SPRINT: Ranking Search Results by Paths

Christoph Böhm[1], Eyk Kny[2], Benjamin Emde[2], Ziawasch Abedjan[1], Felix Naumann[1]
Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
[1]firstname.lastname@hpi.uni-potsdam.de [2]firstname.lastname@student.hpi.uni-potsdam.de

ABSTRACT

Graph-structured data abounds and has become the subject of much attention in the past years, for instance when searching and analyzing social network structures. Measures such as the shortest path or the number of paths between two nodes are used as proxies for similarity or relevance[1]. These approaches benefit from the fact that the measures are determined from some context node, e.g., "me" in a social network. With SPRINT, we apply these notions to a new domain, namely ranking web search results using the link-path-structure among pages.

SPRINT demonstrates the feasibility and effectiveness of Searching by Path Ranks on the INTernet with two use cases: First, we **re-rank intranet search results** based on the position of the user's homepage on the graph. Second, as a live proof-of-concept we **dynamically re-rank Wikipedia search results** based on the currently viewed page: When viewing the Java software page, a search for "Sun" ranks Sun Microsystems higher than the star at the center of our solar system. We evaluate the first use case with a user study. The second use case is the focus of the demonstration and allows users to actively test our system with any combination of context page and search term.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Search and Retrieval

General Terms

Performance, Experimentation

Keywords

Directed Graph, Shortest Path, Ranking

1. INTRODUCTION

The idea of using the link structure within a set of web pages is not new; Google's PageRank is a prominent example. However, such rankings are determined globally, often independent of the search term, but, more importantly, independent of some context page: A computer science student searching for "dean" is most likely interested in the dean of the computer science department and not of the political sciences department. The page of the former should be ranked higher than that of the latter. This context could be simulated by extending/refining the search to "dean computer sciences". If the search engine knew, however, the departmental homepage of the student, this context could be computed automatically.

As in [3] the underlying hypothesis of this work is that a page in the web graph can represent such context. As opposed to [3], we specifically examine the eligibility of path lengths among page as a sole criterion (independent of textual page content) to represent query contexts, i.e., the students homepage is simply "closer" to the page of the dean of computer sciences rather than any other dean.

These ideas have been successfully used in the context of social networks, where users are interconnected in friendship-graphs [4]. We extend this approach to deal with large *directed web graphs* such as intranets or Wikipedia. A main challenge of these ideas is to efficiently determine these measures: Even in intranets, the number of nodes and edges is enormous. Therefore, exhaustively analyzing paths in the extremely short time allowance of online search is impossible. We tested two techniques to overcome this difficulty.

A first, and obvious technique is to precalculate closeness scores for all pairs of context pages and web pages. In the intranet scenario, we can set homepages or department pages as context pages. The number of such context pages is equal to or less than the number of users, and thus pre-calculation and updates after new crawls are feasible. In a scenario where a context page should not be fixed in advance, as is the case for Wikipedia search or documentation pages search, precalculation is prohibitively expensive. Thus, we make use of a second technique, namely using seed nodes throughout the graph and precalculating distances only to such dedicated nodes [6].

SPRINT demonstrates the surprisingly effective results of re-ranking Wikipedia search results based on a context page. Search queries are first sent to the original keyword-based Wikipedia search engine, but its results are intercepted and re-ranked based on their distance to the page that was viewed by the user when issuing the query. For instance, when viewing any software or programming page on Wikipedia and searching for "python", the page for the Python scripting language is automatically ranked before any other Python page, such as the animal, the movie or the British comedy troupe. Re-ranking is performed online with only very little overhead. The offline seed selection is based on [5] but is optimized to reduce seed selection computation time. Thus, SPRINT makes for a fast (and fun) demonstration:
`http://tinyurl.com/wikirank-01`

What follows is a brief description of two case studies to underpin the use of path information for ranking: Intranet search is evaluated with a user study. Then, Wikipedia search, which is subject of the demonstration, is examined theoretically. We close with a description of our implementation and the demonstration.

2. CASE STUDIES

Before discussing implementation details, we further motivate the use of paths for search result ranking. Specifically, we present results for two studies: The first compares our ranking approach with Google's ranking. In particular, we show results for ranking by shortest paths, the total number of paths as well as the average path length between two nodes. The second study justifies the use of web pages as search context and illustrates its influence on the ranking.

2.1 Intranet Search Ranking

In this scenario, we use path information for personalizing intranet search result rankings, i.e., we address users searching the intranet of their organization. Entering a query such as "latest publications" or "patent guidelines", the hypothesis is that the user is more interested in results concerning the own working group rather than in results specific to other groups. We further hypothesize that these more relevant results lie closer to the user's context page.

Setup. We asked 16 staff members from our research institute to fill out a very simple questionnaire (20 questions) stating the following: "Imagine you were using a *personalized* intranet search engine. For the given questions/tasks in natural language, please formulate a keyword query and enter the URL of the desired top search result." For each task we ran the keyword query against the Google search engine (with *site:hpi.uni-potsdam.de*) and re-ranked all results by paths using the staff member's working group main page as context node. As for shortest path and average path length a page is ranked high if the respective path length is small. Path count causes a high ranking if the value is high. We then compared the ranks of the desired top result given by Google and our personalized ranking. For this study we used a crawl of our website and an implementation that considers all available paths among two nodes (allowing a comparison of different ranking approaches).

Results. Figure 1 (left) depicts the shortest path ranking compared to the Google ranking (sorted by Google rankings)[1]. In most cases, shortest path ranks desired results among the top ten. There are only few interesting pages ranked so poorly that a user could not consider them. Specifically, shortest path ranks those pages better that Google fails to consider important. Also, the average shortest path rank for desired top results (4.8) is better than Google's average (9.2). Since Google ranks 60% of the desired results very high it achieves a better Mean Reciprocal Rank of 0.57 (shortest path: 0.46, avg. path length: 0.43, path count: 0.27). The right part of the figure shows different queries as well as desired top result rankings and thus allows for a distinction of query types: The results of *general* queries, such as "colloquium" (for the task "Find out who gave a talk at the group's colloquium."), or "project partner", are ranked well by shortest path. In contrast, Google ranks results for *descriptive* queries better than any other path-based ranking, e.g., "duplicate detection".

2.2 Context Node Sensitivity

In a second study, we examine the sensitivity of the shortest path ranking with regard to different context pages. The hypothesis is

[1]The number of usable query/result pairs amounts to only 100 due to searches that do not yield desired top results.

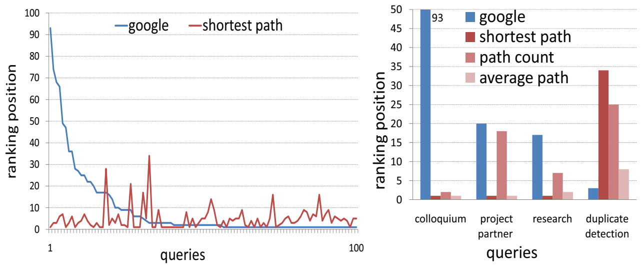


Figure 1: Results for the intranet search result ranking (the lower the better). Left: Shortest path and Google ranking for (user-defined) interesting pages. Right: Comparison of different path rankings for selected queries.

that a context node enables a high ranking of pages from a specific area of interest, disregarding pages from other areas that also match a query.

Setup. Figure 2 (left) illustrates the study design. Given an ambiguous query, such as "worm", we select two matching pages p_1 and p_2 with distinct topics from the Wikipedia *disambiguation page*, e.g., p_1 = computer worm and p_2 = parasitic worm. For these pages, we determine concepts c_1 and c_2 as well as the least common ancestor c_a in a concept hierarchy (here Yago [2]). We then compute path scoresfor p_1 and p_2 given a context page p_c for each concept c on the *concept path* $c_1 \to \cdots \to c_a \leftarrow \cdots \leftarrow c_2$.

Results. Figure 2 (right) depicts score changes (the higher the better) for varying context pages, i.e., context pages were chosen randomly for each concept along the concept path, and scores have been computed. The figure shows that a context page causes a better scoring for (conceptually) closer pages than for distant pages. This demonstrates the discriminative feature of context pages combined with shortest paths. Further, one can see a vanishing scoring when using context pages close to the common ancestor concept c_a (mostly at 50% of the concept path): At this position paths among context page p_c and p_1 as well as p_2 have comparable lengths.

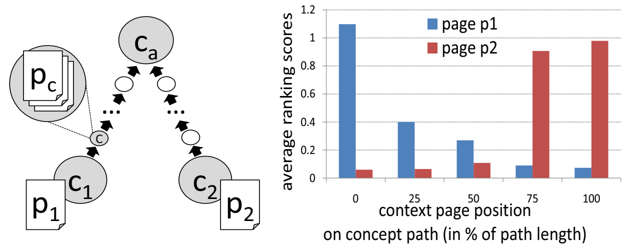


Figure 2: Context node sensitivity in Wikipedia. Left: Study design; pages p_1 and p_2 that match a query as well as respective concept path $c_1 \to \cdots \to c_a \leftarrow \cdots \leftarrow c_2$. Right: Average ranking scores (for 50 page pairs p_1, p_2) using selected context pages on concept path. Higher scores indicate better ranking.

3. IMPLEMENTATION AND DEMO

Depending on the use case the web graph at hand might be very large. Table 1 shows sizes of the web graphs we considered for evaluations. Computing shortest paths among all pairs of nodes clearly does not scale sufficiently: The Floyd-Warshall algorithm is in $O(|V|^3)$.

Existing approaches often deal with undirected graphs: Friendship relations in social networks are usually undirected, whereas links in intranets should be considered directed to support topical

Web graph	num. of nodes	num. of edges	avg. degree
CS at Humboldt	9,633	57,480	11.9
CS in Potsdam	51,341	177,371	6.1
HPI in Potsdam	39,424	2,236,432	113.5
Wikipedia	3.2m	175.5m	109.7

Table 1: Typical sizes of graphs under consideration. Graphs obtained with in-house crawler.

connections. Therefore, our implementation considers directions. Apart from semantics, it is an elegant way to prune the immense number of paths among nodes: Considering directed edges decreases the number of paths. Table 2 depicts the varying number of paths from the Java Programming Language node in Wikipedia. It is 5-10 times as high for undirected paths. Note that all numbers and the demo, however, refer to directed paths for semantic reasons.

FROM node	TO node	directed	undirected
Java (Prog. Lang.)	Sun Microsystems	159	557
Java (Prog. Lang.)	Oracle Corp.	12	130
Java (Prog. Lang.)	Sun (the star)	0	9

Table 2: Number of paths for selected nodes in Wikipedia.

To gain experience with different types of ranking and various ways of their computation we implemented two approaches: In a first step we aimed at testing the use of different path measures for ranking, namely the total number of paths, the average path length, and the length of the shortest path between two nodes. Since enumerating all paths in a large graph is not feasible we implemented the enumeration in a reduced search space, i.e., a *reasonable* subgraph of a smaller size. Here, *reasonable* means that the subgraph contains a given context node and all nodes reachable within distance r. However, given the average degree of an intranet node (e.g. > 100), even when choosing $r = 4$, we found that the subgraph is too large for enumerating all paths in user response time. Nevertheless, we used this approach for generating first evaluation results for the intranet use case discussed in Sec. 2.1.

As a second approach, we approximate path information. In [6] one can find a comprehensive overview of techniques for approximating distances in graphs. In particular, we redefine the one used in [4] to deal with large *directed* graphs and enhance it with proper seed selection [1, 5] as the basis for computing an index – also introduced in the following.

3.1 Approximation using Seeds

This approach allows for an approximation of shortest paths and proceeds in three phases: The first step is to select a set of seed nodes from the graph. This set of nodes serves as navigational backbone for the index computation. Given the seed nodes, step two is to compute distances from these nodes to all reachable node. These distances (pruned efficiently) form the actual index structure that enables phase three – online search result ranking. For didactic purposes, we first describe the index computation as well as the online ranking and then discuss seed selection.

Index Computation. Given z dedicated seed nodes $\{s_1, \ldots, s_z\}$, we compute two seed distance vectors (SDVs) $\overrightarrow{v}_n$ and $\overleftarrow{v}_n$ for each node n in the directed graph. The vectors $\overrightarrow{v}_n$ and $\overleftarrow{v}_n$ capture distances *from* the seeds *to* the node and vice versa, respectively. Let $dist(n_1, n_2)$ be the minimal distance *from* n_1 *to* n_2. SDVs are defined as follows:

$$\begin{aligned}\overrightarrow{v}_n &= [dist(s_1, n), \ldots, dist(s_z, n)] \\ \overleftarrow{v}_n &= [dist(n, s_1), \ldots, dist(n, s_z)]\end{aligned}$$

Note that we use only values of $dist(n_1, n_2) \le \theta$ where θ is a pruning threshold. The pruned vectors, on the one hand, allow an efficient computation as well as storage, because they are relatively sparse; on the other hand, they limit the approximation result since they restrict the considered path lengths to 2θ (see next paragraph). Consider the example in Fig. 3 and Tab. 3: The graph has 13 nodes; e, i, and m have been selected as seed nodes. The table shows SDVs in its original and pruned version. Given the size of the graph and a large number of seeds, we used a modified version of the map/reduce solution given in [4] to compute all SDVs.

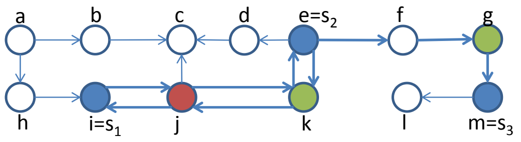


Figure 3: Example graph with e, i, and m as seeds. j is a context node and k as well as g shall be ranked.

	original			pruned		
seed distance vectors	s_1	s_2	s_3	s_1	s_2	s_3
$\overrightarrow{v}_j$	1	2	∞	1	2	-
$\overleftarrow{v}_j$	1	2	5	1	2	-
$\overrightarrow{v}_k$	2	1	∞	2	1	-
$\overleftarrow{v}_k$	2	1	4	2	1	-
$\overrightarrow{v}_g$	5	2	∞	-	2	-
$\overleftarrow{v}_g$	∞	∞	1	-	-	1

Table 3: Example seed distance vectors for graph in Fig. 3. Middle part: Original vectors. Right part: Vectors for $\theta = 2$.

Online Search Result Ranking. Given the SDVs, the authors of [4] propose a ranking function. This function combines lengths of available paths via seeds and derives a score that is high if there are many short paths. We redefine the original approach to incorporate directed links: Let c be a context node in the graph, let n be a node to be ranked with respect to c and let $\overleftarrow{v}_c$, $\overrightarrow{v}_n$, $\overrightarrow{v}_c$, $\overleftarrow{v}_n$ be SDVs. $dist_x(\overleftarrow{v}_c, \overrightarrow{v}_n)$ indicates the number of seeds whose sum of distances (here from c to n) is equal to x ($x = 0 \ldots 2\theta$). Let $k_0 > \cdots > k_{2\theta}$ be weighting parameters and let z be the number of seeds used. The ranking function $r_c(n)$ is defined as follows:

$$r_c(n) = \frac{\sum_{x=0\ldots2\theta} k_x * (dist_x(\overleftarrow{v}_c, \overrightarrow{v}_n) + dist_x(\overrightarrow{v}_c, \overleftarrow{v}_n))}{z}$$

Consider again Fig. 3 and Tab. 3 as an example where j is the current context: Then, k is ranked better than g, because the ranking scores for k and g with respect to j are the following:

$$\begin{aligned}r_j(g) &= (k_0 * 0 + k_1 * 0 + k_2 * 0 + k_3 * 0 + k_4 * (1 + 0))/3 \\ r_j(k) &= (k_0 * 0 + k_1 * 0 + k_2 * 0 + k_3 * (2 + 2) + k_4 * 0)/3\end{aligned}$$

Depending on the number of seeds we achieve different response times for ranking a set of nodes. For instance, if we were to rank 1,000 nodes at a time it takes 12s in a pure database-solution. We therefore propose to perform online ranking computation in a distributed manner, e.g., a map task (without reduce) can compute the $r_c(n)$ value for a single node n. For the demo, however, we use the db approach to save map/reduce overhead.

Seed Selection. The given ranking scheme requires a set of seed nodes as input. Such a set must meet the following criteria: (1) Given the threshold θ, the seeds plus the nodes within distance θ of any seed should cover a large portion of the graph's nodes – at least $p\%$. This ensures that there is at least one $dist_x(\dots) > 0$ such that a ranking score can be computed (for $p\%$ of the nodes). (2) The number of seeds required to cover $p\%$ of the input graph should be relatively small because the fewer seeds we consider, the faster is the online ranking computation [4].

In [5] the authors propose several heuristics for this purpose and $MaxOut$ is shown to be the best performing approach in terms of node coverage. $MaxOut$ initially selects the node with the highest (out-)degree as a seed; it then removes from the graph the seed as well as all nodes within distance θ, updates degree values, and again selects the node with the highest degree as seed. It proceeds until $p\%$ of the nodes have been removed from the graph. Note that reachable nodes as well as new node degrees have to be determined in every iteration, causing high runtimes. Therefore, we introduce the additional requirement that a run (the selection of one or more seed nodes) must cover at least $q\%$ of the remaining nodes. If this is not the case, the next run considers $s' = \lceil (q/l) * s \rceil$ seed nodes (with highest out-degrees) at once. Here, l and q are the current and required coverage, respectively, and s indicates the currently considered number of seed nodes. This technique considerably reduces seed selection runtime, because the time per run grows sublinearly with the number of seeds under consideration. The seed selection flow is depicted in Fig. 4: Given a set of seeds (initially one), a map/reduce program removes reachable nodes from the graph. After computing the current coverage, a second map/reduce programm determines a required number of seed node candidates. Since different reducers output independent candidate sets, these need to be merged before starting the next iteration.

For the SPRINT demo on Wikipedia, we have selected 12,000 seeds of the Wikipedia link graph and computed seed distance vectors $\overleftarrow{v}_n$, $\overrightarrow{v}_n$ for each Wikipedia page n. Table 4 summarizes runtimes for these phases on the graph that comprises > 3m nodes and > 175m edges. The graph we use has been extracted from a Wikipedia dump as of 12/2009, the map/reduce programs ran on a Hadoop cluster of 8 (old) PCs.

Phase	time (h:mm:ss)
Seed selection	1:03:00
Index computation	1:12:11

Table 4: Runtimes for Wikipedia; $\theta = 2$, $p = 95\%$, $q = 1\%$

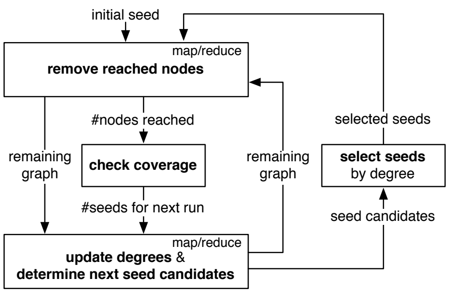


Figure 4: The seed selection process.

3.2 The Demo

The demo of SPRINT leverages all discussed techniques and is straightforward: Users can issue arbitrary queries to Wikipedia as shown in Fig. 6. Here, the left frame displays our search box and the re-ranked search results. We place our ranked results in a separate frame to enable comparisons with Wikipedia's original ranking. We supply several suggestions for search terms that make our approach obvious. We also encourage participants to conceive ambiguous search terms and observe how the expected result page is ranked depending on the current context. Additionally, we display respective seeds of shortest paths used for the ranking which facilitates an analysis of the ranking.

Figure 5 shows the components involved in the online tool for ranking Wikipedia search results. Given a user query, we leverage the Wikipedia API to retrieve a set of search results. Next, we retrieve two seed distance vectors per result page from our DB. Then, we optionally use a cluster to compute the new ranking.

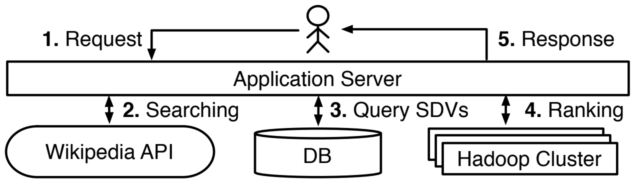
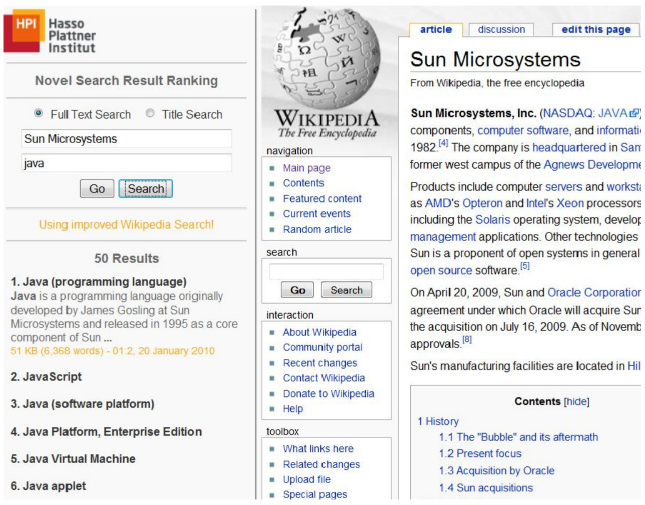


Figure 5: The Architecture of the Ranking Tool.



Sun Microsystems

java

Using improved Wikipedia Search!

50 Results

1. Java (programming language)
Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun ...
51 KB (6,368 words) - 01:2, 20 January 2010

2. JavaScript

3. Java (software platform)

4. Java Platform, Enterprise Edition

5. Java Virtual Machine

6. Java applet

Sun Microsystems

From Wikipedia, the free encyclopedia

Sun Microsystems, Inc. (NASDAQ: JAVA
components, computer software, and informati
1982.[4] The company is headquartered in San
former west campus of the Agnews Developme

Products include computer servers and workst
as AMD's Opteron and Intel's Xeon processors
including the Solaris operating system, develop
management applications. Other technologies
Sun is a proponent of open systems in general
open source software.[5]

On April 20, 2009, Sun and Oracle Corporatior
agreement under which Oracle will acquire Sur
the acquisition on July 16, 2009. As of Novemb
approvals.[8]

Sun's manufacturing facilities are located in Hil

Contents [hide]

1 History
1.1 The "Bubble" and its aftermath
1.2 Present focus
1.3 Acquisition by Oracle
1.4 Sun acquisitions

Figure 6: The Wikipedia Search Result Ranking Tool.

4. REFERENCES

[1] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *Proc. of the Int. CIKM*, 2009.

[2] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proc. of the Int. WWW Conf.*, pages 697–706, 2007.

[3] A. Ukkonen, C. Castillo, D. Donato, and A. Gionis. Searching the Wikipedia with Contextual Information. In *Proc. of the Int. CIKM*, 2008.

[4] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. de Castro Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *Proc. of the Int. CIKM*, 2007.

[5] S. Zheng, P. Dmitriev, and C. L. Giles. Graph-based seed selection for web-scale crawlers. In *Proc. of the Int. CIKM*, 2009.

[6] U. Zwick. Exact and Approximate Distances in Graphs - A Survey. In *Proceedings of the Annual European Symposium on Algorithms*, 2001.