Partitioned Multi-Indexing: Bringing Order to Social Search

Bahman Bahmani *
Stanford Univesrity
Stanford, CA
bahman@stanford.edu

Ashish Goel†
Stanford Univesrity
Stanford, CA
ashishg@stanford.edu

ABSTRACT

To answer search queries on a social network rich with user-generated content, it is desirable to give a higher ranking to content that is closer to the individual issuing the query. Queries occur at nodes in the network, documents are also created by nodes in the same network, and the goal is to find the document that matches the query and is closest in network distance to the node issuing the query. In this paper, we present the "Partitioned Multi-Indexing" scheme, which provides an approximate solution to this problem. With m links in the network, after an offline $\tilde{O}(m)$ pre-processing time, our scheme allows for social index operations (i.e., social search queries, as well as insertion and deletion of words into and from a document at any node), all in time $\tilde{O}(1)$. Further, our scheme can be implemented on open source distributed streaming systems such as Yahoo! S4 or Twitter's Storm so that every social index operation takes $\tilde{O}(1)$ processing time and network queries in the worst case, and just two network queries in the common case where the reverse index corresponding to the query keyword is much smaller than the memory available at any distributed compute node.

Building on Das Sarma et al.'s approximate distance oracle (which itself is similar to Bourgain's embeddings [3]), the worst-case approximation ratio of our scheme is $\tilde{O}(1)$ for undirected networks. Our simulations on the social network Twitter as well as synthetic networks show that in practice, the approximation ratio is actually close to 1 for both directed and undirected networks. We believe that this work is the first demonstration of the feasibility of social search with real-time text updates at large scales.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Online computation*

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Partitioned Multi-Indexing, Social Search, Scalable, Real Time

1. INTRODUCTION

In contrast to traditional search where search ranking is primarily based on document-based relevance and quality measures such as tf-idf [17] or PageRank [21], "Social Search" also takes into account the social graph of the person issuing the query, normally by giving a higher rank to content generated or consumed by proximate users in the social graph. This type of search not only has applications such as name, entity, or content search on social networks [29], and social question and answering [13], it is also very effective for personalization of web search [4, 2]. The rapid rise of user generated content (on online social networks, blogs, forums, and social bookmarking or tagging systems) has added to the importance of social search. This is reflected not only in the growing literature on the topic [4, 13, 29, 31, 1], but also in the attempts made by both major and small Internet companies, such as Google, Microsoft, Twitter, Aardvark, etc. to develop social search technologies.

With the massive scale of today's social data, e.g. on online social networks, and noting the fact that social content is being constantly generated, an ideal social search engine needs to have the following properties:

- Very high efficiency and speed at query time
- Real-time updatability, to keep up with content being generated or modified
- Capability to mix social-graph-based personalization with more traditional (e.g. document-based) relevance and quality measures
- High scalability

Given the number of users in a typical social network, and the volume of updates, any solution to the above problem must be amenable to a distributed computation. In this paper, we will assume that the underlying computational substrate is an *Active DHT*. A DHT (Distributed Hash Table) is a distributed (Key, Value) store which allows Lookups, Inserts, and Deletes on the basis of the "Key". The term *Active* refers to the fact that, in addition to these DHT operations,

*Research supported in part by William R. Hewlett Stanford Graduate Fellowship

†This research was supported in part by NSF awards IIS-0904325 and DC-0915040. Part of the research was also sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

we assume that an arbitrary User Defined Function (UDF) can be executed on a (Key, Value) pair. The Active DHT model is broad enough to act as a distributed stream processing system and as a continuous version of Map-Reduce. Yahoo's S4 [20] and Twitter's Storm are two examples of Active DHTs which are now gaining widespread use. We will implicitly assume that all the (Key, Value) pairs in a node of the active DHT are stored in main memory; this is equivalent to assuming that no one (Key, Value) pair is too large and that the distributed deployment has sufficient number of nodes.

In this paper, we present the "Partitioned Multi-Indexing" scheme for indexing graph structured data which as we will show, when applied to the problem of social search, satisfies all the above-mentioned properties. At the core, the scheme is an indexing method which, for any query, allows for very quickly finding the closest nodes (to the node issuing the query) in a social graph which answer the query. While our scheme handles what we call social index operations (search, content addition, and content deletion) in real-time, it does not handle social graph updates in real-time; we assume that the social graph is pre-processed (perhaps daily) in a separate initialization step.

The paper is organized as follows. First, we present some background, the formal statement of the problem, an overview of our scheme, and a summary of our results in the rest of this section. Section 2 presents the preliminaries necessary for presenting our main algorithms. In section 3, we present the basic partitioned multi-indexing scheme, including the algorithms and space and time complexity analyses. In the same section, we will also present the extension of the basic scheme to directed graphs, its integration with document-based relevance measures, and the distributed implementation of the scheme. In section 4, we present the results of our experiments.

1.1 Background

With the rapid rise of social data in recent years, the social search problem has gained increasingly more attention both in the academic literature [29, 22, 30, 31, 4, 13, 11], and in industry. Yahia et al. [30] study the problem of ranking search results in collaborative tagging networks. Vieira et al. [29] focus on ranking name search results on social networks. Horowitz et al. [13] focus on social question and answering. Carmel et al. [4] consider personalization of search results based on the user's social network, show its much higher quality in comparison with topic-based personalization, and provide heuristic methods to re-rank the search results based on the social graph. Similarly, Yin et al. [31] show the high effectiveness of social search for personalization of web search.

Shortest path distances have been proposed as the main proxy for social graph based personalization [29, 26, 30, 22]. Clearly, any social search system based on this proxy needs a way to compute or approximate shortest path distances, which has also been an active area of research [32, 28, 24, 10, 9, 5, 8]. Among these, the family of methods known as "approximate distance oracles" [9, 28, 8, 32] are best suited for the social search application. The methods in this family preprocess the graph such that any subsequent distance query can be answered very quickly.

To solve the social search problem, even given a fast distance oracle, one still needs to find the closest nodes to the querying node which answer the query. The basic method of using the oracle to find the distances to all the candidates and then finding the closest ones does not scale to today's massive social networks where the number of search result candidates itself can be very large. The previous works in the social search literature (e.g., [29, 26, 30, 22]) provide no additional efficiency compared to this basic scheme.

We introduce a method for indexing graph structured data, called partitioned multi-indexing, based on the oracle introduced by Das Sarma et al. [8], which allows for a very efficient search scheme. Our scheme inherits two parameters k, r from Das Sarma et al.'s oracle, which, to provide approximation guarantees, need to be set to $r = \log_2 n, k = \tilde{O}(1)$.

With $r = 0$, this oracle reduces to the landmark-based distance approximation [22, 7], and our indexing method reduces to an efficient way of finding the search results based on landmark-based approximate distances. In this case, there is no theoretical guarantee on the approximation quality, and our experiments also show that landmark-based approximate distances perform poorly in social search. Potamias et al. [22] study a number of heuristics for landmark selection, and report a centrality-based heuristic to work best across their experiments. We also implemented this scheme but did not observe any improvement in search quality, compared to the random landmark selection scheme. With $r > 0$, the partitioning property that we prove for our scheme allows for maintaining space and time efficiency while using whole seed sets instead of single node landmarks to approximate the distances. This leads to significantly higher quality search results.

Our method can be also compared to the family of Approximating and Eliminating Search Algorithms (AESA) for metric space near neighbor search [23, 18, 25]. The algorithm proposed by Vidal [23] requires quadratic space and preprocessing time which is clearly infeasible. The methods proposed by Shapiro [25] and Micó et al. [18] address this issue; however, in a graph with n nodes, they may need to compute, at query time, the distance from the querying node to up to $\Omega(n)$ nodes, which is clearly infeasible. Even without distance computations, they can not provide any efficiency guarantees. Also, without exact distance computations, they can not provide any guarantees on the quality (i.e., approximation factor) of the results they find. Finally, our scheme can also be considered as a member of the "distance-based indexing" methods for metric space similarity searching. Chávez et al. [6] and Hjaltason et al. [12] provide great surveys of these methods. To the best of our knowledge, ours is the only scheme to provide theoretical guarantees on approximation factor, preprocessing space and time complexity, and query time complexity, and also scale up to today's social networks.

Before presenting an overview of our scheme, we first present, in the next section, the formal statement of the problem we study.

1.2 Notations and Problem Statement

We have a (social) graph $G = (V, E)$ with $|V| = n, |E| = m$. The nodes of this graph may represent people, documents, entities, etc., and the edges may represent friendships, page visits, or any other social interactions. For now, we assume G to be undirected. In section 3.1, we will also study the case of directed graphs. Also, our scheme works in exactly the same way and with exactly the same guar-

antees for graphs with weighted edges. So, for simplicity of presentation, we will assume in the rest of the paper that the edges are not weighted.

We also have a corpus $C =< C_v >_{v \in V}$, where for each $v \in V$, C_v is the document(s) (e.g. tags, bookmarks, tweets, etc.) associated with node v. In this paper, we will assume that C_v is a set of words. Also, even though we start with an initial corpus, we will allow words to be added to or deleted from any document over time. This corresponds to, e.g., receiving new tweets, bookmarks, or wall posts.

For each word ω, we will denote:

$$I(\omega) = \{v \in V | \, \omega \in C_v\}$$

and let $l(\omega) = |I(\omega)|$. Furthermore, we denote:

$$|C| := \sum_{v \in V} |C_v| = \sum_{\omega \in \cup_v C_v} l(\omega)$$

We also have an approximate distance oracle, which for any two nodes $u, v \in V$, outputs $\tilde{d}(u, v)$, an approximation of the shortest path distance $d(u, v)$ between u and v. For now, we do not restrict the choice of this oracle, but later in the paper, we will base our algorithms on the oracle introduced by Das Sarma et al. [8].

We will need to answer search queries of the form (u, ω, J), where $u \in V$ is the node issuing the query, ω is the word being queried, and $J \geq 0$, an integer, is the desired number of search results for the query. Each search result is a node $v \in I(\omega)$, and we would like to find, among all such nodes, the J nodes having the smallest approximate distances to u (as measured by $\tilde{d}(u, \cdot)$), and return them in a ranked list sorted in the increasing order of approximate distance to u. We will assume that $J \leq l(\omega)$, as $l(\omega)$ is clearly the maximum possible number of search results for the query.

Having set all the necessary notation, the problem statement is then as follows:

Real-Time Social Search Problem: Preprocess the social graph G and the corpus C in a space and time efficient way to construct a data structure that allows for:

1. Answering any social search query very quickly
2. Distributed storage and processing in an Active DHT
3. Very fast incremental updates as soon as words are added to or deleted from any document

Having presented the formal statement of the basic problem, we will next present an overview of our solution scheme; we will consider extensions of our scheme later in the paper.

1.3 Overview of Our Scheme

In this section, we give a high level overview of our scheme, called partitioned multi-indexing. Our scheme has an offline phase and a query phase. In the offline phase:

1. We first pick a number of random seed sets $S_0, \ldots, S_{h-1} \subseteq V$. The number of these sets, h, and the cardinality of each set will be specified later in the paper.
2. $\forall \, u \in V$, $0 \leq i < h$, we find $L_i[u]$, the closest node to u among all the nodes in S_i, and $D_i[u] = d(u, L_i[u])$. This can be accomplished using $O(h)$ calls to a breadth first search subroutine.
3. $\forall \, 0 \leq i < h, x \in S_i$, we construct an inverted index, $I_{i,x}$, over all documents stored at nodes $v \in V$ which are closer to x than to any other node in S_i. For each indexed word ω, the corresponding list of nodes, $I_{i,x}(\omega)$, will be kept in the increasing order of distances to x, and these distances will also be stored in this list.

Then, at query time, when a node u issues a query, we use the indexes $I_{i,L_i[u]}$ $(0 \leq i \leq h-1)$, i.e., intuitively speaking, the closest indices to u, to find the search results. We will see that since u is closer to $L_i[u]$ than to any other node in S_i, and also the nodes in each entry of $I_{i,L_i[u]}$ are sorted in terms of their distance to $L_i[u]$, then at query time, we can find the search results by sweeping through the beginning nodes in the index entries being looked up. This will result in a very fast search algorithm at query time. We will, furthermore, show that our index allows for very fast incremental updates upon addition or deletion of words.

Note that, for each $0 \leq i < h$, any node $x \in S_i$ indexes a different part of the graph (i.e., the part closer to x than to any other node in S_i), and also, every node u in the graph is indexed at one (and only one) node of S_i, i.e., the one closest to u. This means that the union of the indexes constructed at the nodes in each S_i $(0 \leq i < h)$ constitutes a full inverted index of the graph, partitioned across different nodes of S_i. Thus, in the offline phase, we construct h inverted indexes, each partitioned across the nodes of one seed set. Hence, the name partitioned multi-indexing for our scheme.

Quite interestingly, this schemes maps naturally to an Active DHT. Consider (for illustration) the common scenario where the reverse index corresponding to any word has size much smaller than the amount of main memory of each individual node in the Active DHT. Then we can use the query word w as the key used to store the part of each index $I_{i,v}$ which pertains to w. This allows us to perform social index operations using just two network calls, without any corresponding increase in the total processing time. This is important, because small network data transfers such as the one needed here are often much more expensive than large network transfers in terms of data rate. This careful mapping of the social search problem onto a practically feasible distributed computing platform is one of our main contributions.

1.4 Our Results

We present the partitioned multi-indexing scheme for indexing graph structured data, which not only has strong theoretical guarantees, but also when applied to the social search problem, satisfies all the properties mentioned in section 1 for an ideal social search engine. Our scheme consists of an offline preprocessing phase and an online query phase. We show that given a (social) graph G and a corpus C as in section 1.2, the preprocessing phase requires $\tilde{O}(m + |C|)$[1] time and $\tilde{O}(n + |C|)$ space. After preprocessing, whenever any node u queries for any word ω, the top J personalized results can be found in $\tilde{O}(J)$ time. Also, in the distributed setting, the number of network accesses and the total amount of communication needed to answer the query are, respectively, 2 and $\tilde{O}(J)$.

Also, our index can be very quickly updated whenever a word is added to or deleted from a document in the corpus.

[1]The $\tilde{O}(\cdot)$ notation hides factors that are poly-logarithmic in m.

More exactly, updating the index upon each word addition or deletion can be done in $\tilde{O}(1)$ time, and in the distributed setting, the total number of network accesses and the total amount of communication required per update are, respectively, 2 and $\tilde{O}(1)$.

Superficially, it might seem that this work is incremental over that of Das Sarma et al. [8]. However, as we mentioned before, there are many shortest path oracles, and it was not clear up front which of these, if any, could be extended to social search, specially with the constraints of distributed implementation, real-time index updates, and mixing in other relevance features; the novelty of this work lies in identifying the right oracle and carefully adapting it to obtain each of the desired properties, with strong theoretical guarantees.

In addition to theoretical bounds, we also perform an empirical study of our scheme, to evaluate both its efficiency and its quality. We use synthetic data as well as data from the social network Twitter. On both sets of networks, and for both evaluation criteria, our scheme performs much better than the (already strong) theoretical bounds would suggest. Hence, we believe that our scheme can indeed facilitate large scale, real-time social search.

2. PRELIMINARIES

As explained in section 1.2, one of the ingredients of the social search problem is an approximate distance oracle $\tilde{d}(\cdot,\cdot)$. Given such an oracle, to solve the social search problem, we still need to very quickly find the nodes answering the query which have the smallest approximate distances to the querying node. To do so, one can define a basic personalized social search scheme as follows.

Baseline Social Search Scheme: The scheme is composed of an offline phase and a query phase. At the offline phase, a single inverted index I is constructed, which maps each word ω to the list $I(\omega)$ of all the nodes v having ω in their associated document C_v. At query time, receiving a query (u,ω,J) issued by the node u for the word ω, one goes through the list at the entry $I(\omega)$ of the precomputed index, for each node $v \in I(\omega)$ uses the oracle to compute $\tilde{d}(u,v)$, and keeps the top results in a priority queue of size J. This baseline scheme is clearly inefficient for query processing; however, it is a useful benchmark to compare the pre-processing efficiency and the quality of our scheme against.

Das Sarma et al.'s Distance Oracle: This oracle has two integer parameters $k \geq 1, 0 \leq r \leq \log_2 n$. It first preprocesses the graph offline. The preprocessing, presented in Algorithm 1, picks a number, $h = k(r+1)$, of random subsets S_i $(0 \leq i < h)$ of the graph, and by performing a BFS from each one, computes, for each node $u \in V$, the closest node to u in S_i, $L_i[u]$, as well as $D_i[u] = d(u, L_i[u])$. Note that, since each BFS takes $O(m)$ time (assuming $m = \Omega(n)$, which is the case in all networks of our interest), the time and space complexity of Algorithm 1 are, respectively, $O(hm)$ and $O(hn)$.

Afterwards, for any two nodes $u, v \in V$, their approximate distance is computed as follows:

$$\tilde{d}(u,v) = \min\{D_i[u] + D_i[v] \,|\, 0 \leq i < h, L_i[u] = L_i[v]\} \quad (2.1)$$

In the rest of the paper, we will always denote $h = k(r+1)$. For this oracle, independent of the choice of parameters k, r, we clearly have $\forall\, u, v \in V : \tilde{d}(u,v) \geq d(u,v)$. If $r = 0$, this oracle reduces to the landmark-based distance approximation [22, 14, 29, 27]. Kleinberg et al. [14] prove approximation guarantees for this case (even with small values of k), but their result, which assumes the graph to have a bounded doubling dimension, does not apply to social graphs which exhibit expander properties. However, increasing the value of r clearly makes the approximation tighter, and Das Sarma et al. [8] prove the following theorem:

Algorithm 1 Distance Sketching Algorithm.

1: **Input:** Undirected graph G, $k \geq 1, 0 \leq r \leq \log_2 n$
2: Let $h = k(r+1)$
3: **for** $i = 0$ to $h-1$ **do**
4: Sample, uniformly at random, a subset $S_i \subseteq V$ of size $|S_i| = 2^{i \mod (r+1)}$.
5: Do a BFS from S_i, and compute, for all $u \in V$, $L_i[u] = \text{argmin}_{x \in S_i}\{d(u,x)\}$, and $D_i[u] = d(u, L_i[u])$.
6: **end for**
7: $\forall\, u \in V$, let $E[u] =< (L_0[u], D_0[u]), \ldots, (L_{h-1}[u], D_{h-1}[u]) >$.

THEOREM 1. *For $\tilde{d}(\cdot,\cdot)$ defined in equation 2.1, with $r = \lfloor \log_2 n \rfloor$ and $k = \tilde{O}(n^{1/c})$ (with any $c > 1$), with high probability (i.e., probability at least $1 - 1/n^{O(1)}$), we have for any two nodes u, v:*

$$d(u,v) \leq \tilde{d}(u,v) \leq (2c-1)d(u,v)$$

Letting $c = O(\log n)$, this gives:

COROLLARY 2. *To guarantee an $O(\log n)$ approximation factor for the oracle defined by Algorithm 1 and formula 2.1, one can choose $r = \lfloor \log_2 n \rfloor$, and $k = \tilde{O}(1)$.*

Das Sarma et al. [8] observe that in practice this scheme (with r, k chosen as in corollary 2) provides much better approximation factors than is guaranteed in theory. This means one can expect that ranking the search results based on this oracle will also result in high quality search results. Our experiments presented in section 4 verify this.

3. PARTITIONED MULTI-INDEXING

We already presented an overview of our scheme in section 1.3. In this section, we present our scheme, called Partitioned Multi-Indexing, in detail and analyze its algorithms. Before presenting the scheme, we need a definition:

DEFINITION 3. *For any $0 \leq i < h$, node $z \in S_i$, and word ω, define:*

$$I_{i,z}(\omega) := \{v \in V \,|\, \omega \in C_v, L_i[v] = z\}$$

and let $l_{i,z}(\omega) = |I_{i,z}(\omega)|$. We will denote

$$I_{i,z}(\omega) = \{x^r_{i,z}(\omega)\}_{1 \leq r \leq l_{i,z}(\omega)}$$

where $d(z, x^1_{i,z}(\omega)) \leq d(z, x^2_{i,z}(\omega)) \leq \ldots \leq d(z, x^{l_{i,z}(\omega)}_{i,z}(\omega))$.

The scheme is composed of an offline phase and a query phase. The offline phase of our scheme constructs a map (i.e., an index) PMI which, for any $0 \leq i < h$, node $z \in S_i$, and word ω, such that $I_{i,z}(\omega) \neq \emptyset$, maps (i,z,ω) to the list of nodes in $I_{i,z}(\omega)$, sorted in the increasing order of distance to z. This is presented in Algorithm 2. Later in this section,

we will show that the constructed index will allow for a very fast query answering algorithm. But, before that, we analyze the space and time complexities of the offline phase.

Offline Phase Analysis: We analyze the space and time complexity of Algorithm 2. We start by a small lemma:

LEMMA 4. *For any $0 \leq i < h$, and word ω, $\{I_{i,z}(\omega)\}_{z \in S_i}$ partitions $I(\omega)$, that is*

- $\cup_{z \in S_i} I_{i,z}(\omega) = I(\omega)$
- $\forall\, z, z' \in S_i,\ z \neq z' :\ I_{i,z}(\omega) \cap I_{i,z'}(\omega) = \emptyset$

PROOF. The result follows from the observation that any node $v \in I(\omega)$, appears in $I_{i,L_i[v]}(\omega)$, and in no other $I_{i,z}(\omega)$ $(z \in S_i)$. □

Using this lemma, we have the following result:

PROPOSITION 5. *For Algorithm 2:*

- *The space complexity is $O(h|C|)$*
- *The time complexity is $O(h \sum_{\omega \in \cup_v C_v} l(\omega) \log l(\omega))$*

PROOF. Fix an $0 \leq i < h$. For any node $z \in S_i$ and word $\omega \in \cup_v C_v$, the space and time used to construct $PMI[i, z, \omega]$ are, respectively, equal to $O(l_{i,z}(\omega))$ and $O(l_{i,z}(\omega) \log l_{i,z}(\omega))$. Hence, by the previous lemma, the total space and time used to construct all queues $PMI[i, z, \omega]$ $(\forall z \in S_i, \omega \in \cup_v C_v)$, are, respectively,

$$O(\sum_{\omega \in \cup_v C_v} \sum_{z \in S_i} l_{i,z}(\omega)) = O(\sum_{\omega \in \cup_v C_v} l(\omega)) = O(|C|)$$

and

$$O(\sum_{\omega \in \cup_v C_v} \sum_{z \in S_i} l_{i,z}(\omega) \log l_{i,z}(\omega)) = O(\sum_{\omega \in \cup_v C_v} l(\omega) \log l(\omega))$$

Then, considering all $0 \leq i < h$ proves the proposition. □

Choosing the values of r, k as in corollary 2, we get that both space and time complexities of our indexing scheme are within $\tilde{O}(1)$ factor of the baseline indexing method. Furthermore, we will next show that our index leads to a significantly faster search algorithm at query time.

The search algorithm is presented in Algorithm 3. Briefly speaking, upon receiving a query (u, ω, J), we sweep through the queues $PMI[i, L_i[u], \omega]$ $(0 \leq i < h)$ until we find the top J results. More elaborately, upon receiving the query, we initiate a priority queue H, that will keep track of the (next) top result candidates, as well as h pointers p_i $(0 \leq i < h)$, where p_i points to the beginning of the sorted list $PMI[i, L_i[u], \omega]$, i.e., the node $x^1_{i,L_i[u]}(\omega)$, which we add, with priority $D_i[u] + D_i[x^1_{i,L_i[u]}(\omega)]$, to H. Then, we pop the node with the lowest priority, say $x^1_{i_1,L_{i_1}[u]}(\omega)$, from H, report it as the top search result, forward p_{i_1}, and add the node it is now pointing to, i.e., $x^2_{i_1,L_{i_1}[u]}(\omega)$ to H, with priority $D_{i_1}[u] + D_{i_1}[x^2_{i_1,L_{i_1}[u]}(\omega)]$. We then pop the node with the lowest priority from H, report it as the second top result (unless it happens to be the same as the first result), forward the corresponding pointer, and so on. We continue in this way till we find J results. Next, we analyze this algorithm.

Query Phase Analysis: We first prove that the search algorithm 3 actually works correctly. We start with a definition.

Algorithm 2 Partitioned Multi-Indexing Algorithm.

1: **Input:** Social graph G, corpus C, and the distance sketches $\{E[u]\}_{u \in V}$
2: Initialize the map PMI: $\forall 0 \leq i < h$, node $z \in S_i$, and word ω, $PMI[i, z, \omega]$ is an empty priority queue.
3: **for** $v \in V$ **do**
4: **for** $0 \leq i < h, \omega \in C_v$ **do**
5: Insert v into $PMI[i, L_i[v], \omega]$ with priority $D_i[v]$.
6: **end for**
7: **end for**

Algorithm 3 Partitioned Multi-Index Query Algorithm.

1: **Input:** Distance sketches $\{E[u]\}_{u \in V}$, Partitioned multi-index PMI, and a query (u, ω, J)
2: Let $\forall\, 0 \leq i < h :\ p_i = 1$
3: Initialize H to be an empty priority queue.
4: **for** $0 \leq i < h$ **do**
5: Insert $x^{p_i}_{i,L_i[u]}(\omega)$ into H with priority $D_i[u] + D_i[x^{p_i}_{i,L_i[u]}(\omega)]$
6: **end for**
7: Let $j = 1$.
8: **while** $(j \leq J)$ **do**
9: Pop the node with the smallest priority from H, and let $s := \text{argmin}_{0 \leq i < h}\{D_i[u] + D_i[x^{p_i}_{i,L_i[u]}(\omega)]\}$
10: **if** $(\forall j' < j :\ x^{p_s}_{s,L_s[u]}(\omega) \neq v_{j'})$ **then**
11: $v_j := x^{p_s}_{s,L_s[u]}(\omega)$
12: $j = j + 1$
13: **end if**
14: $p_s = p_s + 1$
15: Insert $x^{p_s}_{s,L_s[u]}(\omega)$ into H with priority $D_s[u] + D_s[x^{p_s}_{s,L_s[u]}(\omega)]$.
16: **end while**
17: **return** $\{v_j\}_{1 \leq j \leq J}$ as the ranked list of search results

DEFINITION 6. *For a query (u, ω, J), we say two sets of ranked results $\{v_j\}_{1 \leq j \leq J}$ and $\{v'_j\}_{1 \leq j \leq J}$, are equivalent, and we write $\{v_j\}_{1 \leq j \leq J} \sim \{v'_j\}_{1 \leq j \leq J}$, if $\forall\, 1 \leq j \leq J :\ \tilde{d}(u, v_j) = \tilde{d}(u, v'_j)$.*

Essentially, an equivalent pair of search result sets are equally good and can not be distinguished, as far as (approximate) distances to the querying node are concerned. Now, we prove the correctness of Algorithm 3.

THEOREM 7. *For a query (u, ω, J), assume $\{\tilde{v}_j\}_{1 \leq j \leq J}$, is the true ranked list of search results according to $\tilde{d}(u, \cdot)$, and $\{v_j\}_{1 \leq j \leq J}$ is defined as in Algorithm 3. Then, $\{v_j\}_{1 \leq j \leq J} \sim \{\tilde{v}_j\}_{1 \leq j \leq J}$.*

PROOF. We need to prove that $\forall\, 1 \leq j \leq J :\ \tilde{d}(u, v_j) = \tilde{d}(u, \tilde{v}_j)$. We first prove this for $j = 1$. Let:

$$i_1 = \text{argmin}\{D_i[u] + D_i[\tilde{v}_1] |\, 0 \leq i < h, L_i[u] = L_i[\tilde{v}_1]\}$$

Then, we have:

$$\begin{aligned} \tilde{d}(u, \tilde{v}_1) &= D_{i_1}[u] + D_{i_1}[\tilde{v}_1] \\ &\geq D_{i_1}[u] + D_{i_1}[x^1_{i_1,L_{i_1}[u]}(\omega)] \\ &\geq \tilde{d}(u, x^1_{i_1,L_{i_1}[u]}(\omega)) \\ &\geq \tilde{d}(u, v_1) \geq \tilde{d}(u, \tilde{v}_1) \end{aligned}$$

where the first line is by definition of $\tilde{d}(u, \tilde{v}_1)$, the second is by definition of $x^1_{i_1,L_{i_1}[u]}(\omega)$, the third is by definition of $\tilde{d}(u, x^1_{i_1,L_{i_1}[u]}(\omega))$, the fourth is by definition of v_1, and the last is by definition of $\tilde{v}_1$.

Therefore, $\tilde{d}(u, v_1) = \tilde{d}(u, \tilde{v}_1)$, that is, v_1 indeed has the smallest approximate distance to u among all the nodes in $I(\omega)$. Now, notice that to find v_2, the algorithm is essentially removing v_1 from $I(\omega)$, and finding the node having the smallest distance to u among the rest of the nodes in $I(\omega)$, in exactly the same way as it found v_1. A simple induction then proves the result for general $1 \leq j \leq J$. □

Hence, Algorithm 3 outputs a correct ranking. Next, we analyze the time complexity of this algorithm.

PROPOSITION 8. *The worst case running time of Algorithm 3 is $O(Jh(\log l(\omega) + \log h))$.*

PROOF. Reading each node from PMI takes $O(\log l(\omega))$ time. Also, adding a node to or popping a node from H takes $O(\log h)$ time. During the run of algorithm, each search result is read from PMI, and added to or popped from H at most h times. Also, the total number of nodes that get read from PMI and added to H but do not show up in the search results is at most h. Hence, the total running time of the algorithm is at most $O(Jh(\log l(\omega) + \log h)) + O(h(\log l(\omega) + \log h)) = O(Jh(\log l(\omega) + \log h))$. □

REMARK 9. *Choosing r, k as in corollary 2, we get that the total query time is just $\tilde{O}(J)$. Using the baseline scheme, presented in section 2, with the same oracle, the query time, as analyzed in section 2, would be $\tilde{O}(l(\omega))$. In today's huge social networks, one can easily expect $l(\omega)$, i.e. the number of nodes the word ω appears on, to be much (even orders of magnitude) larger than J. For instance, in a name search application on a huge social network, there may be tens or hundreds of thousands of people sharing a same name, but the querying node may be interested only in at most the top $10 - 20$ results. Hence, our scheme is expected to be significantly faster at query time in practice. Our experimental results, presented later in the paper, verify this as well.*

REMARK 10. *The same analysis as in proposition 8 shows that if we have already found the first J results, then by keeping the values of the pointers in the algorithm, finding the next J' results will take only $O(J'h(\log l(\omega) + \log h))$. This feature can be useful in practice. For instance, the search engine can first generate the results to be presented on the first results page, and then only if the user decides to proceed to the next page, it can, at that time, quickly compute the results to be presented in the next page, and so on.*

Having analyzed the query phase of our scheme, we will next show that our indexing scheme also allows for very fast incremental updates upon addition or deletion of words to the documents.

Incremental Updates: So far we focused on the case where the documents were static, that is, the sets C_v did not change over time. Here, we show that any changes to these sets can be efficiently reflected in our index. This is more formally stated in the following proposition:

PROPOSITION 11. *If a word ω is added to (or removed from) C_v, for some $v \in V$, the index can be updated in $O(h \log l(\omega))$ time to incorporate this insertion (or deletion).*

PROOF. To update the index, we only need to update the queues $PMI[i, L_i[v], \omega]$ $(0 \leq i < h)$, by adding (or removing) v with priority $D_i[v]$. Updating the queue $PMI[i, L_i[v], \omega]$ takes $O(\log l_{i,L_i[v]}(\omega)) = O(\log l(\omega))$ time. Hence, the total update time is $O(h \log l(\omega))$. □

Choosing the parameters r, k as in corollary 2, we see that the update time is just $\tilde{O}(1)$. Hence, our index can be updated very quickly as soon as any of the documents in the network gets modified. This wraps up the analysis of our scheme. We will now discuss several interesting extensions.

3.1 Extensions

Directed Graphs: So far, we assumed the social graph G to be undirected. However, our scheme can be extended to directed graphs, though with no theoretical approximation factor guarantees. Our experiments show our scheme also works very well for directed graphs.

The sketching algorithm, presented in Algorithm 1, gets modified such that instead of computing $L_i[u], D_i[u]$ using a single BFS, at line 5, we compute $L^{\mathrm{O}}_i[u], D^{\mathrm{O}}_i[u]$ via a BFS along incoming edges, and $L^{\mathrm{i}}_i[u], D^{\mathrm{i}}_i[u]$ via a BFS along outgoing edges. We can then use the quantities $L^{\mathrm{i}}_i[u], D^{\mathrm{i}}_i[u]$ at indexing time and the quantities $L^{\mathrm{O}}_i[u], D^{\mathrm{O}}_i[u]$ at query time to obtain a heuristic solution for directed graphs. We omit the details of the implementation from this version; simulation results show that this heuristic works well in practice.

Combining Personalization with Other Relevance Measures: So far, we focused on ranking the search results only based on their distance to the querying node. However, in practice a combination of distance and other relevance measures is used to rank the results. These relevance measures can be text-based scores such as tf-idf [17], link-based authority scores such as PageRank [21], or, in a real-time setting (where more recent results are of more interest) the recency of the document. Here, we show how our scheme can be extended to allow for elegantly combining all such measures with the distance-based personalization, without any change in space or time efficiency.

Assume that associated with each $v \in V$ and $\omega \in C_v$ is a score $\alpha_v(\omega)$ (a real number), and hence the following combined score is used to rank search results:

$$s_{u,\omega}(v) = \lambda d(u, v) + (1 - \lambda)\alpha_v(\omega)$$

For a query (u, ω, J), we need to find the J nodes $v \in I(\omega)$ with the smallest values of $s_{u,\omega}(v)$. Here, $\lambda \in [0, 1]$ is a weight trading off between distance-based personalization and document-based scores, and in practice is learned from the data to optimize the search quality. Replacing the exact distance with its approximation, the following approximate scores can be used:

$$\tilde{s}_{u,\omega}(v) = \lambda \tilde{d}(u, v) + (1 - \lambda)\alpha_v(\omega)$$

However, from equation 2.1 we have:

$$\tilde{s}_{u,\omega}(v) = \min\{\lambda D_i[u] + (\lambda D_i[v] + (1 - \lambda)\alpha_v(\omega))\}$$

Where, as before, min is over $\{0 \leq i < h | \, L_i[u] = L_i[v]\}$. To rank based on this score, we modify the indexing Algorithm 2 such that at line 5, v is inserted into $PMI[i, L_i[v], \omega]$ with priority

$$\pi_v(\omega) = \lambda D_i[v] + (1 - \lambda)\alpha_v(\omega)$$

Also, we modify the search Algorithm 3 such that the priority of each $x^{p_i}_{i,L_i[u]}(\omega)$ in H is

$$\lambda D_i[u] + \pi_v(\omega) = \lambda D_i[u] + \lambda D_i[v] + (1-\lambda)\alpha_v(\omega)$$

Then, a similar analysis as in theorem 7 shows that these modified algorithms, rank the results based on $\tilde{s}_{u,\omega}(v)$. The space and time complexities of these algorithms are also exactly the same as Algorithms 2, 3.

EXAMPLE 12. *The scores $\alpha_v(\omega)$ can represent a whole range of document-based scores. Here, we consider the real-time search scenario, where associated with each node $v \in V$ and word $\omega \in C_v$ is a timestamp $t_v(\omega)$ representing the time instance at which the word ω was added to C_v, and upon receiving a query (u, ω, J) at time t, we would like to not only personalize the results but also bias the results towards the more recent documents.*

At the time of query, the recency of ω on $v \in I(\omega)$, is $t - t_v(\omega)$ (note that $t_v(\omega) \leq t$, as ω is already in C_v when the query arrives). Hence, we would like to rank the results based on $\lambda d(u,v) + (1-\lambda)(t - t_v(\omega))$. Since t is independent of v, ranking based on this score is exactly the same as ranking based on $\lambda d(u,v) + (1-\lambda)(-t_v(\omega))$. Hence, letting $\alpha_v(\omega) = -t_v(\omega)$, we can use the framework explained above to do the search and ranking. This together with the possibility of quick incremental index updates explained earlier in the paper (which lets each new word $\omega \in C_v$ to be indexed as soon as it arrives, i.e., at time $t_v(\omega)$), allows for a real-time personalized social search system.

Distributed Implementation: In order to scale up our scheme to today's huge social networks, one would want to implement it in a distributed fashion. Since finding the sketches, using Algorithm 1, only requires a number of BFS's, it can easily adopt a distributed implementation, e.g., using MapReduce [16]. Hence, we focus on implementing the rest of the scheme in a distributed fashion, on an *Active DHT*. Note that the offline index construction can be regarded as a sequence of word additions. So, if real-time updates can be done efficiently, the offline phase can be done efficiently as well. Hence, we will first focus on efficient distributed implementation of query and update algorithms. Later, we will show that the offline phase can be done even more efficiently than through a sequence of real-time updates.

For a distributed implementation of our scheme, we need to shard both the distance sketches and the index entries across a number of machines in an Active DHT, using appropriate (Key, Value) pairs. As pointed out above, we would like to shard in a way that not only the loads (in terms of space) on different machines are balanced, but also answering queries or updating the index can be done with little network usage, i.e., both few network accesses and small amount of communications. We will show that sharding the distance sketch using the id of the querying social graph node as the Key, and the inverted index using the word ω as the Key, satisfies all these properties, and results in surprising efficiency bounds.

To formalize this, we consider the following architecture: we have one master machine, which interfaces the outside world, and a set of M machines, labeled $0, 1, \ldots, M-1$, which can be used to distribute the data structures. We will use two hash functions $f : V \to [M]$, $g : \cup_v C_v \to [M]$ (where $[M] = \{0, 1, \ldots, M-1\}$) to distribute our data structures as follows:

- The entry $E[u]$ of the distance sketch is kept on machine $f(u)$
- For any $\omega \in \cup_v C_v$, all the entries $PMI[i, x, \omega]$ of the index, where $0 \leq i < h, x \in S_i$, are kept on machine $g(\omega)$

We assume f, g to be random hash functions. We will further assume that the reverse index corresponding to any word ω is much smaller than the amount of memory at any compute node[2]. Then, a simple Chernoff bound [19] shows that, with high probability, the load (i.e., space used) on each machine is $\Theta(\frac{h(n+|C|)}{M})$. Hence, the load is very well balanced across different machines. Also, note that choosing r, k as in corollary 2, this is just $\tilde{\Theta}((n+|C|)/M)$, which is close to what would be needed to only distribute the corpus across the machines. Next, we show that answering queries and updating the index can be done with little network usage.

At query time, when the master machine receives a query (u, ω, J), it will first retrieve $E[u]$ by accessing the machine $f(u)$ once. Note that, by Algorithm 3, the top J results for the query are definitely in the set

$$\{x^j_{i,L_i[u]}(\omega) |\, 0 \leq i \leq h-1, 1 \leq j \leq J\}$$

Hence, after retrieving $E[u]$, the master machine can retrieve the above set by sending the query along with $\{L_i[u] |\, 0 \leq i \leq h-1\}$ to machine $g(\omega)$. Having retrieved this set, the master machine can then just run Algorithm 3 to find and rank the search results. Hence, the total number of network accesses and the total amount of communication needed to answer the query are, respectively, 2 and $O(Jh)$. Note that choosing r, k as in corollary 2 bounds the total amount of communication at $\tilde{O}(J)$, which is only slightly more than what would be needed to just communicate the search results (i.e. $\Omega(J)$)! This implementation can be done on top of a Distributed Hash Table such as memcached. Further improvements can be obtained by assuming that the DHT is *Active*; in this case, the set $E[u]$ can be directly communicated to the compute node $g(\omega)$ which will perform the search operation, resulting in a total network transfer of $O(J+h)$.

Next, we consider the required network usage to update the index. If a word ω is added to or deleted from the document at node $u \in V$, i.e. C_u, then to update the index, first $E[u]$ is retrieved from machine $f(u)$, and then u and ω are sent along with $E[u]$ to machine $g(\omega)$, which can then insert or delete u into or from all the queues $PMI[i, L_i[u], \omega]$ $(0 \leq i < h)$. Hence, the total number of network accesses and the total amount of communication required to update the index are, respectively, 2 and $O(h)$. Choosing r, k as in corollary 2 then bounds the total amount of communication at $\tilde{O}(1)$.

As mentioned above, offline index construction can be regarded as a sequence of index updates. Hence, directly using the above update scheme, the offline phase can be done with

[2]This assumption is only for a clean illustrative statement of the results; we can fan out the index for ω into multiple nodes at the expense of an extra network call if needed.

a total of $2|C|$ network accesses, and $O(h|C|)$ communications. However, by accessing the sketch of each node only once, the offline phase can be done even more efficiently: for each node u, we retrieve $E[u]$ by communicating with machine $f(u)$ once, and then for each word $\omega \in C_u$, we send u, ω, and $E[u]$ to machine $g(\omega)$ to be indexed. Hence, the offline phase can be done with only $n + |C|$ network accesses and $O(h|C|)$ total communications, which reduces to $\tilde{O}(|C|)$ communications, by choosing r, k as in corollary 2.

4. EXPERIMENTS

We experimented with our scheme to study its quality and efficiency in practice, specially in comparison with the benchmarks from the related literature. In this section, we present the algorithms, datasets, and the methodology used in our experiments, as well as their results.

4.1 Algorithms

As explained in section 2, landmark-based distance approximation, together with the baseline search scheme, has been proposed as a solution to the social search problem, in multiple previous works in the literature [22, 29]. Thus, in our experiments, we compared the quality of our scheme with the landmark-based scheme. The simplest way of selecting landmarks is by picking them randomly from the graph. However, Potamias et al. [22] study different landmark selection methods, show that they influence the quality of the approximations, and state that a centrality-based method, in which the nodes with the highest values of closeness centrality (i.e., smallest average distance to all nodes) are selected as landmarks, works best across all their experiments. Therefore, in addition to the random landmark selection method, we also implemented this centrality-based method, and used both as benchmarks to compare the quality of our scheme against.

For efficiency, we compared our scheme with that of the baseline scheme (explained in section 2) using the same oracle as our scheme. This comparison will show the effect of our partitioned multi-index structure on the efficiency of finding and ranking the search results (as compared to using a simple inverted index). We used $r = \lfloor \log_2 n \rfloor$ for our scheme in all the experiments.

4.2 Datasets

We experimented with four networks, two undirected and two directed, two synthetic and two from real-world data. Table 1 summarizes the networks that we used.

	Undirected	Directed
Synthetic	Grid	ForestFire
Real-world	Undirected Twitter	Directed Twitter

Table 1: Networks used in the experiments.

We now explain each of these networks. The grid network we constructed was an 11-dimensional grid with side length 3. Associated with each node was a single word chosen uniformly at random from a dictionary of 1000 words. This network had $4^{11} > 4$M nodes and around 70M edges.

The ForestFire network, which had more than 1M nodes and around 2.5M edges, was generated using the ForestFire model [15], known to model many of the features of real world networks. Similar to the grid network, we associated each node with a single word chosen uniformly at random from a dictionary of 1000 words.

The undirected Twitter network was a sample of more than 4M nodes from the social network Twitter, and all the reciprocated edges between them. The resulting sampled network had more than 100M edges. We associated with each node the words in the bio and the screen name of the corresponding user.

The directed Twitter network was the giant connected component of a sample of the social network Twitter. The resulting graph had over 4M nodes and more than 380M edges. Similar to the undirected case, we associated with each node the words in the bio and the screen name of the corresponding user.

The samples of the twitter graph were not chosen uniformly at random, and the two samples are not the same, since a random sample would allow inference about the density of the Twitter network which Twitter considers confidential. Also, as explained below, our experiments methodology has the interesting feature that the evaluations are completely automated and do not require any human inspection of the search results, adding an additional layer of privacy and confidentiality.

4.3 Experiments Methodology and Results

We performed experiments studying the quality and the efficiency of our scheme. Here, we present the methodology used in these experiments as well as their results. Before performing the experiments with each of our networks, we preprocessed the network, and constructed, for each node v, a subset $C'_v \subseteq C_v$ of its associated words. For our synthetic networks (having only a single word associated with each node), we simply let $C'_v = C_v$. For the real-world networks (from Twitter), after computing, for each word ω, the frequency (i.e., the fraction) of the nodes v having $\omega \in C_v$, we removed the 100 words with the largest frequencies, as stop words. Then, for each node v, we let C'_v to be the set composed of the following three words: the lowest frequency non-stop word on v, the highest frequency non-stop word on v, and a random non-stop word on v. The sets C'_v were going to later get used for constructing queries (as we will explain below), so we wanted to make sure, by including representatives from low-frequency, high-frequency, and randomly selected non-stop words, that our constructed queries would cover a wide range of possibilities.

After this preprocessing, for each experiment, we generated a number of queries. Each of these queries, q, was constructed as follows: We first picked a length $l^q \in \{2, 3\}$, and a random node u^q from the graph. Then, we performed a random walk starting at u^q for l^q steps, to arrive at a node v^q, and then we picked a random word ω^q from C'_{v^q}. Then, a query for word ω^q was issued by node u^q. In each experiment, for half the queries, we used $l^q = 2$, and for the other half, we used $l^q = 3$. Each of these queries, in accordance with the random walk based intuition behind PageRank [21], simulates the behavior of a random social network user starting at his own page, browsing through random links for a few steps, finding an interesting document, and then later searching for it in the hopes of finding the same page or even closer pages (in terms of social graph proximity) related to that document.

Having explained the query generation method used in all our experiments, we now explain, in further detail, each of our experiments as well as their results.

Quality Experiments: For each network, we generated a set Q of 1000 queries, as explained above, and found the top J results, with $J = 1, 5, 10$, using our scheme, random landmark scheme, and central landmark scheme. For our scheme, we chose, as mentioned earlier in this section, $r = \lfloor \log_2 n \rfloor$, and let k to take all the values from 1 to 10. For each k, when comparing with the landmark-based schemes, we selected $k(r+1)$ landmarks, so they had the same preprocessing time and space as our scheme (ignoring the load of centrality computations for the central landmarks scheme).

For each scheme, finding the top J search results $\{v_j^q\}_{1 \le j \le J}$ for each query q, we considered the set of failed queries to be:

$$F = \{q \in Q | \, d(u^q, v_j^q) > d(u^q, v^q) \, \forall \, 1 \le j \le J\}$$

Then, denoting, for each $q \in Q - F$, the depth of the first good result as:

$$j^q = \min\{1 \le j \le J | \, d(u^q, v_j^q) \le d(u^q, v^q)\}$$

we computed the fraction of failed queries (FFQ) and the average depth of the first good result (ADFGR) as our quality measures:

$$\text{FFQ} = \frac{|F|}{|Q|}, \; \text{ADFGR} = \frac{\sum_{q \in Q-F} j^q}{|Q - F|}$$

Clearly, one would ideally like to have:

$$\text{FFQ} = 0, \; \text{ADFGR} = 1$$

in which case, hundred percent of the queries get a good answer in the very first search result. Our experiments show that our scheme actually gets very close to these ideals. The fraction of failed queries in our experiments with our scheme and the landmark-based schemes, for $J \in \{1, 5, 10\}$, is presented in Figures 4.1, 4.2. These figures show that our scheme consistently outperforms both landmark-based schemes across all the networks, and for all the values of J. Also, we note that selecting the landmarks using centralities did not help the landmark-based scheme, and often even lowered its quality (as measured by FFQ). Furthermore, we note that increasing the number of seed sets (by increasing k) consistently improved the quality of our scheme, while increasing the number of landmarks usually did not help much with the quality of the landmark-based schemes.

The results for ADFGR are also similar for different values of J, and hence we present them only for $J = 10$ in Figure 4.3. We see that across all networks, our scheme performs better than the landmark-based schemes. This, together with the results for FFQ, shows that not only our scheme finds good answers to queries much more frequently, but also it does a much better job in ranking those good results higher in the list of results.

Efficiency Experiments: We compared the efficiency of our scheme against the benchmark provided by the baseline scheme explained in section 2. To do so, we generated a set of 20000 queries as explained earlier in this section. Letting $r = \lfloor \log_2 n \rfloor$, we generated the seed sets defining the approximate distance oracle. Since the efficiencies of both our scheme and the baseline scheme are nearly linear in k, we used $k = 1$ in our efficiency experiments. Then, for our

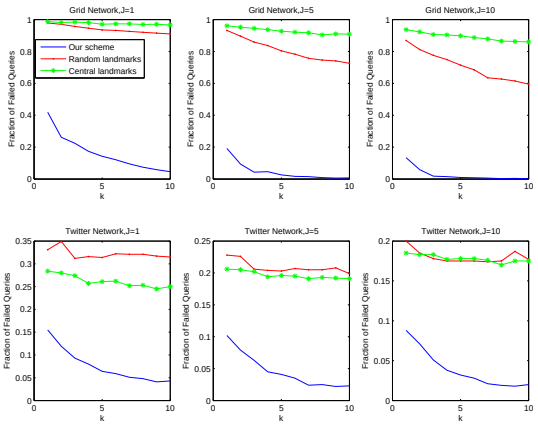

Figure 4.1: Fraction of failed queries for undirected networks

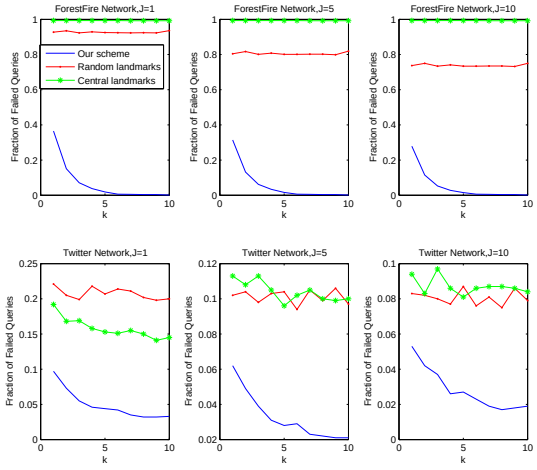

Figure 4.2: Fraction of failed queries for directed networks

scheme, we constructed the corresponding partitioned multi-index, and for the baseline scheme we constructed a simple inverted index of the whole network. Finally, using the constructed indices, we found the top 10 results for each query by each scheme.

As efficiency measures, we measured the total preprocessing (sketching plus indexing) time, as well as the total query time (over 20000 queries) for each scheme. The results are presented in Tables 2, 3. As can be observed from these tables, even though the baseline scheme takes, of course, less preprocessing time, our scheme is still very efficient at preprocessing time. Note that unlike query time which, in practice, has a harsh deadline of few milliseconds, offline preprocessing time is much more flexible.

The real strength of our scheme is then evident from the query time results (Table 3), where our scheme is significantly (i.e., depending on the network, 20 to 60 times) more efficient than the baseline scheme, and is insensitive to the size of the network, as predicted by our theoretical analyses.

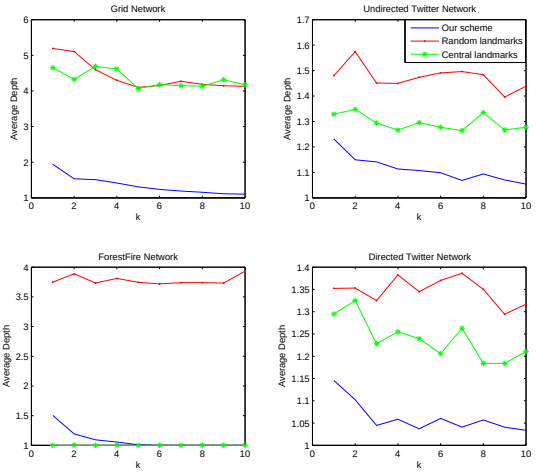

Figure 4.3: Average depth of the first good result

	Our schme	Baseline
Grid Network	58	18
Undirected Twitter Network	930	71
ForestFire Network	74	5
Directed Twitter Network	1384	163

Table 2: Total preprocessing time (sec).

5. REFERENCES

[1] S. Bao, G. Xue, X. Wu, Y. Yu, B. Fei, and Z. Su. Optimizing web search using social annotations. In *WWW '07*, pages 501–510.

[2] C. Böhm, E. Kny, B. Emde, Z. Abedjan, and F. Naumann. Sprint: ranking search results by paths. In *EDBT/ICDT '11*, pages 546–549.

[3] J. Bourgain. The metrical interpretation of superreflexivity in banach spaces. *Israel J. of Mathematics*, 56(2):222–230, 1986.

[4] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. In *CIKM '09*, pages 1227–1236.

[5] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in o(mn) time. In *SODA '06*, pages 514–523.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33:273–321, September 2001.

[7] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *SODA '02*, pages 937–946.

[8] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *WSDM '10*, pages 401–410.

[9] D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM J. Comput.*, 29:1740–1759, March 2000.

[10] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *SODA '05*, pages 156–165.

[11] L. Gou, X. L. Zhang, H.-H. Chen, J.-H. Kim, and C. L. Giles. Social network document ranking. JCDL '10, pages 313–322.

[12] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28:517–580.

[13] D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *WWW '10*, pages 431–440.

[14] J. Kleinberg, A. Slivkins, and T. Wexler. Triangulation and embedding using small sets of beacons. *J. ACM*, 56:32:1–32:37, September 2009.

[15] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD '05*, pages 177–187.

[16] J. Lin and C. Dyer. *Data-intensive text processing with mapreduce*, Morgan and Claypool, 2010.

[17] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15:9–17, January.

[19] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[20] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. *Proceedings of the International Conference on Data Mining Workshops*, pages 170–177, 2010.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[22] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM '09*, pages 867–876.

[23] E. V. Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recogn. Lett.*, 4:145–157, July 1986.

[24] R. Seidel. On the all-pairs-shortest-path problem. In *STOC '92*, pages 745–749.

[25] M. Shapiro. The choice of reference points in best-match file searching. *Commun. ACM*, 20:339–343, May 1977.

[26] P. Singla and M. Richardson. Yes, there is a correlation: - from social networks to personal behavior on the web. In *WWW '08*, pages 655–664.

[27] L. Tang and M. Crovella. Virtual landmarks for the internet. In *IMC '03*, pages 143–152.

[28] M. Thorup and U. Zwick. Approximate distance oracles. In *STOC '01*, pages 183–192.

[29] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *CIKM '07*, pages 563–572.

[30] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *Proc. VLDB Endow.*, 1:710–721.

[31] P. Yin, W.-C. Lee, and K. C. Lee. On top-k social web search. In *CIKM '10*, pages 1313–1316.

[32] U. Zwick. Exact and approximate distances in graphs - a survey. In *ESA '01*, pages 33–48.

	Our schme	Baseline
Grid Network	2	39
Undirected Twitter Network	1	61
ForestFire Network	2	44
Directed Twitter Network	2	63

Table 3: Total query time (sec) over 20000 queries.