Compact and Scalable Graph Neighborhood Sketching

Takuya Akiba
National Institute of Informatics
JST PRESTO & CIT STAIR Lab
takiba@nii.ac.jp

Yosuke Yano[*]
National Institute of Informatics
JST PRESTO
yyano@nii.ac.jp

ABSTRACT

The *all-distances sketch (ADS)* has recently emerged as a promising paradigm of graph neighborhood sketching. An ADS is a probabilistic data structure that is defined for each vertex of a graph. ADSs facilitate accurate estimation of many useful indicators for network analysis with the guarantee of accuracy, and the ADSs for all the vertices in a graph can be computed in near-linear time. Because of these useful properties, ADS has attracted considerable attention. However, a critical drawback of ADS is its space requirement, which tends to be much larger than that of the graph itself. In the present study, we address this issue by designing a new graph sketching scheme, namely, *sketch retrieval shortcuts (SRS)*. Although SRSs are more space-efficient than ADSs by an order of magnitude, an ADS of any vertex can be quickly retrieved from the SRSs. The retrieved ADSs can be used to estimate the aforementioned indicators in exactly the same manner as with plain ADSs, inheriting the same accuracy guarantee. Our experiments on real-world networks demonstrate the usefulness of SRSs as a practical back-end of large-scale graph data mining.

Keywords

Graphs; Min-hash sketches; All-distances sketches

1. INTRODUCTION

Many types of indicators and measures play key roles as building blocks for graph analysis and mining. *Vertex centralities* constitute one of the the most fundamental classes of indicators. They are defined for a vertex to measure the relative importance of every other vertex [9, 8, 12]. Other common classes include *vertex similarities*, which are defined for a pair of vertices to estimate their similarity or strength of relevance [23, 13]. Graph properties, which are defined for a whole graph, have also attracted considerable attention for understanding the underlying mechanisms and developing realistic models [21, 4, 6].

Computing these measures in an ad-hoc manner is almost always unrealistic for large-scale graphs. Let us take vertex centralities as an example. Most types of centralities require time at least proportional to the graph size. Moreover, centralities are usually employed to compare vertices in a graph; thus, centrality values are necessary for a subset of vertices, or sometimes, even for all vertices.

Therefore, algorithmic frameworks called *sketching* or *indexing* are employed to compute these indicators and properties [2, 28, 6, 25, 12, 19, 11, 1, 3]. In such frameworks, we first compute a data structure called a *sketch* or an *index* from a graph, which can substantially accelerate the computation of these useful indicators. This not only results in better scalability of graph analysis but also makes it possible to use such indicators in applications that require quick interactive response, such as drill-down analytics and network-aware search systems [27, 24].

The *all-distances sketch (ADS)* [11, 12] has recently emerged as a paradigm of graph neighborhood sampling. An ADS is defined for a vertex, and is an extension of the min-hash sketch [18, 11] (see Section 3.1 for the definition). The ADSs for all the vertices in a graph can be computed efficiently, and once they are obtained, the values of the many useful indicators can be efficiently and accurately estimated. To the best of our knowledge, ADS is the only sketching paradigm that combines the following three properties.

- **Multi-Functionality**: Using ADSs, we can estimate many types of useful indicators such as shortest-path distance, closeness centrality, closeness similarity, and reverse ranking (see Section 3.2).
- **Guaranteed Accuracy**: Estimation using ADSs is quite accurate and has tight theoretical error bounds.
- **Scalability** (in theory): The ADSs for all the vertices in a graph can be computed in near-linear time, and their total size is also near-linear to the number of vertices.

Because of these useful properties, ADS has attracted considerable attraction. However, in practice, ADS has turned out to be not as scalable as expected theoretically. The main drawback is its space requirement. An ADS is an array of length approximately $k \ln n$, where n is the number of vertices and k is an accuracy-space trade-off parameter. In order to achieve accurate estimation, k is set to a value of the order of tens or hundreds. Thus, the length of an ADS is of the order of hundreds or thousands. Therefore,

[*]Y. Yano's current affiliation: Recruit Holdings Co., Ltd.

KDD '16, August 13 - 17, 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4232-2/16/08...$15.00
DOI: http://dx.doi.org/10.1145/2939672.2939762

ADSs require more space than the graph itself, which is unreasonable for large graphs. Moreover, because there are few repetitions in an ADS, standard compression methods, such as the LZ family, do not work well, as discussed in Section 8.2. Thus, substantial space reduction techniques are required to make ADS practical.

1.1 Contributions

To address the above-mentioned issue, we propose a new graph sketching data structure, namely, *sketch retrieval shortcuts (SRS)*. Instead of ADSs, we propose that SRSs be computed for the all vertices in a graph. Then, using the SRSs, an ADS of a vertex can be quickly retrieved. SRSs can be computed with acceptable additional computation cost in comparison to ADSs. Although SRSs are an order of magnitude smaller than ADSs, once we obtain them, an ADS of a vertex can be quickly retrieved from the SRSs; thus, the various aforementioned measures and indicators can be estimated in exactly the same manner as with ADSs, inheriting the same accuracy guarantee.

The idea underlying the definition of SRSs is to consider the ADSs of a graph as another weighted graph, and sparsify it such that an ADS of a vertex can still be quickly retrieved by a *restricted* search on it. The retrieval algorithm is designed such that it visits only vertices in an ADS; thus, an ADS is efficiently retrieved in time that is almost linear to the ADS size. SRSs are carefully defined to ensure that an ADS of any vertex can be correctly retrieved using this procedure.

We propose two types of construction algorithms: the first constructs SRSs from ADSs, whereas the second constructs SRSs directly from graphs. The former is faster, whereas the latter requires less working memory and provides a flexible trade-off between time and space consumption.

Our experiments on real-world networks demonstrate the usefulness of SRSs as a practical back-end of large-scale graph data mining. Specifically, we can confirm that *(1)* SRSs can be constructed for very large graphs with millions of vertices and hundreds of millions of edges, *(2)* SRSs are an order of magnitude smaller than ADSs, and *(3)* ADS retrieval is sufficiently quick, i.e., the retrieval time is of the order of milliseconds.

Organization. The remainder of this paper is organized as follows. Section 2 provides the relevant definitions and notations. Section 3 reviews the definition and usage of ADSs. Section 4 presents the definition and crucial properties of our SRSs. Sections 5 and 6 propose efficient SRS construction algorithms with different performance characteristics. Section 7 proposes techniques to further improve the practicality of SRSs. Section 8 presents our experimental results. Section 9 concludes the paper.

2. PRELIMINARIES

2.1 Notations

Table 1 lists the frequently used notations in the paper. Let $G = (V, E)$ be a graph where V and E are a vertex set and an edge set, respectively. We denote $|V|$ and $|E|$ by n and m, respectively. For generality, we assume that G is a weighted and directed graph. We denote the length of an edge e by $\ell(e)$. Further, we assume that $\ell(e) > 0$ for any $e \in E$. Our discussions can be applied to unweighted graphs

Table 1: Frequently used notations.

Notation	Description
$G = (V, E)$	A graph.
n, m	Numbers of vertices and edges in G.
$\ell(u, v)$	Length of an edge $(u, v) \in E$.
$d(u, v)$	Distance from u to v.
$P(u, v)$	Vertices on the shortest paths from u to v.
k	Accuracy-space trade-off parameter.
$r(u)$	Rank of vertex u.
$k^{\text{th}}_r(S)$	k-th smallest rank in vertex set S.
$\pi(u, v)$	Threshold rank of vertex v against vertex u.
$\mathcal{A}(u)$	All-distances sketch (ADS) of vertex u.
$\mathcal{B}(u)$	Sketch retrieval shortcuts (SRS) of vertex u.

by introducing a constant length function, e.g., $\ell(e) = 1$ for any $e \in E$. Let $d(u, v)$ denote the distance from u to v. Let $P(u, v)$ denote the vertices on the shortest paths from u to v, i.e., $P(u, v) = \{w \in V \mid d(u, w) + d(w, v) = d(u, v)\}$.

We always compare tuples by lexicographic order. For example, $(a, b) < (c, d)$ if and only if *(1)* $a < c$ or *(2)* $a = c$ and $b < d$. We also use this lexicographic comparison in sorting and priority queues, unless otherwise stated.

2.2 Distance Tie-Breaking

The strict definition of ADSs assumes that distances are unique [12]. In theory, an arbitrary tie-breaking rule can be employed. However, when closely looking at algorithms and implementation, the selection of the tie-breaking rule is highly important for simplicity and performance.

In this paper, we propose that the following tie-breaking rule be used. We conduct lexicographic comparison using the pair of *(1)* the original distance and *(2)* the destination vertex ID. In other words, v is *closer* to u than w if and only if $(d(u, v), v) < (d(u, w), w)$. For the definition of ADSs, we do not need to break all the ties among any pairs of vertices; instead, we just need to break the ties among the distances from a single vertex to other vertices. Therefore, this simple tie-breaking rule works, and we find that this rule can be effectively accommodated in our algorithms.

In this paper, this tie-breaking rule is explicit: the definition of "distance" $d(u, v)$ remains unchanged, and we explicitly conduct this tie-breaking as part of our algorithm. On the other hand, our algorithms and data structures are essentially compatible with any other tie-breaking rule.

3. ALL-DISTANCES SKETCHES

In this section, we review the all-distances sketch (ADS). We first explain the definition and basic properties, and then introduce the examples of the useful indicators that can be accurately estimated by ADSs.

3.1 Definition and Construction

All-distances sketches (ADSs) are defined with respect to an integer parameter k and a random rank assignment to vertices. The parameter k gives the trade-off between sketch size and estimation precision. Throughout this paper, we use a random function $r : V \to [0, 1]$ as the rank function, where $r(v) \sim U[0, 1]$ for any vertex v. In other words, $r(v)$ is independently drawn from the uniform distribution on $[0, 1]$.

For $u, v \in V$, we define $N(u, v)$ as the set of vertices that are closer to u than v, i.e., $N(u, v) = \{w \in V \mid (d(u, w), w) < (d(u, v), v)\}$. For a vertex subset $X \subseteq V$, we define the function $k_r^{\text{th}}(X)$ as the k-th smallest value of ranks for vertices in X. If $|X| < k$, we define $k_r^{\text{th}}(X) = 1$. For vertices u and v, we define *threshold rank* $\pi(u, v)$ as $\pi(u, v) = k_r^{\text{th}}(N(u, v))$, which denotes the k-th smallest value of ranks for vertices that are closer to u than v. Using $\pi(u, v)$, ADSs are defined as follows[1].

Definition 3.1 (ADS [12])**:** *The* all-distances sketch (ADS) *of a vertex u is defined as $\mathcal{A}(u) = \{(v, \delta_{uv}) \mid v \in V, r(v) < \pi(u, v)\}$, where $\delta_{uv} = d(u, v)$.*

In this paper, we use the singular form, i.e., "an ADS," to indicate the sketch of a single vertex (i.e., $\mathcal{A}(v)$ for a vertex v), and the plural form, i.e., "ADSs," sometimes indicates the whole set of sketches for all the vertices in a graph (i.e., $\{\mathcal{A}(v)\}_{v \in V}$). For directed graphs, we distinguish between the *forward ADS* $\overrightarrow{\mathcal{A}}(u)$ and the *backward ADS* $\overleftarrow{\mathcal{A}}(u)$. For the forward ADS $\overrightarrow{\mathcal{A}}(u)$, we use the distances *from* u (as in the above definition), and, for the backward ADS $\overleftarrow{\mathcal{A}}(u)$, we use the distances *to* u. Note that, on an undirected graph, the forward and backward ADSs are exactly the same.

3.1.1 Size

The size of an ADS can be calculated as follows.

Lemma 3.2 (ADS Size [12])**:** *For a vertex u, let n_u be the number of reachable vertices from u, and $H(i)$ be the i-th harmonic number. The expected size of $\mathcal{A}(u)$ is $k(1 + H(n_u) - H(k))$.*

As $n_u \leq n$ and $H(n) = O(\log n)$, the expected size of $\mathcal{A}(u)$ is $O(k \log n)$. Therefore, the expected total storage usage of the ADSs for all vertices is $O(nk \log n)$.

3.1.2 Construction

There are two efficient approaches for computing ADSs from a graph. The first is the *pruned-search approach* [11], which conducts a pruned version of shortest-path searches from all vertices in ascending order of their ranks. The other approach is the *scan-and-merge approach* [25, 6], which repeats iterations until convergence. In each iteration, the ADS of each vertex is grown by scanning all outgoing edges and merging the ADSs of the neighbors into it.

3.2 Estimation Using ADSs

Once we obtain ADSs, many types of graph properties and features can be efficiently estimated with the guarantee of accuracy, as follows.

Neighborhood Function [11, 16, 12]. The neighborhood function $n_\delta(v)$ is the number of vertices that can be reached from v within distance δ. The value of $n_\delta(v)$ can be estimated from ADS $\mathcal{A}(v)$, and its coefficient of variation is bounded by $1/\sqrt{2(k-1)}$.

Shortest-Path Distance [13, 26]. For $u, v \in V$, distance $d(u, v)$ can be estimated by ADSs $\mathcal{A}(u)$ and $\mathcal{A}(v)$. It is $O(\log n)$-approximation for constant k, and $(2a-1)$-approximation when $k = n^{1/a}$ (for some $a \geq 1$).

Closeness Centrality [12]. Closeness centrality is one of the most fundamental measures of vertex importance. it is defined for a vertex u, *distance decay function* α, and vertex weight function β, as $C_{\alpha,\beta}(u) = \frac{1}{n}\sum_{v \in V} \alpha(d(u, v))\beta(v)$. Using ADS $\mathcal{A}(u)$, the estimation of $C_{\alpha,\beta}(u)$ can be obtained with coefficient of variation bounded by $1/\sqrt{2(k-1)}$.

Closeness Similarity [13]. Closeness similarity is a proximity measure that is used to estimate the strength of relevance of a pair of vertices. It is based on the similarity of their distances to all other nodes. For two vertices $u, v \in V$, the closeness similarity between u and v can be estimated using ADSs $\mathcal{A}(u)$ and $\mathcal{A}(v)$, and the root of its expected square error is guaranteed as $O(1/\sqrt{k})$.

Average Distance and Effective Diameter [6]. Average distance and effective diameter are fundamental properties of a graph, and of considerable interest because of their relation to so-called *small world phenomenon*. They can be accurately estimated with confidence intervals.

Reverse Raking and Nearest Neighbors [10]. Reverse ranking measures the relevance of vertex u to vertex v by the number of vertices that are closer to v than u. Based on the neighborhood function estimation, approximated reverse nearest neighbors (rNNs) of arbitrary size can be efficiently obtained.

Continuous-Time Influence [17, 15, 14]. The continuous-time influence model is an influence propagation model with the time decay property. The expected influence of vertex set S can be estimated using a *combined ADS* $\mathcal{A}(S)$, which can be computed from ADSs $\mathcal{A}(v)$ for all $v \in S$. Its coefficient of variation is bounded by $1/\sqrt{2(k-1)}$. ADSs can also facilitate efficient influence maximization for such a model.

4. SKETCH RETRIEVAL SHORTCUTS

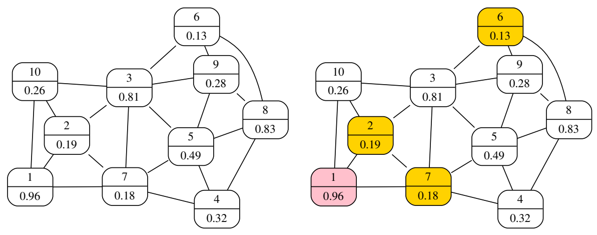
In this section, we present our new graph sketching scheme, namely, *sketch retrieval shortcuts (SRS)*. First, we explain the underlying idea. Then, we mathematically define SRSs. Finally, we present the retrieval algorithm that quickly instantiates an ADS from SRSs.

4.1 Underlying Idea

Before explaining the idea underlying SRSs, we discuss the difficulty in ad-hoc ADS retrieval, i.e., we cannot obtain an ADS quickly from a graph without any preprocessing. Figure 1a shows a graph. Each vertex is drawn as a box, where its ID and rank are denoted in the upper and lower parts, respectively. In, Figure 1b the vertices in the ADS $\mathcal{A}(1)$ are highlighted ($k = 2$). We observe that in this example, vertex 6 is *isolated* from the other vertices in $\mathcal{A}(1)$, in the sense that we cannot reach vertex 6 from vertex 1 without passing vertices that are not in the ADS. This suggests that without any preprocessing, we cannot avoid a full search on the graph to collect all the vertices that are contained in an ADS. A full search requires time at least linear to the graph size, and is thus too time-consuming for large-scale network analysis. On the other hand, full precomputation of ADSs for all vertices requires too much storage. Therefore, we aim to achieve a good trade-off such that an ADS can be quickly retrieved with less precomputed data.

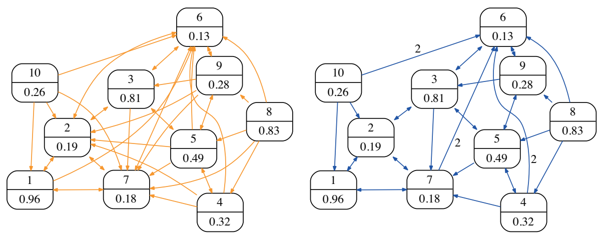
The general idea underlying SRSs is to overcome the need for a full search on a graph and realize the retrieval of an ADS by a smaller search on another graph. Toward this end, we consider a search that starts from a specified vertex u and is allowed to visit only vertices in the ADS $\mathcal{A}(u)$. As

[1] The definition of an ADS has some variations, though, in this paper, we focus on the *bottom-k ADS* [12]. Cohen proved that they have similar estimation ability, but, among them, the bottom-k ADSs are slightly better [12].



(a) A graph. (b) The vertices in $\mathcal{A}(1)$.

Figure 1: Examples of a graph and an ADS ($k = 2$).



(a) ADSs as a graph. (b) SRSs as a graph.

Figure 2: ADSs and SRSs as graphs.

the number of vertices in an ADS is $O(k \log n)$ and usually $k \log n$ is much smaller than n, we expect that this search is much more efficient than a full search.

To realize such searches for ADS retrieval, we first take a different view of ADSs. We consider the ADSs of the graph as another weighted graph, where each entry $(v, \delta_{uv}) \in \mathcal{A}(u)$ corresponds to an edge from u to v with weight δ_{uv} (Figure 2a). In this "graph", obviously, the vertices in an ADS $\mathcal{A}(u)$ are directly connected from vertex u; thus, the above-mentioned search from u can successfully visit all these vertices (with only 1 hop).

Intuitively, for this purpose, this ADS graph has a considerable number of redundant edges, and herein lies the concept of our SRSs. In general, SRSs are yet another weighted graph whose edges are selected from the ADS graph such that the above-mentioned search can always succeed from any vertex (Figure 2b). In other words, SRSs are defined such that, when considering a subgraph induced by the vertex set of $\mathcal{A}(u)$, the subgraph is connected, and the distances from u to all these vertices remain unchanged (Figure 3). We call this the *reachability property*.

This might seem like an intuitive and simple idea. However, to guarantee the above property, the definitions and algorithms need to be designed carefully. Indeed, there is a common misunderstanding about the definition of SRSs, which is explained with a counterexample in Appendix A.

4.2 Definition and Properties

4.2.1 Definition

Now, we mathematically define SRSs. As with ADSs, SRSs are defined for a graph $G = (V, E)$, a trade-off parameter k, and a random rank function $r : V \to [0, 1]$.

Let Δ be the set of distances between any pair of vertices, i.e. $\Delta = \{d(u, v) \mid u, v \in V\}$. We assume that $\Delta = \{d_0, d_1, \ldots, d_h\}$, where $d_0 < d_1 < \cdots < d_h$. Note that $d_0 = 0$ and d_h corresponds to the diameter of the graph. We define $\mathcal{B}_i$ $(i = 0, 1, \ldots, h)$ and $\mathcal{C}_i, \mathcal{D}_i$ $(i = 1, 2, \ldots, h)$ recursively as follows.

- $\mathcal{B}_0(u) = \emptyset$ and $\mathcal{B}_i(u) = \mathcal{B}_{i-1}(u) \cup \mathcal{D}_i(u)$ for $i > 0$.
- $\mathcal{C}_i(u, v) = \{w \in P(u, v) \mid w \in \mathcal{A}(u), v \in \mathcal{B}_{i-1}(w)\}$.
- $\mathcal{D}_i(u) = \{(v, \delta_{uv}) \in \mathcal{A}(u) \mid \delta_{uv} = d_i, \mathcal{C}_i(u, v) = \emptyset\}$.

Intuitively, $\mathcal{C}_i(u, v)$ corresponds to possible transit vertices from u to v for the reachability property, and $\mathcal{B}_i(u)$ contains ADS entries with distance at most d_i that have no such transit vertices. The SRSs are defined as follows.

Definition 4.1 (SRS)**:** *The* sketch retrieval shortcuts (SRS) *of vertex u is $\mathcal{B}_h(u)$.*

Hereafter, for simplicity, we simply denote $\mathcal{B}_h(u)$ by $\mathcal{B}(u)$. Similarly to ADSs, we use the singular form to indicate the sketch of a single vertex, and the plural form to indicate the whole set of sketches for all vertices in a graph. For directed graphs, as with ADSs, we distinguish between the *forward SRS* $\overrightarrow{\mathcal{B}}(v)$ and the *backward SRS* $\overleftarrow{\mathcal{B}}(v)$. For the forward SRS, we use the forward ADS $\overrightarrow{\mathcal{A}}(u)$ for defining $\mathcal{C}_i$. The backward SRS is similarly defined on backward ADSs.

4.2.2 Reachability Property

The following lemma mathematically states the property discussed in Section 4.1.

Lemma 4.2: *If $v \in \mathcal{A}(u)$, there exists a sequence of vertices $\mathcal{W}(u, v) = (w_1, w_2, \ldots, w_p)$ such that, (1) $w_1 = u$, $w_p = v$, (2) $w_i \in A(u)$ for all $1 \le i \le p$, (3) $w_{i+1} \in \mathcal{B}(w_i)$ for all $1 \le i \le p - 1$, and (4) $\sum_{i=1}^{p-1} d(w_i, w_{i+1}) = d(u, v)$.*

Proof. We prove the lemma by mathematical induction on the distance $d(u, v)$. Since $\mathcal{W}(u, u) = (u)$ satisfies the above conditions, it is true for distance zero. Now, we assume that it holds for pairs within distance d_{i-1} and prove it also holds for pairs with distance d_i. Let u, v be a pair of vertices such that $v \in \mathcal{A}(u)$ and $d(u, v) = d_i$. If $v \in \mathcal{B}(u)$, then $\mathcal{W}(u, v) = (u, v)$ satisfies the conditions. Otherwise, $v \notin \mathcal{D}_i(u)$, and thus, $\mathcal{C}_i(u, v) \neq \emptyset$. Let $w \in \mathcal{C}_i(u, v)$. Then, $\mathcal{W}(u, v)$ can be obtained by appending v to $\mathcal{W}(u, w)$. $\square$

4.2.3 Theoretical Size Upper Bound

By definition, $\mathcal{B}(u)$ is a subset of $\mathcal{A}(u)$ for any vertex u. Therefore, the upper bound of the expected size is as below.

Lemma 4.3: *The expected size of $\mathcal{B}(u)$ is $O(k \log n)$.*

In total, the theoretical upper bound of the expected space usage of the all SRSs is $O(nk \log n)$. This bound is tight, as it is $\Theta(k \log n)$ on some pathological cases such as a clique. In practice, SRSs are much smaller than ADSs, as seen in our experiments.

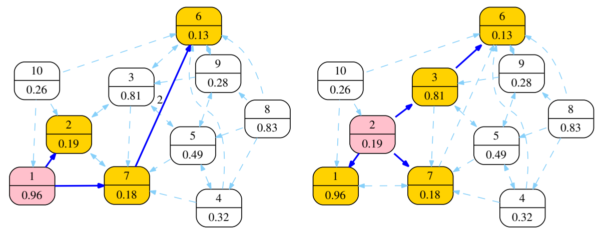
4.3 Retrieving an ADS from SRSs

The main feature of SRSs is to enable quick retrieval of an ADS of any vertex. By obtaining an ADS, the various graph properties can be estimated in exactly the same manner as with a plain ADS as mentioned in Section 3.2. The retrieval algorithm Retrieve-ADS is explained as Algorithm 1. Note that, unless otherwise stated, sorting and priority queues on tuples use the ascending order in lexicographic comparison.

To retrieve the ADS of vertex u, in general, the retrieval algorithm conducts a pruned version of Dijkstra's algorithm

Algorithm 1: Retrieving the ADS of vertex u

```
Procedure Retrieve-ADS($\mathcal{B}, u, k$)
1   $A \leftarrow$ an empty all-distances sketch;
2   $Q \leftarrow$ a priority queue with only one element $(0, u)$;
3   while $Q$ is not empty do
4       $(\delta_{uv}, v) \leftarrow Q$.Pop;
5       if $v \notin A$ and $r(v) < \pi(u, v)$ then
6           Add $(v, \delta_{uv})$ to $A$;
7           for all $(\delta_{vw}, w) \in \mathcal{B}(v)$ do
8               $Q$.Push$(\delta_{uv} + \delta_{vw}, w)$;
9   return $A$;
```



(a) Retrieving $\mathcal{A}(1)$. (b) Retrieving $\mathcal{A}(2)$.

Figure 3: Retrieval of ADSs from SRSs.

from u on the SRSs such that it only visits vertices in $\mathcal{A}(u)$ (Figure 3). We start with an empty sketch A, and entries are added in increasing order of distances to build A. For each visited vertex v with distance δ_{uv}, we check whether (v, δ_{uv}) is *necessary* for A, i.e., whether (v, δ_{vu}) should be included in $\mathcal{A}(u)$ (Line 5). As A contains all the entries in $\mathcal{A}(v)$ with smaller distances at this point, $\pi(u, v)$ can be computed from A. We can use a priority queue that manages the top-k ranks in A to obtain $\pi(u, v)$ quickly. If v is unnecessary for A, we prune the search and do nothing. Otherwise, we add v to A and expand the search by entries in $\mathcal{B}(v)$. Note that the lexicographic comparison in the priority queue substantiates the distance tie-breaking rule discussed in Section 2.2. Lemmas 4.4 and 4.5 state the correctness and complexities of the algorithm.

Lemma 4.4: *Retrieve-ADS returns $\mathcal{A}(u)$.*

Proof Sketch. This is almost immediate from Lemma 4.2. From the lemma, any vertex in $\mathcal{A}(u)$ can be reached with correct distance on SRSs through other vertices in $\mathcal{A}(u)$. □

Lemma 4.5: *Retrieve-ADS runs in $O(k^2 \log^2 n \log(k \log n))$ expected time and $O(k^2 \log^2 n)$ expected space.*

Proof Sketch. Line 8 is executed for $|\mathcal{B}(v)|$ time for each vertex $v \in \mathcal{A}(u)$, and the expected size of $\mathcal{B}(u)$ and $\mathcal{A}(v)$ is $O(k \log n)$. The priority queue takes $O(\log(k \log n))$ time to push an element. □

We stress that the retrieval time only has a logarithmic dependence on the graph size. Therefore, ADSs can be efficiently retrieved even on very large graphs.

5. CONSTRUCTION VIA ADS

Then, we study construction algorithms for SRSs. In this section, we design algorithms that construct SRSs from ADSs. First, we explain the basic algorithm with our *retrieve-and-verify* principle, which is common to all of our construction algorithms. Second, we speed up the algorithm by introducing *eager entry generation.*

Algorithm 2: Constructing SRSs from ADSs (naive).

```
Procedure Construct-SRS-Naive($G = (V, E), \mathcal{A}, k$)
1   $B[u] \leftarrow \emptyset$ for all $u \in V$;
2   $T \leftarrow \{(\delta_{uv}, v, u) \mid (v, \delta_{uv}) \in \mathcal{A}(u)\}$;
3   Sort $T$;
4   for $(\delta_{uv}, v, u) \in T$ do
5       $A \leftarrow$ Retrieve-ADS$(B, u, k)$;
6       if $(v, \delta_{uv}) \notin A$ then Add $(v, \delta_{uv})$ to $B[u]$;
7   return $B$;
```

5.1 Basic Algorithm

We know that SRSs are recursively defined, and the inclusion of pairs with longer distances depends on pairs with shorter distances. Therefore, in general, we need to construct SRSs in ascending order of distances. Algorithm 2 outlines algorithm Construct-SRS-Naive, our first construction algorithm. It examines the entries of all the ADSs in ascending order of distances. We check each ADS entry to determine whether it is necessary for the SRSs, and if so, we add it.

The tricky part of this algorithm is that, even for construction, we use the retrieval algorithm Retrieve-ADS. At Line 5, algorithm Retrieve-ADS retrieves an "ADS" A from the current incomplete SRSs B. As the current SRSs are incomplete, A does not always match the correct ADS $\mathcal{A}(u)$. However, interestingly, we can use A to check whether entry (v, δ_{uv}) is necessary for the SRS of u as follows.

Lemma 5.1: *At Line 5 in Algorithm 2, assuming $B[w]$ contains SRS entries with distances less than δ_{uv} for any $w \in V$, $(v, \delta_{uv}) \notin A$ if and only if $(v, \delta_{uv}) \in \mathcal{B}(u)$.*

Proof Sketch. Let $\delta_{uv} = d_i$. If $(v, \delta_{uv}) \in A$, then $\mathcal{C}_i(u, v) \neq \emptyset$, and thus $(v, \delta_{uv}) \notin \mathcal{B}(u)$. Otherwise, from Lemma 4.2, (v, δ_{uv}) must be in $\mathcal{B}(u)$. □

As a corollary of this lemma, we obtain the correctness of the algorithm with mathematical induction on the distance.

Corollary 5.2: *Construct-SRS-Naive returns $\{\mathcal{B}(u)\}_{u \in V}$.*

The time and space complexities of this algorithm can be bounded as follows.

Lemma 5.3: *Construct-SRS-Naive runs in $O(nk^3 \log^3 n \log(k \log n))$ expected time and $O(nk \log n)$ expected space.*

Proof Sketch. The ADSs contain $O(nk \log n)$ entries, and Retrieve-ADS is called for each entry. □

5.2 Eager Entry Generation

Algorithm 3 describes another SRS construction algorithm called Construct-SRS-Fast that shares the general idea, but is faster than the previous algorithm. In the previous algorithm, repeated call of Retrieve-ADS was the bottleneck. The high-level difference from the previous algorithm is that, instead of calling Retrieve-ADS, we eagerly generate what can be retrieved from the current incomplete SRSs, immediately after adding an entry to a SRS.

Algorithm 3: Constructing SRSs from ADSs (fast).

```
Procedure Construct-SRS-Fast($G = (V, E), \mathcal{A}, k$)
1    $B[u] \leftarrow \mathcal{A}(u)$ for all $u \in V$;
2    $T \leftarrow \{(\delta_{uv}, v, u) \mid (v, \delta_{uv}) \in \mathcal{A}(u)\}$;
3    Sort $T$;
4    for $(\delta_{uv}, v, u) \in T$ do
5        if $(v, \delta_{uv}) \notin B[u]$ then continue;
6        for $x \in V$ s.t. $(u, \delta_{xu}) \in \mathcal{A}[x]$ do
7            Remove $(v, \delta_{xu} + \delta_{uv})$ from $B[x]$ if it exists;
8    return $B$;
```

The eager generation corresponds to Lines 6–7. As the inclusion of entry (v, δ_{uv}) in SRS $\mathcal{B}(u)$ has been determined, we generate the ADS entries that can be retrieved through that SRS entry. Compared to the previous algorithm, the time complexity is improved as follows.

Lemma 5.4: *Construct-SRS-Fast runs in $O(nk \log n \log(nk \log n) + |\mathcal{B}|\, k \log n)$ expected time and $O(nk \log n)$ expected space, where $|\mathcal{B}|$ is the total size of whole SRSs.*

Proof Sketch. We first sort the whole $O(nk \log n)$ ADS entries, and then, for each SRS entry, we traverse $O(k \log n)$ ADS entries. □

6. DIRECT CONSTRUCTION

Finally, we present a direct construction algorithm for SRSs. Previous algorithms explicitly instantiate ADSs in memory once, the required working space for which may be too large. The following direct algorithm achieves a trades-off between the time consumption and the space efficiency by building SRSs and *virtually* constructing ADSs simultaneously. We start by explaining the basic form, and then make the trade-off more flexible by introducing the *partial ADS caching technique.*

6.1 Basic Algorithm

For simplicity, we first assume that G is an unweighted graph. Our direct algorithm Construct-SRS-Direct is described as Algorithm 4. At a high level, this algorithm combines the retrieve-and-verify principal of the previous indirect SRS construction algorithms with scan-and-merge ADS construction algorithms.

We generally repeat iterations until convergence, where each iteration merges ADSs of neighbors into the SRS of each vertex. Specifically, in the i-th iteration, the SRSs are grown so that the retrieved ADS of any vertex is correct with regard to entries with distances less than or equal to i. In i-th iteration, for each vertex u, we first retrieve the incomplete "ADSs" of its neighbors by Retrieve-ADS, and extract the entries with distance $i - 1$ as possible entries for u with distance i (Lines 5–9). Then, we again use Retrieve-ADS to retrieve an incomplete "ADS" A of vertex u to judge whether *(1)* the entry is an actual ADS entry (Line 13), and *(2)* the entry is necessary as an SRS entry (Line 14).

The correctness and complexities of the algorithm are as follows. This space consumption is almost just for the graph and SRSs themselves and essentially minimal.

Lemma 6.1: *Construct-SRS-Direct returns $\{\mathcal{B}(u)\}_{u \in V}$.*

Proof Sketch. By mathematical induction on the distance δ, we can prove that B always matches the SRSs for entries with distances at most δ. From the initial condition, it holds for $\delta = 0$. Assuming it holds for distances less than δ, retrieved ADSs are also correct for entries with distances less than δ; thus, all ADS entries with distance δ are obtained from them. Then, similarly to Lemma 5.1, we can prove that B is also updated for distance δ. □

Algorithm 4: Constructing SRSs directly.

```
Procedure Construct-SRS-Direct($G = (V, E), k$)
1    $B[u] \leftarrow \emptyset$ for all $u \in V$;
2    for $i = 1, 2, \ldots$ do
3        $f \leftarrow$ FALSE;
4        for $u \in V$ do
5            $T \leftarrow \emptyset$;
6            for $v \in V$ such that $(u, v) \in E$ do
7                $A \leftarrow$ Retrieve-ADS($B, v, k$);
8                for $(w, \delta_{vw}) \in A$ do
9                    if $\delta_{vw} = i - 1$ then Add $w$ to $T$;
10           Sort $T$;
11           $A \leftarrow$ Retrieve-ADS($B, u, k$);
12           for $w \in T$ do
13               if $r(w) \geq \pi(u, w)$ then continue;
14               if $w \notin A$ then Add $(w, i)$ to $B[u]$;
15               $f \leftarrow$ TRUE;
16       if $f =$ FALSE then break;
17   return $B$;
```

Lemma 6.2: *Construct-SRS-Direct runs in $O(D(n + m)k^2 \log^2 n \log(k \log n))$ expected time and $O(n+m+|\mathcal{B}|+k \log n)$ expected space, where D is the diameter of G.*

Proof Sketch. The number of iterations is at most D. In each iteration, Retrieve-ADS is called for each edge. With regard to space consumption, we just need additional space for an ADS and extracted possible entries. □

6.2 Partial ADS Caching

To improve the running time of the above algorithm, we here introduce a caching technique that can flexibly adjust the trade-off between space and time consumption. The most time-consuming part is retrieving the current ADS of each neighbor (Line 7 in Algorithm 4). We cache these retrieval results to reuse them.

We propose to use the following strategy, which is simple but sufficiently effective in practice. Let us first assume that we receive a parameter α ($0 \leq \alpha \leq 1$). Before each iteration, we retrieve and cache the current ADSs for the top-αn vertices with the highest degrees. As each cached ADS takes $O(k \log n)$ expected space, the expected space consumption increases to $O(n + m + |\mathcal{B}| + \alpha nk \log n)$. On the other hand, as the ADSs of the ends of at least αm edges are cached, the expected running time improves to $O(D(n + (1 - \alpha)m)k^2 \log^2 n \log(k \log n) + \alpha mk \log n)$. The latter term is the total time consumption for reading cached entries, as the total number of entries that pass through each edge in total corresponds to the size of an ADS.

The space consumption can be further improved as follows. During the i-th iteration, the entries of our interest in cached ADSs are only those with distance $i - 1$, and thus other entries are unnecessary to store in the cache. Therefore, interestingly, even with the full cache setting (i.e.,

$\alpha = 1$), this algorithm may work with less space than the size of a full ADS.

In practice, if necessary, we can adjust the space consumption more precisely as below. Given the cache size limit, we retrieve and cache ADSs of vertices in the decreasing order of their degrees, until the total cache size exceeds the limit.

Weighted Graphs. Although the indirect algorithms such as Construct-SRS-Fast perfectly work with weighted graphs, the direct approach has some difficulty with them. If weight values are small integers, the above direct algorithm still works fine. Otherwise, one way is to consider a little relaxed version of ADSs called the $(1+\epsilon)$-*ADSs* [12], which is designed to be efficiently built with scan-and-merge-based algorithms.

7. PRACTICAL IMPROVEMENT

7.1 Implicit Neighborhood

To further reduce the storage requirement of SRSs, we propose the *implicit neighborhood technique*. The idea underlying this technique is that, when we use SRSs for analyzing a graph, the original graph itself is often also available for access. Therefore, under such circumstances, we do not need to store what we can immediately obtain from the original graph, i.e., the neighbors in the original graph.

The implicit neighborhood technique redefines the SRS of vertex u as

$$\mathcal{B}'(u) = \{(v, \delta_{uv}) \in \mathcal{B}(u) \mid (u, v) \notin E \text{ or } \ell(u, v) > \delta_{uv}\}.$$

That is, we basically remove v in $\mathcal{B}(u)$ if (u, v) is in E. However, if $\delta_{uv} < \ell(u, v)$, then it cannot be removed, since otherwise the correct distance from u to v cannot be obtained. We modify the retrieval algorithm Retrieve-ADS as follow. When we expand the search from v (Line 7 in Algorithm 1), in addition to entries in $\mathcal{B}'(v)$, we also use $(w, \ell(v, w))$ for all $(v, w) \in E$. The time complexity becomes $O((\deg_{\max} + k \log n) k \log n \log(k \log n))$, where $\deg_{\max}$ is the maximum degree in the original graph. To construct $\{\mathcal{B}'(u)\}_{u \in V}$, we once construct $\{\mathcal{B}(u)\}_{u \in V}$ and then remove entries that correspond to original edges.

As seen in Section 8, this technique reduces the storage requirement of SRSs drastically. On the other hand, the retrieval time does not degrade significantly.

7.2 Parallelization

The construction algorithms described above can be easily parallelized to exploit the thread-level parallelism of modern computer systems. As confirmed in our experiments, their performance well scales with the number of threads.

Specifically, in Construct-SRS-Fast, different tuples in the list can be processed in parallel. To guarantee the same result, we need to restrict the tuples processed in parallel to those with the same distance. Similarly, in Construct-SRS-Direct, different vertices can be processed in parallel.

8. EXPERIMENTS

8.1 Setup

Environment. All the experiments were conducted on a Linux server with two Intel Xeon X5650 processors (2.67 GHz, 6 cores, 12 threads) and 96 GB of main memory. All the algorithms were implemented in C++ and compiled using gcc 4.8.4. We used Google Snappy as a general-purpose LZ-family compression algorithm. Each entry of ADSs and SRSs was represented by 32 bits, where 26 bits are for the vertex ID and 6 bits are for the distance.

Table 2: Datasets used in our experiments.

Name	Type	$\|V\|$	$\|E\|$
wiki-Vote	Social (d)	7,115	103,689
email-Enron	Social (u)	36,692	367,662
web-NotreDame	Web (d)	325,729	1,497,134
com-dblp	Social (u)	317,080	2,099,732
web-Google	Web (d)	875,713	5,105,039
com-youtube	Social (u)	1,134,891	5,975,248
in-2004	Web (d)	1,382,870	16,917,053
flickr-links	Social (u)	1,715,256	31,101,563
soc-LiveJournal1	Social (d)	4,847,571	68,993,773
hollywood-2009	Social (u)	1,139,905	113,891,327
indochina-2004	Web (d)	7,414,768	194,109,311
com-orkut	Social (u)	3,072,441	234,370,166

Methods. We compared five sketch construction methods in our experiments. *(1)* ADS denotes the all-distances sketches construction method. *(2)* ADS-c denotes an ADS construction method with compression, which simply compresses sketches using an LZ-family compressor for each vertex. *(3)* SRS denotes the SRS construction algorithm using ADSs (Algorithm 3). *(4)* SRS-d denotes the direct SRS construction algorithm (Algorithm 4). *(5)* SRS-i denotes the SRS construction algorithm with the implicit neighborhood technique described in Section 7.1. Note that the algorithm of SRS and SRS-i are the same, and the constructed indexes of SRS and SRS-d are also the same. The SRS construction algorithms are parallelized, and run for 24 threads unless otherwise specified. For SRS-d, we set $\alpha = 1$ unless otherwise specified.

Datasets. We used publicly available real-world networks from the Stanford Large Network Dataset Collection [22], the Koblenz Network Collection [20] and Laboratory for Web Algorithms [5, 7], which include social networks and web graphs of various sizes. Table 2 summarizes the details of the real-world networks used in the experiments, where (d) and (u) stand for directed and undirected networks, respectively. It shows the number of directed edges, i.e., an undirected edge corresponds to two directed edges. For directed graphs, we constructed forward ADSs and SRSs.

8.2 Sketch Size

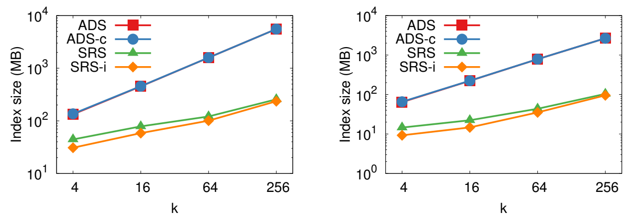
Table 3 summarizes the performances of each method for $k = 16$. The sketch size for each method and dataset is shown in the upper-left part of the table. ADS-c performed slightly worse than ADS, which suggests that simple compression does not work for ADSs. This is because an ADS for each vertex is too small to compress, although the ADSs for all the vertices can be large. SRS consistently generated smaller sketches than ADS, which ranged in 7% to 34% of the sizes of the original ADSs. The SRSs become even smaller with the implicit neighborhood technique, i.e., less than half of the size of the original SRSs in some datasets.

Effect of *k*. Figures 4a and 4b show the sketch size for different k values in web-Google and com-DBLP, respectively. It is should be emphasized that the relative size of SRSs as

Table 3: The columns on the left show the sizes of the sketches and the average ADS retrieval time, and the columns on the right show the time and memory consumption for construction ($k = 16$).

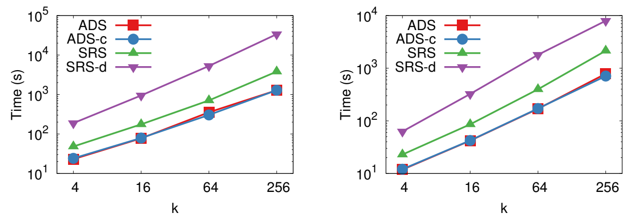
Dataset	Sketch size (MB)				Construction space (MB)			
	ADS	ADS-c	SRS	SRS-i	ADS	ADS-c	SRS	SRS-d
wiki-Vote	1.96	1.98	0.29	0.06	8.93	8.93	20.65	7.08
email-Enron	19.46	19.63	1.53	0.56	59.11	59.40	182.85	40.57
web-NotreDame	59.90	60.89	8.75	4.48	201.12	201.12	579.60	234.02
com-dblp	222.15	223.73	22.30	14.66	529.67	532.26	1929.51	303.22
web-Google	451.38	455.01	78.42	58.45	1055.01	1052.66	3956.93	734.14
com-youtube	873.19	878.85	30.37	13.79	1869.00	1863.96	7512.91	1103.72
in-2004	597.66	603.18	138.63	92.68	1468.30	1489.94	5049.08	1272.42
flickr-links	1277.11	1285.42	59.56	16.84	2866.05	2893.68	11344.86	2045.52
soc-LiveJournal1	3552.90	3575.24	552.34	367.78	7479.33	7480.42	30822.70	5341.82
hollywood-2009	843.54	849.01	289.21	165.82	5189.92	5190.38	7997.62	5189.80
indochina-2004	4255.40	4286.35	1443.24	1048.27	9440.30	9556.09	35403.26	9629.31
com-orkut	2534.30	2549.65	813.91	431.16	7801.51	7803.18	23531.55	7802.66

	Retrieval time (μs)				Construction time (s)			
	ADS	ADS-c	SRS	SRS-i	ADS	ADS-c	SRS	SRS-d
wiki-Vote	—	0.40	73.68	119.92	0.26	0.27	0.56	0.81
email-Enron	—	0.84	182.71	538.69	2.26	2.40	5.32	11.46
web-NotreDame	—	0.54	36.26	53.47	5.86	6.14	13.70	142.44
com-dblp	—	1.33	413.76	494.32	45.83	46.12	84.60	324.12
web-Google	—	1.16	271.13	260.74	82.57	81.32	173.13	952.80
com-youtube	—	2.33	391.58	4522.63	145.57	146.67	301.00	977.78
in-2004	—	0.91	160.90	189.82	64.67	66.12	171.73	3473.29
flickr-links	—	1.85	517.60	8163.60	341.40	319.07	599.38	1954.50
soc-LiveJournal1	—	1.54	1008.34	2273.50	1584.42	1627.21	2685.72	9142.96
hollywood-2009	—	2.70	1073.94	4270.85	845.68	802.59	1082.13	2709.84
indochina-2004	—	1.36	357.28	412.61	597.31	602.28	1685.47	34237.70
com-orkut	—	1.40	1629.66	11186.20	3798.51	3772.43	4557.96	5093.08



(a) web-Google (b) com-DBLP

Figure 4: Sketch sizes for varying k.



(a) web-Google (b) com-DBLP

Figure 5: Construction time for varying k.

compared to ADSs further decreases as the value of k increases in both datasets, which suggests that SRS is more efficient when k is large. In contrast, the implicit neighborhood technique is more effective when k is small because the ratio of entries of neighboring vertices in an SRS will also be small when the size of the SRS increases.

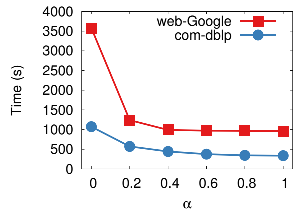
8.3 Construction Time and Space

The right-hand side of Table 3 compares the construction time and space for $k = 16$. The construction time for SRS includes the ADS construction time. In all datasets, ADS and ADS-c perform the best in terms of construction time. SRS constructed sketches for the com-Orkut dataset with 234 million edges in around 75 minutes. SRS-d took more time for SRS construction, i.e., up to 21 times longer than SRS, and it processed the indochina-2004 dataset in around 10 hours, which was the toughest instance for SRS-d in the experiments. In social networks, however, SRS-d took up to four times longer than SRS. This is attributed to the difference of diameters between web graphs and social networks.

As for the construction space, SRS-d performs the best in most of the datasets, and shows quite similar performance to ADS in the other datasets. The construction space of SRS-d tends to be small when the SRSs themselves are also small, since it does not keep the whole ADSs during the construction. SRS basically required two to four times more space than ADS because it internally keeps ADSs and other data structures linear to the size of the ADSs.

Effect of k. Figures 5a and 5b show the construction time for different values of k in web-Google and com-DBLP, respectively. Although the construction time of SRS-d grows slightly faster than that of the other methods, all the methods constructed sketches for $k = 256$ within several hours in web-Google.

Effect of Partial ADS Caching. Figure 6 shows the trade-off between time and space consumption of our direct construction algorithm for varying α, which is the ratio of cached ADSs ($k = 16$). Partial ADS caching significantly improves the construction time even when $\alpha = 0.2$. This is because a small fraction of vertices cover a large number of


(a) Construction time

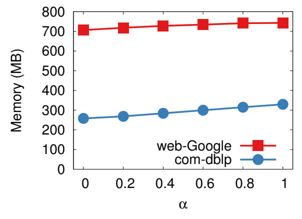
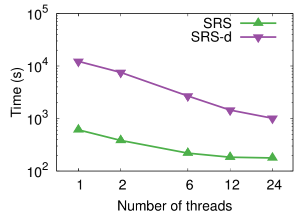

(b) Construction space

Figure 6: The effect of partial ADS caching.


(a) Construction time

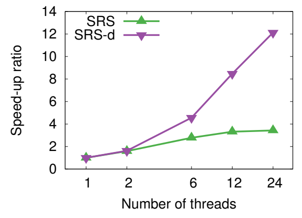
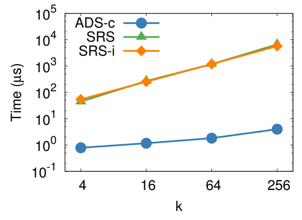

(b) Speed-up ratio

Figure 7: The effect of parallelization for varying numbers of threads in web-Google.


(a) web-Google

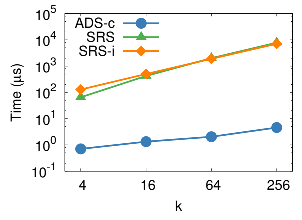

(b) com-DBLP

Figure 8: Retrieval time for varying values of k.

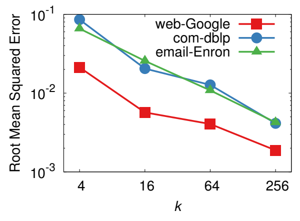
number of edges in real-world networks owing to their power law degree distributions. In contrast, the construction space does not seem to increase fast and it is not very large even when $\alpha = 1$.

Effect of Parallelization. Figure 7 shows the effect of parallelization of SRS construction algorithms ($k = 16$). These methods, especially SRS-d, have quite high parallelizability. SRS-d constructed SRSs around 12 times faster for 24 threads than for a single thread, which makes the algorithm more practical.

8.4 Retrieval Time

The lower-left part of Table 3 shows the average time of retrieval of an ADS for $k = 16$. For each method and dataset, we conducted retrievals from 10,000 randomly chosen vertices and took their average. The retrieval times of SRS are within around 1 ms in those datasets. SRS-i showed similar performance to SRS in some datasets, whereas it took up to seven times longer in other datasets. The retrieval times of these methods seem to depend not only on the size of ADSs or SRSs. We can observe that the retrieval tends to takes a long time when the sketch size is small compared with the original ADS.

Effect of k. Figures 8a and 8b show the retrieval time for different k values in web-Google and com-DBLP. The retrieval times of SRS and SRS-i grow almost linearly to k.


(a) Closeness centrality

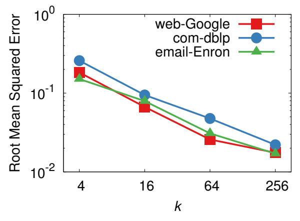
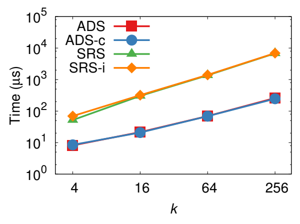

(b) Closeness similarity

Figure 9: Accuracy of estimated values.


(a) Closeness centrality

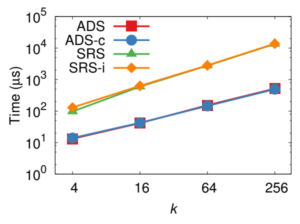

(b) Closeness similarity

Figure 10: Estimation time in web-Google.

The difference between SRS and SRS-i becomes less significant as k increases.

8.5 Estimation

Further, we conducted experiments for estimating graph properties to examine the practical applicability of SRS. It has been shown that several indicators of graphs can be estimated using ADS, as discussed in Section 3.2. We estimated the closeness centrality [12] and closeness similarity [13] of 1,000 uniformly chosen vertices or pairs of vertices for each dataset using the constructed ADSs and SRSs, and evaluated the approximation error and estimation time for varying values of k. Note that the accuracy results of SRSs and ADSs are exactly the same.

Accuracy. Figures 9a and 9b show the root-mean-square error (RMSE) of estimated centrality and similarity values compared with the exact values. For the three datasets shown in the figures, the RMSEs of both centrality and similarity basically decrease as the value of k increases. The other datasets also showed similar results, as suggested theoretically.

Estimation Time. Figures 10a and 10b show the average estimation time of closeness centrality and closeness similarity values in web-Google. The bottleneck of the estimation is ADS retrieval because a vertex centrality or similarity is estimated in linear time to the size of the ADSs. Thus, these figures show a similar trend to retrieval time. The values can be estimated within 20 ms even for $k = 256$, which should be acceptable in typical situations.

9. CONCLUSIONS

In this paper, we proposed a new scheme for graph neighborhood sketching, called the *sketch retrieval shortcuts* (SRS). The SRS complements the recently-emerging powerful sketching method *all-distances sketches (ADS)* [11,12]. The ADS combines preferable properties: multi-functional, accuracy guaranteed, and scalable (in theory). However, in practice, it is not as scalable as expected, due to its large storage consumption. The SRS is designed so that, while it is much smaller than the ADS, from SRSs, an ADS of any vertex can be quickly retrieved. Therefore, once we have obtained SRSs, various graph properties can be estimated on massive graphs with the exactly same procedure as that of plain ADSs. In our experiments, we observed that SRSs are orders of magnitude smaller than ADSs, we can construct SRSs for million-scale graphs, and the retrieval time is in milliseconds and sufficiently quick. These results show the high practicality of SRSs as a back-end for large-scale graph analysis.

Acknowledgement. This work was supported by JSPS Grant-in-Aid for Research Activity Startup (No. 15H06828) and JST, PRESTO. The authors thank E. Cohen, D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck for fruitful discussion.

10. REFERENCES

[1] T. Akiba, T. Hayashi, N. Nori, Y. Iwata, and Y. Yoshida. Efficient top-k shortest-path distance queries on large networks by pruned landmark labeling. In *AAAI*, pages 56–67, 2015.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

[3] T. Akiba, C. Sommer, and K. Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, pages 144–155, 2012.

[4] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *WebSci*, pages 33–42, 2012.

[5] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.

[6] P. Boldi, M. Rosa, and S. Vigna. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *WWW*, pages 625–634, 2011.

[7] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–601, 2004.

[8] P. Boldi and S. Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW*, pages 107–117, 1998.

[10] E. Buchnik and E. Cohen. Reverse ranking by graph structure: Model and scalable algorithms. *CoRR*, abs/1506.02386, 2015.

[11] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[12] E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis. *TKDE*, 27(9):2320–2334, 2015.

[13] E. Cohen, D. Delling, F. Fuchs, A. V. Goldberg, M. Goldszmidt, and R. F. Werneck. Scalable similarity estimation in social networks: closeness, node labels, and random edge lengths. In *COSN*, pages 131–142, 2013.

[14] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *CIKM*, pages 629–638, 2014.

[15] E. Cohen, D. Delling, T. Pajor, and R. F. Werneck. Timed influence: Computation and maximization. *CoRR*, abs/1410.6976, 2014.

[16] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *PODC*, pages 225–234, 2007.

[17] N. Du, L. Song, M. Gomez-Rodriguez, and H. Zha. Scalable influence estimation in continuous-time diffusion networks. In *NIPS*, pages 3147–3155, 2013.

[18] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, Sept. 1985.

[19] Y. Fujiwara, M. Nakatsuji, T. Yamamuro, H. Shiokawa, and M. Onizuka. Efficient personalized pagerank with accuracy assurance. In *KDD*, pages 15–23, 2012.

[20] J. Kunegis. Konect: The koblenz network collection. In *WWW Companion*, pages 1343–1350, 2013.

[21] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[22] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

[23] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[24] S. Maniu and B. Cautis. Network-aware search in social tagging applications: Instance optimality versus efficiency. In *CIKM*, pages 939–948, 2013.

[25] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *KDD*, pages 81–90, 2002.

[26] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

[27] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *PVLDB*, 1(1):710–721, 2008.

[28] A. D. Zhu, X. Xiao, S. Wang, and W. Lin. Efficient single-source shortest path and distance queries on large graphs. In *KDD*, pages 998–1006, 2013.

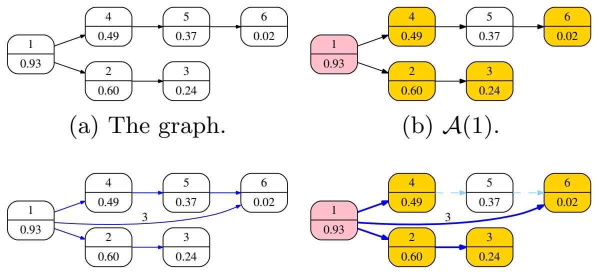
APPENDIX

A. COMMON MISUNDERSTANDING

A common misunderstanding about the definition of SRSs is that $\mathcal{B}(u)$ is equivalent to

$$\mathcal{E}(u) = \{(v, \delta_{uv}) \in \mathcal{A}(u) \mid P(u, v) \cup \mathcal{A}(u) = \emptyset\} .$$

Intuitively this may seem true, but it is false. Figure 11 shows a counterexample ($k = 1$). As vertex $4 \in \mathcal{A}(1)$ and $4 \in P(1, 6)$, $6 \notin \mathcal{E}(1)$. However, $6 \in \mathcal{B}(1)$, and indeed, the SRS entry from vertex 1 to vertex 6 is necessary for retrieval; otherwise, vertex 6 would be isolated.



(a) The graph. (b) $\mathcal{A}(1)$.

(c) The SRSs. (d) Retrieving $\mathcal{A}(1)$.

Figure 11: A counterexample where $\mathcal{B}(1) \neq \mathcal{E}(1)$.

B. RETRIEVING A COMBINED ADS

To estimate the expected influence of a vertex set under the continuous-time influence model, we need its *combined ADS* [17, 15]. The combined ADS can also be quickly retrieved from SRSs by slightly modifying the retrieval algorithm. At Line 2 in Algorithm 1, we push all all vertices in S. It works in $O(|S| \log |S| + k^2 \log^2 n \log(k \log n))$ time and $O(|S| + k^2 \log^2 n)$ space.