming link prediction.

C.3 Space Cost

As described in the above sections, we make use of concise graph sketches to approximate target measures for streaming link prediction. Such sketches are expected to be space-efficient and easy to maintain and update in main memory. In this experiment, we report the *total memory consumption* of the graph sketches used in each of the approximate link prediction methods. As shown in Table VII, the memory used (in megabytes) by App-Ja, App-CN, and App-Adar is very small for all real-world graph streams. Specifically, the space cost of App-Adar is the highest because we need to maintain both a vertex-biased sampling based reservoir and an inverted index for each vertex in graph streams as its graph sketch. However, the overall space cost is still much affordable even for very large graph streams. For example, on the Wikipedia graph stream, the memory consumed for the graph sketches in App-Adar is only 13.83MB. In comparison with the exact methods and other snapshot-based link prediction methods, which need to explicitly materialize a series of *entire* graph snapshots from the graph streams, our proposed graph sketches are very concise with small space cost.

The space requirement of App-Ja , App-CN, and App-Adar is also closely related to the algorithmic parameters, such as the number of minwise hash functions, K, and the size budget of reservoirs, L. We then examine the space cost of different graph sketch based algorithms with respect to these key parameters in the three real-world graph streams. As shown in Figure 4, by increasing the number K of minwise hash functions for App-Ja, and the budge size L for App-CN and App-Adar, the memory consumption grows slightly in all three graph streams. The total space requirement for graph sketches, however, is still much affordable (within 14MB for the largest Wikipedia graph stream). Therefore, our graph sketch based methods are very space-efficient for streaming link prediction in graph streams.

VIII. Conclusion

In this paper, we considered a new streaming link prediction problem that is defined in large-scale, dynamic graph streams. Graph streams have been pervasive in a wide range of real-world networked applications. They comprise a huge number of fast, transient, and dynamic interactions among entities over time. In this paper, we demonstrated how the elementary, neighborhood-based link prediction measures, such as common neighbor, Jaccard coefficient, and Adamic-Adar, can be accurately estimated and efficiently computed in order to address the streaming link prediction problem in graph streams. We designed the MinHash based graph sketches to estimate Jaccard coefficient, and the vertex-biased sampling based graph sketches to estimate common neighbor and Adamic-Adar with both theoretically guaranteed accuracy and highly accurate empirical results. Our experimental studies demonstrated that the proposed graph sketches and their corresponding estimation algorithms have enabled streaming link prediction in a series of real-world graph streams without significant loss in link prediction accuracy. Moreover, the graph sketch based methods are extremely efficient and cost-effective, and thus can be practically employed for streaming link prediction in real-world graph streams.

References

[1] N. Barbieri, F. Bonchi, and G. Manco, "Who to follow and why: Link prediction with explanations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*, 2014, pp. 1266–1275.

[2] C. Lee, B. Nick, U. Brandes, and P. Cunningham, "Link prediction with social vector clocks," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, 2013, pp. 784–792.

[3] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM'03)*, 2003, pp. 556–559.

[4] J. Menche, A. Sharma, M. Kitsak, S. D. D. Ghiassian, M. Vidal, J. Loscalzo, and A.-L. L. Barabási, "Uncovering disease-disease relationships through the incomplete interactome," *Science*, vol. 347, no. 6224, 2015.

[5] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang, "CoupledLP: Link prediction in coupled networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*, 2015, pp. 199–208.

[6] P. Sarkar, D. Chakrabarti, and M. I. Jordan, "Nonparametric link prediction in dynamic networks." in *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, 2012, pp. 1687–1694.

[7] L. A. Adamic and E. Adar, "Friends and Neighbors on the Web," *Social Networks*, vol. 25, pp. 211–230, 2001.

[8] R. N. Lichtenwalter and N. V. Chawla, "LPmade: Link prediction made easy," *J. Mach. Learn. Res.*, pp. 2489–2492, 2011.

[9] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla, "New perspectives and methods in link prediction," in *Proceedings of the 16th ACM SIGKDD International Conference (KDD'10)*, 2010, pp. 243–252.

[10] Z. Lu, B. Savas, W. Tang, and I. S. Dhillon, "Supervised link prediction using multiple sources," in *Proceedings of the IEEE International Conference on Data Mining (ICDM'10)*, 2010, pp. 923–928.

[11] M. A. Hasan and M. J. Zaki, "A survey of link prediction in social networks," in *Social network data analytics*, 2011, pp. 243–275.

[12] S. Guha, A. McGregor, and D. Tench, "Vertex and hyperedge connectivity in dynamic graph streams," in *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS'15)*, 2015, pp. 241–247.

[13] A. McGregor, "Graph stream algorithms: A survey," *SIGMOD Rec.*, vol. 43, no. 1, pp. 9–20, 2014.

[14] S. Pan and X. Zhu, "Continuous top-k query for graph streams," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, 2012, pp. 2659–2662.

[15] P. Zhao, C. C. Aggarwal, and M. Wang, "gSketch: On query estimation in graph streams," *Proc. VLDB Endow.*, vol. 5, no. 3, pp. 193–204, 2011.

[16] C. C. Aggarwal, "On classification of graph streams," in *SIAM SDM Conference*, 2011, pp. 652–663.

[17] C. C. Aggarwal, Y. Zhao, and P. S. Yu, "Outlier detection in graph streams," in *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE'11)*, 2011, pp. 399–409.

[18] A. Das Sarma, S. Gollapudi, and R. Panigrahy, "Estimating PageRank on graph streams," in *Proceedings of the Twenty-seventh ACM Symposium on Principles of Database Systems (PODS'08)*, 2008, pp. 69–78.

[19] C. C. Aggarwal, "Mining text and social streams: A review," *SIGKDD Explor. Newsl.*, vol. 15, no. 2, pp. 9–19, 2014.

[20] J. Zhang, P. S. Yu, and Z.-H. Zhou, "Meta-path based multi-network collective link prediction," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*, 2014, pp. 1286–1295.

[21] J. Tang, T. Lou, and J. Kleinberg, "Inferring social ties across heterogenous networks," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM'12)*, 2012, pp. 743–752.

[22] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and recommending links in social networks," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM'11)*, 2011, pp. 635–644.

[23] C. Aggarwal and K. Subbian, "Evolutionary network analysis: A survey," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–36, 2014.

[24] C. Budak, D. Agrawal, and A. El Abbadi, "Structural trend analysis for online social networks," *Proc. VLDB Endow.*, vol. 4, no. 10, pp. 646–656, 2011.

[25] R. V. Oliveira, B. Zhang, and L. Zhang, "Observing the evolution of internet as topology," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'07)*, 2007, pp. 313–324.

[26] W. Yu, C. C. Aggarwal, S. Ma, and H. Wang, "On anomalous hotspot discovery in graph streams," in *2013 IEEE 13th International Conference on Data Mining (ICDM'13)*, 2013, pp. 1271–1276.

[27] Y. Sun, J. Han, C. C. Aggarwal, and N. V. Chawla, "When will it happen? — relationship prediction in heterogeneous information networks," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM'12)*, 2012, pp. 663–672.

[28] P. Sarkar, D. Chakrabarti, and A. W. Moore, "Theoretical justification of popular link prediction heuristics," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*, 2011, pp. 2722–2727.

[29] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC'98)*, 1998, pp. 327–336.

[30] C. C. Aggarwal, T. Huang, and G.-J. Qi, "Link prediction across networks by biased cross-network sampling," in *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE'13)*, 2013, pp. 793–804.

[31] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[32] F. Gao, K. Musial, C. Cooper, and S. Tsoka, "Link prediction methods and their accuracy for different social networks and network metrics," *Sci. Program.*, vol. 2015, pp. 1:1–1:1, 2015.

[33] L. Lv and T. Zhou, "Link prediction in complex networks: A survey," *Physica A*, vol. 390, no. 6, pp. 1150–1170, 2011.

[34] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu, "Scalable proximity estimation and link prediction in online social networks," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC'09)*, 2009, pp. 322–335.

[35] H. Kashima and N. Abe, "A parameterized probabilistic model of network evolution for supervised link prediction," in *Proceedings of the Sixth International Conference on Data Mining (ICDM'06)*, 2006, pp. 340–349.

[36] J. Kunegis and A. Lommatzsch, "Learning spectral graph transformations for link prediction," in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*, 2009, pp. 561–568.

[37] R. Chitnis, G. Cormode, M. Hajiaghayi, and M. Monemizadeh, "Parameterized streaming: Maximal matching and vertex cover," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, 2015, pp. 1234–1251.

[38] Y. Lim and U. Kang, "Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*, 2015, pp. 685–694.

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

[40] C. Song, T. Ge, C. Chen, and J. Wang, "Event pattern matching over graph streams," *Proc. VLDB Endow.*, vol. 8, no. 4, pp. 413–424, 2014.

[41] S. Guha and A. McGregor, "Graph synopses, sketches, and streams: A survey," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2030–2031, 2012.

[42] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "On graph problems in a semi-streaming model," *Theor. Comput. Sci.*, vol. 348, no. 2, pp. 207–216, 2005.

[43] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[44] C. C. Aggarwal, Y. Xie, and P. S. Yu, "Towards community detection in locally heterogeneous networks," in *SIAM SDM Conference (SDM'11)*, 2011, pp. 391–402.

[45] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[46] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, 1999, pp. 251–262.