ABSTRACT

Title of dissertation: COMPUTING APPROXIMATE CUSTOMIZED RANKING

Yao Wu, Doctor of Philosophy, 2009

Dissertation directed by: Professor Louiqa Raschid
Department of Computer Science

As the amount of information grows and as users become more sophisticated, ranking techniques become important building blocks to meet user needs when answering queries. PageRank is one of the most successful link-based ranking methods, which iteratively computes the importance scores for web pages based on the importance scores of incoming pages. Due to its success, PageRank has been applied in a number of applications that require customization.

We address the scalability challenges for two types of customized ranking. The first challenge is to compute the ranking of a subgraph. Various Web applications focus on identifying a subgraph, such as focused crawlers and localized search engines. The second challenge is to compute online personalized ranking. Personalized search improves the quality of search results for each user. The user needs are represented by a personalized set of pages or personalized link importance in an entity relationship graph. This requires an efficient online computation.

To solve the subgraph ranking problem efficiently, we estimate the ranking scores for a subgraph. We propose a framework of an exact solution (IdealRank) and

an approximate solution (ApproxRank) for computing ranking on a subgraph. Both IdealRank and ApproxRank represent the set of external pages with an external node Λ and modify the PageRank-style transition matrix with respect to Λ. The IdealRank algorithm assumes that the scores of external pages are known. We prove that the IdealRank scores for pages in the subgraph converge to the true PageRank scores. Since the PageRank-style scores of external pages may not typically be available, we propose the ApproxRank algorithm to estimate scores for the subgraph. We analyze the L_1 distance between IdealRank scores and ApproxRank scores of the subgraph and show that it is within a constant factor of the L_1 distance of the external pages. We demonstrate with real and synthetic data that ApproxRank provides a good approximation to PageRank for a variety of subgraphs.

We consider online personalization using ObjectRank; it is an authority flow based ranking for entity relationship graphs. We formalize the concept of an aggregate surfer on a data graph; the surfer's behavior is controlled by multiple personalized rankings. We prove a linearity theorem over these rankings which can be used as a tool to scale this type of personalization. DataApprox uses a repository of precomputed rankings for a given set of link weights assignments. We define DataApprox as an optimization problem; it selects a subset of the precomputed rankings from the repository and produce a weighted combination of these rankings. We analyze the L_1 distance between the DataApprox scores and the real authority flow ranking scores and show that DataApprox has a theoretical bound. Our experiments on the DBLP data graph show that DataApprox performs well in practice and allows fast and accurate personalized authority flow ranking.

COMPUTING APPROXIMATE CUSTOMIZED RANKING

by

Yao Wu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2009

Advisory Committee:
Professor Louiqa Raschid, Chair/Advisor
Professor Amol Deshpande
Professor Samir Khuller
Professor Bobby Bhattacharjee
Professor William Michael Rand
Professor Min Wu

Acknowledgments

First and foremost, I would like to thank my advisor Professor Louiqa Raschid. It was my great fortune to be able to work under her supervision. This thesis would not have been possible without her endless support and encouragement. I thank her for constant words of encouragement and reassurance in my ability to complete the research, countless hours spent reading and editing all kinds of drafts, and a lot of constructive suggestions along the way.

I wish to thank Professor Samir Khuller for his excellent lectures and all the help. The techniques I learned from his courses eventually became powerful tools and enabled me to complete this thesis.

It was my great pleasure to have the opportunity working with my collaborators: Professor María Esther Vidal, Professor Vagelis Hristidis, Professor Felix Naumann, and Dr. Panayiotis Tsaparas. I owe my sincere gratitude to Professor Vidal, who spent a lot of time to work with me when she visited Maryland and she was always patient to answer my questions. I would like to thank other graduate students, Woei-Jyh (Adam) Lee, Hassan Sayyadi, Jens Bleiholder for discussion and help.

I am indebted to Professor Amol Deshpande, Professor Min Wu, Professor Bobby Bhattacharjee, and Professor William Michael Rand for spending their precious time to serve on my thesis committee. I thank Professor William Gasarch for his help. I also thank Jennifer Story, Fatima Bangura, Arlene E. Schenk and other staff members in Computer Science Department, the Institute for Advanced

Computer Studies, and the Robert H. Smith School of Business. They handled the administrative matters for me patiently.

This material is based in part upon work supported by the National Science Foundation under Grant Numbers IIS0430915 and CMMI0753124.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Lastly, I appreciate the support and love from my parents and, in particular, my husband Yingchuan for helping me through difficulties and for being there all the time. Finally, I thank my lovely son Kesler, who has made all the difference in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The explosion of information available on the Web has made the ranking of Web pages an expensive but unavoidable component of query answering. Since hyperlinks from one page to another usually implies an "endorsement" or "recommendation", link analysis plays a critical role in determining the importance of web pages. PageRank[1, 20, 76] and HITS[61] are two seminal approaches in the area. PageRank iteratively computes the score of a page based on the scores of its parent pages. HITS separates the role of each web page into a *hub* or *authority*. The hub score estimates the value of its links to other pages and the authority score estimates the importance of the page.

While both link analysis ranking algorithms assign importance scores based on the hyperlink structure, PageRank became the dominant ranking model because of two main advantages: query independence and spam resistance. The first advantage of PageRank is that PageRank is query independent, because PageRank can be precomputed before the keyword query arrives. At query time, the precomputed importance score for a page and relevance score are combined. HITS, on the other hand, is query dependent. HITS first computes a local neighborhood graph and then calculates the importance scores for the neighborhood graph. The second advantage of PageRank is that PageRank is more spam-resistant. Compared to HITS, where

the ranking scores can be affected by small changes to the neighborhood graph, PageRank is more resistant to the changes occurring on a small portion of the Web graph. In this thesis, therefore, we consider extensions of the PageRank algorithm to support customized rankings.

In this thesis we address the scalability challenges for two types of customized ranking. The first challenge is to compute the ranking of a subgraph. The PageRank computation typically takes many hours on large graphs. Various Web applications focus on identifying a subgraph, such as focused crawlers and localized search engines. These applications focus on ranking the pages contained within a subgraph in order to avoid the global computation of PageRank. The second challenge is to compute online personalized ranking. Personalized search improves the quality of search results for each user. The user needs are represented by a personalized set of pages or personalized link importance in an entity relationship graph. This requires an efficient online computation.

1.1 Challenges of Ranking a Subgraph

We first describe the applications that identify a subgraph. In January 2005, the indexable Web for search engines was estimated to be more than 11.5 billion pages [50]. According to [2], the Web is growing at a rate of 25% per year. To make ranking manageable, and to reflect the diversity of clients' information needs, web applications such as semantic search, focused crawlers, and localized search engines have emerged. They all have a common objective to rank a subgraph.

To solve the subgraph ranking problem efficiently, our objective is to estimate the ranking scores for a subgraph. The first intriguing application is a *focused crawler* [24, 35], also called a thematic crawler. A focused crawler is interested in collecting a subset of the Web pages that are related to a specific topic. Compared to a standard crawler which can easily get lost and waste resources, a focused crawler acquires relevant pages using a Best First Search; it assigns scores to outgoing links based on the likelihood that they reach relevant pages, then selects links based on the scores [35]. In contrast to focused crawlers which are topic specific, a *localized search engine* indexes a subset of web pages that are within a specific domain.

The web fragment retrieved by the focused crawler (or localized search engine) is a subgraph of the global web graph. Only PageRank scores for local pages in the subgraph are of interest to users. Figure 1.1 shows the typical infrastructure of a focused crawler (or a localized search engine). Users submit queries to the subgraph collected by a focused crawler and local query answers are returned to the user. The ranking on this local graph, however, should take into account the link structure of all web pages.

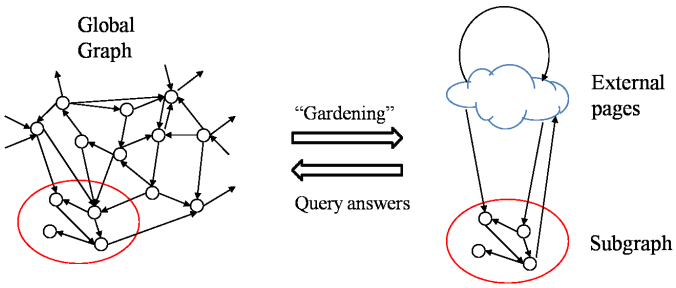

Figure 1.1: The infrastructure of a focused crawler or a localized search engine.

PageRank assumes that all edges in the Web graph are of the same type.

ObjectRank [15] is an extension that imposes a schema or Entity-Relation graph on an untyped graph. The details are in Section 1.2 and 2.7. An example of an Entity-Relation graph is in Figure 1.2. In the same spirit as a focused crawler or a localized search engine, the Entity-Relation graph of interest to ObjectRank is a subgraph. Here, too, the objective is to compute or estimate the ranking of a Entity-Relation subgraph.

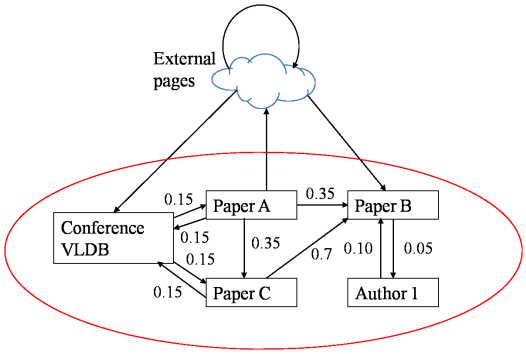

Figure 1.2: An example of subgraph ranking for an Entity-Relation graph.

Another application that involves ranking a subgraph is peer-to-peer network. The advent of peer-to-peer(P2P) technology has further boosted web information retrieval by leveraging distributed computing power, storage, and connectivity[49, 86, 94]. A distributed or decentralized system has multiple peers or servers, each of which stores its own subgraph of the Web. A user may ask queries on one peer and ranked query answers that are available locally are presented to the user. The ranking depends on the context of the query.

A similar situation is presented in meta-searcher as well. A study shows that search engines are more different than expected[3]. For the 500 most popular search terms, Google and Yahoo! shared only 3.8 of their top 10 results on average. Part of

the reasons behind this inconsistency is that the search engines fetch the web pages using different crawling algorithms. According to a recent study [50], the major search engines including Google, Yahoo!, MSN, Ask/Teoma fetch different portions of the whole indexable web. Figure 1.3 shows the percentage of the indexable web fetched by each search engine and their overlaps. It stands to reason that a meta-searcher better aggregates relevant results, which may require ranking computation on multiple subgraphs.

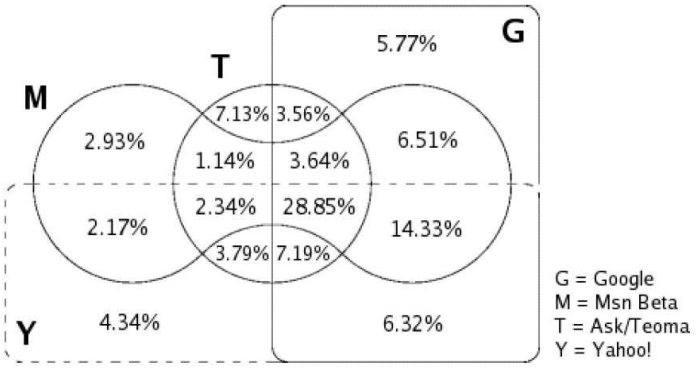

Figure 1.3: The percentage of the indexable web that lies in each search engine's index.

A final scenario is a reflection of the constant change of the Web. The ranking of pages needs to be updated frequently, especially for the subgraph of the Web that experiences the most change. This subgraph can be either a set of dangling pages that crawlers have not as yet crawled, referred to as the web "frontier" [41], or the set of pages that are most affected by updates [66]. It is desirable that any strategy to update the ranking of this subgraph exploits existing PageRank scores for other regions of the graph which may remain largely unchanged.

In response to these many motivating applications, we address the problem of computing ranking scores for a subgraph in this thesis. The challenges of ranking a subgraph include the follows:

- The approximate ranking for local pages should reflect the global link structure of the Web graph, i.e. it should be close to the true PageRank.
- Due to the dramatic growth of the World Wide Web, ranking techniques for a subgraph must be significantly computationally cheaper computation, when compared to executing the PageRank algorithm on the global Web graph.
- PageRank scores for external pages may be available for some applications. However, it is also possible scores for external pages are unobtainable. The subgraph ranking algorithm should address both cases.

The problem of estimating PageRank values for a small portion of the Web graph has been studied in [26, 34]. However, [26] estimates the PageRank value for single target node. The approach in [26] cannot be applied directly on subgraphs, since the expansion will quickly reach a very large graph where the computation is expensive. The SC algorithm in [34] is the best existing approach. SC predicts a supergraph with restricted size that will have the most significant impact on the ranking of the nodes of the subgraph. However, SC pays a big penalty whenever it mis-estimates and misses pages in the supergraph. In addition, the process of expansion of the subgraph brings immense extra cost to the algorithm. The details of these approaches are in Section 2.5. Our approach to estimate the ranking of a subgraph has similar to superior quality to SC and it outperforms SC in computational

cost.

1.2 Challenges of Personalization

The idea of a personalized ranking that provides users customized views of Web pages is attractive. Web personalization includes any action that tailors the Web experience to a particular user, or set of users [71]. Web personalization targets to provide users with accurate information for query answers, without always having to obtain explicit preference descriptions from users [73]. According to [39, 71], principal elements of Web personalization include modeling of Web pages and users, categorization and preprocessing of objects, the extraction of correlations between and across objects, and determination of the set of actions to be recommended for personalization.

Personalized ranking is one of the latest trends in search engines, which adjusts the order of Web pages presented to users and narrows down the retrieval space for users. For instance, while PageRank gives one global ranking for all the Web pages, users frequently have different point of views. The personalized ranking is an ordered list for the current (active) user, which should reflect the current user's preference or profile. Major commercial search engines improve the search quality by accommodating the topics of interests, prior search history, or other descriptions of users' preference.

There are numerous approaches to provide personalized ranking, including the following:

- To describe the importance with respect to a particular topic, [51, 76] proposed computing a set of PageRank vectors that are biased using a set of representative topics. At query time, the query-specific importance scores are combined using precomputed biased PageRank vectors.

- [15, 75] proposed capturing the importance with respect to the edge types. For the Entity-Relation graph where the query is received, an weight assignment of authority flow weights is used to describe the user preference.

- The concepts of trust and similarity are used to compute personalized rankings [84]. These concepts are captured from explicit user input and implicit user behavioral patterns to describe the users taste and preference.

- [62] considers the Web community of a specific user in personalization. In this way, past interactions of the user with the search engine are used to improve future search results. For each Web community, its neighborhoods including the documents linked to, or from, documents in the community are determined. The query answers are ordered to reflect the number of times these community neighborhoods have been visited.

- [87, 88] proposed adapting search results by constructing user profiles based on users' navigational history, browsing history, or query history. There are multiple data mining techniques can be applied to extract usage patterns from Web logs [31, 83].

Among all these factors, the first two factors are the two key ways to achieve

personalization in authority flow-based search systems like PageRank. Below we focus on the first two types of personalized ranking.

The first type of personalization involves selecting user-specific entities as the source of authority in the graph. Topic-sensitive PageRank [51] proposed to personalize using a set of representative topics, which can be considered a *personalized base set* of pages.

The second type of personalization assumes a typed graph and allows users to assign different importance to different types of edges. We present an example of the second personalization as follows: a biologist querying NCBI Entrez genomic resources [4] may assign a high weight to the gene-to-protein link type whereas a practitioner may assign a higher weight to the publication-cites-publication link type.

ObjectRank [15] is defined on an Entity-Relation graph and was the first work to propose personalization of the weight associated with link types. This type of ranking is referred to as *authority flow ranking*. Figure 5.1 [15] shows the Entity-Relation graph for the DBLP database, a bibliographic database for computer science publications [5]. The values along each edge type represent the relative importance of that edge type.

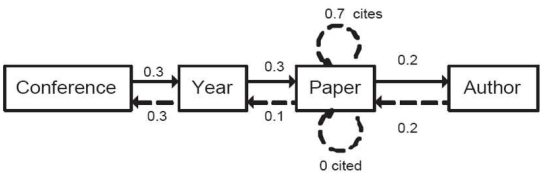

Figure 1.4: The DBLP authority transfer schema graph in ObjectRank ([15]).

Since users submit queries and expect query answers on-the-fly, a key challenge for personalization is to compute personalized rankings online and provide the answers to the user quickly. Since the personalized ranking is an expensive algorithm on the global graph, computing a personalized ranking at query time is infeasible. Since the space of possible queries is very large, computing all possible personalized rankings off-line and storing them is also impractical. Therefore, a hybrid solution is to maintain a set ,called repository, of personalized rankings. At query time, an approximate personalized ranking is computed using the personalized rankings in the repository and this is the approach we explore in this thesis.

We summarize the challenges of approximate personalized ranking as follows:

- The approximate personalized ranking should be computed efficiently.
- The ranking repository of precomputed personalized rankings should be managed and maintained up-to-date.
- The approximate personalized ranking should be close to the ideal personalized ranking to guarantee quality.

The problem of achieving scalable personalization based on a personalized base set, i.e., a personalization vector, has been studied before [23, 46, 51, 55, 57]. However, little work has addressed the problem of scalable link-based personalization based on user-dependent authority transfer schema graph. We study the latter problem in this thesis.

The only existing work that computes approximate authority flow ranking is [90], where sampling techniques are applied for lgOR [79]. However, lgOR is a

special case of authority flow ranking, where the result graph is a layered graph. For general case of authority flow ranking like ObjectRank, it is not straightforward to apply sampling techniques. The details of sampling techniques are presented in Section 2.8.2.

1.3 Contributions

We study two customized ranking problems in this thesis – approximating the ranking for a subgraph and approximating personalized authority flow rankings. For both problems, we present efficient algorithms and conduct analysis of the quality and precision of the algorithms.

1.3.1 Ranking a subgraph

For the subgraph ranking problem, we present a framework based on an exact and an approximate solution to compute PageRank on a subgraph. The IdealRank algorithm is an exact solution. It assumes that the PageRank scores of external pages are known. We prove that the IdealRank scores for pages in the subgraph converge to the true PageRank scores. Since the PageRank scores of external pages may not be available, we present the ApproxRank algorithm to estimate PageRank scores for the subgraph. Both IdealRank and ApproxRank represent the set of external pages with an external node Λ and extend the subgraph with links to Λ. They also modify the PageRank transition matrix with respect to (the links to) Λ.

The IdealRank and ApproxRank framework formalizes the problem of ranking

a subgraph. It allows us to model multiple scenarios where ranking a subgraph is important. IdealRank can be used to model scenarios where PageRank scores of the global graph are known a priori and can potentially be re-used. This includes the case where the subgraph contains the pages that have been updated, or the subgraph represents the pages that represent all the semantic types of interest to a domain expert in ObjectRank [15]. ApproxRank can be applied in general to all these problems, when we do not know the PageRank scores of external pages.

For the subgraph ranking problem, our contributions are as follows:

- We define an efficient algorithm, IdealRank, to compute PageRank scores for a subgraph when PageRank scores of the external pages are known. IdealRank performs a random walk on a modified local graph called the *extended local graph*, where an external node Λ is added to the local graph. Λ represents the set of pages that are not local. The random walk defined by IdealRank utilizes the PageRank scores of the external pages. The IdealRank algorithm can be applied when the Web graph is updated.

- We prove that the IdealRank scores converge to the true PageRank scores for all local pages in the subgraph, and the IdealRank score for the external node Λ converges to the sum of true PageRank scores for all external pages. Since IdealRank converges to the true PageRank, it provides a golden standard for the approximate solution, ApproxRank.

- When PageRank scores of external pages are not known, we define an efficient algorithm ApproxRank to estimate the PageRank scores for a subgraph. The

ApproxRank random walk is defined on the extended local graph as well. Since there is no knowledge about the external pages, ApproxRank assumes the authority flow from external pages are equally important. We conduct error analysis for ApproxRank scores. We show that the error of the ApproxRank scores of the subgraph depends on the accuracy of estimation of external page ranking scores.

- We show through empirical results that the ApproxRank ranking accuracy is similar (sometimes superior) to the best competitor SC, and it overwhelmingly outperforms the runtime efficiency of SC. We use two real datasets, on which we conduct experiments on three types of subgraph: topic specific subgraph, domain specific subgraph, and BFS subgraph (gathered by a Breadth First Search crawler). We compare ApproxRank against three algorithms, local PageRank, LPR2, and SC. We use two ranking distance metrics to evaluate the accuracy of the algorithms, L_1 distance and the Spearman's Footrule distance. The experiments show that, even without assuming any knowledge about the external pages, ApproxRank behaves well.

1.3.2 Personalized authority flow rankings

We formalize the problem of approximating authority flow ranking defined by ObjectRank [15] for Entity-Relation graphs. A user query is associated with an authority flow weight assignment. A ranking repository is a set of precomputed authority flow rankings for a set of candidate authority flow weight assignment

vectors. We explain the ranking repository in our approximation. We define the DataApprox and SchemaApprox problem that approximate the authority flow ranking of a user-specified assignment. This thesis focuses on the DataApprox problem. SchemaApprox can be solved as a Least Squares problem (See Section 5.2) or a quadratic programming problem. We leave SchemaApprox as future work.

We formally define two approximate approaches, SchemaApprox and DataApprox. Both algorithms target to find a combination of existing rankings such that the combined ranking is close to the ideal personalized ranking. SchemaApprox is defined at the schema level and considers the weight assignment vector in the Entity-Relation graph. It is a Least Squares problem that requires quadratic programming solution. We do not solve SchemaApprox in this thesis. The DataApprox approach is defined at the data graph level, i.e., the actual nodes and edges described by the Entity-Relation graph. DataApprox considers the transition matrix of the data graph; this matrix is defined using the authority schema graph. The objective of the DataApprox algorithm is to combine the best existing rankings such that the transition matrix is close to the query transition matrix.

We have the following contributions as follows:

- We define two optimization problems to approximate the authority flow rankings, SchemaApprox and DataApprox. SchemaApprox is defined at the schema graph level, and DataApprox is defined at the data graph level.

- We introduce the concept of the aggregate surfer and prove the authority flow linearity theorem for authority flow rankings. We show that, given two weight

assignment vectors, there exists a random walk that is defined by combining the random walks of the two weight assignment vectors, and whose ranking vector is a linear combination of the two ranking vectors.

- The DataApprox algorithm is defined as an optimization problem to find the aggregate surfer that combines the best candidates from the ranking repository. It solves the optimization problem by employing a Linear Programming sub-procedure.

- We perform a theoretical analysis of the approximation quality of DataApprox. We show that the L_1 distance between DataApprox scores and the accurate personalized ranking scores depends on the objective of the DataApprox algorithm.

- We apply a set of heuristics to dramatically reduce the search space and the complexity of the DataApprox and makes the computation feasible even for very large data graphs.

- We conduct extensive experiments to evaluate the execution time and the quality for DataApprox, i.e., how close the approximate DataApprox ranking is to the exact ranking. The experiments are conducted on the complete DBLP data graph. We compare DataApprox algorithm with a baseline algorithm PickOne, which chooses the best candidate with the minimum Euclidean distance in the ranking repository. We evaluate the accuracy of the algorithms using a ranking distance metric, Spearman's Footrule distance. The experi-

ments show that DataApprox performs well both in terms of execution time as well as in terms of quality.

1.4 Outline

In Chapter 2, we survey related work and describe recent ranking techniques. Chapter 3 presents our ranking framework for the subgraph ranking including two ranking algorithms: IdealRank and ApproxRank, along with their properties. In Chapter 4, we report experimental results for the ApproxRank algorithm. Chapter 5 presents two algorithms to approximate authority flow rankings in entity-relation graphs, SchemaApprox and DataApprox. We also show the theoretical model and properties for DataApprox. Experimental results for the DataApprox algorithm are reported in Chapter 6. Finally Chapter 7 concludes the thesis and discusses the future work.

Chapter 2

Related Work

This chapter reviews the PageRank algorithm and presents an overview of related research in the areas of efficiently computing PageRank, computing PageRank in a distributed system, approximating PageRank for a subgraph, updating PageRank scores. We review the ObjectRank algorithm and scaling personalized PageRank. We refer to [17, 19, 65, 63, 64] for related work.

2.1 Preliminaries

In link analysis ranking, the Web is considered to be a massive *directed graph* in which web pages are represented by nodes and hyperlinks between web pages are represented by directed edges in the graph. The Web graph is a *simple* graph, i.e., even if there are multiple hyperlinks between two web pages, only one edge is considered in the Web graph. The *forward links set*, F_i, of page i denotes the set of pages reached by outgoing edges from page i. The set of *backlinks*, B_i, is the set of pages that point to page i. The cardinality of the forward links set is the outdegree for the page and the cardinality of the backlinks is its indegree. The Web graph is *unweighted*.

The distribution of indegrees and outdegrees (especially indegrees) of the Web graph follows a power law [16, 21, 45]. The power law states that the probability

that a node has indegree I is proportional to $1/I^x$ for some $x > 1$. The Web graph is also shown to have the bow-tie structure [21]. The Web can be broke into four pieces: the *CORE* that is a central strongly connected component (SCC), the *IN* set that consists of pages reaching the *CORE* but not accessible from the *CORE*, the *OUT* set that consists of pages accessible from the *CORE* but not linked to it, and the *TENDRILS* containing pages that are not accessible from the *CORE* and cannot reach the *CORE*.

There are two types of ranking algorithms based on how the ranking algorithm proceeds. The HITS algorithm [61] is a *query dependent* ranking algorithm which starts from a subset of the Web pages related to a query. On the other hand, PageRank [76] is a *query independent* ranking which ranks all Web pages. In the Web graph, ranking scores for all pages are represented by a ranking vector, where each entry in the vector is the ranking score for one web page.

2.2 Link Analysis Ranking

In 1998, there were two algorithms, PageRank [20, 76] and HITS [61], that initiated the area of link analysis ranking. Both algorithms propose to rank web pages based on the link structure of the Web graph. The basis of these approaches is that hyperlinks convey information about pages. A hyperlink from page A to page B is evidence that page A suggests that page B is important. Link analysis is a prominent approach in determining the importance of web pages.

Given a Web graph, where the pages are represented by nodes, and hyperlinks

are represented by directed edges, a link analysis ranking algorithm produces a score assignment.

Link analysis ranking is applied to the context of databases. ObjectRank considers a database as a weighted schema graph [15], where the random walk in PageRank is adjusted according to authority transfer on the database schema graph. [47] instead models the database and a set of queries as a weighted graph, and applies PageRank and HITS algorithms on such graphs. There is a combination of PageRank and HITS [36].

Link analysis ranking is also an important tool and is applied in different areas: it is applied to evaluate the similarity between data objects [56]; it is used to improve the quality of crawlers [30, 80]; it is used to characterize the Web structure [77]; and it is used for relationship search operator [54].

2.2.1 The PageRank Algorithm

PageRank was introduced in [20, 76] to capture the intuition that important pages have a large number of important pages pointing to them. [17, 19, 64] are excellent surveys of the PageRank computation. A link from page i to page j is evidence that i is suggesting that j is important. The importance contributed to page j by i is inversely proportional to the outdegree of i. The PageRank score of page j, denoted by $R(j)$, is the sum of the PageRank scores along incoming edges in its backlink set B_j. Let D_i be the outdegree of an incoming page i and $R(i)$ be

the PageRank score of page i.

$$R(j) = \sum_{i \in B_j} \frac{R(i)}{D_i} \tag{2.1}$$

The PageRank scores are computed through iterations. $R(j)$ in Equation (2.1) refers to the current iteration and $R(i)$ at the right hand side of the equation refers to the previous iteration. Initially all pages are assigned the same score of 1. This formula appears meaningful since the contribution of authority from all pages in backlinks set is distinguished based on their importance and connectivity (outdegree). However, there are a large number of pages with no outgoing links, referred as *dangling pages*. According to Equation (2.1), dangling pages only receive authority flow but they do not distribute their own weights. The consequence is that dangling pages accumulate more and more PageRank scores, and the ranking vector does not converge. This is the *rank sinks* problem [76].

To overcome the rank sinks problem, PageRank proposes to add links from dangling pages to all other pages. This can be modeled by the behavior of a random web surfer. With certain probability, the surfer gets bored by following links presented in the graph and randomly jumps to any page with equal probability. Let *damping factor* ϵ be the probability that a web surfer follows hyperlinks, and let $(1 - \epsilon)$ be the probability of a surfer making a random jump to a page, where ϵ is usually set to be 0.85. Let n be the number of web pages in the graph, Equation (2.1) can be rewritten as follows:

$$R(j) = \epsilon \sum_{i \in B_j} \frac{R(i)}{D_i} + (1 - \epsilon)\frac{1}{n} \tag{2.2}$$

The set of dangling pages is referred as "web frontier" [41]. There are multiple reasons that one page can become dangling. 1) The page truly contains no outgoing links, 2) the page is protected by robots.txt, 3) the crawler has not yet crawled the page, or 4) the page does not exist any more (a 404 HTTP code). Several algorithms in [41] are proposed for handling dangling pages.

The above random walk can be represented in matrix form. Let A denote the transition matrix:

$$A[i,j] = \begin{cases} \frac{1}{D_i} & \text{if there is an edge from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Let R be the PageRank vector to be computed over the web pages. Initially R can be an arbitrary vector representing the probability of visiting web pages. The *personalization vector* P (also called *teleportation vector*) can be used to bias PageRank to prefer certain pages. In standard PageRank, P is a uniform distribution to indicate the equal probability of randomly jumping to any page; it is as follows:

$$P = [\frac{1}{n}]_{n \times 1} \tag{2.4}$$

Let A^T denote the transpose of A. The PageRank vector R is recursively defined as follows in Equation (2.5):

$$R = \epsilon A^T \cdot R + (1 - \epsilon)P \tag{2.5}$$

In graph theory, a directed graph is *aperiodic* if there is no integer $k > 1$ that divides the length of every cycle of the graph. A Markov Chain is *irreducible* if its underlying graph consists of a single strongly connected component. According

to the Ergodic Theorem [60, 72] for Markov chains, if the graph is aperiodic and irreducible, then a unique steady state distribution exists. Since the Web graph is generally aperiodic, and is made irreducible by adding a damping factor, R converges to the stationary distribution for the Web graph.

The PageRank vector R can be represented as a eigenvector for matrix G^T. Let e be a vector of all 1s, and e^T be a row vector of all 1s. ee^T is a matrix of all 1s. Let $G = \epsilon A + (1 - \epsilon)\frac{1}{n}ee^T$. The PageRank vector is the dominant eigenvector for matrix G^T [64]. Since G^T is a stochastic matrix, the corresponding dominant eigenvalue $\lambda_1 = 1$. R can be written as:

$$R = G^T R \tag{2.6}$$

R is the stationary vector for a Markov Chain with transition matrix G^T. The power method is used in the original PageRank paper [20]. There are three reasons that power method has been chosen:

- The power method is simple and easy to implement. During the computation, there is only matrix-vector multiplication and no matrix-matrix multiplication.

- The power method is storage efficient. During the computation, only the transition matrix and the PageRank vector for the current iteration needs to be stored. Other iterative method for PageRank will require multiple vectors through iterations.

- The power method only requires 50–100 for convergence for large Web graphs. Although this number depends on the accuracy of the convergence, it is hard

to find other methods that beat 50 power iterations.

Edges in a graph usually are associated with weights to indicate the strength of the relationship. Random walk on weighted graph has been considered in different contexts. In [32], undirected weighted graph are dealt with to design on line algorithms. In [40], usage data are considered to rank paths.

2.2.2 The HITS Algorithm

The other pioneer link analysis ranking algorithm, HITS [61], considers each web page has two roles: hub and authority. Hub score estimates the value of its links to other pages, and authority score estimates the importance of the page. A page is a good hub if it points to good authorities; a page is a good authority if it is pointed to by good hubs. Different from the PageRank algorithm, HITS is a *query-dependent* algorithm – HITS builds a neighborhood graph based on a small set of relevant pages, and then executes the ranking algorithm on this neighborhood graph.

HITS is an iterative algorithm. Let $x^{<p>}$ be the non-negative authority weight assigned for page p. Let $y^{<p>}$ be the non-negative hub weight assigned for page p. There are two types of operations $\mathcal{I}$ and $\mathcal{O}$ to update authority weights and hub weights respectively. In $\mathcal{I}$ operations,

$$x^{<p>} = \sum_{q:(q,p)\in E} y^{<q>} \tag{2.7}$$

In $\mathcal{O}$ operations,

$$y^{<p>} = \sum_{q:(p,q)\in E} x^{<q>} \tag{2.8}$$

HITS can provide two ranked lists to the user. This is beneficial since in different applications, users may be interested in the best authorities or the best hubs. However, HITS is query-dependent. For each query, HITS needs to build a neighborhood graph and solve at least one matrix eigenvector problem. HITS is also vulnerable to TKC effect, which will be described in 2.2.3.

2.2.3 The SALSA Algorithm

Lempel and Moran proposed SALSA (Stochastic Approach for Link Structure Analysis) algorithm [68] that combines PageRank and HITS. Similar to HITS, SALSA algorithm creates a neighborhood graph and calculates both authority scores and hub scores. SALSA also takes advantages of stochastic matrices which are used by PageRank.

By defining these stochastic matrices, SALSA is able to overcome the tightly knit community (TKC) effect. TKC effect refers to the phenomenon that a small but highly connected sites can boost each other's scores considerably. However, SALSA is query-dependent, which means it is expensive to calculate at query time.

2.3 Efficiently Computing PageRank

Because the Web is constantly changing, and because web pages are crawled periodically, the PageRank vector needs to be calculated regularly. Another reason that obstructs the efficient computation of the PageRank algorithm is that the personalized and topic-sensitive PageRank requires the computation of many PageRank

vectors. The efficient computation of the PageRank scores for the global graph has been studied in [22, 33, 58, 59, 60].

2.3.1 The adaptive method

The adaptive method is exploited in [58] where pages whose scores have converged are not recomputed in a new iteration. The adaptive PageRank algorithm partitions web pages into two sets N and C at each iteration, where N denotes the set of m pages that have not yet converged, and C denotes the set of $n - m$ pages that have converged. The transition matrix A is accordingly divided into two submatrices A_N and A_C, each of which corresponds to the inlinks of web pages that have not yet converged and pages that have already converged respectively. According to the power method, the scores at the $(k+1)$th iteration are calculated based on scores at the kth iteration as follows in Equation (2.9):

$$\begin{pmatrix} R_N^{k+1} \\ R_C^{k+1} \end{pmatrix} = \epsilon \begin{pmatrix} A_N^T \\ A_C^T \end{pmatrix} \cdot \begin{pmatrix} R_N^k \\ R_C^k \end{pmatrix} + (1-\epsilon)P \tag{2.9}$$

In Equation (2.9), R_N^j denotes the PageRank scores for N at iteration j and R_C^j denotes the PageRank scores for C at iteration j. As we expect no changes between elements of R_C^{k+1} and R_C^k, the computation can be simplified as in Equation (2.10):

$$R_N^{k+1} = \epsilon A_N^T \cdot R_N^k + (1-\epsilon)P_N \tag{2.10}$$

The adaptive method simply avoids computation on the web pages that have already converged, and the experiments report about a 20% saving in wallclock time to compute the PageRank vectors. While the adaptive method takes a few more

iterations for convergence compared to the standard power method, the adaptive method gains overall savings due to the saving on computation cost in each iteration.

2.3.2 Extrapolation methods

Two extrapolation methods, Aitken Extrapolation and Quadratic Extrapolation, are proposed in [60]. The basis of these methods is that the initial ranking vector R^0 can be represented as a linear combination of the eigenvectors $(u_1, u_2, \cdots, u_m)$ of the transition matrix A. The power method converges to the principal eigenvector of A after many iterations.

$$R^0 = u_1 + \alpha_2 u_2 + \cdots + \alpha_m u_m \tag{2.11}$$

Since, for Markov chains, the first eigenvalue λ_1 is 1, R_k can be written as:

$$R^k = A^k R^0 = u_1 + \alpha_2 \lambda_2^k u_2 + \cdots + \alpha_m \lambda_m^k u_m \tag{2.12}$$

Since $1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_m$, an approximation can be made by truncating the tail of the summation.

The Aitken Extrapolation method approximates u_1 and therefore R by preserving two terms in the summation for R^{k-2}, after expressing R^{k-2} as a linear combination of u_1 and u_2:

$$R^{k-2} = u_1 + \alpha_2 u_2 \tag{2.13}$$

$$R^{k-1} = u_1 + \alpha_2 \lambda_2 u_2 \tag{2.14}$$

$$R^k = u_1 + \alpha_2 \lambda_2^2 u_2 \tag{2.15}$$

Unlike the Aitken Extrapolation method where the expansion of R^{k-2} is truncated by the first two eigenvectors, the Quadratic Extrapolation method assumes $R^{k-3} = u_1 + \alpha_2 u_2 + \alpha_3 u_3$. R^{k-2}, R^{k-1} and R^k are expressed using $u_1, u_2, u_3, \alpha_2, \alpha_3, \lambda_2, \lambda_3$. With some matrix computation, u_1 can be approximated.

The experiments report the Aitken Extrapolation method speeds up PageRank calculation by about 40% and the Quadratic Extrapolation speeds up PageRank calculation by about 60%.

2.3.3 The BlockRank algorithm

[59] presents a 3-step algorithm for speeding up the PageRank computation by exploiting the block structure of the Web. The block structure of the Web describes that the vast majority (about 80%) of hyperlinks are intra-host links, and only a small portion (about 20%) are inter-host links. The intuition of the BlockRank algorithm is that we first compute local PageRank for each host, then we aggregate local PageRank reasonably, and we finally run standard PageRank on the global graph using the aggregated scores as starting scores.

Figure 2.1 provides the outline of the BlockRank algorithm.

The BlockRank algorithm converges fast, as local PageRank vectors converge quickly and it allows parallel implementation. The experiments show that BlockRank gives a speedup of factors up to 1.55 and 2 for two datasets respectively.

In [22, 33], other graph aggregation approaches are presented to approximate the PageRank scores.

Algorithm BlockRank [59]

1. Compute local PageRank scores for each host.
2. Construct a block graph, where every node represents a block and every edge represents a set of hyperlinks from a block to another block (or itself).
   Compute the importance of hosts on this block graph.
3. Run the standard PageRank on the global graph using the weighted aggregation of the local PageRank score as its starting vector.

Figure 2.1: The outline of the BlockRank algorithm.

2.4 Computing PageRank in a distributed system

2.4.1 Distributed system without overlaps

Recent research efforts in distributed systems have addressed the case where the Web graph is partitioned into disjoint web sites or domains [13, 91]. In the ServerRank algorithm [91], the Web is modeled as numerous disjoint web servers. The hyperlinks in the Web are divided into two categories, *intra-sever* links and *inter-server* links. Intra-server links are links between pages within a server, and these links are used to compute a local PageRank vector on each server. Inter-server links are links between pages in different servers, and they are used to compute ServerRank. ServerRank measures the relative importance of the different web servers.

The outline of the ServerRank algorithm is described in Figure 2.2.

Algorithm SERVERRANK [91]

1. Each web server constructs a web link graph based on intra-server links and computes its local PageRank vector.
2. Web servers exchange inter-server links information and compute the ServerRank vector.
3. Web servers use the ServerRank vector to refine their local PageRank vectors.
4. The submitting server of the query fuses the ServerRank vector and local PageRank vectors to get a single ranked URL list.

Figure 2.2: The outline of the ServerRank algorithm.

Step 1 and 2 seem straightforward. The purpose of Step 3 (Local PageRank Refinement) is first to update the local PageRank computed in Step 1 by considering the ServerRank acquired in Step 2 and then to use this vector as the initial ranking vector and to run the local PageRank algorithm for a single iteration. The local PageRank is performed for only one iteration to avoid the convergence of the PageRank vector back to the local PageRank vector in Step 1. Step 4 (Result Fusion) merges the ranking lists from all servers into a single ranked list. The basic idea of ServerRank is visualized in Figure 2.3, where the cycles represent disjoint web servers, and edges in the figure represent inter-server links.

A similar approach is presented in [22]. The basic idea of [22] is to partition the Web into equivalence classes of web pages, where a class contains all the pages of a host. For a web graph with m hosts, a $m \times m$ transition matrix $\tilde{T}$ is defined,

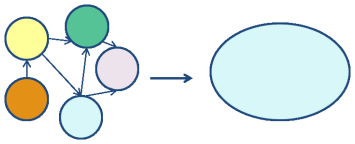

Figure 2.3: The ServerRank algorithm

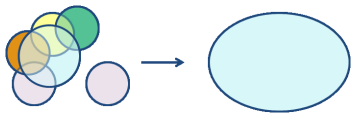

Figure 2.4: The JXP algorithm

and the stationary distribution of dimension m for $\tilde{T}$ is calculated. Finally, the local stationary distribution and the stationary distribution for $\tilde{T}$ are aggregated.

In [13], a ranking algebra is proposed to deal with rankings at different granularity levels, which can also be applied to the aggregation of local rankings and site rankings to get global rankings.

2.4.2 Distributed system with overlaps

There has been work on PageRank approximation in a fully decentralized system [78], in which each peer is autonomous, and peers may overlap with each other. In the proposed JXP algorithm, each peer computes local PageRank scores, randomly meets other peers, gradually increases its knowledge about the global web graph by exchanging information, and then recomputes the PageRank scores on the local peer. This meeting and recomputation process is repeated until the peer gathers enough information. The JXP scores converge to the true global PageRank scores if peers eventually meet a sufficient number of times to exchange information. The assumption is that the outdegree of each page in the global graph is known. The system is visualized in Figure 2.4, and the outline of the JXP algorithm is described in Figure 2.5.

Algorithm JXP(G_A, Λ_A, L_A) [78]

▷ The input for peer A includes local graph G_A, world node Λ_A, and score

▷ list L_A from G_A.

1. Contact a random peer B.
2. Merge graphs G_A and G_B, world nodes Λ_A and Λ_B, and score lists L_A and L_B for two peers.
3. Run PageRank on Merged graph and update L_A.
4. Repeat until scores converge.

Figure 2.5: The outline of the JXP algorithm.

2.5 Estimating PageRank for a Subgraph

The problem of estimating PageRank values for a small portion of the Web graph has received recent attention in the literature [26, 34].The goal of [26] is to estimate the PageRank value for one target node. [34] addresses the problem of estimating the score for a subgraph. Another paper [93] aims to do link-based ranking on a small graph exploiting users' access patterns.

The common approach in all of these papers is to expand the subgraph to a supergraph and then to run PageRank on this augmented graph. They differ in the procedure for augmenting the subgraph. In [93], besides the existing hyperlinks that indicate recommendation, implicit recommendation links determined by mining user access patterns are also added into the supergraph.

2.5.1 Estimating PageRank for one target node

In [26], the expansion proceeds backwards by following reverse hyperlinks. PageRank scores for *boundary nodes* in the augmented graph that have incoming edges from outside of the subgraph are estimated. Figure 2.6 shows the expansion of a target node to a subgraph. Then standard PageRank is performed on this graph.

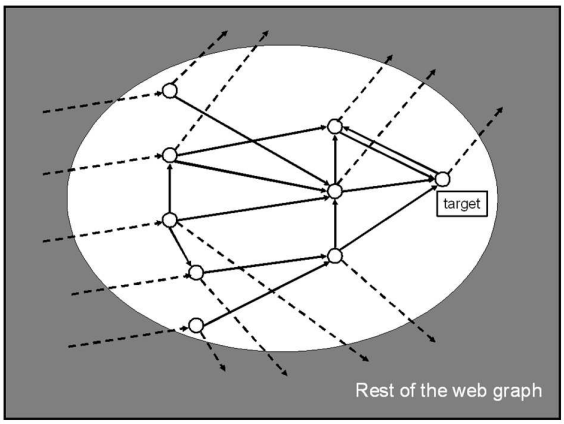

Figure 2.6: The expansion from the target node in [26].

For choosing boundary nodes, several approaches are presented. The first one is a naive method where a subgraph is built by simply following backward links from the target node for a fixed number of levels k, which may quickly expand to the whole Web graph. In other approaches, a fraction of the PageRank value at one node that will eventually reach the target node without a random jump is defined as the *influence* or *indegree-based influence* for a page and estimated in order to choose a proper set of pages to expand.

The experiments report a reasonable estimate of the PageRank value for the

target node – the relative error of a target node is between 5% and 30%.

2.5.2 Estimating PageRank for a subgraph

In contrast, [34] estimates the global PageRank on a local domain, which is motivated by Localized Search Engines. Localized search engines focus on a particular community, for example, all computer science related websites. This paper presents a method of approximating the global PageRank of a local graph. From the given local graph of size n, a supergraph of size $O(n)$ is constructed iteratively by selecting a set of k nodes. This supergraph has the property that the PageRank on this supergraph is close to the true global PageRank.

When the supergraph is constructed, a page selection algorithm is used to decide which nodes are to be included. The priority order of selecting nodes is defined as *influence*, with a definition different from that of [26]. For a candidate page j, the influence is:

$$influence(j) = \sum_{k \in L} |f_j[k] - f[k]| \tag{2.16}$$

where L denotes the local domain, f denotes the PageRank of the local graph, and f_j denotes the L_1-normalized PageRank of the local graph extended by page j and restricted to the local pages.

For each candidate page j, if f_j is accurately calculated, then it is extremely expensive to give the priority order for all candidate pages. The stochastic complement for each page j is constructed and utilized to estimate f_j; the stochastic complementation theory is extensively studied in [69]. The stochastic complemen-

tation approach is able to give an estimation of f_j with a tight lower bound of 0.

The experiments show that the stochastic complementation approach outperforms three other methods, random selection and outlink count, and a heuristic [29] used to choose outgoing URLs when crawling. Running the standard PageRank algorithm on a carefully selected supergraph of $O(n)$ pages provides an estimation of global PageRank scores that is up to 10 times better than just using local PageRank.

2.6 Updating PageRank without Global Computation

Since the Web changes constantly (about 40% of all web pages changed within a week in a sample according to [28]), the PageRank computation faces an updating challenge in two scenarios. The first situation is that only hyperlinks changed and no page insertion or deletion occurs. The second situation is that, aside from hyperlink changes, the set of pages may change as well. The problem of updating PageRank has been studied in [27, 66].

In [27], only link evolution is considered. A single edge insertion algorithm is proposed, and an analysis of this algorithm is provided. The approach is first to construct a small graph G that is close to the two pages involved in a hyperlink and models the rest of the Web with one supernode Ω. Let P denote the transition matrix for the global graph, and the transition matrix T for the new smaller Markov chain is defined as follows [27]:

1. For two pages k and l within G, the transition probability remains the same.

$t_{kl} = p_{kl}$.

2. For transition probabilities from page k in G to Ω is the summation of probabilities to pages represented by Ω. $t_{k\Omega} = \sum_{s \in \Omega} p_{ks}$.

3. The transition probabilities from Ω to page k in G is computed using existing PageRank scores π and P. $t_{\Omega k} = \sum_{s \in \Omega} \frac{\pi_s}{\sum_{s \in \Omega} \pi_s} p_{sk}$.

The analysis of the algorithm gives a bound on the difference between the updated PageRank score for a page s and its obsolete score. Both experiments on synthetic data and real web data support the conclusion that this efficient algorithm yields excellent approximation.

An Iterative Aggregation/Disaggregation (IAD) method [85] is applied to PageRank in [66] to update PageRank scores when either the links or the pages are changed. The Iterative Aggregation Updating Algorithm gradually improves the quality of the ranking vector and it is shown the ranking vector eventually converges to the true PageRank vector.

2.7 Authority Flow Ranking: The ObjectRank Algorithm

Recently personalization attracts a lot of interests in the area of ranking [15, 51, 55, 46, 23]. Topic-sensitive PageRank [51] proposes to precompute a set of personalized PageRank vectors and boosts the ranking quality based on the use of multiple precomputed PageRank vectors.

ObjectRank was introduced in [15] to personalize ranking in Entity-Relation graphs. ObjectRank [15, 53] models the entity sets and semantic connections among

them as a schema graph, where the authority transfer assignment is defined by the user. PopRank [75] presents a similar idea to personalize to give a personalized ranking for web objects. Section 2.8 provides more details about scaling personalized search.

Let A_{OR}^T denote the transpose of A_{OR}. The ObjectRank vector R_{OR} is recursively defined as follows in Equation (5.3):

$$R_{OR} = \epsilon A_{OR}^T \cdot R_{OR} + (1 - \epsilon)P \tag{2.17}$$

In ObjectRank, the transition matrix A_{OR} depends on the authority transfer specified on the *schema graph* (Figure 5.1); it defines the authority transferred along each edge type. To demonstrate the relationship of the ObjectRank transition matrix and the PageRank transition matrix, without loss of generality we assume that the objects of the same type are grouped together. Suppose an authority transfer schema graph contains t types of objects.

Let $\Theta = \{\alpha_{1,1}, \alpha_{1,2}, ..., \alpha_{1,t}, \alpha_{2,1}, \alpha_{2,2}, ..., \alpha_{2,t}, ..., \alpha_{t,1}, \alpha_{t,2}, ..., \alpha_{t,t}\}$ be the vector of authority transfer weights assigned to the semantic types. We refer to this vector as the *weight assignment vector* henceforth. These terms are followed in Chapter 5 and Chapter 6.

A_{OR} can be expressed as follows: Each entry of the transition matrix for A_{OR} is multiplied by the authority transfer weight for the corresponding semantic type. A_{OR} contains $t \times t$ submatrices.

$$A_{OR} = \begin{pmatrix} \alpha_{1,1}A_{1,1} & \alpha_{1,2}A_{1,2} & \cdots & \alpha_{1,t}A_{1,t} \\ \alpha_{2,1}A_{2,1} & \alpha_{2,2}A_{2,2} & \cdots & \alpha_{2,t}A_{2,t} \\ \vdots & \vdots & & \vdots \\ \alpha_{t,1}A_{t,1} & \alpha_{t,2}A_{t,2} & \cdots & \alpha_{t,t}A_{t,t} \end{pmatrix} \tag{2.18}$$

The submatrix $A_{p,q}$ contains authority transfer probabilities from objects of type p to objects of type q. Let $e^T(v_i, v_j)$ be the semantic type of edge (v_i, v_j) and let $\alpha(e^T(v_i, v_j))$ denote the weight assignment for $e^T(v_i, v_j)$. $OutDeg(v_i, e^T(v_i, v_j))$ is the number of outgoing edges from page v_i, of type $e^T(v_i, v_j)$. The submatrix $A_{p,q}$ is defined as follows:

$$A_{p,q}[i,j] = \begin{cases} \frac{1}{OutDeg(v_i, e^T(v_i, v_j))} & \text{if there is an edge from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

2.8 Personalized Search

Personalized ranking adjusts the order of Web pages based on users' preferences, which can be implicit or explicit. The two key ways to achieve personalization for the PageRank algorithm are:

1. Personalize with a personalized base set. This involves selecting user-dependent entities as the source of the authority in the graph [51, 76].

2. Personalize by adjusting the authority flow weight of the edges. This allows users to assign different importance to different types of edges [15, 75].

These personalization approaches suffer from scalability issues, as these algorithms are as expensive as the PageRank algorithm – they all require multiple iterations to reach convergence. While there are numerous approaches to scale the first type of personalization, scaling the second type of personalization remains an open problem. Next we present the existing approaches to scale personalization.

2.8.1 Scaling personalization with a base set

Personalization of PageRank using a base set has been proposed in the original PageRank [76]. The problem of achieving scalable personalization based on a personalized base set, i.e., a personalization vector, has been studied in [23, 46, 51, 57, 55].

Recall that in the PageRank definition, vector v_P is the personalization vector that replaces P. Then the personalized PageRank equation can be written as follows:

$$R_P = \epsilon A^T \cdot R + (1 - \epsilon) v_P \tag{2.20}$$

R_P is the personalized PageRank vector (PPV) for personalization vector v_P.

Theorem 1 (The Linearity Theorem) [51, 57] *For any personalization vectors v_{P1} and v_{P2}, if R_{P1} and R_{P2} are the two corresponding PPVs, then for any constants β_1, $\beta_2 \geq 0$ such that $\beta_1 + \beta_2 = 1$,*

$$\beta_1 R_{P1} + \beta_2 R_{P2} = \epsilon A^T \cdot (\beta_1 R_{P1} + \beta_2 R_{P2}) + (1 - \epsilon)(\beta_1 v_{P1} + \beta_2 v_{P2})$$

[51] is the first work to scale personalized PageRank. Haveliwala proposed to precompute a set of topic-specific PageRank vectors, and to use these vectors to

generate query-specific ranking scores at query time. These results are based on the linearity theorem, which is used to combine multiple personalization vectors.

Based on Theorem 1, [57] proposed a technique that encodes personalized ranking vectors as partial vectors, which are shared across multiple personalized PageRank vectors. They also presented efficient dynamic programming algorithms to compute partial vectors and an algorithm to compute personalized PageRank using partial vectors. In [46], an algorithm that simulates random walks is used to precompute an *index database* of personalized PageRank vectors (fingerprints).

Recently [23, 55] consider personalized ranking on entity-relation graphs, however, we categorize these work as personalization with a base set instead of personalization with a weight assignment. The reason is that these work are all based on the same mathematical tool – linearity theorem for personalized ranking [51, 57], therefore, the techniques do not utilize precomputed authority flow rankings with different weight assignments.

HubRank [23] employs query log statistics to select a small fraction of nodes, and computes and stores fingerprints for these nodes. A small subgraph is identified at query time to form approximate personalized PageRank vectors. BinRank [55] stores subgraphs such that any keyword query can be answered by performing ObjectRank on one subgraph.

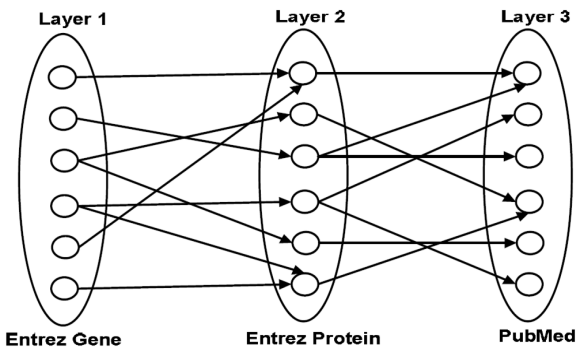

Figure 2.7: The layered graph ([90])

2.8.2 Approximation methods for personalization with a weight assignment

The only work that approximates authority flow rankings for personalization with weight assignments, as far as we know, is to ues sampling techniques [90] for lgOR [79]. lgOR is a specialization of ObjectRank. In lgOR, navigational queries are considered. A navigational query produces a layered result graph (Figure 2.7), and the authority scores for objects are only determined based on the scores of objects in the previous layer. The lgOR is terminated until the scores for the target objects in the last layer are computed. Therefore, compared to ObjectRank algorithm, which is an iterative algorithm, lgOR algorithm is computed through just one iteration and authority flows from the first layer to the target layer.

Since lgOR [90] deals with layered graphs, a graph-sampling technique is applied to approximate the lgOR scores. The problem is reduced to estimating a subgraph for the result graph, such that with high confidence the relative error of computing lgOR on the subgraph is small. lgOR considers a special type of queries,

navigational queries. Hence, the sampling technique applied to lgOR does not apply to the ObjectRank which is computed on general Web graphs.

PopRank [75] applied the idea of authority flow rankings to the Web objects. A simple simulated annealing algorithm is presented to learn a good query weight assignment. [75] does not estimate the authority flow rankings for a weight assignment as well.

Although computing personalization with a base set is well-studied, computing authority flow ranking remains an open problem. There are few research utilizes the precomputed authority flow rankings to approximate ranking for a new weight assignment vector. Therefore, we study approximation methods for authority flow rankings in this thesis.

2.9 Using Relevance Feedback for Authority Flow Ranking

Authority flow ranking has been successfully applied to different scenarios, including bibliographic databases (ObjectRank), and biological databases. These papers assumed that the user or domain expert provide weight assignment vectors. An extension is to use feedback from users to determine the weight assignment vector [14, 89].

Relevance feedback has been widely used to improve result quality in Information Retrieval [38, 81, 82]. Query expansion technique plays the dominant role in relevance feedback, where keywords are added to the original query based on the user's feedback.

[89] proposed two types of query reformulation for ObjectRank: Content-based Reformulation and Structure-based Reformulation. The Structure-based reformulation adjusts the weight assignment vector for the schema graph. Firstly an explaining subgraph is constructed, which contains all edges that transfer authority to a given page. Intuitively, in this explaining subgraph, if the edges of a particular edge type have high probability to be followed in random walk, then the weight for this edge type is increased. The original weight for an edge type is adjusted based on this intuition.

[14] considers a different random walk from ObjectRank, where the probabilities of following outgoing links depend on the weight assignments solely. While in ObjectRank, the probabilities of following outgoing links depend on two factors: the weight assignment and the outdegree of each node (See Equation 5.1 and 5.2). In [14], a dummy node d is added and dummy edges are added between d and every other node in the graph. To learn the authority flow weight assignments on different edge types, [14] defines an optimization problem. The ranking order of a pair of pages is used to enforce the amount of authority flow to the two pages, which has an impact on the weight assignment vector.

Chapter 3

Estimating Rank for a Subgraph

We present a framework of an exact solution and an approximate solution, IdealRank and ApproxRank, for computing PageRank on a subgraph. The subgraph ranking problem has been motivated by a variety of applications, including ranking for a focused crawler, personalized search, the updated ranking when the Web is changed, and PageRank computation in distributed system like peer-to-peer networks. These applications identify a subgraph and the ranking scores for the subgraph are needed. The details of these applications are described in Section 1.1.

For both IdealRank and ApproxRank algorithms, the global graph and the local graph are given and the local graph is contained in the global graph. The IdealRank and ApproxRank framework can be used to compute ranking for the local graph. It is also possible to execute the global PageRank algorithm to compute ranking scores for the global graph. However, motivated by multiple applications that focus on a subgraph, the cost of IdealRank and ApproxRank for ranking a subgraph is orders of magnitude cheaper than applying the global PageRank, while IdealRank achieves the exact ranking and ApproxRank achieves an approximation with bound.

If the global graph is not available, depending on the amount of available information about the global graph, there are a few possibilities for computing

ranking on a subgraph. The experimental results are reported in Chapter 6 for different algorithms.

- When no information about the global graph is given, local PageRank is applied on the local graph.
- In some cases, partial information about the global graph is given. e.g. for each local page, we know if it is connected to some external page in the global graph. LPR2 algorithm (See Section 3.2.3) can be applied.
- In some cases, the neighborhood graph around the local graph is available. e.g. the local graph can be expanded to a supergraph. SC algorithm [34] (details in 2.5.2) can be applied.
- When the global graph is given, we can apply IdealRank and ApproxRank to compute ranking on a subgraph.

The IdealRank algorithm assumes the scores of external pages are known, and the IdealRank scores for pages in the local graph converge to the true PageRank scores. The ApproxRank is an approximation with bound, without assuming the scores of external pages. The experiments with real and synthetic data show that ApproxRank provides a good approximation to PageRank for a variety of subgraphs.

The chapter is organized as follows. In Section 3.1, we present the IdealRank algorithm and show that IdealRank scores converge to the true PageRank scores which are obtained through global computation. In Section 3.2, we present the ApproxRank algorithm that estimate PageRank scores for a subgraph, and we conduct

error analysis for the ApproxRank scores. These results are presented in [92].

3.1 IdealRank Approach

We formally define the IdealRank algorithm to compute PageRank scores for a local graph. Our approach is inspired by research on collapsing matrices with the same eigenvector [52].

Consider two graphs; a global graph of size N, and a local graph of size n. The local graph is a subgraph of the global graph. The pages in the local graph are called *local pages* while pages in the global graph and that are not in the local graph are called *external pages*. In the IdealRank problem, we assume that PageRank scores of all external pages in Λ are known. This assumption will be relaxed in the next section where we present an approximate solution. The goal is to provide the true PageRank for the local graph without running PageRank on the global graph.

Table 3.1 lists the symbols used to define our algorithms.

Symbol	Meaning
Λ	External node, the artificial node representing all external pages.
G_l	A subgraph of the Web with n pages
G_g	The global Web graph with N pages.
G_e	The extended local graph with $n+1$ pages.

Table 3.1: Symbols used by algorithms

3.1.1 The IdealRank algorithm

IdealRank performs a random walk on a modified local graph called the *extended local graph*, where an external node Λ is added to the local graph. Λ represents the set of pages that are not local. The transition matrix probabilities of IdealRank are derived from the transition matrix of PageRank for the global graph.

Recall that in [91], an artificial node represents the external world. There are edges between the artificial node and local nodes based on the global Web graph. However, this solution cannot distinguish between the case of one link or multiple links between a local page and the external pages as seen in the following example:

Let Figure 3.1 be a global graph. Node A,B,C, and D are local pages, and node X, Y and Z in the cloud are external pages. Figure 3.2 provides an example of adding an artificial external node to represent the external pages. Edges are added from local pages to the external node. However, there is no strategy to adjust the probability flow in the random walk. Ideally, the random walk should reflect that each edge may represent multiple edges in the global graph. When computing the standard PageRank algorithm on this graph, the probability flow from a page is proportional to the inverse of its outdegree. Page C which has 3 incoming edges from the external pages is treated similarly to page D which has only 1 incoming edge from the external pages. Intuitively, however, we should expect a higher probability of following links from the external pages to page C. Similarly, the probability of following links from page A to Λ is 1/3. This too is lower than the transition probability based on the global graph.

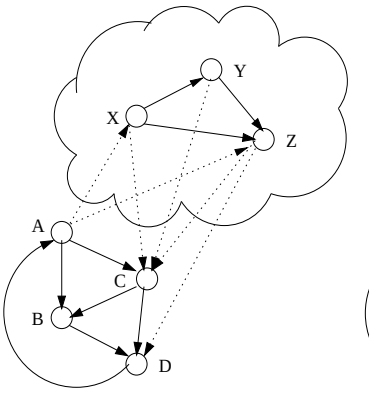

Figure 3.1: A global graph of both local pages and external pages.

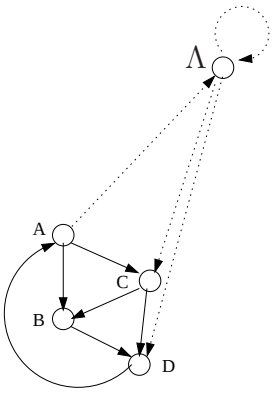

Figure 3.2: An extended local graph without a strategy to adjust transition probabilities

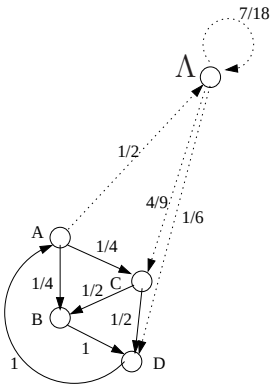

Figure 3.3: An extended local graph marked with transition probabilities in ApproxRank

IdealRank addresses this shortcoming with the following solution: The first step is to add an external node Λ to the subgraph to represent all external pages. The second step is to construct the *extended local graph* G_e, the Λ enriched graph of size $n+1$. There is an edge from Λ to a local page in G_e if there is an edge from an external page to that local page. The same hold for edges out of local pages. Similarly, there is an edge from Λ to Λ if there is an edge between external pages. The next step is to define a transition matrix A_{ideal} and a personalization vector P_{ideal}. The details will be discussed in Section 3.1.2. Finally, a random walk is performed on G_e. The IdealRank vector R_{ideal} is defined as follows:

$$R_{ideal} = \epsilon A_{ideal}^T \cdot R_{ideal} + (1 - \epsilon) P_{ideal} \tag{3.1}$$

Figure 3.4 is the IdealRank algorithm outline:

Algorithm IdealRank(G_l, G_g)

1. Add external node Λ to G_l.
2. Create edges associated with Λ and get G_e.
3. Assign values to P_{ideal} and A_{ideal}.
4. Perform a random walk on the extended local graph according to Formula (3.1).

Figure 3.4: The outline of the IdealRank algorithm.

3.1.2 A_{ideal} and P_{ideal}

We define an $(n+1) \times (n+1)$ transition matrix A_{ideal} and a length $(n+1)$ personalization vector P_{ideal}. Let A represent the $N \times N$ transition matrix for PageRank on the global graph. Entry $A_{i,j}$ has the value of the inverse outdegree of page i, if there is an edge (i, j); the value is the probability of a random surfer following this edge from i. Without loss of generality, we consider the local pages to be the first contiguous n pages in A and the external pages are indexed from $n+1$ to N in A.

Assume that the PageRank scores for all external pages are known. The values are $\{R[n+1], R[n+2], \cdots, R[N]\}$, respectively. Let $EXTSum = \sum_{i=n+1}^{N} R[i]$. A_{ideal}

is defined as follows, based on the entries in the original PageRank transition matrix A:

$$A_{ideal} = \left(\begin{array}{ccc|c} A_{1,1} & \cdots & A_{1,n} & \sum_{i=n+1}^{N} A_{1,i} \\ \vdots & & \vdots & \vdots \\ A_{n,1} & \cdots & A_{n,n} & \sum_{i=n+1}^{N} A_{n,i} \\ \hline \underbrace{\sum_{j=n+1}^{N} R[j]A_{j,1}}_{EXTSum} & \cdots & \underbrace{\sum_{j=n+1}^{N} R[j]A_{j,n}}_{EXTSum} & \underbrace{\sum_{j=n+1}^{N} R[j] \sum_{i=n+1}^{N} A_{j,i}}_{EXTSum} \end{array}\right) \tag{3.2}$$

Next we explain the elements in A_{ideal}. These values are as follows:

1. The $n \times n$ submatrix at upper left is identical to the corresponding elements in transition matrix A for the global graph. They represent the probability of transition between edges in the local graph.

2. The $n \times 1$ submatrix at upper right represents the probability flow from a local page to the node Λ. We note that the probability of reaching Λ is the sum of the probability of reaching any external page from the local page. For local page k, the value is $\sum_{i=n+1}^{N} A_{k,i}$.

3. The $1 \times n$ submatrix at lower left corresponds to the probability flow from Λ to local pages. For local page k, the value is $\frac{\sum_{j=n+1}^{N} R[j]A_{j,k}}{EXTSum}$.

4. The entry at the lower right corner denotes the probability flow from Λ to Λ.

The last row has entries that are each a weighted sum of probabilities summed

over all external pages. The weight is determined by the PageRank score of the external page. This is a key feature of A_{ideal} and will be discussed next.

We define A_{ideal} formally as follows: $A_{ideal} = Q_1 A Q_2$, where Q_1 is an $(n+1) \times N$ matrix and Q_2 is an $N \times (n+1)$ matrix. Let Q_2 be an $N \times (n+1)$ matrix as follows:

$$\begin{pmatrix} I_n & B \\ C & D \end{pmatrix} \tag{3.3}$$

where I_n is an $n \times n$ identity matrix, B is an $n \times 1$ 0-matrix, C is a $(N-n) \times n$ 0-matrix, and D is a $(N-n) \times 1$ matrix with all 1's. The effect of AQ_2 on the ranking vector is to aggregate the authority flow from local pages to all external pages, which indicates the authority goes to Λ.

Let Q_1 be the following $(n+1) \times N$ matrix:

$$\begin{pmatrix} I_n & C^T \\ B^T & E \end{pmatrix} \tag{3.4}$$

where I_n is an $n \times n$ identity matrix, C^T is an $n \times (N-n)$ 0-matrix and B^T is a $1 \times n$ 0-matrix.

The matrix of interest is E, a $1 \times (N-n)$ matrix. It considers the PageRank scores for all external pages. Recall that $EXTSum$ is the sum of PageRank scores for all external pages, $EXTSum = \sum_{i=n+1}^{N} R[i]$. Then, E can be expressed as follows:

$$E = \begin{pmatrix} \frac{R[n+1]}{EXTSum}, \frac{R[n+2]}{EXTSum}, \cdots, \frac{R[N]}{EXTSum} \end{pmatrix} \tag{3.5}$$

The idea of multiplying the values of entries in A with the two matrices Q_1 and Q_2, where Q_1 derived from the ranking vector for external pages, is *key* to the

approach of A_{ideal}. It has the effect of distributing the probability flow from the external nodes, in a manner that is proportional to the importance of each of the external pages in the original PageRank vector.

Recall that the personalization vector in the original PageRank is defined as a uniform vector $P = [\frac{1}{n}]_{n\times 1}$. Instead, for IdealRank we define the personalization vector P_{ideal} according to the number of external pages and total number of pages in the graph. More specifically, the i-th entry of P_{ideal}, $P_{ideal}[i]$ can be expressed as follows:

$$P_{ideal}[i] = \begin{cases} \frac{1}{N} & \text{if page } i \text{ is local,} \\ \frac{N-n}{N} & \text{if page } i \text{ is the external node } \Lambda. \end{cases} \tag{3.6}$$

3.1.3 Convergence of IdealRank

Let R_{ideal} be the final ranking vector of *IdealRank*, where the first n elements are scores for local pages and the $(n+1)$-th element is the score for the external node Λ. Let R be the PageRank vector for the global graph of N pages. We show that the scores of first n elements are identical to the true PageRank scores. The score for the $(n+1)$th element, Λ, converges to the sum of true PageRank scores for all external pages.

Theorem 2 *For local pages with indices* $i = 1, 2, \cdots, n$, $R_{ideal}[i] = R[i]$. *For* Λ, $R_{ideal}[n+1] = \sum_{i=n+1}^{N} R[i]$.

Proof. Let R be the true PageRank vector such that $R = \epsilon A^T \cdot R + (1-\epsilon)P$, i.e., R is the converged stationary distribution for A. Let $R' = Q_2^T R$ be a vector with

$n+1$ entries. We also know that $R = Q_1^T R'$. It is obvious that $R'[i] = R[i]$ for first n elements and $R'[n+1] = \sum_{i=n+1}^{N} R[i]$. We will show that R' is the IdealRank vector.

We know that $\epsilon A^T R + (1-\epsilon)P = R$. Next consider a left multiply with Q_2^T to obtain the following:

$$\begin{aligned}
\epsilon A^T R + (1-\epsilon)P &= R \quad \Rightarrow \\
Q_2^T \epsilon A^T R + Q_2^T (1-\epsilon)P &= Q_2^T R \quad \Rightarrow \\
\epsilon Q_2^T A^T Q_1^T R' + (1-\epsilon) Q_2^T P &= Q_2^T R \quad \Rightarrow \\
\epsilon (Q_1 A Q_2)^T R' + (1-\epsilon) P_{ideal} &= R' \quad \Rightarrow \\
\epsilon A_{ideal} R' + (1-\epsilon) P_{ideal} &= R'
\end{aligned} \tag{3.7}$$

Since A_{ideal} is stochastic and Markov Chain defined by IdealRank is irreducible and aperiodic, there is a unique stationary distribution for A_{ideal}. Therefore, $R' = R_{ideal}$. ∎

The IdealRank algorithm addresses several applications. One is where some subgraph of the Web graph has been updated. A second case is when the personalized authority transfer is limited to the subgraph. In these cases, the knowledge of PageRank scores can be potentially relied on to estimate new ranking scores.

3.2 The ApproxRank algorithm

Unlike the previous scenario where PageRank values for external pages are known, we now consider scenarios where the PageRank scores are not known a priori. To cover this situation, our framework has an approximate solution ApproxRank.

The key difference is that for ApproxRank, the algorithm is not able to differentiate the (previously weighted) contribution of authority from each individual external page (since these PageRank scores are unknown). Instead, ApproxRank will consider the authority flow from external pages assuming they are equally important. We analyze the L_1 distance between IdealRank scores and ApproxRank scores of the subgraph and reveal that it is within a constant factor of L_1 distance between the true PageRank scores and uniform scores of the external pages. We will show through experiments that ApproxRank is a good approximation and is superior to the best existing approach.

ApproxRank estimates the PageRank scores for web pages within a local graph efficiently. Google reported that they have indexed 1 trillion pages in July 2008 [6]. However, in different applications such as focused crawler, a user may be only interested in a subgraph of size of a few thousand pages. ApproxRank applies when the global computation is expensive and the user does not care about the external pages. ApproxRank avoids the cost of the global computation and produce an approximation which guarantees quality. The suitable applications include focused crawler, personalized search, and meta-searcher (see Section 1.1).

3.2.1 The ApproxRank algorithm

The ApproxRank vector R_{approx} is defined as follows:

$$R_{approx} = \epsilon A_{approx}^T \cdot R_{approx} + (1 - \epsilon) P_{ideal} \tag{3.8}$$

ApproxRank adopts the same personalization vector as IdealRank. It however,

defines its own transition matrix A_{approx}.

3.2.2 A_{approx} definition

A_{approx} is an $(n+1) \times (n+1)$ matrix. It is defined as follows:

$$A_{ideal} = \left(\begin{array}{ccc|c} A_{1,1} & \cdots & A_{1,n} & \sum_{i=n+1}^{N} A_{1,i} \\ \vdots & & \vdots & \vdots \\ A_{n,1} & \cdots & A_{n,n} & \sum_{i=n+1}^{N} A_{n,i} \\ \hline \frac{\sum_{j=n+1}^{N} A_{j,1}}{N-n} & \cdots & \frac{\sum_{j=n+1}^{N} A_{j,n}}{N-n} & \frac{\sum_{i=n+1}^{N}\sum_{j=n+1}^{N} A_{i,j}}{N-n} \end{array}\right) \tag{3.9}$$

A_{approx} is different from A_{ideal} in the last row (see Section 3.1.2), since IdealRank does not utilize knowledge about PageRank scores of external pages in the first n rows. For the first n entries in the last row, the value represents the (average) probability flow accumulated from $(N-n)$ external pages to each local page. The last entry in this n-th row of the matrix is the (average) probability flow from external pages to other external pages. Similar to $A_{ideal} = Q_1 A Q_2$, A_{approx} can be formally defined as $A_{approx} = Q'_1 A Q_2$, where the vector E is replaced by a vector E_{approx} in Q'_1:

$$E_{approx} = \left(\frac{1}{N-n}, \frac{1}{N-n}, \cdots, \frac{1}{N-n} \right) \tag{3.10}$$

In A_{approx}, the values at the last row are as follows:

1. For the first n values, $(1 <= k <= n)$, the probability from Λ to a local page

k is assigned the summation of flow from all external pages to k, divided by the number of external pages. For local page k, it is $\frac{\sum_{j=n+1}^{N} A_{j,k}}{N-n}$.

2. For the $(n+1)$-th value, the probability for the self-loop edge is determined by the total authority flow among external pages, divided by the number of external pages.

Given the global graph example in Figure 3.1, the probabilities assigned by A_{approx} are shown in Figure 3.3. We provide some examples of edge weight calculation following these rules. According to rule 1, the authority flow on edge AB, AC, CB, BD, CD, DA are the outdegree inverse. Since A points to page X, Z, the authority flow on edge (A, Λ) is $1/2$. The authority flow on edge (Λ, C), $\frac{\frac{1}{D_X}+\frac{1}{D_Y}+\frac{1}{D_Z}}{3} = \frac{\frac{1}{3}+\frac{1}{2}+\frac{1}{2}}{3} = \frac{4}{9}$. The self-loop edge authority flow will be $\frac{\frac{2}{D_X}+\frac{1}{D_Y}}{3} = \frac{\frac{2}{3}+\frac{1}{2}}{3} = \frac{7}{18}$.

The ApproxRank algorithm reduces the computation cost of global PageRank dramatically. Let E_g denote the number of edges in the global graph and E_l denote the number of edges in the extended local graph. Using the cheapest power method for eigenvectors, the runtime for the PageRank algorithm is $O(E_g)$ and ApproxRank complexity is $O(E_l)$. Note that the constant involved in the big O notation is larger for global PageRank as well. Since E_l is usually a small fraction of E_g, ApproxRank would bring significant runtime savings.

Another advantageous quality about ApproxRank is that it is suitable to adopt precomputation for various subgraphs. With the same global graph, A_{approx} can be figured out easily from the difference between the local values and the global values.

This is especially beneficial for applications where there are multiple subgraphs.

ApproxRank scores converge to a unique vector R_{approx}. There are two reasons. First, the transition matrix A^T_{approx} is a column stochastic matrix, as the sum of each column is 1. Second, since we complement the random walk with jumps from dangling pages, the Markov Chain we defined is irreducible and aperiodic. ApproxRank satisfies the two conditions of being irreducible and aperiodic of the Ergodic Theorem for Markov chains [60]. Next we will investigate how close is R_{approx} to R_{ideal}, which we have shown to be the true PageRank scores for local pages.

3.2.3 Error analysis of ApproxRank ranking vector R_{approx}

In this section we provide important properties of ApproxRank scores through iterations. We show that the L_1 distance between IdealRank scores and ApproxRank scores of the subgraph is within a constant factor of L_1 distance between the true PageRank scores and assumed scores of the external pages. This relationship can be utilized to improve ApproxRank algorithm, which will be our future work. Our experiments show that, however, even assume that the external pages are equally important, ApproxRank behaves well and produces comparable results to existing approach. To our best knowledge, similar analysis has not been conducted in previous work for PageRank estimation [34, 26, 91]. There are analysis results of the same flavor through different approaches in the area of analysis of PageRank [18, 74] and in the area of updating PageRank scores [27].

Let R_{ideal} and R_{approx} be the ranking vectors from IdealRank and ApproxRank

respectively, each with length $n+1$, where the $(n+1)th$ elements in vectors are scores for the external node Λ. We abuse notations and let R_{ideal} and R_{approx} be the subvector of the first n elements, as we are interested in accuracy of ApproxRank for the n local pages. Let R^m_{ideal} and R^m_{approx} be the ranking vectors after the m-th iteration from IdealRank and ApproxRank.

Let E and E_{approx} in Equation (3.5) and (3.10) be the vector used to define A_{ideal} and A_{approx}. Both E and E_{approx} are vectors of length $N-n$, where each element denotes the relative importance of $N-n$ external pages. We note that we index these elements with $n+1, \cdots, N$ to reference the scores for the corresponding external pages.

Theorem 3.2.3 states that after m iterations,

$$\| R^m_{ideal} - R^m_{approx} \|_1 \leq (\epsilon^m + \epsilon^{m-1} + \cdots + \epsilon) \| E - E_{approx} \|_1$$

When the number of iterations goes to infinity, this becomes

$$\| R^\infty_{ideal} - R^\infty_{approx} \|_1 \leq \frac{\epsilon}{1-\epsilon} \| E - E_{approx} \|_1$$

This shows that the accuracy of ApproxRank is dependent on the knowledge of relative importance of external pages. When ϵ is set to be 0.85, which is usually the case, the error of ApproxRank is bounded by a constant factor of 5.67 of the error of E_{approx}.

We first derive the base case and recurrence relation for the L_1 distance between ApproxRank ranking vector and IdealRank ranking vector. Then an error bound are obtained based on a priori error of the external pages in Theorem 3.2.3.

Lemma 3.2.1 *After the first iteration, the ApproxRank ranking vector R^1_{approx} satisfies:*

$$\| R^1_{ideal} - R^1_{approx} \|_1 \leq \epsilon \| E - E_{approx} \|_1$$

Proof.

$$\begin{aligned}
& \| R^1_{ideal} - R^1_{approx} \|_1 \\
= \; & \textstyle\sum_{k=1}^{n} |R^1_{ideal}[k] - R^1_{approx}[k]| \\
= \; & \textstyle\sum_{k=1}^{n} |\epsilon \sum_{i=1}^{n} A_{ik} \cdot 1 + \epsilon \sum_{j=n+1}^{N} A_{jk} E[j] + (1-\epsilon)\frac{1}{N} \\
& \textstyle -\epsilon \sum_{i=1}^{n} A_{ik} \cdot 1 - \epsilon \sum_{j=n+1}^{N} A_{jk} E_{approx}[j] - (1-\epsilon)\frac{1}{N}| \\
= \; & \textstyle \epsilon \sum_{k=1}^{n} | \sum_{j=n+1}^{N} A_{jk} (E[j] - E_{approx}[j])| \\
\leq \; & \textstyle \epsilon \sum_{k=1}^{n} \sum_{j=n+1}^{N} A_{jk} |E[j] - E_{approx}[j]| \\
\leq \; & \textstyle \epsilon \sum_{j=n+1}^{N} \sum_{k=1}^{n} A_{jk} |E[j] - E_{approx}[j]| \\
\leq \; & \textstyle \epsilon \sum_{j=n+1}^{N} |E[j] - E_{approx}[j]| \\
\leq \; & \epsilon \| E - E_{approx} \|_1
\end{aligned}$$

To derive the inequality, we first express the L_1 distance based on its definition, then calculate $R^1_{ideal}[k]$ and $R^1_{approx}[k]$ assuming that the initial vectors for IdealRank and ApproxRank are the same (e.g. 1) for local pages. Because transition matrix A is row stochastic, $\sum_{k=1}^{n} A_{jk} \leq 1$. The definition of L_1 distance concludes the proof.

■

Next we explore a recurrence relation for the L_1 distance between R_{approx} and R_{ideal} after m iterations. Lemma 3.2.2 shows that after each iteration, the L_1 distance deviate not too much.

Lemma 3.2.2 *After an arbitrary positive integer $m > 1$ iterations, we have the following recurrence relation:*

$$\| R^m_{ideal} - R^m_{approx} \|_1 \;\leq\; \epsilon \| R^{m-1}_{ideal} - R^{m-1}_{approx} \|_1 + \epsilon \| E - E_{approx} \|_1$$

Proof. Albeit more terms involved, the proof follows the same vein with the base case in Lemma 5.7.1.

$$\begin{aligned}
& \| R^m_{ideal} - R^m_{approx} \|_1 \\
= \;& \textstyle\sum_{k=1}^{n} |R^m_{ideal}[k] - R^m_{approx}[k]| \\
= \;& \textstyle\sum_{k=1}^{n} |\epsilon \sum_{i=1}^{n} A_{ik} \cdot R^{m-1}_{ideal}[i] + \epsilon \sum_{j=n+1}^{N} A_{jk} E[j] + (1-\epsilon)\frac{1}{N} \\
& \textstyle -\epsilon \sum_{i=1}^{n} A_{ik} \cdot R^{m-1}_{approx}[i] - \epsilon \sum_{j=n+1}^{N} A_{jk} E_{approx}[j] - (1-\epsilon)\frac{1}{N}| \\
= \;& \textstyle \epsilon \sum_{k=1}^{n} | \sum_{i=1}^{n} A_{ik} \cdot (R^{m-1}_{ideal}[i] - R^{m-1}_{approx}[i]) + \sum_{j=n+1}^{N} A_{jk}(E[j] - E_{approx}[j])| \\
\leq \;& \textstyle \epsilon \sum_{k=1}^{n} | \sum_{i=1}^{n} A_{ik} \cdot (R^{m-1}_{ideal}[i] - R^{m-1}_{approx}[i])| + \epsilon \sum_{k=1}^{n} | \sum_{j=n+1}^{N} A_{jk}(E[j] - E_{approx}[j])| \\
\leq \;& \textstyle \epsilon \sum_{k=1}^{n} \sum_{i=1}^{n} A_{ik} |R^{m-1}_{ideal}[i] - R^{m-1}_{approx}[i]| + \epsilon \sum_{k=1}^{n} \sum_{j=n+1}^{N} A_{jk} |E[j] - E_{approx}[j]| \\
\leq \;& \textstyle \epsilon \sum_{i=1}^{n} \sum_{k=1}^{n} A_{ik} |R^{m-1}_{ideal}[i] - R^{m-1}_{approx}[i]| + \epsilon \sum_{j=n+1}^{N} \sum_{k=1}^{n} A_{jk} |E[j] - E_{approx}[j]| \\
\leq \;& \textstyle \epsilon \sum_{i=1}^{n} |R^{m-1}_{ideal}[i] - R^{m-1}_{approx}[i]| + \epsilon \sum_{j=n+1}^{N} |E[j] - E_{approx}[j]| \\
\leq \;& \epsilon \| R^{m-1}_{ideal} - R^{m-1}_{approx} \|_1 + \epsilon \| E - E_{approx} \|_1
\end{aligned}$$

■

Theorem 3.2.3

$$\| R^m_{ideal} - R^m_{approx} \|_1 \leq (\epsilon^m + \epsilon^{m-1} + \cdots + \epsilon) \| E - E_{approx} \|_1$$

Proof. The proof is straightforward by combining Lemma 5.7.1 and Lemma 3.2.2.

$$
\begin{aligned}
& \| R^m_{ideal} - R^m_{approx} \|_1 \\
\leq\ & \epsilon(\epsilon \| R^{m-2}_{ideal} - R^{m-2}_{approx} \|_1 + \epsilon \| E - E_{approx} \|_1) + \epsilon \| E - E_{approx} \|_1 \\
\leq\ & \epsilon^{m-1} \| R^1_{ideal} - R^1_{approx} \|_1 + (\epsilon^{m-1} + \epsilon^{m-2} + \cdots + \epsilon) \| E - E_{approx} \|_1 \\
\leq\ & (\epsilon^m + \epsilon^{m-1} + \cdots + \epsilon) \| E - E_{approx} \|_1
\end{aligned}
$$

■

When PageRank scores of external pages are not known, ApproxRank estimates the PageRank scores for local pages. We provide important properties of ApproxRank scores. This bound shows that the quality of ApproxRank depends on the accuracy of estimation of external page ranking scores. If we start with a good estimation for external scores, we may approach a better approximation in ApproxRank.

Although ApproxRank is designed to efficiently compute ranking for a subgraph, ApproxRank can be personalized without any extra effort, for personalization with a base set (See Section 2.8). In ApproxRank, the personalization vector P_{ideal} can be adjusted based on the users' preferences, and the algorithm remains unchanged and the bound for ApproxRank still holds.

ApproxRank exploits the global graph structure to produce an accurate approximation, as the outdegrees for local pages and external pages are used in ApproxRank transition matrix. If the outdegrees for external pages are not available, there exists a similar approximation method LPR2 [91].

The LPR2 algorithm is a component of the ServerRank algorithm. For a subgraph of size n, an artificial page ξ is added to construct a local graph with $n+1$

pages. If there is an edge connecting local page i to an out-of-domain page, then page i and ξ are connected in the constructed graph. The standard PageRank is computed on this graph. The LPR2, however, does not have any quality bound. We compare ApproxRank and LPR2 in Chapter 4.

Chapter 4

The Evaluation for ApproxRank

Note that IdealRank is of theoretical interests and not applicable for estimating rank for a subgraph where the PageRank scores of the external pages are not known a priori. Therefore, we limit our experimental evaluation in this chapter to ApproxRank.

4.1 Experiment Description

To evaluate our approach, we consider two goals in experiments. The first goal is to compare the ApproxRank with the stochastic complementation (SC) approach [34], which is the best existing approach for the problem. The second goal of experiments is to study the effect of size and type of the subgraphs on accuracy of the ApproxRank vector.

Ideally we would run experiments on the whole web graph, which is obviously infeasible. In choosing appropriate datasets, we first surveyed a few recent ranking papers and we list the key characteristics of their datasets in Table 4.1. We will take a similar approach of crawling a relatively small portion of the Web, and let it reflect the whole Web.

We consider the following three types of subgraph in our experiments:

- **TS subgraph**: The first type of subgraph is a topic specific subgraph.

Paper	data description	#pages (million)	#links (million)
[34]	"edu": crawl of 100 CS domains	4.7	22.9
	"politics": crawl under politics hierarchy	4.4	17.3
[75]	web objects including papers, authors etc	1.65	7
[78]	Amazaon.com data	0.055	0.237
	Web crawl	0.103	1.63
[91]	A breadth first search crawl within domain www.standford.edu	1.05	4.98

Table 4.1: Dataset characteristics from recent ranking papers.

- **DS subgraph**: This type of subgraph is a domain specific subgraph, where each subgraph contains *all* pages from the domain and hyperlinks between local pages within the local domain.

- **BFS subgraph**: This subgraph is constructed by a Breadth First Search (BFS) crawler which starts from a seeded URL. The crawler may follow hyperlinks and fetch Web pages across multiple domains.

For ApproxRank and PageRank implementation, we set the damping factor ϵ to be 0.85. The convergence of the algorithms is identified when the absolute value of the L_1 norm is less than 0.00001. For SC experiments for a subgraph of size n, we use the similar setting in [34] and expand the subgraph for 25 iterations to select another n external pages. The experiments were run on a Solaris machine with 12

GB RAM.

4.2 Evaluation Method

We compute the PageRank vector for the global graph. This ranking vector for the global graph is then limited to pages in the subgraph, denoted by ranking vector R_1. Let R_2 be the PageRank estimation on the local graph. We evaluate the difference between R_1 and R_2. Without considering the actual scores, these two ranking vectors produce two ranked list σ_1 and σ_2.

We use two ranking metrics in our experiments. The SC approach [34] reported on the L_1 distance. The L_1 distance is the absolute value of the differences between the PageRank estimation and the global PageRank scores, for the subgraph.

$$\| R_1 - R_2 \|_1 = \Sigma_{i=1}^{n} |R_1[i] - R_2[i]|$$

Other research [78, 37] use the Spearman's Footrule distance to measure the success of their PageRank approximations. Thus, we also report on the Spearman's Footrule distance between the ApproxRank vector σ_2 and the global PageRank vector σ_1.

Note that there may be a substantial number of tied pages with the same score. A ranking with ties is referred to as a *partial ranking*. We consider an extension of the Spearman's Footrule distance for ranking with ties [43].

The set of pages in ties is called a *bucket*. Each list σ_1, and σ_2 can be viewed as ranked buckets $B_1, B_2, \cdots, B_t$. The *bucket position* for bucket B_i, $pos(B_i)$, is

defined as follows:

$$pos(B_i) = (\sum_{j<i} |B_j|) + \frac{|B_i| + 1}{2}$$

Intuitively, $pos(B_i)$ is the average location within the bucket. The position for a page x, $\sigma(x)$ in list σ is assigned the bucket position for B where x belongs to B.

Spearman's Footrule distance for two partial rankings σ_1 and σ_2 is defined as follows:

$$F(\sigma_1, \sigma_2) = \frac{\Sigma_{i=1}^{n} |\sigma_1(i) - \sigma_2(i)|}{\lfloor |\sigma_1|^2/2 \rfloor}$$

.

We use the following symbols in our figures and tables:

- ApproxRank is labeled (▲).
- The first baseline algorithm, local PageRank, is labeled (■).
- The second baseline algorithm, LPR2, is labeled (●). See Section 3.2.3
- SC is labeled (♦).

4.3 Performance on the TS Subgraphs

We conduct experiments on the same dataset used by the SC approach and compare the distance from the global PageRank for the two approaches. The dataset we consider is labeled **politics**. Starting from the set of pages under the "politics" hierarchy in the dmoz open directory project [7], the dataset is a crawl of pages up to four links away from the set of seeded pages. This dataset contains 4.4 million pages and 17.3 million links. Within the **politics** dataset, we consider the following three

TS subgraphs, **liberalism, conservatism, socialism**. These subgraphs pages are identified by their corresponding dmoz categories, as well as by crawling to all pages within three links.

subgraph	SC (KDD) L_1 distance	SC (Implemented) L_1 distance	ApproxRank L_1 distance	SC (Implemented) Spearman's Footrule	ApproxRank Spearman's Footrule
conservatism	0.0496	0.0476	0.0450	0.0632	0.0255
liberalism	0.0622	0.0733	0.0494	0.0917	0.0293
socialism	0.04318	0.0442	0.104	0.0316	0.0193

Table 4.2: The distance comparison for TS subgraphs on the Politics dataset.

We report on the L_1 distance and the Spearman's Footrule distance for SC and ApproxRank in Table 4.2. We note that we have two values for the L_1 distance for SC. The values in column *SC (KDD)* were reported in [34] and *SC (Implemented)* was our implementation of SC. Since the SC approach expands subgraphs based on the influence scores of external pages, which may have ties, it is possible that a subgraph is expanded to different supergraphs. This explains our SC implementation may produce different L_1 distance compared to results in [34].

For the L_1 distance, ApproxRank has slightly superior behavior to SC for the subgraphs **liberalism** and **conservatism**. SC outperforms ApproxRank for **socialism**. For all the subgraphs reported in Table 4.2, ApproxRank significantly outperforms SC for the Spearman's Footrule distance value.

To summarize, ApproxRank shows similar (sometimes superior) behavior to SC for the L_1 distance and outperforms SC for the Spearman's Footrule distance.

We note that in many applications, e.g., Top-K query answering, the accuracy of the ordering (measured by Spearman's Footrule distance) is more important than the accuracy of the scores (measured by L_1 distance).

4.4 Performance on the DS Subgraphs

Next, we present results of experiments on dataset **AU**. We report on the Spearman's Footrule distance on each of the **DS** subgraphs from the **AU** dataset for ApproxRank, SC, and the two baseline algorithms, in Table 4.3. The performance of ApproxRank (▲), in the last column, is typically an order of magnitude better compared to local PageRank (■) and significantly outperforms the SC (♦) and LPR2 (●) – the distance values are at least 5 times smaller.

In **AU** dataset, the global graph consists of 38 domains and there are 3884199 pages and 23898513 links. Table 4.3 lists 12 domains in ascending order of number of pages in **AU** dataset. The second column, *(%) of global graph*, reports on the size of the domain as a percentage of the global graph; the size ranges from 0.35% to 10.42%. We note that this is an independent variable, i.e., the domains are pre-defined.

First, we observe that as the size increases (as a percentage of the global graph), the distance decreases, for all algorithms. For example, the first row of Table 4.3 is domain *acu.edu.au* which is 0.35% of the global graph. The distance for local PageRank is as poor as 0.19171 whereas the distance for ApproxRank is 0.012112. The last row is domain *anu.edu.au* which is 10.42% of the global graph.

Domain	(%) of global graph	Average outdegree	local PageRank (■)	SC (♦)	LPR2 (●)	ApproxRank (▲)
acu.edu.au	0.35	4.71	0.19171	0.15654	0.10938	0.012112
bond.edu.au	0.50	5.31	0.11049	0.09679	0.09102	0.013611
canberra.edu.au	0.66	5.92	0.10839	0.09197	0.07839	0.012554
cdu.edu.au	0.75	8.74	0.11999	0.09418	0.07898	0.012589
ballarat.edu.au	0.82	5.80	0.07317	0.06471	0.05762	0.006625
cqu.edu.au	0.95	3.80	0.11344	0.09033	0.06722	0.011167
csu.edu.au	2.58	4.26	0.07583	0.05745	0.04826	0.008273
adelaide.edu.au	2.91	5.27	0.08901	0.08321	0.06970	0.009757
curtin.edu.au	2.91	5.55	0.05306	0.03118	0.02771	0.005799
jcu.edu.au	5.04	4.44	0.04823	0.02957	0.02719	0.004614
monash.edu.au	8.45	6.54	0.04101	0.02048	0.02022	0.003934
anu.edu.au	10.42	5.03	0.04516	0.02446	0.02760	0.004945

Table 4.3: The Spearman's Footrule distance for DS subgraphs on the AU dataset.

The distance for local PageRank has now improved to 0.04516 while the distance for ApproxRank is 0.004945.

The second and more interesting observation is that based on the Spearman's Footrule distance, SC shows poor accuracy of ranking compared to ApproxRank The performance of SC lies between LPR2 and local PageRank in these domains. For example, the distance for SC ranges from 0.02048 to 0.15654; it is similar to the distance for LPR2 which ranges from 0.02022 to 0.10938. In contrast, the corresponding distances for ApproxRank is significantly better (distance is less) and ranges from 0.003934 to 0.013611.

To summarize, ApproxRank significantly outperforms SC and both baseline algorithms for the DS subgraphs.

Below we study quality of ApproxRank with respect to top K objects. Since users often are only concerned about the query answers that are ranked high, the properties of the top K objects are often studied. We report the precision of top K objects for all four algorithms in Table 4.4, where we consider the different K value and different subgraphs for ApproxRank.

The experiments are conducted on three DS subgraphs: *acu.edu.au*, *adelaide.edu.au*, *anu.edu.au*. The size of the subgraphs as a percentage of the global graph varies from 0.35% to 10.42%. For each subgraph, we consider different top K values, $(10, 50, 100, 200, 500, 1000)$.

Among these four algorithms, local PageRank and PR2 have similar performance. SC is slightly superior and ApproxRank's performance is more robust. For instance, for domain *acu.edu.au*, the precision of ApproxRank top 10 objects is 0.8,

Domain	(%) of global graph	top K	local PageRank	SC	LPR2	ApproxRank
acu.edu.au	0.35	10	0.0	0.0	0.0	0.8
		50	0.3	0.32	0.3	0.7
		100	0.44	0.47	0.44	0.74
		200	0.515	0.545	0.515	0.745
		500	0.632	0.656	0.632	0.864
		1000	0.715	0.742	0.715	0.94
adelaide.edu.au	2.91	10	0.1	0.2	0.1	0.8
		50	0.2	0.52	0.2	0.86
		100	0.3	0.6	0.29	0.84
		200	0.395	0.545	0.345	0.795
		500	0.494	0.57	0.458	0.796
		1000	0.631	0.696	0.615	0.861
anu.edu.au	10.42	10	0.1	0.1	0.1	0.2
		50	0.38	0.7	0.38	0.88
		100	0.35	0.7	0.35	0.84
		200	0.495	0.75	0.49	0.88
		500	0.666	0.776	0.652	0.87
		1000	0.745	0.812	0.713	0.882

Table 4.4: The precision of top K lists for DS subgraphs on the AU dataset.

indicating that among the top 10 objects of ApproxRank, there are 8 top-10 objects from the real PageRank. The precision of the other three algorithms is 0, indicating the top 10 objects from these algorithm do not include any top-10 objects from the real PageRank.

For each subgraph in Table 4.4, when the top K value is increased, the precision of each algorithm is generally increased. However, the amount of the increase for different algorithms is different – ApproxRank has the smallest increase. This is because ApproxRank is the most robust algorithm among the four.

4.5 Performance on the BFS Subgraph

We next experiment on graphs created by a Breadth First Search crawler, **BFS** subgraphs. We use a BFS crawler, where the crawl starts from seeded page *http://www.sounddesign.unimelb.edu.au/web/biogs/gallery/P000517g.htm*. We consider a sequence of **BFS** subgraphs, as the subset of pages that are reached by the crawler ranges from $0.1\%, 0.5\%, 2\%, 5\%, 8\%, 10\%, 12\%, 15\%$, to 20%. We note that the pages in a BFS subgraph can be in different domains.

Since a majority of links in the Web graph are intra-domain links [59], and these intra-domain links may connect local pages and external pages in **BFS** subgraphs, the interaction between local pages and the external pages can have a more significant impact on the ranking of the subgraph. If this is true, we can expect a negative impact on the performance of the algorithms for **BFS** subgraphs.

Figure 4.1 reports on the distances for the **BFS** datasets. We first observe

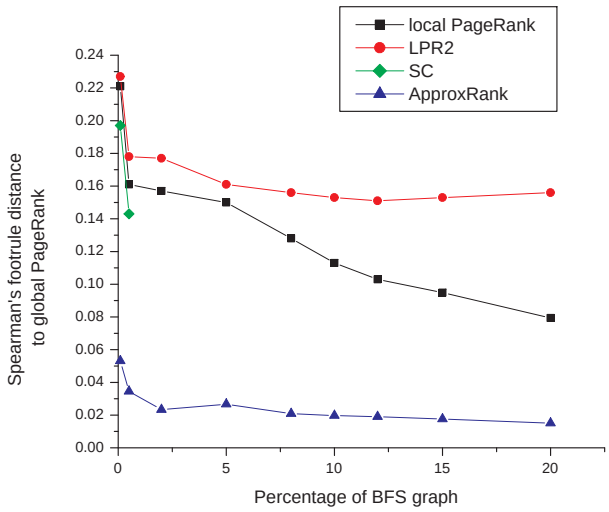

Figure 4.1: Spearman's Footrule distance for BFS subgraphs on AU dataset.

that the distances are much larger compared to those in Table 4.3 that reports on **DS** graphs, for the same **AU** dataset. For example, for the **BFS** subgraph of size 10%, the distances of ApproxRank and local PageRank are 0.0197 and 0.153, respectively. The corresponding values for the **DS** subgraph for `anu.edu.au`, of size 21.86%, (the last row of Table 4.3), is 0.004945 and 0.04516, respectively. In general, the distances on the **BFS** subgraphs appear to be an order of magnitude greater, compared to a **DS** subgraph of similar size.

Our second observation is that ApproxRank generally shows an order of magnitude improvement in comparison to the two baseline algorithms. Since the interaction between local pages and external pages may be intra-domain links, there are much more number of external pages for **BFS** subgraphs. SC becomes very expensive to estimate the influence scores for all external pages so we did not obtain the SC ranking for the larger subgraphs. For the smallest two **BFS** subgraphs in

Figure 4.1, ApproxRank outperforms SC significantly.

We also note that the worst accuracy was shown by LPR2 for all **BFS** subgraphs. This again can be explained by the heavy connectivity between the subgraph and external pages. Unlike ApproxRank that modifies the transition probabilities, LPR2 simply connects a local page to the artificial page even when there are multiple links in the global graph. Hence on **BFS** subgraphs, LPR2 further underestimates this connectivity.

4.6 Runtime Performance

We compare the runtime efficiency of ApproxRank in comparison to the SC approach. We also report on the runtime of the global PageRank algorithm and local PageRank to provide a context.

The disadvantage for SC runtime performance is that it computes the supergraph for each subgraph. In the process of creating the supergraph, it expands the local graph of size n by estimating the influence of each candidate outgoing page on the local graph. To decide the influence of each page, it estimates the PageRank for a graph of size $(n+1)$. This implies that the creation of the supergraph involves the PageRank estimation for many graphs of size $(n+1)$. ApproxRank, on the other hand, processes the global graph for one time and determines the transition matrix A_{approx} for its random walk. When the rankings on multiple subgraphs need to be computed, we can preprocess the global graph for one time, and decide A_{approx} for each subgraph with only local cost.

subgraph	conservatism	liberalism	socialism
#nodes in local graph	42797	61724	12991
local PR (seconds)	63	69	7
ApproxRank (seconds)	542	571	484
SC (seconds)	3002	3483	652
k	1711	2468	519
#ext nodes in 1st expansion	25870	51283	4170
#ext nodes in 2nd expansion	55156	93653	11540
#ext nodes in 3rd expansion	71336	110481	15936

Table 4.5: The runtime comparison on TS subgraphs.

Table 4.5 and 4.6 provide runtime details for ApproxRank, SC, and local PageRank, for the **TS** subgraphs and the **DS** subgraphs. The second column, *#nodes in local graph*, reports on the number of pages in the subgraph; the third column to the fifth column report on the runtime of local PageRank, ApproxRank, and SC, respectively. The sixth column, the value k, shows the number of external pages selected by SC and added to the local graph through each expansion. The last 3 columns report the number of external pages in the first three expansions of SC, which reveal the cost of SC to some extent.

For the global graph **politics** with 4382829 pages, the global PageRank computation takes 5480 seconds. ApproxRank shows an order of magnitude or better runtime performance, and its execution ranges from 484 to 571 seconds. The runtime of the SC approach largely depends on the number of external pages reached

by the local graph through expansions. For example, for TS subgraph **socialism**, the initial graph is 12991 pages and SC considers 15936 pages in the third expansion. The runtime for SC is 652 seconds and is slightly worse than ApproxRank. However, for the larger TS subgraphs, **conservatism** of 42797 pages, and **liberalism** of 61724 pages, the SC solution is at least five times as expensive compared to ApproxRank.

Table 4.6 reports the runtime on DS subgraphs for the **AU** dataset. The cost of global PageRank on this global graph of 3884199 pages is 7035 seconds with 131 iterations. The runtime for ApproxRank ranges from 110 to 468 seconds. SC shows

subgraph	#nodes in local graph	local PR (sec)	Approx-Rank (sec)	SC (sec)	k	#ext nodes in 1st expansion	#ext nodes in 2nd expansion	#ext nodes in 3rd expansion
acu.edu.au	13785	8	319	894	551	1172	6519	13769
bond.edu.au	19559	11	110	1310	782	1826	7918	16502
canberra.edu.au	25501	15	114	1700	1020	3590	10521	20705
cdu.edu.au	29039	25	152	2059	1161	4068	14176	24767
ballarat.edu.au	31724	22	134	2037	1268	1501	15215	27242
cqu.edu.au	36948	16	128	2047	1477	4029	15709	28955
csu.edu.au	100191	59	165	5306	4007	7609	36445	58557
adelaide.edu.au	113181	91	267	6276	4527	13714	45358	73579
curtin.edu.au	113221	80	197	6552	4528	6924	41595	67271
jcu.edu.au	195691	135	272	10327	7827	15705	60966	108644
monash.edu.au	328062	346	468	20292	13122	15489	90993	150890

Table 4.6: The runtime comparison on DS subgraphs.

very poor runtime performance. For the first few rows of the table the runtime ranges from 894 to 2047 seconds. However, for the last rows, where the graph is much larger, the performance of SC sharply degrades. In some cases, e.g., the last two rows, the SC performance is even worse than the exact computation of global PageRank. The high overhead of SC is a trade-off with the lack of access to the global graph.

The runtime for SC on BFS subgraphs is much higher than the runtime on TS and DS subgraphs. The running time of SC was 14655 seconds for the BFS subgraph of size 19420, while for the other types of subgraphs of similar size, the runtime of SC was 652 seconds for TS subgraph **socialism** of size 12991 and 1310 seconds for DS subgraph *bond.edu.au* of size 19559. ApproxRank, on the other hand, seems not as sensitive to the subgraph types. For example, the runtime for ApproxRank on the BFS subgraph of size 19420 is 142 seconds, and ApproxRank takes 484 seconds on **socialism** and 110 seconds on *bond.edu.au*.

Chapter 5

Approximating Authority Flow Rankings in Entity-Relation Graphs

The power of the Web was the ability to support efficient search and a global PageRank score to all Web content. Personalized ranking improves on a global ranking and further filters content and adjusts the order of pages presented to users. A personalized ranking is an ordered list for the current (active) user which reflects the user's preference or profile. Search engines and e-commerce sites improve search quality by accommodating topics of interests, prior search history, or other descriptions of users' preferences. Google [1] provides personalized ranking for users based on user history and bookmarks. Amazon [8] recommends products to users using Collaborative Filtering, based on their previous purchases.

A personalized base set of pages [51, 57, 76] and a weight assignment vector (WAV) Θ of authority flow weights [15, 75] are two important factors achieving personalization for authority flow-based search and ranking such as PageRank. Scalable personalization based on a personalized base set, i.e., a personalization vector, has been studied [57, 51, 23, 46, 55]. However, little work has addressed the problem of scalable personalization based on the second approach of a personalized WAV Θ; we study this latter problem in this chapter.

Figure 5.1 [15] shows the *authority transfer schema graph* for the DBLP database, a bibliographic database for computer science publications [5]. Each edge type is

associated with a numeric value representing the personalized authority weight in Θ.

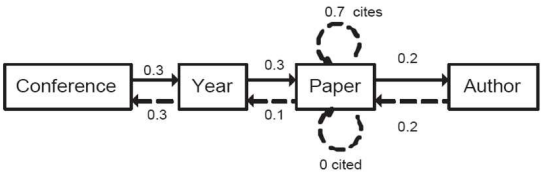

Figure 5.1: The DBLP authority transfer schema graph in ObjectRank ([15]).

Since users submit their queries and personalized WAV Θ on-the-fly, a key challenge is to compute personalized rankings online and to provide the answers to the user quickly. Two extreme solutions include 1) computing each personalized ranking at query time, and 2) computing all possible personalized rankings a priori and storing them. Both solutions are infeasible. A more pragmatic hybrid solution is to maintain a repository of precomputed rankings. At query time, an *approximate personalized ranking* may be computed using some chosen set of candidate rankings from the repository.

This thesis makes the following contributions on approximate personalized authority flow ranking:

- Consider a query q, its WAV Θ^q, its transition matrix for ObjectRank computation A_q, and the ideal ranking R_q. We consider the following two approximation algorithms: (a) SchemaApprox is defined at the schema level and employs a least squares formulation to choose the m-best candidates so that the combined Euclidean distance of these m candidates Θ^{comb}, to Θ^q, is minimized. (b) DataApprox is defined at the data level and solves an optimization

problem so that the maximum norm (δ), over all elements of the aggregate transition matrix of DataApprox and A_q, is minimized. DataApprox computes a weighted combination of m candidate rankings.

- We introduce the concept of an aggregate surfer and prove the authority flow linearity theorem for authority flow rankings. DataApprox's behavior depends on the properties of the aggregate surfer. We show that, given two WAVs Θ_{c1} and Θ_{c2}, there exists a random walk that is defined by combining the two independent random walks; the resulting ranking vector is a linear combination of the two independent ranking vectors for Θ_{c1} and Θ_{c2}.

- We perform a theoretical analysis of the approximation quality of DataApprox. We show that the L_1 distance between DataApprox approximate rankling scores and the ideal personalized ranking scores of R_q is bounded by the maximum norm objective δ of the DataApprox algorithm.

- We abuse the name of DataApprox to its approximation, which has been applied a set of heuristics to dramatically reduce the search space and the complexity of DataApprox. The approximated DataApprox chooses m candidates from the repository based on their Euclidean distance to Θ^q of the query. It efficiently solves the DataApprox optimization problem by employing a Linear Programming sub-procedure. These heuristics make the computation feasible even for large data graphs.

- We conduct extensive experiments to evaluate the execution time and the

quality for DataApprox, i.e., how close the approximate ranking is to the ideal ranking R_q. The experiments are conducted on the complete DBLP data graph. We compare DataApprox with a baseline algorithm PickOne, which chooses the best candidate in the repository with the minimum Euclidean distance to Θ^q. We evaluate the accuracy of the algorithms using the Spearman's Footrule distance. Our experiments show that DataApprox performs well both in terms of execution time as well as in terms of quality.

The chapter is organized as follows. Section 5.1 revisits ObjectRank and related work to scale personalized rankings. Section 5.2 presents the Least Squares problem, which is related to our optimization problems. In Section 5.3, we compare various distances between a target Ranking and an existing Ranking. In Section 5.4, we present the three optimization problems. Two concrete problems are called SchemaApprox and DataApprox. In Section 5.5, we present the mathematical model aggregate surfer behind DataApprox. Section 5.6 and 5.7 present DataApprox architecture and DataApprox respectively.

5.1 Authority Flow Ranking: ObjectRank

5.1.1 ObjectRank Revisited

ObjectRank [15] personalizes ranking in Entity-Relationship graphs; it models nodes as entity types and groups edges by their edge type or semantic type. Authority flow is personalized for the semantic edge type.

The transition matrix A_{OR} of ObjectRank depends on the authority trans-

fer specified on the *schema graph* (Figure 5.1); it defines the authority transferred along each edge type. To demonstrate the relationship of the ObjectRank transition matrix and the PageRank transition matrix, without loss of generality we assume that the objects of the same type are grouped together. Suppose an authority transfer graph with t semantic edge types. A *weight assignment vector* (WAV) $\Theta = \{\alpha_{1,1}, \alpha_{1,2}, ..., \alpha_{1,t}, \alpha_{2,1}, \alpha_{2,2}, ..., \alpha_{2,t}, ..., \alpha_{t,1}, \alpha_{t,2}, ..., \alpha_{t,t}\}$ represents the authority transfer weights.

A_{OR} can be expressed as follows: Each entry of the transition matrix for A_{OR} is multiplied by the authority transfer weight for the corresponding semantic edge type. A_{OR} contains $t \times t$ submatrices.

$$A_{OR} = \begin{pmatrix} \alpha_{1,1}A_{1,1} & \alpha_{1,2}A_{1,2} & \cdots & \alpha_{1,t}A_{1,t} \\ \alpha_{2,1}A_{2,1} & \alpha_{2,2}A_{2,2} & \cdots & \alpha_{2,t}A_{2,t} \\ \vdots & \vdots & & \vdots \\ \alpha_{t,1}A_{t,1} & \alpha_{t,2}A_{t,2} & \cdots & \alpha_{t,t}A_{t,t} \end{pmatrix} \tag{5.1}$$

The submatrix $A_{p,q}$ contains authority transfer probabilities from objects of type p to objects of type q. Let $e^T(v_i, v_j)$ be the semantic type of edge (v_i, v_j) and let $\alpha(e^T(v_i, v_j))$ denote the weight assignment for $e^T(v_i, v_j)$. $OutDeg(v_i, e^T(v_i, v_j))$ is the number of outgoing edges from page v_i, of type $e^T(v_i, v_j)$. The submatrix $A_{p,q}$ is defined as follows:

$$A_{p,q}[i,j] = \begin{cases} \frac{1}{OutDeg(v_i, e^T(v_i, v_j))} & \text{if there is an edge from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Let A_{OR}^T denote the transpose of A_{OR}. The ObjectRank vector R_{OR} is recursively defined as follows in Equation (5.3):

$$R_{OR} = \epsilon A_{OR}^T \cdot R_{OR} + (1 - \epsilon)P \tag{5.3}$$

Below we show an example of a schema graph, a weight assignment vector on the schema graph, a data graph, and a transition matrix. Figure 5.2 is an example of a schema graph, which contains four semantic types A, B, C, D. The weight assignment vector in Figure 5.2 can be written as {0.7,0.5,0.3,0.7,0.6,0.4,0.1}. Figure 5.3 shows an example of a data graph that conforms the schema graph in Figure 5.2. For example, object $A1$ has outgoing links to object $C1$ and $C2$.

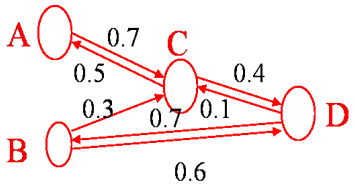

Figure 5.2: An example of a schema graph with a weight assignment vector.

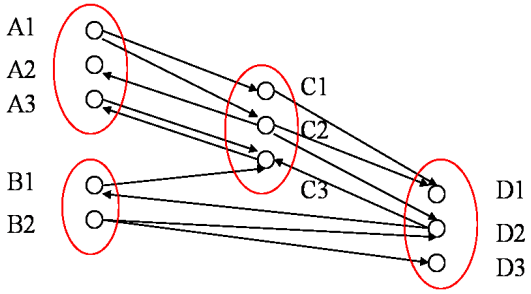

Figure 5.3: An example of a data graph.

According to Equation 5.2, the probability of following the edge (v_{A1}, v_{C1}) should be the edge weight of 0.7 (based on the assignment in Figure 5.2) divided by 2 (the outdegree of v_{A1}, of type $e^T(v_{A1}, v_{C1})$). Therefore, the probability flow on edge (v_{A1}, v_{C1}) is 0.35. Other entries in the transition matrix is defined similarly.

Figure 5.4 shows the transition matrix for Figure 5.3:

	A1	A2	A3	B1	B2	C1	C2	C3	D1	D2	D3
A1	0	0	0	0	0	0.35	0.35	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0
A3	0	0	0	0	0	0	0	0.7	0	0	0
B1	0	0	0	0	0	0	0	0.5	0	0	0
B2	0	0	0	0	0	0	0	0	0	0.3	0.3
C1	0	0	0	0	0	0	0	0	0.4	0	0
C2	0	0.5	0	0	0	0	0	0	0.2	0.2	0
C3	0	0	0.5	0	0	0	0	0	0	0	0
D1	0	0	0	0	0	0	0	0	0	0	0
D2	0	0	0	0.7	0	0	0	0.1	0	0	0
D3	0	0	0	0	0	0	0	0	0	0	0

Figure 5.4: An example of a transition matrix for ObjectRank

5.1.2 ObjectRank convergence

Interestingly, although [53] reported that ObjectRank converges by experiments, no one has proved its convergence. Next we provide a simple proof in Theorem 3, which states that when $\sum_{q=1}^{q=t} \alpha_{p,q} = 1$, the ObjectRank converges.

Lemma 5.1.1 *Each submatrix $A_{p,q}$ is row stochastic.*

Proof. To simplify the proof, we assume that there are no dangling pages (see Section 2.2.1 for the definition of dangling pages). This assumption can be removed by adding the damping factor into the ranking algorithm. It is obvious that each row of $A_{p,q}$, the probability flow from one page of semantic type p to all pages of semantic type q, add to 1. ∎

Next we show a property in Theorem 3. Theorem 3 shows that A_{OR} is stochastic for certain weight assignment vectors, which is a requirement for the convergence of the ObjectRank algorithm.

Theorem 3 *If* $\sum_{q=1}^{q=t} \alpha_{p,q} = 1$ *for all* $0 \leq q \leq t$*, then* A_{OR} *is row stochastic.*

Proof. We know that each submatrix $A_{p,q}$ in Equation 5.1 is row stochastic from Lemma 5.1.1. Given $\sum_{p=1}^{p=t} \alpha_{p,q} = 1$, it is straightforward that A_{OR} is row stochastic. ∎

5.1.3 Approximation methods for personalization

The problem of achieving efficient personalization based on a base set, i.e., a personalization vector, has been studied in [51, 57, 46, 23, 55]. The approach in [51] is based on a linearity theorem that is used to combine multiple personalized PageRank vectors. [57] proposed a technique that encodes personalized ranking vectors as partial vectors. They also presented efficient dynamic programming algorithms. [46] simulates random walks to precompute an *index database* of personalized PageRank vectors (fingerprints). [23, 55] consider using a personalized base set on entity-relationship graphs. They do not consider personalized weight assignments a la ObjectRank.

The only work that approximates authority flow rankings for personalization with weight assignments, as far as we know, is to apply sampling techniques [90] for lgOR [79]. Since lgOR is a special case of ObjectRank, the sampling techniques, the sampling technique applied to lgOR does not directly apply to the ObjectRank

which is computed on general Web graphs, we refer to Section 2.8.2 for more details about the sampling techniques.

5.2 The Least Squares Problem

Although scaling authority flow rankings is an open problem in ranking community, an important application of linear algebra, Least Squares Problem, is related to our problem. In this section, we briefly review the Least Squares Problem and its solution. We refer to [67] for a complete explanation.

In real world applications, data from experiments are often prone to errors and are inconsistent. In consequence, these data lead to systems of equations which are unsolvable. Least squares is a method to deal with this kind of obstacles and to give an approximate solution.

Below we give a brief overview for the Least squares problem. A linear system is *overdetermined* if there are more equations than unknowns, which often requires an approximate solution. Below is an overdetermined system.

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i, (i = 1, 2, \cdots, m)$$

There are m linear equations, n unknowns $(\beta_1, \beta_2, \cdots, \beta_n)$, and $m > n$. Let X be the $m \times n$ matrix representing all coefficients X_{ij} and β be the vector of all β_j. The system can be written as $X\beta = y$.

Since an overdetermined system usually has no solution, the goal is to find the β vector that is most consistent to the constraints. *Least squares* is one of the commonly used approximation criteria. The intuition is to minimize the Euclidean

distance between $X\beta$ and y. Below is the least squares problem:

$$argmin_\beta \sum_{i=1}^{m} |y_i - \sum_{j=1}^{n} X_{ij}\beta_j|^2 = argmin_\beta ||y - X\beta||_2$$

The least squares problem can be solved using the Singular Value Decomposition (SVD) of a matrix [70]. SVD is an important factorization of matrices. The SVD of an $m \times n$ matrix X can be computed in $O(mn^2)$ by a 2-stage algorithm [67].

The Least squares can be applied to approximate personalized rankings. Some precomputed personalized rankings along with different user preferences provide inconsistent information. When a new user comes with her new preference, the approximate approach should pick a subset of precomputed personalized rankings and combine them. We model the approximating the personalized ranking problem as a least squares problem, SchemaApprox, in Section 5.4.

5.3 Comparing Distances between a Target Ranking and any Candidate Ranking

To motivate our approximation approach, we first consider the problem of identifying the best candidate from a ranking repository. To do so we consider some metric to identify the candidate ranking as well as a distance to determine its quality. We compare the behavior of any candidate ranking R_{cand} and the target ranking R_q for some query q.

Consider a query q, WAV Θ^q, the ObjectRank transition matrix A_q, and ranking R_q. Further consider any candidate Θ^{cand}, A_{cand} and R_{cand} from the repository that is used to approximate R_q.

For SchemaApprox, which is defined at the schema level, we consider a schema level metric to choose a candidate; it should minimize the distance of this metric between Θ^{cand} and Θ^{q}. A natural candidate is the Euclidean distance $\pi = ||\Theta^{cand} - \Theta^{q}||_2$.

For DataApprox, which is defined at the level of the data graph, we must consider an objective function that will minimize the distance between A_{cand} and A_q. Let matrix $A_{diff} = A_{cand} - A_q$. The difference between A_{cand} and A_q can be naturally represented by the matrix norms of A_{diff}. The definition for entry-wise matrix norm is equivalent to the definition for vector norm [48]. Using the p-norm for vectors, we have the following:

$$\|A_{diff}\|_p = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{diff}[i,j]|^p \right)^{1/p}$$

When $p = 2$, this is the Frobenius norm, and when $p = \infty$ this is the maximum norm. The alternatives for the distance to be considered for DataApprox are as follows:

- The maximum norm for A_{diff}, $\delta = max_{\{i,j\}}\{|A_{diff}[i,j]|\}$.
- The 1-norm for A_{diff}, $\sigma = \sum_{i,j} |A_{diff}[i,j]|$.
- The Frobenius norm for A_{diff}, $\phi = \sqrt{\sum_{i,j} A_{diff}[i,j]^2}$.

We consider the Spearman's Footrule Distance between the candidate ranking R_{cand} and the ideal ranking R_q as a measure of the quality of our approximate solution.

To understand the correlation behavior between the potential objective function distance metrics of SchemaApprox or DataApprox, and the Spearman's Footrule distance, we generate the ScaleRank ranking repository of 1000 random authority flow rankings for the DBLP dataset. We consider a test set of 20 target queries Θ^q. The details of the values for Θ^q and Θ^{cand} used in this evaluation DBLP dataset are described in detail in the experiment section 6.1.

We compute the average Euclidean distance π used for SchemaApprox, as well as the 3 norms δ, σ and ϕ used for DataApprox, over the 20 queries. We also compute the average Spearman's Footrule Distance over the 20 queries. We average the distances π, δ, σ and ϕ.

We first provide a scatterplot of the distances π, δ, σ and ϕ, plotted against the average Spearman's Footrule distance. For example, in Figure 5.5 through 5.8, the X axis is the average distance π, δ, σ or ϕ, and the Y axis is the Spearman's Footrule distance. Each point in the plot is a candidate ranking in the repository. We report distance values averaged over 20 queries. The lines in the figures are the linear trendlines.

We fit the data in Figure 5.5 through 5.8 using a linear regression model $Y_i = A + BX_i$, where the parameters, A and B, are estimated. The estimated correlation coefficients for Figure 5.5 through 5.8 are 0.4191, 0.3459, 0.4036, and 0.45968 respectively. A value of 1 for the correlation coefficient means a perfect fit, and a value of 0 means no relationship between the two variables.

We conclude that π and δ should be chosen as the distance metrics for SchemaApprox and DataApprox, respectively. As a final step, problems in Section 5.4, we

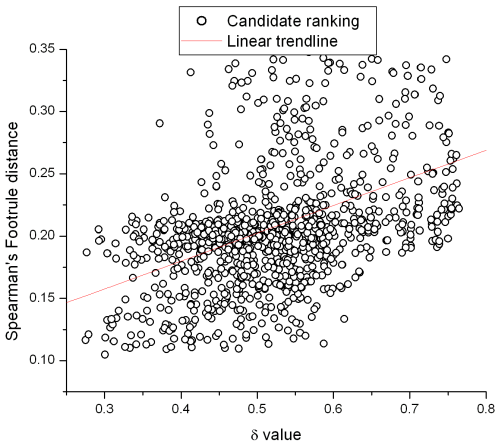

Figure 5.5: The correlation between δ and Spearman's Footrule distance.

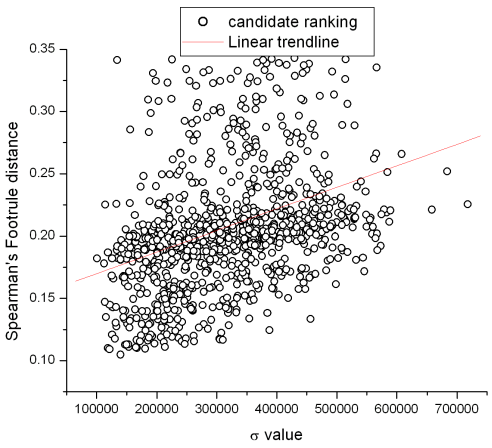

Figure 5.6: The correlation between σ and Spearman's Footrule distance.

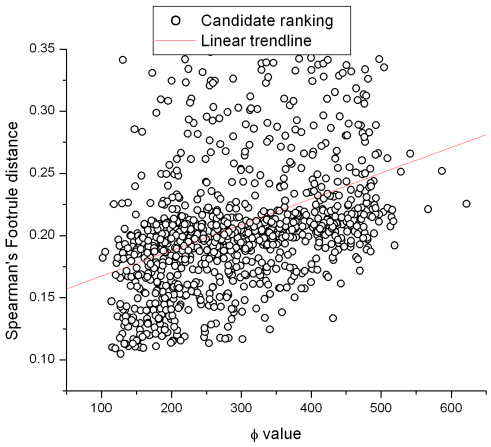

Figure 5.7: The correlation between ϕ and Spearman's Footrule distance.

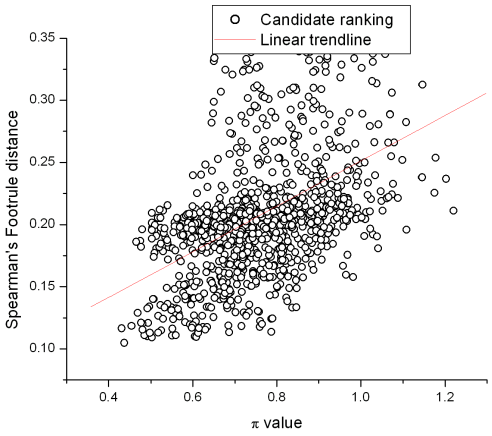

Figure 5.8: The correlation between π and Spearman's Footrule distance.

report on the scatterplot of $\delta - \pi$ correlation in Figure 5.9. The estimated correlation coefficient is 0.894 and this appears to be a strong correlation. The correlation indicates that although the DataApprox and SchemaApprox will solve optimization problem at different levels, the metrics that they use to choose candidate rankings

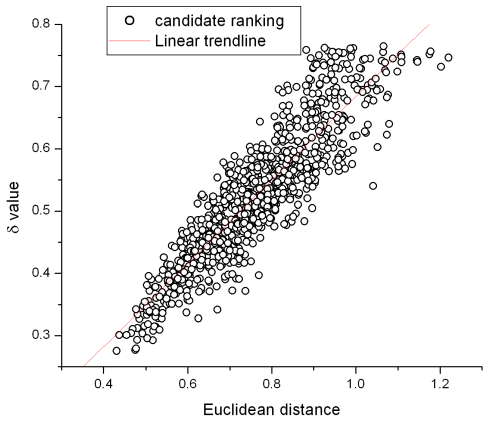

Figure 5.9: The correlation between π and δ.

are correlated.

5.4 The Problem Definition

Our problem can be described informally as follows: Given a set of candidate rankings in a repository, choose the m best candidates using some metric, so that it provides an approximate rankings of the highest quality. We describe the problem as follows:

Problem definition: Let $\mathcal{S} = \{(\Theta_1, R_1), (\Theta_2, R_2), \cdots, (\Theta_m, R_m)\}$ be the ranking repository of m precomputed ranking vectors and their corresponding weight assignment vectors. Let Θ^q be the query weight assignment vector. The goal is to approximate the authority flow ranking for query Θ^q efficiently, with the maximal quality, by utilizing the precomputed ranking vectors in the repository.

The objective for the concrete problem needs to satisfy the following requirements: 1) We should choose an appropriate distance metric to choose the candidate

rankings. 2) The distance should be easy to compute. During the search process of the optimal objective value, it may be necessary to compute the objective function repeatedly. 3) The distance metric should be correlated to the approximation quality. We expect that a the stronger correlation will improve the choice of candidate rankings and the approximation quality.

SchemaApprox: to approximate at the schema graph level

Recall that we chose the Euclidean distance π for SchemaApprox. Let Θ^{comb} be a linear combination of m weight assignment vectors (WAVs) selected from the repository $\mathcal{S}$. The goal is to minimize the Euclidean distance between the linear combination Θ^{comb} and the query Θ^q. Let $\pi = ||\Theta^{comb} - \Theta^q||_2$. SchemaApprox is defined as follows:

$$\begin{aligned} &\text{minimize} \quad \pi = ||\Theta^{comb} - \Theta^q||_2 \\ &\text{subject to:} \\ &\qquad \Theta^{comb} = \textstyle\sum_{l=1}^{m} \beta_l \Theta_l \\ &\qquad \textstyle\sum_{l=1}^{m} \beta_l = 1 \\ &\qquad 0 \leq \beta_l \qquad\qquad for\ all\ 1 \leq l \leq m \end{aligned} \tag{5.4}$$

SchemaApprox can be solved using an approach to solve the Least Squares Problem [67]. A linear system of equations is *overdetermined* if there are more equations than unknowns, which often requires an approximate solution. Below is

an overdetermined system.

$$\sum_{j=1}^{n} X_{ij}\beta_j = y_i, (i = 1, 2, \cdots, m)$$

There are m linear equations, n unknowns $(\beta_1, \beta_2, \cdots, \beta_n)$, and $m > n$. Let X be the $m \times n$ matrix representing all coefficients X_{ij} and β be the vector of all β_j. The system can be written as $X\beta = y$.

Since an overdetermined system usually has no solution, the goal is to find the β vector that is the closest to satisfying all the equations. *Least squares* is one of the commonly used approximation criterion. The intuition is to minimize the distance between $X\beta$ and y. The least squares problem formulation is as follows:

$$argmin_\beta \sum_{i=1}^{m} |y_i - \sum_{j=1}^{n} X_{ij}\beta_j|^2 = argmin_\beta ||y - X\beta||_2$$

The least squares problem can be solved using the Singular Value Decomposition (SVD) of a matrix [70]. SVD is an important factorization of matrix. The SVD of an $m \times n$ matrix X can be computed in $O(mn^2)$ using a 2-stage algorithm [67].

We note that solving SchemaApprox is computationally expensive. Further, there is no known properties that can be proved to describe the behavior of SchemaApprox, i.e., there is no linearity theorem that describes the behavior of a random walk based on Θ^{comb}, with respect to the random walk based on Θ^q. As a result there is no proof that SchemaApprox will converge or a bound on the quality of SchemaApprox. Finally, our observation with experiments is that Θ^{comb} is insensitive to the properties of the data graph, in particular, the edge distribution for the different edge types. For all of these reasons, we do not attempt to solve SchemaApprox.

DataApprox: to approximate at the data graph level

$$
\begin{aligned}
&\text{minimize} && \delta \\
&\text{subject to:} && \\
&|A_{agg}(\mathcal{S})[i,j] - A_q[i,j]| && \leq \ \delta, \ for\ all\ entry\ (i,j) \\
&\textstyle\sum_{l=1}^{m} \beta_l = 1 && \\
&0 \leq \beta_l && for\ all\ 1 \leq l \leq m
\end{aligned}
\tag{5.5}
$$

Equation 5.7 defines the DataApprox optimization problem. Recall that we chose the maximal norm δ over the elements of $A_{diff} = (A_{cand}$ - $A_q)$ for DataApprox.

5.5 The Aggregate Surfer

A key result used in [46, 51, 57] to scale personalization using a base set of pages is the linearity theorem [51] that combines multiple personalized PageRank vector. In this section, we present some important theoretical results for personalization based on authority transfer weight assignments. We formalize a model of *aggregate surfer* whose behavior is controlled by multiple authority flow rankings and prove the linearity theorem for authority flow ranking.

5.5.1 The Authority Transfer Weights Linearity Theorem

Without loss of generality, we show how we can combine the ranking vectors two candidate weight assignment vectors. Multiple ranking vectors can be combined

in the same way. In this section we show that, given two weight assignment vectors, there exists a random walk that is defined by combining the random walks of the two weight assignment vectors, and whose ranking vector is a linear combination of the two ranking vectors. The intuition can be described later as an aggregate surfer.

Theorem 4 (Authority Transfer Weights Linearity Theorem) *Let R_1 and R_2 be two ranking vectors for weight assignment vectors Θ_1 and Θ_2 respectively. Let A_1 and A_2 be the corresponding transition matrices. Let β_1, β_2 be constants such that $\beta_1, \beta_2 \geq 0$ and $\beta_1 + \beta_2 = 1$. For a random walk with transition matrix A, where $A[i,j] = \frac{\beta_1 A_1[i,j] R_1[i] + \beta_2 A_2[i,j] R_2[i]}{\beta_1 R_1[i] + \beta_2 R_2[i]}$, the ranking vector $R = \beta_1 R_1 + \beta_2 R_2$.*

Proof. We first show that A is row stochastic given that A_1 and A_2 are row stochastic. We know that $\sum_{j=1}^{n} A_1[i,j] = 1$ and $\sum_{j=1}^{n} A_2[i,j] = 1$. For any row i, we have:

$$\begin{aligned} \sum_{j=1}^{n} A[i,j] &= \frac{\beta_1 R_1[i] \sum_{j=1}^{n} A_1[i,j] + \beta_2 R_2[i] \sum_{j=1}^{n} A_2[i,j]}{\beta_1 R_1[j] + \beta_2 R_2[j]} \\ &= \frac{\beta_1 R_1[i] + \beta_2 R_2[i]}{\beta_1 R_1[i] + \beta_2 R_2[i]} \\ &= 1 \end{aligned}$$

If we complement the random walk with jumps from dangling pages then the Markov Chain we defined is irreducible and aperiodic. The ranking scores converge to a unique vector R. Next we show that R converges to $\beta_1 R_1 + \beta_2 R_2$.

$$\begin{aligned} & \beta_1 R_1 + \beta_2 R_2 \\ = \; & \beta_1(\epsilon A_1^T R_1 + (1-\epsilon)P) + \beta_2(\epsilon A_2^T R_2 + (1-\epsilon)P) \\ = \; & \epsilon(\beta_1 A_1^T R_1 + \beta_2 A_2^T R_2) + (\beta_1 + \beta_2)(1-\epsilon)P \\ = \; & \epsilon(\beta_1 A_1^T R_1 + \beta_2 A_2^T R_2) + (1-\epsilon)P \end{aligned}$$

Let $v_1 = \beta_1 A_1^T R_1 + \beta_2 A_2^T R_2$ and $v_2 = A^T(\beta_1 R_1 + \beta_2 R_2)$ Next we show that $v_1 = v_2$. For the j-th entry $v_1[j]$ in vector v_1, we have:

$$\begin{aligned}
& v_1[j] \\
= \; & \beta_1(\textstyle\sum_{i=1}^{n} A_1[i,j]R_1[i]) + \beta_2(\sum_{i=1}^{n} A_2[i,j]R_2[i]) \\
= \; & \textstyle\sum_{i=1}^{n} \beta_1 A_1[i,j]R_1[i] + \beta_2 A_2[i,j]R_2[i] \\
= \; & \textstyle\sum_{i=1}^{n} \frac{\beta_1 A_1[i,j]R_1[i] + \beta_2 A_2[i,j]R_2[i]}{\beta_1 R_1[i] + \beta_2 R_2[i]}(\beta_1 R_1[i] + \beta_2 R_2[i]) \\
= \; & \textstyle\sum_{i=1}^{n} A[i,j](\beta_1 R_1[i] + \beta_2 R_2[i]) \\
= \; & v_2[j]
\end{aligned}$$

Since $\beta_1 A_1^T R_1 + \beta_2 A_2^T R_2 = A^T(\beta_1 R_1 + \beta_2 R_2)$, we have:

$$\beta_1 R_1 + \beta_2 R_2 = \epsilon A^T(\beta_1 R_1 + \beta_2 R_2) + (1-\epsilon)P$$

This concludes the proof that $R = \beta_1 R_1 + \beta_2 R_2$. ∎

5.5.2 The intuition behind the Authority Transfer Weights Linearity Theorem

An intuitive view of Theorem 4 (Authority Transfer Weights Linearity Theorem) is that if we know the behavior of two individual random surfers, including the way they walk (A_1, A_2) and the expected probability reaching all the pages (R_1, R_2), then for another random surfer, who behaves like two individual random surfers with probability β_1 and β_2 respectively, the expected probability reaching all the pages (R) is a linear combination of R_1 and R_2. $R = \beta_1 R_1 + \beta_2 R_2$. We call this random surfer an *aggregate surfer*. This is shown in Figure 5.10. The two surfers

with vertical line or horizontal lines in the figure represent two random surfers, and the aggregate surfer mimics both surfers.

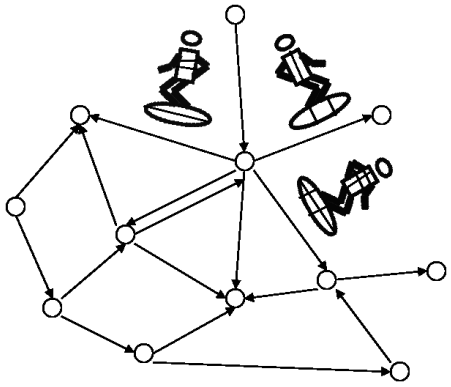

Figure 5.10: The aggregate surfer and two individual surfers.

The Linearity Theorem indicates that a ranking vector can be represented by a linear combination of ranking vectors, which can be naturally used as a mathematical tool to scale personalized ranking based on weight assignments.

To make the results more general, Theorem 4 is extended to multiple surfers as follows. We are given a set of m weight assignment vectors and their corresponding ranking vectors $\mathcal{S} = \{(\Theta_1, R_1), (\Theta_2, R_2), \cdots ,$

$(\Theta_m, R_m)\}$. Let A_l be the transition matrix for Θ_l. We define the behavior of an aggregate surfer with transition matrix $A_{agg}(\mathcal{S})$ as follows:

$$A_{agg}(\mathcal{S})[i,j] = \frac{\sum_{l=1}^{m} \beta_l A_l[i,j] R_l[i]}{\sum_{l=1}^{m} \beta_l R_l[i]} \tag{5.6}$$

5.5.3 The application of the Authority Transfer Weights Linearity Theorem

The Authority Transfer Weights Linearity Theorem is an important tool. Here are two use cases.

- Given a query and some existing authority flow rankings, the theorem can be used to approximate the ranking for the query, by combining multiple existing rankings. Let $A_{agg}(\mathcal{S})$ in Equation 5.6 be the transition matrix for the aggregate surfer, and let A_q be the transition matrix for query Θ^q. A combination means an appropriate β vector such that the $A_{agg}(\mathcal{S})$ is close to A_q.

- Given some user preferred ranking (or the top K objects), the theorem can be used to learn a weight assignment vector. Let R_{agg} be the linear combination of multiple ranking vectors. Let R_q be the preferred ranking from user input. This problem searches for a combination, such that R_{agg} is close to R_q.

With the trend that personalization is considered in another dimension for links, the Authority Transfer Weights Linearity Theorem complements the previous linearity theorem (Theorem 1).

5.6 The DataApprox System Architecture

Figure 5.11 shows the architecture of the system, which inputs a query (a weight assignment vector Θ^q) and outputs the top K objects based on their authority score. The system maintains a repository of M candidate rankings. For each

candidate ranking we store its weight assignment vector, and its ranking vector. Given a query, the Candidate Ranking Selector selects m candidate rankings out of the M in the repository based on a heuristic described below. The reason that only m are selected is that the cost of DataApprox depends on the number of input rankings. DataApprox algorithm then computes the best way to linearly combine these m rankings. Finally a top K algorithm is used to produce the top K objects. Next we briefly describe the elements in the system.

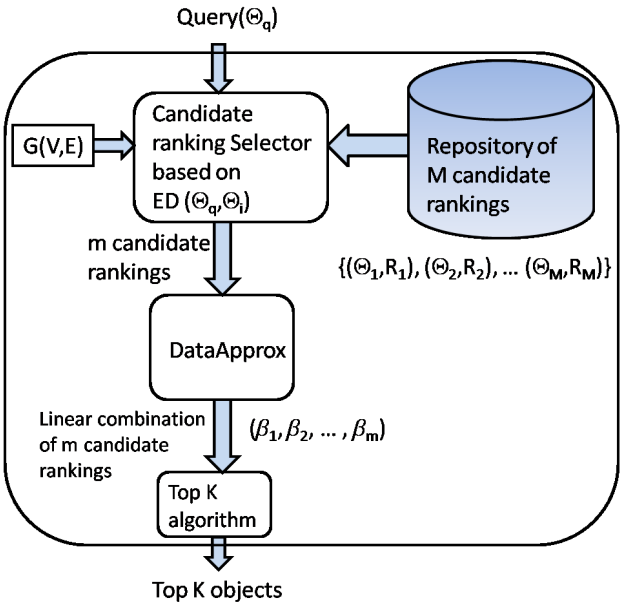

Figure 5.11: The system architecture

Materializing candidate rankings in the repository:

The set of rankings in the repository affects the quality of our approximation. and they are computed using ObjectRank. Ideally we would precompute rankings for a set of weight assignment vectors that users are interested in; this is obviously impossible since we do not know such a set. We use a simple randomized technique

to populate the repository as described in Section 6.2.

For each candidate ranking i in the repository, its weight assignment vector Θ_i and its ranking vector R_i are materialized. The repository can be represented by a set $\{(\Theta_1, R_1), (\Theta_2, R_2), \cdots, (\Theta_M, R_M)\}$.

The candidate ranking selector:

Among the M candidate rankings in the repository, we choose the m closest weight assignments and DataApprox tries to combine them in an optimal way. Each weight assignment vector can be viewed as a point in multi-dimensional space, where the weight for each link type is the coordinate on one dimension. The intuition is that the best candidates should be close to the query Θ^q, therefore we calculate the Euclidean distance $||\Theta^q, \Theta_i||_2$ between Θ^q and each candidate Θ_i and select the closest m candidates.

The DataApprox algorithm:

Given the m closest candidates, the DataApprox algorithm combines their rankings to compute an approximation of the ranking vector of the query on-the-fly. The DataApprox algorithm determines to what extent the aggregate surfer behaves like each candidate i, β_i, and produces a vector of β values, $(\beta_1, \beta_2, \cdots, \beta_m)$. We will elaborate on the DataApprox algorithm in the next section.

Top K algorithm:

The problem of combining multiple ranking vectors (sorted lists) and output the top K objects is well studied and there are numerous efficient algorithms [25, 42, 44]. For example, the famous TA algorithm [44] deals with monotone functions to aggregate the ranking scores. The linear combination as a weighted sum suggested by the

DataApprox algorithm is a monotone function. Hence, we use TA to produce the top K objects.

5.7 The DataApprox algorithm

5.7.1 The algorithm

Given a set of weight assignment vector and their ranking vector $S = \{(\Theta_1, R_1), (\Theta_2, R_2), \cdots, (\Theta_m, R_m)\}$, and the query weight assignment vector Θ^q, the DataApprox algorithm determines a way to combine these m candidate rankings, $R_{DA} = \sum_{l=1}^{m} \beta_l R_l$. For that, DataApprox computes the aggregate surfer of S that closely resembles the random surfer of the query.

We define an optimization problem to find the best aggregate surfer. Let A_q be the transition matrix for the query Θ^q and $A_{agg}(\mathcal{S})$ be the transition matrix for the aggregate surfer. Let δ be the maximum difference between the entries in A_q and $A_{agg}(\mathcal{S})$. Formally, $\delta = max_{\{i,j\}}\{|A_{agg}(\mathcal{S})[i,j] - A_q[i,j]|\}$. The DataApprox algorithm finds a linear combination that minimizes the δ value.

Let fractional variable $0 \leq \beta_l \leq 1$ denote the probability that the aggregate surfer behaves like the individual random surfer (Θ_l, R_l). For the aggregate surfer, $\sum_{l=1}^{m} \beta_l = 1$. We can then simplify $0 \leq \beta_l \leq 1$ to $0 \leq \beta_l$. We define the following optimization problem as follows:

$$
\begin{aligned}
&\text{minimize} && \delta \\
&\text{subject to:} && \\
&|A_{agg}(\mathcal{S})[i,j] - A_q[i,j]| && \leq \ \delta, \ for\ all\ entry\ (i,j) \\
&\textstyle\sum_{l=1}^{m} \beta_l = 1 && \\
&0 \leq \beta_l && for\ all\ 1 \leq l \leq m
\end{aligned}
\tag{5.7}
$$

where $A_{agg}(\mathcal{S})[i,j] = \frac{\sum_{l=1}^{m} A_l[i,j]R_l[i]\beta_l}{\sum_{l=1}^{m} R_l[i]\beta_l}$. The first constraint sets an upper bound for the difference between two matrices.

Note that the above optimization problem (Equation 5.7) can be solved by solving a series of feasibility problems without the objective function, that is, choose a δ and check if the constraints hold. Since δ is the absolute value between two transition matrix entries difference, it is within the range $[0, 1]$. Therefore, we can use binary search to find the minimum δ with upper bound $u = 1$ and lower bound $l = 0$. The search stops until $|u - l| < \tau$, where τ is the user defined accuracy requirement. Given the candidate rankings $\mathcal{S}$, the data graph G, the query weight assignment Θ_q, and accuracy requirement τ for δ, we describe the *DataApprox* algorithm as follows:

The algorithm *DataApprox* finds the minimum δ such that the optimization problem in Equation 5.7 is feasible, and stores the β vector which produces *min_*δ in *Feasibility* algorithm. The *while* loop is usually executed for around 10 times if we choose accuracy requirement $\tau = 0.1$. The Feasibility procedure in Line 5 of algorithm solves the Linear Programming problem of Equation 5.7 without the

Algorithm *DataApprox*($\mathcal{S}$, G, Θ_q, τ)

1. $u = 1$, $l = 0$
2. $min\_\delta = u$
3. **while** $(u - l \geq \tau)$ **do**
4. $\quad \delta = (u + l)/2$
5. $\quad$ **if** $(Feasibility(\mathcal{S}, G, \Theta_q, \delta))$
6. $\quad\quad min\_\delta = \delta$
7. $\quad\quad u = \delta$
8. $\quad$ **else**
9. $\quad\quad l = \delta$
10. **return** min_δ

Figure 5.12: The outline of the DataApprox algorithm.

objective function, that is, for a given δ.

5.7.2 Error analysis of DataApprox ranking vector R_{DA}

In this section, we provide the error approximation analysis for the scores in the ranking vector of the user query, computed by applying DataApprox to combine m candidate rankings. We show that the L_1 distance between DataApprox scores and the accurate personalized ranking scores depends on the value of δ in Equation 5.7. A similar analysis was conducted to a different algorithm to estimate the rank for a

subgraph [92]. Note that although the DataApprox algorithm is solved by finding a linear combination of candidate rankings, the ranking for the aggregate surfer can be alternatively computed by traditional iterative approach until convergence [76, 15, 92]. Therefore, we can bound the distance through iterations. This can be explained by a prominent feature of the DataApprox algorithm: the aggregate surfer does not deviate much from the random walk defined by the query, which is specified by Equation 5.7.

Let R_{DA} and R_{OR} be the ranking vectors from DataApprox (approximate) and ObjectRank (exact) respectively, each with length n, where n is number of nodes in the graph. We will derive the L_1 distance between R_{DA} and R_{OR}, $\| R_{DA} - R_{OR} \|_1$. Let R^m_{DA} and R^m_{OR} be the ranking vectors after the m-th iteration from DataApprox and ObjectRank. We first derive the error bound for the base case when $m = 1$.

Lemma 5.7.1 *After the first iteration, the DataApprox ranking vector R^1_{DA} satisfies:*

$$\| R^1_{DA} - R^1_{OR} \|_1 \leq \epsilon |E| \delta$$

Proof.

$$
\begin{aligned}
& \| R^1_{DA} - R^1_{OR} \|_1 \\
(i) \quad &= \sum_{i=1}^{n} |R^1_{DA}[i] - R^1_{OR}[i]| \\
(ii) \quad &= \sum_{i=1}^{n} |\epsilon \sum_{j=1}^{n} A_{DA}[j,i] \cdot 1 + (1-\epsilon)d(i) \\
& \quad -\epsilon \sum_{j=1}^{n} A_{OR}[j,i] \cdot 1 - (1-\epsilon)d(i)| \\
(iii) \quad &= \epsilon \sum_{i=1}^{n} \sum_{j=1}^{n} |A_{DA}[j,i] - A_{OR}[j,i]| \\
(iv) \quad &\leq \epsilon \sum_{i=1}^{n} \sum_{j=1}^{n} \delta \\
(v) \quad &\leq \epsilon |E| \delta
\end{aligned}
$$

We first expand the L_1 distance by the definition. In inequality (ii), we calculate $R^1_{DA}[i]$ and $R^1_{OR}[i]$ assuming initial scores for DataApprox and ObjectRank are all 1s. $d(i)$ represents the probability to jump to page v_i. In ObjectRank, $d(i) = 1/|S|$ if page v_i contains the keyword, where $|S|$ is the number of pages that contain the keyword; otherwise, $d(i) = 0$. We replace $|A_{DA}[j,i] - A_{OR}[j,i]|$ by δ in inequality (iv), which is specified by Equation 5.7. Since the number of non-zero entries in the graph is the number of edges $|E|$, we concludes the proof in inequality (v). ∎

Next we develop the error bound for the L_1 distance

$\| R^m_{DA} - R^m_{OR} \|_1$ after m iterations.

Lemma 5.7.2 *After an arbitrary positive integer $m > 1$ iterations, the DataApprox ranking vector R^m_{DA} satisfies:*

$$\| R^m_{DA} - R^m_{OR} \|_1 \leq \epsilon \| R^{m-1}_{DA} - R^{m-1}_{OR} \|_1 + \epsilon\delta n$$

Proof.

$$
\begin{aligned}
& \| R^m_{DA} - R^1_{OR} \|_1 \\
(i) \quad &= \textstyle\sum_{i=1}^n |R^m_{DA}[i] - R^m_{OR}[i]| \\
(ii) \quad &= \textstyle\sum_{i=1}^n |\epsilon \sum_{j=1}^n A_{DA}[j,i] \cdot R^{m-1}_{DA}[j] + (1-\epsilon)d(i) \\
& \quad -\epsilon \textstyle\sum_{j=1}^n A_{OR}[j,i] \cdot R^{m-1}_{OR}[j] - (1-\epsilon)d(i)| \\
(iii) \quad &= \epsilon \textstyle\sum_{i=1}^n \sum_{j=1}^n |A_{DA}[j,i] \cdot R^{m-1}_{DA}[j] \\
& \quad -A_{OR}[j,i] \cdot R^{m-1}_{OR}[j]| \\
(iv) \quad &\leq \epsilon \textstyle\sum_{i=1}^n \sum_{j=1}^n |A_{DA}[j,i] R^{m-1}_{DA}[j] - \\
& \quad A_{DA}[j,i] R^{m-1}_{OR}[j]| + \epsilon \textstyle\sum_{i=1}^n \sum_{j=1}^n \delta R^{m-1}_{OR}[j] \\
(v) \quad &\leq \epsilon \textstyle\sum_{i=1}^n \sum_{j=1}^n A_{DA}[j,i] |R^{m-1}_{DA}[j] - R^{m-1}_{OR}[j]| \\
& \quad +\epsilon\delta \textstyle\sum_{i=1}^n \sum_{j=1}^n R^{m-1}_{OR}[j] \\
(vi) \quad &\leq \epsilon \textstyle\sum_{j=1}^n |R^{m-1}_{DA}[j] - R^{m-1}_{OR}[j]| \sum_{i=1}^n A_{DA}[j,i] \\
& \quad +\epsilon\delta \textstyle\sum_{i=1}^n \sum_{j=1}^n R^{m-1}_{OR}[j] \\
(vii) \quad &\leq \epsilon \textstyle\sum_{j=1}^n |R^{m-1}_{DA}[j] - R^{m-1}_{OR}[j]| + \epsilon\delta \sum_{i=1}^n 1 \\
(viii) \quad &\leq \epsilon \| R^{m-1}_{DA} - R^{m-1}_{OR} \|_1 + \epsilon\delta n
\end{aligned}
$$

The idea of the first three inequalities in the proof is identical to the proof for Lemma 5.7.1, except that the scores from previous iteration are R^{m-1}_{DA} and R^{m-1}_{OR} instead of all 1s in inequality (ii). The inequality (iv) is derived based on the constraint that $|A_{DA}[j,i] - A_{OR}[j,i]| \leq \delta$. Reorganizing the terms leads to inequality (v) and (vi). Because the transition matrix A_{DA} is column stochastic, $\sum_{i=1}^n A_{DA}[j,i] = 1$. Considering $\sum_{j=1}^n R^{m-1}_{OR}[j] = 1$, the inequality is reduced to (vii). The definition of L_1 distance concludes proof. ∎

Next we combine Lemma 5.7.1 and Lemma 5.7.2 to develop Theorem 5.

Theorem 5 (DataApprox)

$$\| R_{DA} - R_{OR} \|_1 \leq \delta \frac{\epsilon}{1-\epsilon} n$$

Proof.

$$\begin{aligned}
& \| R_{DA}^m - R_{OR}^m \|_1 \\
\leq\ & \epsilon \| R_{DA}^{m-1} - R_{OR}^{m-1} \|_1 + \epsilon\delta n \\
\leq\ & \epsilon(\epsilon \| R_{DA}^{m-2} - R_{OR}^{m-2} \|_1 + \epsilon\delta n) + \epsilon\delta n \\
\leq\ & \epsilon^2 \| R_{DA}^{m-2} - R_{OR}^{m-2} \|_1 + (\epsilon^2 + \epsilon)\delta n \\
\leq\ & \epsilon^m |E|\delta + (\epsilon^m + \epsilon^{m-1} + \cdots + \epsilon)\delta n
\end{aligned}$$

When the number of iterations m becomes infinity, this gives

$$\| R_{DA} - R_{OR} \|_1 \leq \delta \frac{\epsilon}{1-\epsilon} n$$

■

The theorem shows that when δ is very small, the DataApprox will give accurate ranking. Another interesting observation is that the error bound increases with ϵ, whereas in the case of PageRank perturbation, it has been shown [27] that the error decreases with ϵ. The reason is that in the case of perturbation, for large ϵ, the scores of high-score nodes are influenced by thousands of paths and hence removing a few edges does not make a difference. On the other hand, in our problem, larger ϵ means that the authority transfer weights are multiplied by a larger constant and hence small differences in authority bounds translate to large differences in weights and hence to larger errors.

We conduct error analysis for DataApprox based on L_1 distance, which is a score-based distance. It is difficult to analyze a order-based distance, if it is

possible. In practice, what matters to the user is the order of ranking, instead of ranking scores. Therefore, we will report Spearman's Footrule distance in Chapter 6, which is a order-based distance [37].

5.7.3 Reduce the complexity of the feasibility problem

We now go back to the execution of the DataApprox algorithm, which involves repeatedly solving a linear programming (LP) problem. It is well known that LP problem can be solved by Simplex algorithm in linear time "in practice". There are $|E|$ non-zero matrix entries, where $|E|$ is the number of edges in the graph. Recall that m is the number of rankings. The complexity of the LP problem is $O(|E|+m)$, according to Equation 5.7. In this section, we consider several methods to reduce the number of constraints of the LP.

5.7.3.1 One constraint per semantic type

The first constraint of Equation 5.7 can be rewritten as follows:

$$\sum_{l=1}^{m}(|A_l[i,j] - A[i,j]| - \delta) \cdot R_l[j] \cdot \beta_l \leq 0 \ for\ all\ entry\ (i,j) \tag{5.8}$$

The constraint of Equation 5.8 addresses the transition entry differences between the new random walk and existing random walks. For an important page v_j, if v_j leads to thousands of pages, does this imply thousands of constraints? We will show we can reduce the number of constraints dramatically. To do this, we reformulate this constraint.

We look into the authority flow ranking definition. Let $e^T(v_j, v_i)$ be the seman-

tic type of edge (v_j, v_i). $\alpha(e^T(v_j, v_i))$ denotes the weight assignment for $e^T(v_j, v_i)$ in the new query, and $\alpha_l(e^T(v_j, v_i))$ denotes the weight assignment for $e^T(v_j, v_i)$ in candidate weight assignment l. $OutDeg(v_j, e^T(v_j, v_i))$ is the number of outgoing edges from page v_j, of type $e^T(v_j, v_i)$. According to the authority flow ranking definition, $A[i, j] = \frac{\alpha(e^T(v_j,v_i))}{OutDeg(v_j,e^T(v_j,v_i))}$. Equation 5.8 becomes as follows:

$$\sum_{l=1}^{m}(|\alpha_l(e^T(v_j, v_i)) - \alpha(e^T(v_j, v_i))|\frac{1}{OutDeg(v_j,e^T(v_j,v_i))} - \delta) \cdot R_l[j] \cdot \beta_l \leq 0 \qquad (5.9)$$

From Equation 5.9, it is clear that for outgoing edges from page v_j, if they belong to the same semantic type, the same constraint holds. Therefore, for outgoing edges, the number of constraints can be reduced to the number of semantic types departing from page v_j.

5.7.3.2 Make use of skewed scores

It is known that the PageRank-style ranking scores conform to a power law distribution, therefore, the top ranked nodes/pages have very high scores. Table 5.1 lists the sum of normalized ranking scores of top K pages when K is varied from 20 to 2000, for a data graph with 1707898 nodes.

top K	20	100	500	1000	1500	2000
RankSum	0.1302	0.6349	0.9732	0.9858	0.9888	0.9905

Table 5.1: The sum of ranking scores of top K pages.

Equation 5.9 describes the constraints for type of edge $e^T(v_j, v_i)$. If page v_j is assigned high ranking score in existing ranking R_{l_h}, then the similarity of the weight

assignment Θ_{l_h} to the query weight assignment is more accredited. Since frequently top ranked pages accumulated most ranking scores, we will focus on setting up the LP for these pages.

5.7.3.3 The range for δ

Given the range for δ, we can use binary search to find the smallest δ such that the above LP problem is feasible. Instead of the trivial range $[0, 1]$ for δ, we show that there is a better range. Let $s_h = max_{e^T(j,i)}\{|\alpha_h(e^T(v_j, v_i)) - \alpha(e^T(v_j, v_i)|\}$ be the largest weight assignment difference, among all edge types, for the existing ranking of Θ_h. Let $u = min_{h=1\cdots m}\{s_h\}$ be the minimum s_h among all m existing rankings. Based on formula (5.9), we can see that u is a feasible upper bound. To search a smallest value for δ such that the LP is feasible, we use the range $[0, u]$.

5.7.4 The time complexity of DataApprox

The DataApprox algorithm (Figure 5.12) employs a Linear Programming subroutine to solve an optimization problem. The number of iterations for the while loop in line 3–9 depends on the choice of the accuracy requirement τ. In experiments, we choose $\tau = 0.001$ which leads to good approximations. For all cases, the DataApprox is executed for up to 9 iterations.

In each iteration of the while loop, DataApprox calls the linear programming subroutine. LP problem can be solved by Simplex algorithm in linear time "in practice". That is, the number of iterations is linear in the number of constraints,

which is the union cardinality of top K objects from m candidates and objects.

The ObjectRank algorithm is computed through iterations until convergence. In practice, it takes around 25 iterations on our data graph. In each iteration, the ObjectRank examines all the edges in the graph.

Our analysis shows that the DataApprox is an efficient algorithm, since we typically choose top K less than 500 which leads to good approximation. For ObjectRank, however, there can be millions of links. We will report runtime for both algorithms in Section 6.5.

Chapter 6

The Evaluation for DataApprox

6.1 Experiment Description

The ER dataset. Bibliographic databases (DBLP or CiteSeer) are frequently used to evaluate authority flow ranking [15, 23, 89]. We use the **DBLP** dataset (June 2008) to build a data graph that conforms to the schema graph of Figure 5.1. Part of citation links were crawled from CiteSeerX [9] and added to the data graph; the dataset contains 1707898 objects and 7704633 links.

Below we discuss the impact of the characteristics of the schema graph and data graph. At the schema graph level, the Theorem 3 restricts that in the ObjectRank weight assignment vector, the sum of the outgoing edge weights from the same entity should be 1. The experiments show that when the sum is less than 1, the ObjectRank algorithm converges to a unique ranking vector as well [53]. Therefore, when an entity is connected to few other entities, the weight for its outgoing link is typically larger than the rest weights in the weight assignment vector. The consequence is that at the schema level, ObjectRank favors the semantic link types whose outdegrees are small.

At the data graph level, similar impact exists. If for an object, the outdegree for one link type is significantly smaller than the others, while the weights in the weight assignment vector are comparable, then ObjectRank favors the these links

where the outdegrees are small at data level. In DataApprox, this skewed outdegree distribution has an effect as well. Since $A_{agg}(\mathcal{S})[i,j] = \frac{\sum_{l=1}^{m} A_l[i,j]R_l[i]\beta_l}{\sum_{l=1}^{m} R_l[i]\beta_l}$ (Equation 5.7) is defined based on the linearity theorem, the first constraint in Equation 5.7 indicates the β vector is largely determined by the links with smaller outdegrees.

Therefore, outdegree distribution has an impact on the ranking, at both schema graph and data graph level.

The Evaluation method. For a given query and its authority flow weight assignment Θ^q, we compute the exact ObjectRank ranking vector and the DataApprox vector. We compute the Spearman's Footrule distance between these two vectors. Since there are a many tied pages with the same score, we use an extension for ranking with ties [43]. The normalized Spearman's Footrule distance is reported in Figures 6.1 and 6.2. We consider 20 typical user-preferred queries; they reflect a range of preferences for the link types. Since different weight assignment vectors have an impact on the approximation quality, all results are an average over the 20 queries.

Baseline algorithm PickOne. We compare DataApprox against a baseline algorithm PickOne. The PickOne algorithm follows the intuition that the best candidate Θ_i in the repository is the closest to the query Θ^q (Section 5.6). PickOne calculates the Euclidean distance $||\Theta^q, \Theta_i||_2$ and chooses the candidate with the minimum Euclidean distance. Note that PickOne does satisfy the bound in Theorem 5.

While PickOne chooses the single best candidate, it does not have minimum value of δ. However, the DataApprox algorithm will minimize the value of δ. Theoretically, the DataApprox algorithm should produce a more accurate approximation

compared to the PickOne algorithm.

We implemented our algorithms (DataApprox and PickOne) and the ObjectRank algorithm in Java. Our experiments were run on a Solaris machine with two 2.8 GHz dual-core processors and 12 GB RAM.

6.2 The candidate ranking repository

The candidate rankings available in the repository will have an impact on the quality of the DataApprox approximation. In an extreme case, the query ranking is stored in the repository, and an accurate ranking can be retrieved free at runtime.

A natural way of materializing candidate rankings is to generate a grid to represent all possible weight assignments, e.g., for each edge type in the semantic graph, we can select W uniformly spread weight assignments. One drawback is that we would have to generate a very large number of candidate rankings in order to provide a uniform coverage of all the points in the grid. A small value of W or not covering the grid uniformly may produce poor candidate rankings. For the experiments, we generated 1000 candidate rankings; each candidate can be considered to correspond to a randomly selected point. We will use these candidates to serve as the ranking repository.

Note that the DataApprox algorithm does not need all the scores of the ranking vectors, as addressed in Section 5.7.3.2. DataApprox works well even when $K = 50$ (see Figure 6.1). The storage requirement for the top $K = 1000$ objects of 1000 rankings is estimated to be 30 MB bytes. We note that we used the complete

ranking vector to accurately calculate the Spearman's Footrule distance.

In Table 6.1, we show some query samples and some samples in the repository. The second column lists the weight assignment vectors for some queries and candidates. The third column is the keyword used to select the base set. The weight assignment vector assigns the link importance for 7 edge types. They are ((conference,year),(year,conference),(year,paper),(paper,year),(paper,paper),(paper,author),(author,paper)).

	weight assignment vector	keyword
query1	$(1, 0.5, 0.5, 0.33, 0.33, 0.33, 0.1)$	"OLAP"
query2	$(0.3, 0.3, 0.3, 0.1, 0.7, 0.2, 0.2)$	"OLAP"
candidate1	$(0.99, 0.18, 0.75, 0.35, 0.30, 0.23, 0.26)$	"OLAP"
candidate2	$(0.28, 0.18, 0.75, 0.37, 0.22, 0.36, 0.088)$	"OLAP"

Table 6.1: The samples for queries and candidates.

6.3 The impact of the top K on DataApprox

For this experiment we consider 10 candidate rankings chosen by the Candidate Ranking Selector and 20 queries. We report on the DataApprox quality (Spearman's Footrule distance) as we vary the top K objects. Recall from Section 5.7.3.2 that we set up the constraints in the feasibility problem for the top K objects. (Note that this parameter K is not the prefix of the top-k results that the user may request.) Figure 6.1 reports on the DataApprox distance when K is varied from 50 to 1000.

The left vertical axis shows the scale for the Spearman's Footrule distance and the right vertical axis reports the δ values. The δ values for DataApprox are the triangles and the DataApprox distance values are the blue crosses. The red line with squares is the distance for the baseline algorithm PickOne. The PickOne algorithm's choice is indifferent to the top K objects.

We observe that the quality of DataApprox is much higher than PickOne; the DataApprox improves the distance value typically by 30% – 40%. As K increases, DataApprox sets up constraints in the LP for more objects and it has a more complete view of the candidate rankings. Therefore, when K is larger, δ increases and DataApprox is able to produce better approximation.

To summarize, the quality of DataApprox is very good and it outperforms the ranking chosen by PickOne (the closest single candidate ranking).

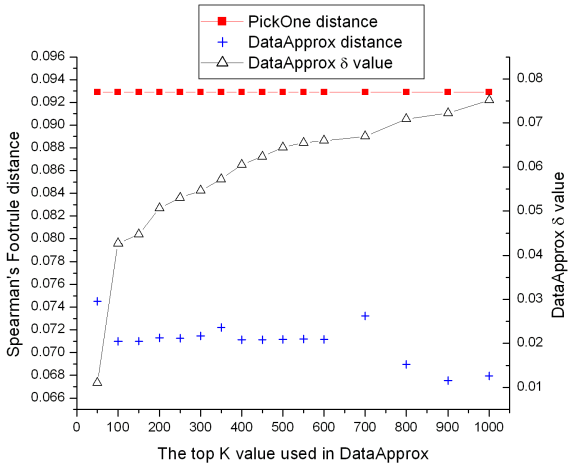

Figure 6.1: The average Spearman's Footrule distance when the value of the top K is varied.

6.4 The impact of the size of the ranking repository

Figure 6.2 reports on the behavior of DataApprox and PickOne when we increase the number of rankings in the repository. We consider 10 different ranking repositories, whose size varies from 100 to 1000. The δ values for DataApprox are the triangles and the DataApprox distance values are the blue crosses and the distance for PickOne are the red squares. When the size of the repository increases, the value of δ for DataApprox decreases, since the algorithm has more candidates to choose from, which leads to the smaller feasible δ. DataApprox outperforms PickOne when we use repository with different sizes.

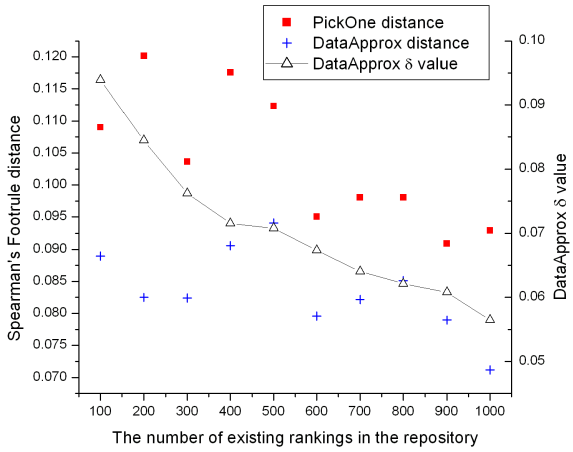

Figure 6.2: The average Spearman's Footrule distance for varying M (M).

We notice that with more candidates, while the δ value decreases monotonically, the quality of DataApprox and PickOne does not uniformly decrease. The correlation between δ and the distance has only limited impact. To summarize, the quality of DataApprox outperforms the baseline algorithm PickOne and both algorithms can be improved by increasing the coverage of the repository.

6.5 DataApprox runtime

We report on the runtime of DataApprox and compare it to the exact ObjectRank algorithm. The figures show that DataApprox can be performed within 2 seconds, e.g., top K = 1000. For the comparison, the optimized ObjectRank takes more than 5 minutes on the same dataset.

DataApprox calls the LP solver during a binary search to find the smallest value of δ. We used an open source LP solver *glpk* [10] and its Java interface [11]. It is reported that the commercial LP solver CPLEX is 10–100 times faster than *glpk* [12]. Thus, we conclude that despite the slower execution of *glpk*, the execution times reported in Figures 6.3 and 6.4 reflect that DataApprox can be performed at query time.

In the ObjectRank implementation, we set the damping factor ϵ to be 0.85 and the convergence of the algorithms is identified when the absolute value of the L_1 norm is less than 0.1. It usually takes $25 - 26$ iterations to converge. The average runtime for ObjectRank on 20 queries is 338 seconds, which is not shown in Figure 6.3 and 6.4 because DataApprox is faster than ObjectRank in several orders of magnitude. Note that this runtime does not include the preprocessing time when the graph is loaded into memory.

Figures 6.3 and 6.4 report the runtime when we vary the top K values and the number of rankings in the repository M respectively. The DataApprox runtime reported in Figures 6.3 and 6.4 also does not include the preprocessing time to load the data graph and the authority transfer weights for all candidates into the

memory. We report the initialization and execution time for the DataApprox with stacked bars. The initialization is to select m rankings from the repository and to load the top K objects for these m rankings, which is the white bar in the figure. In DataApprox execution, the algorithm calls the LP solver multiple times (usually $9-10$ times to satisfy the accuracy requirement of $\tau = 0.1$).

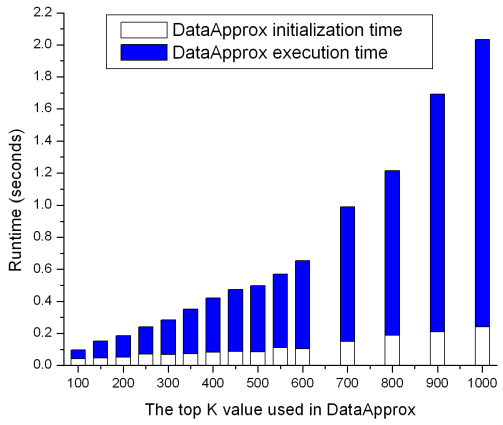

Figure 6.3: Average runtime of DataApprox when the number of top K are varied.

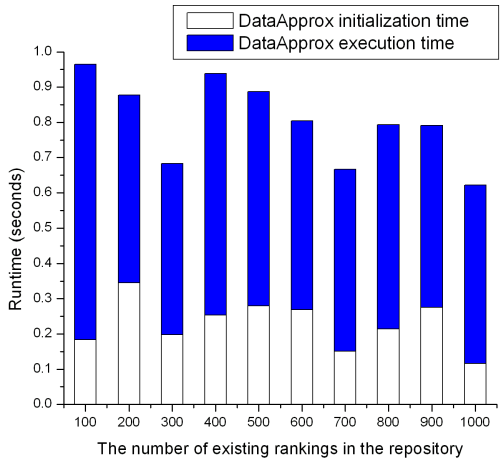

Figure 6.4: Average runtime of DataApprox for varying M (M).

Compared to the ObjectRank execution time of 338 seconds, Figure 6.3 shows

that DataApprox can be executed at query time. As K is increased, the number of constraints in the LP problem is increased. When K increased from 100 to 1000, the runtime for DataApprox ranges from 0.096 second to 2.03 seconds.

Figure 6.4 shows that DataApprox runtime is within 1 second. Note that when the size of the repository is increased, the runtime for DataApprox does not necessarily increase. Adversely, it has a tendency to decrease. This can be explained as follows: with larger ranking repository, DataApprox tends to have a better set of rankings from the Candidate Ranking Selector. However, the runtime of DataApprox does not uniformly decreasing when the repository is larger. This may be explained by the weak correlation between δ and the Spearman's Footrule distance. Considering the results in Figure 6.2 where the quality of the approximation is generally improved, the DataApprox algorithm benefits from a complete repository for accuracy gain without efficiency loss.

To summarize, despite the use of a comparatively slow LP solver *glpk*, DataApprox execution performance is very fast and makes it an option for runtime personalization.

Chapter 7

Conclusion and Future Direction

We have addressed the challenge of computing customized ranking for two problems. The first is subgraph ranking and the second is authority flow based personalized ranking.

For the subgraph ranking problem, we have presented a framework based on an exact and an approximate solution to compute PageRank on a subgraph. The IdealRank algorithm is an exact solution and the ApproxRank algorithm is an approximate solution. The IdealRank algorithm assumes that the PageRank scores of external pages are known, therefore it produces accurate ranking for pages in a subgraph. When the PageRank scores of external pages are not available, ApproxRank algorithm estimates PageRank scores for the subgraph. For IdealRank algorithm, we proved that the IdealRank scores for pages in the subgraph converge to the true PageRank scores. For ApproxRank algorithm, we developed a theoretical bound of the distance between the ApproxRank scores and the PageRank scores. We showed through empirical results that the ApproxRank ranking accuracy is similar (sometimes superior) to the best competitor SC, and it overwhelmingly outperforms the runtime efficiency of SC.

For the problem of computing an approximate personalized ranking, we have defined two approximate approaches, SchemaApprox and DataApprox at different

level. We proved the Authority Flow Linearity Theorem for authority flow rankings for the aggregate surfer; her behavior is controlled by multiple personalized rankings. Based on these results, we presented the DataApprox algorithm in order to approximate authority flow rankings. We modeled DataApprox as an optimization problem of selecting and combining the best rankings from a repository. Then we developed a set of heuristics to dramatically reduce the search space and the complexity of the optimization problem and makes the computation feasible even for very large data graphs. We performed a theoretical analysis of the approximation quality of DataApprox. We conducted extensive experiments on the complete DBLP data graph and showed that DataApprox performs well both in terms of execution time as well as in terms of quality, i.e., how close the approximate DataApprox ranking is to the exact ranking.

Given the dynamic nature of the Web, being able to provide relevant, personalized ranking quickly will continue to present challenges. For both subgraph ranking and personalized ranking, there is considerable room for enhancement and improvement.

We summarize some of these challenges that were identified in completing this research. For subgraph ranking, we will consider extensions to ApproxRank to improve performance. An example is to utilize the error analysis of ApproxRank to improve the accuracy. We expect better estimation for external pages will improve ApproxRank. We will also investigate the possibility to cluster the external pages into multiple external nodes rather than the single external node considered in this thesis. We expect to perform this clustering based on the importance or the

PageRank score associated with the external pages. The challenge is to correctly classify the external pages into multiple nodes (determined by their PageRank score or other importance score) without actually having to compute these PageRank scores. We also will determine the optimal number of such external nodes, e.g., two external nodes, reflecting the power laws controlling the distribution of importance on the Web. We also plan to apply a similar analysis to IdealRank when it is applied to semantic ranking on a subgraph. ApproxRank was a centralized computation. It can be extended in several ways. One possible extension is to adapt it in distributed system like P2P network to speedup ranking computation. A final extension is to extend subgraph ranking to consider updates to the graph. IdealRank can be applied to this case, where the part of the graph that experienced the most changed is considered as the subgraph.

For computing authority flow rankings, we will consider efficient solutions for the SchemaApprox optimization problem. SchemaApprox is solvable as a Quadratic Programming problem, which is very expensive to solve. We may apply some heuristics to reduce the search space in future work. We can also address an inverse personalization problem. In this thesis we assumed that the user provided Θ. This is difficult. It is simpler for the user to provide a preferred ranking. The challenge then is to use this relevance feedback to determine the best Θ.

Bibliography

[1] *http://www.google.com/*.

[2] *http://www.useit.com/alertbox/web-growth.html*.

[3] *http://www.jux2.com/stats.php*.

[4] *http://www.ncbi.nlm.nih.gov/*.

[5] *http://www.informatik.uni-trier.de/~ley/db/*.

[6] *http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html*.

[7] *http://www.dmoz.org/*.

[8] *http://www.amazon.com/*.

[9] *http://citeseerx.ist.psu.edu/*.

[10] *http://www.gnu.org/software/glpk/*.

[11] *http://bjoern.dapnet.de/glpk/index.htm*.

[12] *http://www.go.dlr.de/pdinfo_dv/glpk/GLPK_FAQ.txt*.

[13] Karl Aberer and Jie Wu. A framework for decentralized ranking in web information retrieval. In *APWeb*, pages 213–226, 2003.

[14] Alekh Agarwal, Soumen Chakrabarti, and Sunny Aggarwal. Learning to rank networked entities. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 14–23, New York, NY, USA, 2006. ACM.

[15] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB, 2004*, 2004.

[16] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[17] Pavel Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[18] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Pagerank as a function of the damping factor. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 557–566, New York, NY, USA, 2005. ACM.

[19] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM Trans. Inter. Tech.*, 5(1):231–297, 2005.

[20] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[21] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pages 309–320, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[22] Andrei Z. Broder, Ronny Lempel, Farzin Maghoul, and Jan Pedersen. Efficient pagerank approximation via graph aggregation. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 484–485, New York, NY, USA, 2004. ACM Press.

[23] Soumen Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 571–580, New York, NY, USA, 2007. ACM.

[24] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *WWW '99: Proceeding of the eighth international conference on World Wide Web*, pages 1623–1640, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[25] Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD '02*.

[26] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. Local methods for estimating pagerank values. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 381–389, New York, NY, USA, 2004. ACM Press.

[27] Steve Chien, Cynthia Dwork, Ravi Kumar, Daniel R. Simon, and D. Sivakumar. Link evolution: Analysis and algorithms. *Internet Mathematics*, 1(3):277–304, 2003.

[28] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[29] Junghoo Cho, Hector G. Molina, and Lawrence Page. Efficient crawling through url ordering. *Computer Networks and ISDN Systems*, 30(1–7):161–172, 1998.

[30] Junghoo Cho and Uri Schonfeld. Rankmass crawler: a crawler with high personalized pagerank coverage guarantee. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 375–386. VLDB Endowment, 2007.

[31] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1:5–32, 1999.

[32] Don Coppersmith, Peter Doyle, Prabhakar Raghavan, and Marc Snir. Random walks on weighted graphs and applications to on-line algorithms. *J. ACM*, 40(3):421–453, 1993.

[33] Gianna M. Del Corso, Antonio Gullí, and Francesco Romani. Fast pagerank computation via a sparse linear system. *Journal of Internet Mathematics*, 2(3):251–273, 2005.

[34] Jason V. Davis and Inderjit S. Dhillon. Estimating the global pagerank of web communities. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 116–125, New York, NY, USA, 2006. ACM Press.

[35] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 527–534, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[36] Chris Ding, Xiaofeng He, Parry Husbands, Hongyuan Zha, and Horst D. Simon. Pagerank, hits and a unified framework for link analysis. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 353–354, New York, NY, USA, 2002. ACM.

[37] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2001. ACM Press.

[38] Efthimis N. Efthimiadis. A user-centred evaluation of ranking algorithms for interactive query expansion. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 146–159, New York, NY, USA, 1993. ACM.

[39] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, 2003.

[40] Magdalini Eirinaki and Michalis Vazirgiannis. Usage-based pagerank for web personalization. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 130–137, Washington, DC, USA, 2005. IEEE Computer Society.

[41] Nadav Eiron, Kevin S. McCurley, and John A. Tomlin. Ranking the web frontier. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 309–318, New York, NY, USA, 2004. ACM Press.

[42] Ronald Fagin. Combining fuzzy information from multiple systems (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 216–226, New York, NY, USA, 1996. ACM.

[43] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 47–58, New York, NY, USA, 2004. ACM Press.

[44] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, New York, NY, USA, 2001. ACM.

[45] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 251–262, New York, NY, USA, 1999. ACM.

[46] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3), 2005.

[47] Floris Geerts, Heikki Mannila, and Evimaria Terzi. Relational link-based ranking. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 552–563. VLDB Endowment, 2004.

[48] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996.

[49] A. Gulli and A. Signorini. Building an open source meta-search engine. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1004–1005, New York, NY, USA, 2005. ACM Press.

[50] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *WWW '05: Special interest tracks and posters of the 14th international*

conference on World Wide Web, pages 902–903, New York, NY, USA, 2005. ACM Press.

[51] Taher H. Haveliwala. Topic-sensitive pagerank. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 517–526, New York, NY, USA, 2002. ACM Press.

[52] Donald Hooley. Collapsed matrices with (almost) the same eigenstuff. *The College Mathematics Journal*, 31(4):297–299, 2000.

[53] Vagelis Hristidis, Heasoo Hwang, and Yannis Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 33(1):1–40, 2008.

[54] Heasoo Hwang, Andrey Balmin, Hamid Pirahesh, and Berthold Reinwald. Information discovery in loosely integrated data. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1147–1149, New York, NY, USA, 2007. ACM.

[55] Heasoo Hwang, Andrey Balmin, Berthold Reinwald, and Erik Nijkamp. Binrank: Scaling dynamic authority-based search using materialized subgraphs. In *ICDE*, 2009.

[56] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, New York, NY, USA, 2002. ACM.

[57] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 271–279, New York, NY, USA, 2003. ACM.

[58] Sepandar Kamvar, Taher Haveliwala, and Gene Golub. Adaptive methods for the computation of pagerank. Technical report, Stanford University, 2003.

[59] Sepandar Kamvar, Taher Haveliwala, Chris Manning, and Gene Golub. Exploiting the block structure of the web for computing pagerank. Technical report, Stanford Digital Library Technologies Project, 2003.

[60] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *WWW*, pages 261–270, 2003.

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

[62] Apostolos Kritikopoulos and Martha Sideri. The compass filter: Search engine result personalization using web communities. In Bamshad Mobasher and Sarabjot S. Anand, editors, *ITWP*, volume 3169 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2003.

[63] Amy N. Langville and Carl D. Meyer. Deeper inside pagerank. *Internet Mathematics*, 1(3):335–380, 2004.

[64] Amy N. Langville and Carl D. Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Rev.*, 47(1):135–161, 2005.

[65] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, July 2006.

[66] Amy N. Langville and Carl D. Meyer. Updating markov chains with an eye on google's pagerank. *SIAM J. Matrix Anal. Appl.*, 27(4):968–987, 2006.

[67] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Society for Industrial Mathematics, 1995.

[68] Ronny Lempel and Shlomo Moran. The stochastic approach for link-structure analysis (salsa) and the tkc effect. *Computer Networks*, 33(1-6):387–401, 2000.

[69] C. D. Meyer. Stochastic complementation, uncoupling markov chains, and the theory of nearly reducible systems. *SIAM Rev.*, 31(2):240–272, 1989.

[70] Carl Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, February.

[71] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Automatic personalization based on web usage mining. *Commun. ACM*, 43(8):142–151, 2000.

[72] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[73] Maurice D. Mulvenna, Sarabjot S. Anand, and Alex G. Büchner. Personalization on the net using web mining: introduction. *Commun. ACM*, 43(8):122–125, 2000.

[74] Andrew Y. Ng, Alice X. Zheng, and Michael I. Jordan. Link analysis, eigenvectors and stability. In *IJCAI*, pages 903–910, 2001.

[75] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: bringing order to web objects. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 567–574, New York, NY, USA, 2005. ACM.

[76] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[77] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Using pagerank to characterize web structure. In *COCOON '02: Proceedings of the 8th Annual International Conference on Computing and Combinatorics*, pages 330–339, London, UK, 2002. Springer-Verlag.

[78] Josiane Xavier Parreira, Debora Donato, Sebastian Michel, and Gerhard Weikum. Efficient and decentralized pagerank approximation in a peer-to-peer web search network. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 415–426. VLDB Endowment, 2006.

[79] Louiqa Raschid, Yao Wu, Woei-Jyh Lee, María Esther Vidal, Panayiotis Tsaparas, Padmini Srinivasan, and Aditya Kumar Sehgal. Ranking target objects of navigational queries. In *WIDM '06: Proceedings of the eighth ACM international workshop on Web information and data management*, pages 27–34, New York, NY, USA, 2006. ACM Press.

[80] Matthew Richardson and Pedro Domingos. The intelligent surfer: probabilistic combination of link and content information in pagerank. In *In Advances in Neural Information Processing Systems 14*, pages 1441–1448. MIT Press, 2002.

[81] Ian Ruthven and Mounia Lalmas. A survey on the use of relevance feedback for information access systems. *Knowl. Eng. Rev.*, 18(2):95–145, 2003.

[82] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.

[83] M. Spiliopoulou and L. C. Faulstich. WUM: A tool for Web utilization analysis. *Lecture Notes in Computer Science*, 1590:184–203, 1999.

[84] Lara Srour, Ayman I. Kayssi, and Ali Chehab. Personalized web page ranking using trust and similarity. In *AICCSA*, pages 454–457. IEEE, 2007.

[85] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ, USA, 1994.

[86] Torsten Suel, Chandan Mathur, Jo-Wen Wu, Jiangong Zhang, Alex Delis, Mehdi Kharrazi, Xiaohui Long, and Kulesh Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In *WebDB*, pages 67–72, 2003.

[87] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 675–684, New York, NY, USA, 2004. ACM.

[88] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 449–456, New York, NY, USA, 2005. ACM.

[89] Ramakrishna Varadarajan, Vagelis Hristidis, and Louiqa Raschid. Explaining and reformulating authority flow queries. In *ICDE*, pages 883–892, 2008.

[90] Ramakrishna Varadarajan, Vagelis Hristidis, Louiqa Raschid, Maria-Esther Vidal, Luis Ibanez, and Hector Rodriguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 553–564. ACM, 2009.

[91] Yuan Wang and David J. DeWitt. Computing pagerank in a distributed internet search system. In *VLDB*, pages 420–431, 2004.

[92] Yao Wu and Louiqa Raschid. Approxrank: Estimating rank for a subgraph. In *ICDE*, 2009.

[93] Gui-Rong Xue, Hua-Jun Zeng, Zheng Chen, Wei-Ying Ma, Hong-Jiang Zhang, and Chao-Jun Lu. Implicit link analysis for small web search. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 56–63, New York, NY, USA, 2003. ACM Press.

[94] Demetrios Zeinalipour-Yazti, Vana Kalogeraki, and Dimitrios Gunopulos. Information retrieval techniques for peer-to-peer networks. *Computing in Science and Engg.*, 6(4):20–26, 2004.