Less is More: Sampling the Neighborhood Graph Makes SALSA Better and Faster

Marc Najork
Microsoft Research
Mountain View, CA, USA
najork@microsoft.com

Sreenivas Gollapudi
Microsoft Research
Mountain View, CA, USA
sreenig@microsoft.com

Rina Panigrahy
Microsoft Research
Mountain View, CA, USA
rina@microsoft.com

ABSTRACT

In this paper, we attempt to improve the effectiveness and the efficiency of query-dependent link-based ranking algorithms such as HITS, MAX and SALSA. All these ranking algorithms view the results of a query as nodes in the web graph, expand the result set to include neighboring nodes, and compute scores on the induced neighborhood graph. In previous work it was shown that SALSA in particular is substantially more effective than query-independent link-based ranking algorithms such as PageRank. In this work, we show that whittling down the neighborhood graph through consistent sampling of nodes and edges makes SALSA and its cousins both faster (more efficient) and better (more effective). We offer a hypothesis as to why "less is more", *i.e.* why using a reduced graph improves performance.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process, selection process*

General Terms

Algorithms, Measurement, Experimentation

Keywords

HITS, MAX, SALSA, link-based ranking, retrieval performance, web search

1. INTRODUCTION

One of the fundamental problems in Information Retrieval is the *ranking problem*: how to arrange the documents that satisfy a query into an order such that the documents most relevant to the query rank first. Traditional ranking algorithms proposed by the pre-web IR community were mostly based on similarity measures between the terms (words) in the query and the documents satisfying the query.

In addition to structured text, web pages also contain hyperlinks between web pages, which can be thought of as peer endorsements between content providers. Marchiori suggested early on to leverage incoming hyperlinks as another feature in ranking algorithms [12], and the simplistic idea of merely counting in-links quickly evolved into more sophisticated link-based ranking algorithms that take the quality of an endorsing web page into account.

Link-based ranking algorithms can be grouped into two classes: query-independent ones such as in-link count or Google's famous PageRank [17], and query-dependent ones such as Kleinberg's HITS [7, 8] and Lempel & Moran's SALSA [9, 10]. The aforementioned algorithms were described in seminal papers that inspired a great deal of subsequent work; however, there has been little published work on the effectiveness (that is, the accuracy of the ranking) of these algorithms. A recent study [15] using a 17-billion edge web graph and a set of 28,043 queries with partially judged results concluded that SALSA, a query-dependent link-based ranking algorithm, is substantially more effective than HITS, PageRank and in-degree, although it is not as effective as the state-of-the-art textual ranking algorithm.

HITS, SALSA and similar algorithms extend the result set of a query to a neighborhood graph, and then compute scores on that graph. In this paper, we study how the method for selecting the vertices and edges in the neighborhood graph affects the effectiveness of the ranking algorithm. In particular, we study the effect of using consistent sampling to reduce both the number of vertices and of edges in the graph. We find that reducing the size of the graph in this way not only improves efficiency (which is to be expected, since the graph is smaller), but also effectiveness. Our experiments were performed on the same data set that was used in previous studies [14, 15, 5, 16].

The remainder of this paper is structured as follows: section 2 describes the data sets and effectiveness measure used in this study; section 3 reviews the HITS, MAX and SALSA algorithms; section 4 introduces a method for fixing neighborhood graphs using consistent sampling of the neighboring vertices of each result. Section 5 describes a space-efficient representation of the approximate neighborhood of each vertex in the web graph, which can be computed off-line and then used at query time to compute scores. SALSA computed on the approximate neighborhood graph is more effective than SALSA computed on the neighborhood graph definitions of sections 3 and 4. We investigate this phenomenon in section 6, resulting in two new definitions of neighborhood graphs. Section 7 offers a hypothesis as to why "less is more"; *i.e.* why sampling the neighborhood graph improves effectiveness. Finally, section 8 offers concluding remarks.

WSDM 2009 Barcelona, Spain

2. DATA SETS AND EFFECTIVENESS MEASURE

The experimental evaluations of the algorithms described in this paper are based on the two data sets used in previous work [14, 15]: a large web graph and a substantial set of queries with associated results, some of which were labeled by human judges.

The web graph was obtained by performing a breadth-first search web crawl that retrieved 463,685,607 pages. These pages contain 17,672,011,890 hyperlinks (after eliminating duplicate links embedded in the same web page), which refer to a total of 2,897,671,002 distinct URLs. The mean out-degree of a crawled web page is 38.11; the mean in-degree of discovered pages (whether crawled or not) is 6.10. The graph was not "cleaned up" in any way; in particular, no attempts were made to identify and remove spam web pages.

Our query set was produced by sampling 28,043 queries from the Live Search query log, and retrieving a total of 66,846,214 result URLs for these queries, or about 2,838 results per query on average. It should be pointed out that our web graph covers only 9,525,566 pages or 14.25% of the result set. 485,656 of the results in the query set (about 17.3 results per query) were rated by human judges as to their relevance to the given query using a six point scale, the ratings being "definitive", "excellent", "good", "fair", "bad", and "detrimental". Results were selected for judgment based on their commercial search engine placement; in other words, the subset of labeled results is biased towards documents considered relevant by pre-existing ranking algorithms. Our performance measure treats unlabeled results as "detrimental". Spot-checking the set of unlabeled results suggests that this assumption is indeed reasonable. In the following, given a rank-ordered vector of n results, let $rat(i)$ be the rating of the result at rank i, with 5 being "definitive" and 0 being "detrimental" or "unlabeled".

We use the *normalized discounted cumulative gain* [6] as a measure of ranking effectiveness.[1] NDCG is a non-binary, graded measure that considers all documents in the result set, but discounts the contribution of low-ranking documents. It is normalized to range between 0 and 1; higher values indicate better performance. NDCG is actually a family of performance measures. In this study, we used the following instantiation: We define the *discounted cumulative gain* at document cut-off value k to be:

$$DCG@k = \sum_{i=1}^{k} \frac{1}{\log(1+i)} \left(2^{rat(i)} - 1\right)$$

The *normalized discounted cumulative gain* $NDCG@k$ of a scored result set is defined to be the $DCG@k$ of the result set rank-ordered according to the scores divided by the $DCG@k$ of the result set rank-ordered by an "ideal" scoring function, one that rank-orders results according to their rating. The interested reader is referred to [15, 13] for more detailed definitions of NDCG and other measures.

[1] In addition to NDCG, we also evaluated all the algorithms described in this paper using the *average precision* and the *reciprocal rank* measures. We did not include these results for reasons of space, and using these alternative measures would not lead to any qualitatively different conclusions.

3. A REVIEW OF HITS-LIKE RANKING ALGORITHMS

The World Wide Web consists of web pages interconnected by hyperlinks. Web pages can be viewed as vertices in a graph, and hyperlinks as directed edges from one page to another. Marchiori [12] suggested that a hyperlink pointing to a page may be viewed as an endorsement of that page, and that one could therefore use the in-degree of a web page (the number of hyperlinks referring to it) as an estimate of its quality. Page *et al.* [17] refined this very basic notion of link-based endorsement to take both the out-degree and the quality of referring pages into account – referrals from high-quality pages provide stronger endorsement, and pages split their endorsement ability among the pages they refer to.

Both in-link count and PageRank are query-independent features: Their value is independent of any particular query, and they are estimates of page quality. In contrast, Jon Kleinberg's *Hypertext Induced Topic Selection* (HITS) algorithm computes an estimate of the relevance of a web page to a given query; in other words, it is a query-dependent feature. HITS (and the many algorithms that sprang from it) takes the set of documents that satisfy a query (the *result set*) as input, extracts a *neighborhood graph* from the full web graph consisting of the result set vertices and (some of) their immediate neighbors, and (some of) the edges that connect them, and computes relevance scores for the results based on the neighborhood graph.

Putting it more succinctly, given a web graph (V, E) with vertex set V and edge set $E \subseteq V \times V$ and a result set $R \subseteq V$ of vertices (web pages or documents) that satisfy the query, algorithms in the HITS family perform two steps:

1. Fix a neighborhood graph (V_R, E_R) with vertex set V_R where $R \subseteq V_R \subseteq V$) and edge set E_R where $E_R \subseteq E$ and $E \subseteq V_R \times V_R$.
2. Compute relevance scores for the vertices in R using the neighborhood graph (V_R, E_R).

In order to describe steps (1) and (2) of HITS and its cousins MAX and SALSA more formally, it is helpful to introduce some notation first:

Predecessor and successor sets: Given a vertex $v \in V$, we define its predecessor (or "in-linker") set to be the set of vertices that link to v:

$$I(v) = \{u \in V : (u, v) \in E\}$$

Likewise, we define v's successor (or "out-linker") set to be the set of vertices that v links to:

$$O(v) = \{u \in V : (v, u) \in E\}$$

Uniform random sampling: Given a set X, we write $\mathcal{U}_n(X)$ to denote a set of n elements drawn uniformly at random from X; $\mathcal{U}_n(X) = X$ if $|X| \leq n$.

In Kleinberg's original formulation of HITS, step (1) of the above meta-algorithm (fixing the neighborhood graph) was defined by the following method, which we will call UR(a) ("Uniform random sampling"):

$$V_R = \bigcup_{u \in R} \{u\} \cup \mathcal{U}_a(I(u)) \cup O(u)$$

$$E_R = \{(u, v) \in E : u \in V_R \wedge v \in V_R\}$$

Kleinberg also suggested to only include "transverse edges" into E_R, *i.e.* hyperlinks referring to web pages on a different web site. In previous work [14, 15], we explored how different definitions of what constitutes a transverse edge affect the effectiveness of HITS and SALSA. In this paper, we consider an edge to be transverse if the hyperlink refers to a web page in a different domain than the referring page, and we assume that the edge set E only contains transverse edges.

In the above definition, the neighborhood vertex set V_R depends not only on the result set R, but also on the sampling parameter a. Kleinberg's original motivation for sampling the predecessor set was to limit the size of the neighborhood graph: highly popular web pages may have millions of hyperlinks pointing to them, which would lead to very large neighborhood graphs. By the same line of reasoning, Kleinberg did not consider sampling the successor set, since typical web pages have a fairly manageable number of embedded hyperlinks. In the experiments described in the HITS paper [8], a was set to 50.

In step (2), the HITS algorithm computes two scores for each page v in the result set R: an *authority score* estimating the relevance of v with respect to the query that gave rise to R, and a *hub score* estimating whether v is a good *hub*: a page that links to many highly relevant pages. Conventionally, the equations for authority and hub score are presented in a mutually recursive fashion (and the algorithm is sometimes called a *mutual reinforcement algorithm*). However, it is possible to define the two scores independently of each other. In previous work [14, 15], we showed that HITS and SALSA hub scores are not useful for ranking purposes; therefore, we only define authority scores. Given a neighborhood graph (V_R, E_R), HITS computes an authority score $s(v)$ for each $v \in V_R$ as follows:

1. For all $u \in V_R$ do $s(u) := \sqrt{\frac{1}{|V_R|}}$.
2. Repeat until score vector s converges:
   (a) For all $u \in V_R$ do $s'(u) := \sum_{(v,u)\in E_R} \sum_{(v,w)\in E_R} s(w)$
   (b) For all $u \in V_R$ do $s(u) := \frac{1}{\|s'\|_2} s'(u)$

where $\|s\|_2$ denotes the ℓ^2 (euclidean) norm of the score vector s.

MAX [18] uses the same definition of neighborhood graph as HITS does, but modifies the recurrence relation for hub and authority scores as follows:

1. For all $u \in V_R$ do $s(u) := 1$.
2. Repeat until score vector s converges:
   (a) For all $u \in V_R$ do $s'(u) := \sum_{(v,u)\in E_R} max_{(v,w)\in E_R} s(w)$
   (b) For all $u \in V_R$ do $s(u) := \frac{1}{\|s'\|_\infty} s'(u)$

where $\|s\|_\infty$ denotes the ℓ^∞ norm of the score vector s.

SALSA [9] uses the same definition of neighborhood graph as HITS and MAX, but computes authority scores by performing a random walk on the graph. The *authority walk* commences on a node with in-degree greater than 0; each step in the walk consists of following an edge in the backward direction and then following an edge in the forward direction (this transition may return to the starting vertex).

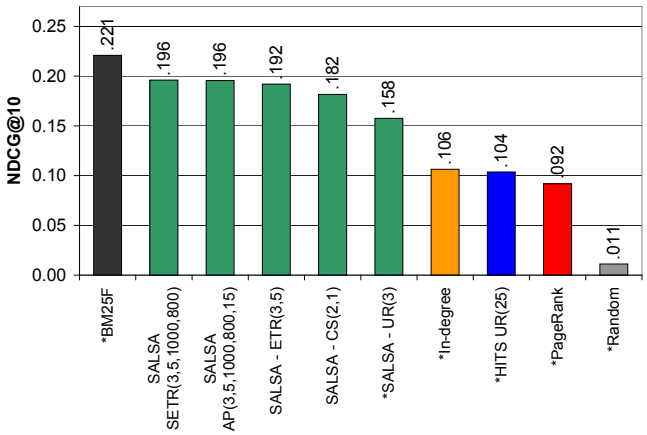

Figure 1: Effectiveness in terms of NDCG@10 of BM25F, PageRank, web page in-degree, "classic" HITS, and SALSA using various different definitions of neighborhood graph.

The authority score vector s is the stationary probability distribution of the authority walk. This leads us to the following algorithm:

0. Let V_R^A be $\{u \in V_R : in(u) > 0\}$
1. For all $u \in V_R$ do $s(u) := \begin{cases} \frac{1}{|V_R^A|} & \text{if } u \in V_R^A \\ 0 & \text{otherwise} \end{cases}$
2. Repeat until s converges:
   (a) For all $u \in V_R^A$ do $s'(u) := \sum_{(v,u)\in N} \sum_{(v,w)\in N} \frac{s(w)}{out(v)in(w)}$
   (b) For all $u \in V_R^A$ do $s(u) := s'(u)$

In previous work [14, 15], we used the data sets described above in section 2 to evaluate the effectiveness of the HITS and SALSA scoring functions using Kleinberg's original definition of neighborhood graph, and compared their effectiveness to web page in-degree (ignoring all intra-domain links) and PageRank. In this paper, we take the algorithm for computing scores on the neighborhood graph as a given, and instead explore different definitions of neighborhood graphs. Figure 1 reviews the findings of these earlier papers (the starred bars), and previews the results of this paper. While SALSA using the original definition of neighborhood graph outperformed "classic" HITS as well as web page in-degree and PageRank, SALSA using our improved definitions of neighborhood graph leads to another substantial improvement, making it by far the best link-based feature we are aware of, and coming quite close to BM25F [19], the state-of-the-art textual feature.

4. FIXING THE NEIGHBORHOOD VERTEX SET USING CONSISTENT SAMPLING

HITS, MAX and SALSA authority scores expose a "co-citation" relationship: scores are propagated from a vertex w to a vertex u by virtue of a third vertex v that links to ("co-cites") both u and w. We postulate that a "good" method for sampling vertices into the neighborhood graph

Table 1: Effectiveness of SALSA-CS(a,b) in terms of NDCG@10, varying a and b between 0 and 10

$a \setminus b$	0	1	2	3	4	5	6	7	8	9	10
0	0.1707	0.1715	0.1717	0.1710	0.1701	0.1692	0.1681	0.1672	0.1662	0.1652	0.1645
1	0.1728	0.1800	0.1800	0.1793	0.1789	0.1784	0.1778	0.1771	0.1764	0.1758	0.1752
2	0.1762	**0.1816**	0.1813	0.1807	0.1801	0.1797	0.1795	0.1791	0.1787	0.1783	0.1778
3	0.1751	0.1807	0.1805	0.1803	0.1799	0.1798	0.1795	0.1794	0.1793	0.1789	0.1787
4	0.1733	0.1798	0.1798	0.1797	0.1795	0.1793	0.1793	0.1791	0.1790	0.1790	0.1787
5	0.1718	0.1791	0.1796	0.1795	0.1792	0.1791	0.1793	0.1791	0.1791	0.1789	0.1789
6	0.1702	0.1788	0.1790	0.1789	0.1789	0.1788	0.1790	0.1789	0.1788	0.1785	0.1785
7	0.1688	0.1785	0.1788	0.1789	0.1788	0.1788	0.1790	0.1790	0.1789	0.1789	0.1787
8	0.1674	0.1782	0.1786	0.1788	0.1787	0.1786	0.1788	0.1788	0.1787	0.1786	0.1784
9	0.1659	0.1774	0.1784	0.1785	0.1785	0.1784	0.1785	0.1785	0.1783	0.1783	0.1782
10	0.1648	0.1771	0.1778	0.1779	0.1780	0.1779	0.1779	0.1780	0.1779	0.1778	0.1777

should preserve existing co-citation relationships. In order to preserve co-citation relationships, that sampling method must preserve *set similarity*. The similarity between two sets A and B can be expressed in terms of their similarity (Jaccard) coefficient, defined to be the cardinality of their intersection divided by the cardinality of their union:

$$\sigma(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Note that $\sigma(A, B)$ is 0 if the two sets are completely disjoint, and 1 if they are identical.

Unfortunately, the uniform random sampling used by the original HITS, MAX and SALSA algorithms does not preserve set similarity: $\sigma(\mathcal{U}_n(A),\mathcal{U}_n(B))$ is typically smaller than $\sigma(A, B)$. Therefore, we investigated another sampling technique that has the desired property of preserving set similarity: *consistent unbiased sampling* [2]. Consistent unbiased sampling is deterministic; that is, when sampling n elements from a set X, we will always draw the same n elements. Moreover, any element x that is sampled from set A is also sampled from subset $B \subset A$ if $x \in B$. We write $\mathcal{C}_n(X)$ to denote a consistent unbiased sample of n elements from set X, where $\mathcal{C}_n(X) = X$ if $|X| < n$. One method to compute a consistent sample is *min-hashing* [2].

Definition: Let U denote the universal set. Given a set $A \subseteq U$ and a permutation $\pi : [|U|] \rightarrow [|U|]$, we define the min-hash $MH_\pi(A)$ to be $argmin_x\{\pi(x)|x \in A\}$. Essentially, $MH_\pi(A)$ is the element in A whose value in the permutation π is the minimum. Alternatively, let f be a hash function that maps elements from the universe U to a real number randomly and uniformly in the interval $[0, 1]$. Then $MH_f(A) = argmin_x\{f(x)|x \in A\}$. Therefore, $MH_f(A)$ is the element in A whose hash value into the interval $[0, 1]$ is minimum. Under the above definition of a min-hash, one can show that similar sets are likely to produce the same consistent sample. Formally, given two sets A and B, $\Pr[MH_\pi(A) = MH_\pi(B)] = \sigma(A, B)$ [2].

Note that we can use min-hashing multiple (say, n) times by using different independent permutations $\pi_1, \ldots, \pi_n$ (or different independent hash functions $f_1, \ldots, f_n$) each time to produce n samples. Thus, we can realize consistent sampling as follows:

$$\mathcal{C}_n(A) = \{MH_{\pi_1}(A), \ldots, MH_{\pi_n}(A)\}$$

Alternately, we could pick the elements corresponding to the n smallest hash values from a single permutation. Under either realization, $\mathcal{C}_m(A) \subseteq \mathcal{C}_n(A)$ for all $m \leq n$.

Using consistent unbiased sampling, we now define a method CS(a, b) ("consistent sampling") for performing step 1 of our meta-algorithm for query-dependent ranking:

$$V_R = \bigcup_{u \in R} \{u\} \cup \mathcal{C}_a(I(u)) \cup \mathcal{C}_b(O(u))$$

$$E_R = \{(u, v) \in E : u \in V_R \wedge v \in V_R\}$$

Table 1 shows the effectiveness (in terms of NDCG@10) of using CS for step 1 (fixing the neighborhood graph) and SALSA for step 2 (computing scores on that graph). Three points are worth noting: First, SALSA-CS performs substantially better than SALSA-UR (as evaluated in [15]). Second, effectiveness does not increase monotonically as we increase the number a of sampled predecessors per result, but rather peaks at fairly low values of a. Third, the same is true for the number b of sampled successors per result. In fact, performance is maximal for $a = 2$ and $b = 1$.

The idea of consistently sampling the neighbors of each vertex in the web graph has been used in other contexts, for example in discovering dense subgraphs of the web graph [4].

5. APPROXIMATING NEIGHBORHOOD GRAPHS USING BLOOM FILTERS

When performed on a web-scale corpus, HITS, MAX and SALSA require a substantial amount of query time processing. Much of this processing is attributable to the computation of the neighborhood graph. The reason for this is that the entire web graph is enormous. A document collection of five billion web pages induces a set of about a quarter of a trillion hyperlinks. Storing this web graph on disk would make lookup operations unacceptably slow due to the inherent seek time limitations of hard drives. On the other hand, the graph is too big to be stored in the main memory of any single machine; therefore, it has to be partitioned across many machines. In such a setup, the cost of a link lookup is governed by the cost of a remote procedure call (RPC). A sophisticated implementation of HITS-like algorithms against a distributed link database will batch many lookup operations into a single RPC request to reduce latency and will query all link servers in parallel, but even so it will require two rounds of concurrent requests: the first round to extend the result set to a base set by sampling predecessors and successors of each result; and the second round to determine the edges induced by the base set. The

implementation used to perform the experiments described in [15] required 235 milliseconds per query for computing SALSA-UR(3). Over 90% of the time spent was spent on performing the RPC calls to the link servers in order to assemble the neighborhood graph, as opposed to computing scores on that graph.

The fact that HITS-like algorithms incur substantial computational cost at query-time puts them at a crippling disadvantage to query-independent algorithms such as PageRank: according to Marissa Mayer, Google's VP of Search Products & User Experience, delaying Google's response time by half a second led to a 20% drop in query traffic (and revenue) from the user population subjected to the delay [11].

In earlier work, we presented a technique to substantially lower the query-time cost of HITS-like algorithms [5]. We do so by moving the most expensive part of the computation off-line. At index-construction time, we build a database mapping web page URLs to summaries of their neighborhoods. At query time, we rank the results satisfying a query by looking up each result in the summary database (an operation that requires only one round of RPCs, as opposed to two), approximating the neighborhood graph of the result set based on the neighborhood summaries of the constituent results, and computing authority scores using that approximation of the neighborhood graph. As we will see, this approximation has no detrimental effect on the quality of the ranking; in fact, surprisingly the approximation-based algorithms are more effective than the original ones. As we will show later, this is because the neighborhood graph constructed from the summaries differs from the graph according to the CS method in two aspects, and each of these aspects impacts ranking effectiveness.

Our summarization technique employs Bloom filters. A *Bloom filter* [1] is a space-efficient probabilistic data structure that can be used to test the membership of an element in a given set; the test may yield a false positive but never a false negative. A Bloom filter represents a set using an array A of m bits (where $A[i]$ denotes the ith bit), and uses k hash functions h_1 to h_k to manipulate the array, each h_i mapping some element of the set to a value in [1,m]. To add an element e to the set, we set $A[h_i(e)]$ to 1 for each $1 \leq i \leq k$; to test whether e is in the set, we verify that $A[h_i(e)]$ is 1 for all $1 \leq i \leq k$. Given a Bloom filter size m and a set size n, the optimal (false-positive minimizing) number of hash functions k is $\frac{m}{n} \ln 2$; in this case, the probability of false positives is $(\frac{1}{2})^k$. For an in-depth description of Bloom filters, the reader is referred to [3]. In the following, we will write $BF_k[X]$ to denote the Bloom filter representing the set X and using k hash functions.

At index construction time, we compute an approximate summary of each web page in our corpus and store it in a summary server. The summary of web page u consists of tuple $\langle EI(u), EO(u), BI(u), BO(u)\rangle$, where $EI(u) = \mathcal{C}_a(I(u))$ (an explicitly stored set of a consistently sampled predecessors of u), $EO(u) = \mathcal{C}_b(O(u))$ (an explicitly stored set of b consistently sampled successors of u), $BI(u) = BF_k[\mathcal{C}_c(I(u))]$ (a Bloom filter containing c consistently sampled predecessors of u), and $BO(u) = BF_k[\mathcal{C}_d(O(u))]$ (a second Bloom filter containing d consistently sampled successors of u). We represent the explicitly stored predecessors and successors using 64-bit numbers that uniquely identify a URL. We refer to this approximation as method AP(a,b,c,d,k).

At query time, given a result set R, we first look up the

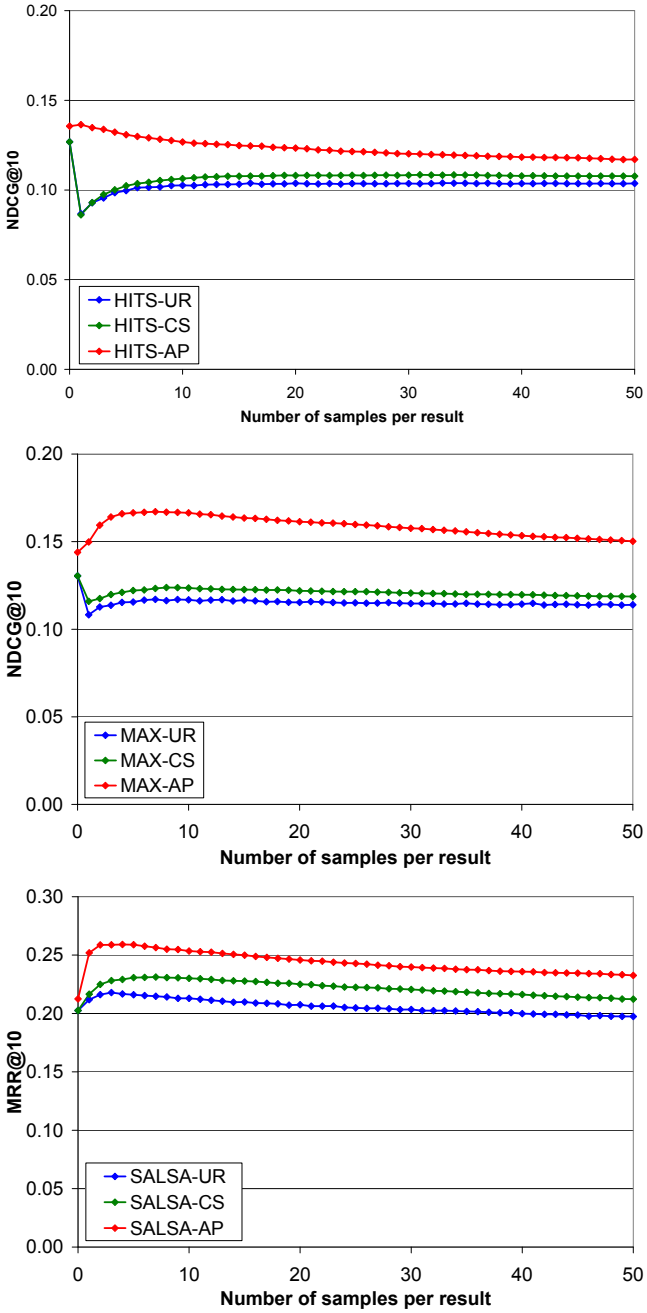

Figure 2: Effectiveness of HITS, MAX, and SALSA authority scores in terms of NDCG@10, using UR(n), CS(n,n) and AP(n,n,1000,1000,10) neighborhoods and varying n between 0 and 50.

Table 2: Effectiveness of SALSA-AP(a,b,1000,1000,10) in terms of NDCG@10, varying a and b between 0 and 10

$a \setminus b$	0	1	2	3	4	5	6	7	8	9	10
0	0.1408	0.1541	0.1546	0.1536	0.1523	0.1510	0.1498	0.1484	0.1472	0.1464	0.1453
1	0.1676	0.1731	0.1733	0.1723	0.1714	0.1700	0.1689	0.1677	0.1666	0.1656	0.1646
2	0.1748	0.1793	0.1794	0.1786	0.1779	0.1770	0.1761	0.1753	0.1744	0.1735	0.1727
3	0.1776	0.1816	0.1817	0.1812	0.1807	0.1800	0.1793	0.1785	0.1779	0.1773	0.1767
4	0.1784	0.1822	0.1827	0.1824	0.1821	0.1815	0.1810	0.1805	0.1800	0.1796	0.1791
5	0.1787	0.1827	0.1830	0.1827	0.1824	0.1821	0.1817	0.1812	0.1809	0.1805	0.1803
6	0.1783	0.1822	**0.1831**	0.1828	0.1826	0.1823	0.1819	0.1816	0.1812	0.1811	0.1809
7	0.1777	0.1819	0.1828	0.1827	0.1826	0.1823	0.1822	0.1820	0.1819	0.1816	0.1813
8	0.1773	0.1813	0.1823	0.1822	0.1821	0.1821	0.1820	0.1817	0.1816	0.1814	0.1811
9	0.1768	0.1804	0.1814	0.1814	0.1814	0.1813	0.1815	0.1812	0.1811	0.1810	0.1809
10	0.1764	0.1800	0.1809	0.1811	0.1813	0.1812	0.1813	0.1811	0.1811	0.1809	0.1808

summaries for all the results in R. Having done so, we fix the neighborhood vertex set as follows:

$$V_R = \bigcup_{u \in R} \{u\} \cup EI(u) \cup EO(u)$$

Then, we compute the neighborhood edge set E_R as follows: For each vertex $u \in R$ and each vertex $v \in V_R$, we perform two tests: If $BI(u)$ contains v, we add an edge (v, u) to the graph; if $BO(u)$ contains v, we add an edge (u, v) to the graph. More formally:

$$E_R = \bigcup_{u \in R, v \in V_R} \{(u, v) : v \in BO(u)\} \cup \{(v, u) : v \in BI(u)]\}$$

Observe that the AP neighborhood graph differs from the CS neighborhood graph in three ways:

- We do not use exact set representations for $\mathcal{C}_c(I(u))$ and $\mathcal{C}_d(O(u))$, but approximate them by using Bloom filters. This introduces additional edges whose number depends on the false positive probability of the Bloom filter. Using k hash functions, we add about $2^{-k+1}|V_R||R|$ spurious edges in the graph.
- The graph determined by the CS method may contain edges where neither endpoint is part of R; the graph determined by AP contains no such edges.
- Furthermore, the AP graph only contains edges from $V_R \cap \mathcal{C}_c(I(u))$ to $u \in R$ and from $u \in R$ to $V_R \cap \mathcal{C}_d(O(u))$, as opposed to all edges between vertices in R and V_R.

Our first goal was to show that AP neighborhood summarizations are indeed suitable for HITS-like link analysis algorithms. To this end, we conducted a series of experiments where we compared the AP method to UR and CS. In order to reduce the dimensionality of AP's parameter space, we fixed three of them and set the remaining two to equal values. Specifically, we used $k = 10$ Bloom filter hash functions, and we fixed the parameters c and d at 1000 (that is, we included a sample of up to 1000 predecessors or successors into each Bloom filter), and we varied a and set $b = a$.

We measured the effectiveness of the HITS, MAX and SALSA scoring methods for the UR, CS and AP methods of determining the neighborhood graph. Figure 2 depicts the results. The figure contains three graphs, one for each method of computing scores (HITS, MAX, and SALSA). The horizontal axis shows a sampling parameter n (ranging from 0 to 50); the vertical axis shows the retrieval performance in terms of NDCG@10. Each graph contains three curves; the blue (dark) curve showing the performance of UR(n); the green (medium) curve showing the performance of CS(n,n); and the red (light) curve showing that of AP(n,n,1000,1000,10). Using CS instead of UR neighborhood graphs substantially improves the performance. However, using the AP method outperforms both UR and CS. Furthermore, we can make the following observations:

- HITS-UR and HITS-CS perform best when no backlinks are sampled. They drop off sharply when one backlink per result is sampled, and then improve gradually as more backlinks are sampled, leveling out at about 10 samples per result. HITS-AP performs best for $a = b = 0$; effectiveness decreases gradually as a and b are increased.
- MAX-UR and MAX-CS resemble HITS in that they perform best when no backlinks are sampled. They drop off when one backlink is sampled (but less so than HITS), improve gradually as more backlinks are sampled per result, peaking at around 8 samples per result and then dropping off again. However, the efficiency of MAX-AP does not decrease monotonically as more samples are drawn (as is the case for HITS-AP), but rather peaks at between 5 and 7 samples per result.
- SALSA-UR and SALSA-CS perform best for around 7 to 8 sampled backlinks per result; they do not show the same unexpected spike at 0 samples that is evident for HITS and MAX. SALSA-AP performs best for a and b between 4 and 5.

Table 3: Effectiveness of SALSA-AP(6,2,c,d,10) in terms of NDCG@10, varying c and d between 600 and 1000

$c \setminus d$	600	700	800	900	1000
600	0.1814	0.1814	0.1814	0.1814	0.1814
700	0.1819	0.1819	0.1820	0.1820	0.1819
800	0.1823	0.1823	0.1824	0.1823	0.1823
900	0.1825	0.1825	0.1826	0.1825	0.1825
1000	0.1831	0.1831	**0.1831**	0.1831	0.1831

Table 4: Effectiveness of SALSA-AP(a,b,1000,800,15) in terms of NDCG@10, varying a and b between 0 and 10

$a \setminus b$	0	1	2	3	4	5	6	7	8	9	10
0	0.1651	0.1723	0.1730	0.1731	0.1730	0.1725	0.1723	0.1721	0.1718	0.1715	0.1713
1	0.1743	0.1870	0.1896	0.1905	0.1907	0.1906	0.1905	0.1904	0.1902	0.1899	0.1896
2	0.1798	0.1914	0.1941	0.1946	0.1949	0.1949	0.1947	0.1947	0.1945	0.1942	0.1940
3	0.1806	0.1920	0.1948	0.1952	0.1955	**0.1956**	0.1955	0.1954	0.1952	0.1951	0.1948
4	0.1807	0.1919	0.1947	0.1951	0.1954	0.1954	0.1954	0.1954	0.1954	0.1951	0.1950
5	0.1801	0.1915	0.1942	0.1948	0.1951	0.1951	0.1951	0.1949	0.1947	0.1946	0.1947
6	0.1795	0.1907	0.1935	0.1943	0.1946	0.1946	0.1945	0.1946	0.1944	0.1942	0.1942
7	0.1787	0.1902	0.1928	0.1938	0.1938	0.1940	0.1940	0.1940	0.1940	0.1939	0.1937
8	0.1778	0.1894	0.1921	0.1929	0.1931	0.1931	0.1932	0.1934	0.1933	0.1931	0.1930
9	0.1769	0.1886	0.1913	0.1923	0.1925	0.1925	0.1925	0.1927	0.1925	0.1924	0.1924
10	0.1762	0.1879	0.1907	0.1919	0.1920	0.1921	0.1921	0.1921	0.1922	0.1920	0.1920

On our web graph, computing neighborhood graph summaries using AP(5,5,1000,1000,10) leads to summaries with an average size of 379 bytes in size (40 bytes for EI, 40 bytes for EO, 227 bytes for BI, and 72 bytes for BO).

Next, we focused on SALSA, which performed best among the three scoring methods, and we explored which parameter settings yield the highest effectiveness. First, we explored the impact of the number of predecessors and successors stored explicitly. We fixed c and d (the number of samples included in each Bloom filter) at 1000, k (the number of Bloom filter hash functions) at 10, and we varied a and b (the number of explicitly stored neighbors) between 0 and 10. Table 2 shows the effectiveness of these parameterizations in terms of NDCG@10. Given the choices of c, d, and k, performance is maximal at $a = 6$ and $b = 2$ with an NDCG@10 value of 0.1831.

Second, we explored the impact of the number of neighbors included in each Bloom filter. We kept k at 10, fixed the parameters a and b at 6 and 2 (the choices corresponding to the highest NDCG values in the previous experiment) and varied c and d between 600 and 1000. Table 3 shows the effectiveness of these parameterizations in terms of NDCG@10. We observed the highest NDCG value for $c = 1000$ and $d = 800$; however, the difference between the NDCG values in the bottom row of the table is not statistically significant, and it appears that performance improves as c and d increase.

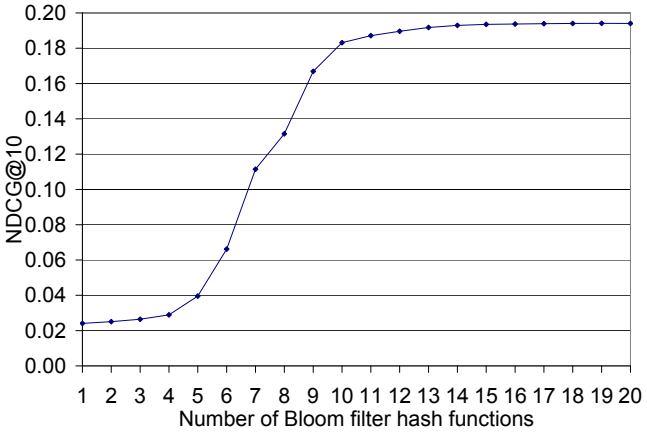

Figure 3: Relationship between k (the number of Bloom filter hash functions) and effectiveness.

Third, we explored the impact of the number of Bloom filter hash functions on the ranking effectiveness. Figure 3 shows the outcome of these experiments. As the figure illustrates, our previous choice of $k = 10$ was not optimal; the NDCG values continue to increase for higher values of k, asymptotically converging at around $k = 15$.

Having established that using 15 Bloom filter hash functions leads to higher NDCG values, we repeated the set of experiments corresponding to table 2. We fixed k at 15, c at 1000, and d at 800; and we varied a and b between 0 and 10. Table 4 shows the effectiveness of these parameterizations in terms of NDCG@10. Given the choices of c, d, and k, performance is maximal at $a = 3$ and $b = 5$ (*i.e.* different values than the optimal choices in table 2) with an NDCG@10 value of 0.1956.

Finally, we repeated the experiments described in table 3 using the improved settings of k, a and b. We kept k at 15, fixed the parameters a and b at 3 and 5 (the parameters leading to the highest NDCG@10 in the previous experiment) and varied c and d between 800 and 1200. Table 5 shows the results. We observed the highest NDCG@10, at 0.1960, for $c = 1000$ and $d = 1200$. However, as in table 3, the difference between the NDCG values in the bottom row of the table is not statistically significant, and it appears that performance improves as c and d increase.

In a production setting, neighborhood summaries would be computed offline (at index construction time) and retrieved online (at query time). In order to achieve acceptable query time performance, summaries would need to be kept in main memory, possibly in a distributed setting. The number of servers required could be substantial. For exam-

Table 5: Effectiveness of SALSA-AP(3,5,c,d,15) in terms of NDCG@10, varying c and d between 800 and 1200

$c \setminus d$	800	900	1000	1100	1200
800	0.1953	0.1953	0.1954	0.1954	0.1953
900	0.1954	0.1954	0.1954	0.1954	0.1954
1000	0.1956	0.1955	0.1956	0.1955	0.1955
1100	0.1958	0.1958	0.1959	0.1958	0.1958
1200	0.1959	0.1959	**0.1960**	0.1959	0.1959

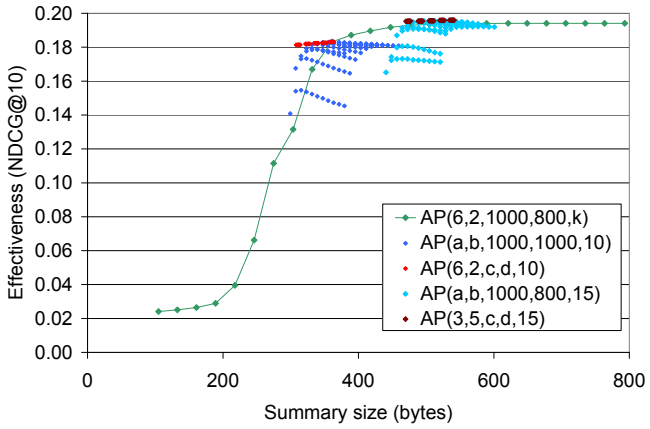

Figure 4: Relationship between effectiveness (in terms of NDCG@10) and feature vector size for various parameterizations of approximate SALSA.

ple, if one summary consumed 400 bytes, then a summary database covering a web corpus of 10 billion pages would require over 250 servers with 16 GB of RAM each. Therefore, it is interesting to explore the trade-off between summary size and ranking effectiveness.

Figure 4 shows the relationship between memory consumption and effectiveness. Each point in the graph corresponds to one parameterization of SALSA-AP; its horizontal position indicates the size of each summary vector (in bytes), its vertical position indicates the effectiveness measured as NDCG@10. As we saw above, the dominant factor governing the effectiveness of SALSA-AP is the number of Bloom filter hash functions. The set of green points connected by a line shows the impact of varying k from 1 to 20. The blue clouds of points visualizes various choices of a and b for $k = 10$ and $c = d = 1000$ (dark blue points) and for $k = 15$, $c = 1000$ and $d = 800$ (light blue points). Note that the choice of a and b not only affects the summary size, but also has a noticeable impact on effectiveness. By contrast, varying c and d (shown by the red and maroon point clouds) affects the memory consumption, but has little impact on effectiveness.

Our current implementation of these summarization-based algorithms does not yet employ a distributed summary server; we use our distributed link server (the *Scalable Hyperlink Store*) to compute summaries. However, since a summary server is similar to, and indeed simpler than, a link server, we can draw on our experiences with the latter to estimate what the query-time performance of a production system would be. To this end, we measured the performance of our current implementation, subtracted the time required to compute the summaries, and added the time we have observed for retrieving a vector of links of the size of an average summary from our link server. These measurements suggest that it would take us an average of 96 milliseconds per query to compute SALSA-AP authority scores. This compares favorably to the 235 milliseconds per query of our implementation of the original SALSA algorithm. Moreover, we have not spent any time on optimizing the code for constructing AP neighborhood graphs, and believe that further speedups are possible.

6. OMITTING NON-RESULT EDGES

As mentioned in the previous section, the neighborhood graphs extracted by the AP method differ from those extracted by the CS method in three ways. It is reasonable to assume that one or more of these differences are responsible for SALSA-AP being more effective than SALSA-CS. Figure 3 suggests that the first difference – the probabilistic nature of Bloom filters – is not the cause, since the effectiveness of SALSA-AP increases as the number of Bloom filter hash functions is increased and thereby the false-positive rate is decreased.

The second difference concerns the edge set. The CS method will include any edge $(u, v) \in E$ as long as both u and v are part of the neighborhood vertex set V_R. The AP method is more selective: At least one of the endpoints u and v must be in the result set R.

This leads us to the ETR ("edges touch result") method of computing the neighborhood graph:

$$V_R = \bigcup_{u \in R} \{u\} \cup \mathcal{C}_a(I(u)) \cup \mathcal{C}_b(O(u))$$

$$E_R = \{(u, v) \in E : (u \in V_R \wedge v \in R) \vee (u \in R \wedge v \in V_R)\}$$

Table 6 shows the effectiveness of the ETR method combined with the SALSA scoring algorithm. The NDCG@10 is maximal at 0.1920 for $a = 3$ and $b = 5$, and the effectiveness is higher than SALSA-CS (maximal NDCG@10 of 0.1816; see table 1), but not as high as SALSA-AP (maximal NDCG@10 of 0.1956; see table 4). In other words, much but not all of the difference between SALSA-CS and SALSA-AP can be attributed to the latter omitting edges that don't touch results.

The third difference distinguishing ETR neighborhood graphs from AP neighborhood graphs is that the former method admits any edge that connects a vertex in R and a vertex in V_R, whereas the latter method samples the edges. To test whether the superior performance of SALSA-AP can be attributed to this edge sampling, we define a method SETR ("sampled edges touch result") for determining the neighborhood graph:

$$V_R = \bigcup_{u \in R} \{u\} \cup \mathcal{C}_a(I(u)) \cup \mathcal{C}_b(O(u))$$

$$\begin{aligned} E_R = \{(u, v) \in E \quad : \quad & (v \in R \wedge u \in V_R \wedge u \in \mathcal{C}_c(I(v))) \\ \vee \quad & (u \in R \wedge v \in V_R \wedge v \in \mathcal{C}_d(O(u)))\} \end{aligned}$$

Table 7 shows the effectiveness of the SETR method combined with the SALSA scoring algorithm. The NDCG@10 is maximal at 0.1961 for $a = 4$ and $b = 5$, and the effectiveness slightly surpasses that of SALSA-AP. In other words, SALSA-AP approximates SALSA-SETR, and slightly suffers from the false positives due to the probabilistic nature of Bloom filters.

It is worth noting that SALSA-SETR is not only effective, but also very efficient. Our implementation (which uses the Scalable Hyperlink Store, a distributed system that partitions the web graph across many SHS servers, with each server maintaining a portion of the graph in main memory) requires about 78 milliseconds to score a query, and we believe that we could reduce this cost further by reducing the number of rounds of RPC calls down to one.

Table 6: Effectiveness of SALSA-ETR(a,b) in terms of NDCG@10, varying a and b between 0 and 10

$a \setminus b$	0	1	2	3	4	5	6	7	8	9	10
0	0.1707	0.1713	0.1714	0.1709	0.1705	0.1700	0.1696	0.1688	0.1684	0.1678	0.1675
1	0.1728	0.1814	0.1845	0.1859	0.1864	0.1864	0.1863	0.1863	0.1861	0.1859	0.1858
2	0.1765	0.1860	0.1894	0.1905	0.1911	0.1911	0.1911	0.1911	0.1909	0.1908	0.1907
3	0.1766	0.1868	0.1902	0.1914	0.1919	**0.1920**	0.1919	0.1919	0.1917	0.1916	0.1915
4	0.1763	0.1865	0.1900	0.1912	0.1916	0.1915	0.1915	0.1913	0.1914	0.1913	0.1911
5	0.1760	0.1863	0.1898	0.1910	0.1916	0.1916	0.1915	0.1915	0.1915	0.1914	0.1913
6	0.1755	0.1860	0.1890	0.1901	0.1909	0.1910	0.1909	0.1911	0.1910	0.1909	0.1909
7	0.1751	0.1858	0.1889	0.1902	0.1908	0.1910	0.1909	0.1912	0.1911	0.1910	0.1907
8	0.1749	0.1855	0.1887	0.1900	0.1903	0.1906	0.1906	0.1907	0.1907	0.1906	0.1905
9	0.1743	0.1849	0.1882	0.1893	0.1897	0.1900	0.1900	0.1902	0.1902	0.1900	0.1900
10	0.1740	0.1849	0.1876	0.1892	0.1894	0.1899	0.1898	0.1900	0.1900	0.1898	0.1899

Table 7: Effectiveness of SALSA-SETR(a,b,1000,800) in terms of NDCG@10, varying a and b between 0 and 10

$a \setminus b$	0	1	2	3	4	5	6	7	8	9	10
0	0.1686	0.1726	0.1736	0.1738	0.1737	0.1736	0.1732	0.1730	0.1726	0.1724	0.1722
1	0.1742	0.1865	0.1897	0.1907	0.1909	0.1909	0.1908	0.1906	0.1904	0.1901	0.1900
2	0.1797	0.1911	0.1939	0.1947	0.1950	0.1950	0.1949	0.1948	0.1945	0.1944	0.1942
3	0.1804	0.1920	0.1948	0.1958	0.1960	0.1961	0.1960	0.1959	0.1959	0.1957	0.1955
4	0.1804	0.1920	0.1947	0.1956	0.1960	**0.1961**	0.1960	0.1960	0.1958	0.1956	0.1954
5	0.1802	0.1916	0.1946	0.1954	0.1958	0.1958	0.1959	0.1958	0.1958	0.1956	0.1955
6	0.1794	0.1911	0.1938	0.1948	0.1953	0.1953	0.1952	0.1954	0.1953	0.1951	0.1952
7	0.1786	0.1905	0.1936	0.1945	0.1947	0.1949	0.1949	0.1950	0.1950	0.1947	0.1946
8	0.1783	0.1899	0.1930	0.1939	0.1940	0.1942	0.1941	0.1944	0.1942	0.1941	0.1941
9	0.1776	0.1893	0.1926	0.1934	0.1937	0.1937	0.1938	0.1940	0.1939	0.1938	0.1938
10	0.1769	0.1890	0.1919	0.1931	0.1932	0.1935	0.1934	0.1936	0.1936	0.1933	0.1934

7. WHY LESS IS MORE

In the previous sections, we showed that SALSA is more effective if we do not consider the entire distance-one neighborhood graph of the result set, but instead use consistent sampling to limit both the number of vertices and the number of edges in the neighborhood graph. It is surprising and somewhat counterintuitive that sampling some vertices and edges should be better than considering them all. In order to understand this seeming paradox, it is helpful to consider what vertices and edges are more likely to be sampled.

The CS, AP, ETR and SETR methods all determine the neighborhood vertex set in the same way:

$$V_R = \bigcup_{u \in R} \{u\} \cup \mathcal{C}_a(I(u)) \cup \mathcal{C}_b(O(u))$$

Given a result vertex v, the probability that a particular predecessor u of v is selected as one of the a samples in $\mathcal{C}_a(I(u))$ is $\frac{a}{|I(u)|}$ (or 1 if $a \geq |I(u)|$). So, the chances of v being sampled are inversely proportional to the in-degree (popularity) of u. Moreover, a vertex v that links to several vertices in R has multiple chances of being sampled as an predecessor. The sampling parameter a controls how pronounced this effect is. In the extreme case of $a \to \infty$, all predecessors u of each result v will be in the neighborhood vertex set; the fact that v has low in-degree (is unpopular) or that u links to multiple $v \in R$ has no impact on the selection. At the other extreme, if a is very low, linking to multiple unpopular results greatly increases an predecessor's chances of being selected.

Similarly, the chances of a successor v of a result u being sampled are inversely proportional to the out-degree of u (its selectivity), and v's chances improve if it is linked to by multiple results. A vertex v that is being linked to by multiple selective results has better chances of being sampled as a successor. The sampling parameter b controls how pronounced this effect is; it is pronounced for low values of b and diminishes as b increases.

As we just remarked, CS, AP, ETR and SETR all compute the neighborhood vertex set in the same way; moreover, as we showed in this paper, they are most effective when parameterized with low values of a and b (between 1 and 6). So, web pages that link to many unpopular pages or are being linked to by many selective pages are much more likely to be part of the neighborhood graph than other pages. It is easy to understand why receiving a link from a selective (low out-degree) web page is more meaningful than a link from a nonselective (high out-degree) page. We postulate that links *to* unpopular pages are also more meaningful: the referring web page endorses something that is not globally popular, and endorsements of things outside of the mainstream are more meaningful.

The ETR method, which is substantially more effective than CS, differs from CS only in its edge section policy: The neighborhood edge set of CS includes all edges in E that are covered by $V_R \times V_R$, while the ETR policy restricts the set to edges in E that are covered by $(V_R \times R) \cup (R \times V_R)$ — in other words, it discards edges that are not connected to the result set. We explain this phenomenon as follows: Web pages in the result set are likely to be topically related

to one another, since they by definition are the results to a query, *i.e.* contain the query terms. So, edges that touch the result set are likely to also be topically related to the query. Edges between vertices in $V_R \setminus R$, on the other hand, are less likely to be topical, since neither of their endpoints contains the query terms. Discarding such off-topic edges improves the efficiency of SALSA-ETR.

Finally, SALSA is yet more efficient if the graph was fixed using the SETR instead of the ETR method. SETR differs from ETR in that for every result vertex v, it samples c of the incoming and d of the outgoing edges, and includes those sampled edges that connect to another vertex in V_R. So, one can think of ETR as an instance of SETR: ETR(a,b) is the same as SETR(a,b,∞,∞). As we showed, SETR(3,5,1000,800) outperforms ETR(3,5), so sampling edges instead of admitting them all is indeed beneficial.

The likelihood that an edge (u, v) from $u \in V_R$ to $v \in R$ is included in the graph is $\frac{c}{|I(v)|}$. So, edges leading to unpopular results are more likely to be sampled. Likewise, edges from selective results are more likely to be included as well. This effect is more pronounced for lower values of c and d. As the results of the previous section show, a moderate bias towards such edges improves ranking effectiveness. Our explanation for this phenomenon is along the same line as the argument for why CS works better for low sampling values: endorsements *by* selective web pages are obviously more meaningful, and endorsements *of* unpopular web pages are also more meaningful since they go against the mainstream opinion.

8. CONCLUSION

In this paper, we propose several definitions of neighborhood graphs for query-dependent link-based ranking algorithms such as HITS, MAX and SALSA, and study their impact on the effectiveness of these algorithms. Specifically, we observe that one can improve the effectiveness of such ranking algorithms by incorporating only a small subset of the neighbors of each result into the neighborhood graph, using consistent sampling as the vertex selection method. Effectiveness is improved even further by avoiding edges that do not touch result vertices, and using consistent sampling to select a subset of the remaining eligible edges. Using a sampled down and thus smaller neighborhood graph not only improves the effective of HITS-like ranking algorithms, but also their efficiency, primarily because less data needs to be transmitted by our distributed hyperlink server. We offer a hypothesis as to why "less is more", *i.e.* why using a sampled-down graph provides better ranking effectiveness.

9. REFERENCES

[1] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13(7):422–426, 1970.

[2] A. Broder, M. Charikar, A. Frieze, M. Mitzenmacher. Min-Wise Independent Permutations. *Journal of Computer and System Sciences* 60(3):630–659, 2000.

[3] A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics* 1(4):485–509, 2005.

[4] D. Gibson, R. Kumar, A. Tomkins. Discovering Large Dense Subgraphs in Massive Graphs. In *31st International Conference On Very Large Data Bases*, pages 721–732, 2005.

[5] S. Gollapudi, M. Najork and R. Panigrahy. Using Bloom Filters to Speed Up HITS-like Ranking Algorithms. In *5th Workshop on Algorithms and Models for the Web Graph*, pages 195–201, 2007.

[6] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[7] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.

[8] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[9] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKC effect. *Computer Networks and ISDN Systems*, 33(1–6):387–401, 2000.

[10] R. Lempel and S. Moran. SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems*, 19(2):131–160, 2001.

[11] G. Linden. Marissa Mayer at Web 2.0. Online at: *http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html*

[12] M. Marchiori. The quest for correct information on the Web: Hyper search engines. In *Computer Networks and ISDN Systems*, 29(8–13):1225–1236, 1997.

[13] F. McSherry and M. Najork. Computing Information Retrieval Performance Measures Efficiently in the Presence of Tied Scores. In *30th European Conference on Information Retrieval*, pages 414–421, 2008.

[14] M. Najork, H. Zaragoza and M. Taylor. HITS on the Web: How does it Compare? In *30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 471–478, 2007.

[15] M. Najork. Comparing the Effectiveness of HITS and SALSA In *16th ACM Conference on Information and Knowledge Management*, pages 157–164, 2007.

[16] M. Najork and N. Craswell. Efficient and Effective Link Analysis with Precomputed SALSA Maps. In *17th ACM Conference on Information and Knowledge Management*, pages 53–61, 2008.

[17] 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.

[18] P. Tsaparas. Using Non-Linear Dynamical Systems for Web Searching and Ranking. In *23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 59–70, 2004.

[19] H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and S. Robertson. Microsoft Cambridge at TREC–13: Web and HARD tracks. In *13th Text Retrieval Conference*, 2004.