3-10-2009

Ranked Search on Data Graphs

Ramakrishna R. Varadarajan
Florida International University, ramkris83@gmail.com

DOI: 10.25148/etd.FI10022554

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

RANKED SEARCH ON DATA GRAPHS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Ramakrishna R. Varadarajan

2009

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Ramakrishna R. Varadarajan, and entitled Ranked Search on Data Graphs, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

_____
Shu-Ching Chen

_____
Tao Li

_____
Raju Rangaswami

_____
Kaushik Dutta

_____
Vagelis Hristidis, Major Professor

Date of Defense: March 10, 2009

The dissertation of Ramakrishna R. Varadarajan is approved.

_____
Dean Amir Mirmiran
College of Engineering and Computing

_____
Dean George Walker
University Graduate School

Florida International University, 2009

DEDICATION

I dedicate this Dissertation to my parents, Krishnan Varadarajan and Suganthi Varadarajan and my brother Venkatanathan Varadarajan. Without their patience, understanding, support, and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

I wish to thank the Department of Computing and Information Sciences for extending me all the support needed for successful completion of my Dissertation. In particular, I would like to thank the Department for consistently providing me with graduate assistantships, travel fellowships and awards that kept me motivated throughout my life as a graduate student. I would like to thank Dr. Yi Deng and Dr. Masoud Milani for their encouragement and support. I wish to thank the members of my Dissertation committee – Dr. Vagelis Hristidis, Dr. Tao Li, Dr. Shu-Ching Chen, Dr. Raju Rangaswami and Dr. Kaushik Dutta for their support and patience in supervising and reviewing my Dissertation. A huge thanks and deep appreciation to Dr. Vagelis Hristidis for providing me with cutting-edge research ideas and spending tremendous amount of time and effort in supervising my Dissertation. I would also like to thank my research Collaborators – Dr. Louiqa Raschid, Dr. Raghuram Krishnapuram, Dr. Gautam Das, Dr. Maria-Esther Vidal and Dr. Prasad DeshPande for providing invaluable feedback, comments, support and encouragement. Great thanks to all the CIS faculty members for providing me with excellent classroom education. A special thanks to Dr. Masoud Sadjadi, Dr. Giri Narasimhan and Dr. Geoffrey Smith for their motivating and stimulating classroom teaching. I would also like to thank my PhD candidacy committee members – Dr. Geoffrey Smith, Dr. Peter Clarke and Dr. Napthali Rishe for their time. A big thanks to my fellow PhD students – Medha, Kasturi, Fernando, Jorge, Luis, Sajib and Ricardo for helping me surpass the numerous barriers I encountered on the road to my PhD. My deepest appreciation to Maria, Martha, Olga, Haydee, Donaley and the rest of the CIS staff for putting up with all my questions and requests and never failing to help me when I needed it.

ABSTRACT OF THE DISSERTATION

RANKED SEARCH ON DATA GRAPHS

by

Ramakrishna R. Varadarajan

Florida International University, 2009

Miami, Florida

Professor Vagelis Hristidis, Major Professor

Graph-structured databases are widely prevalent, and the problem of effective search and retrieval from such graphs has been receiving much attention recently. For example, the Web can be naturally viewed as a graph. Likewise, a relational database can be viewed as a graph where tuples are modeled as vertices connected via foreign-key relationships. Keyword search querying has emerged as one of the most effective paradigms for information discovery, especially over HTML documents in the World Wide Web. One of the key advantages of keyword search querying is its simplicity – users do not have to learn a complex query language, and can issue queries without any prior knowledge about the structure of the underlying data.

The purpose of this dissertation was to develop techniques for user-friendly, high quality and efficient searching of graph structured databases. Several ranked search methods on data graphs have been studied in the recent years. Given a top-*k* keyword search query on a graph and some ranking criteria, a keyword proximity search finds the top-*k* answers where each answer is a substructure of the graph containing all query keywords, which illustrates the relationship between the keyword present in the graph. We applied keyword proximity search on the web and the page graph of web documents

to find top-*k* answers that satisfy user's information need and increase user satisfaction. Another effective ranking mechanism applied on data graphs is the authority flow based ranking mechanism. Given a top-*k* keyword search query on a graph, an authority-flow based search finds the top-*k* answers where each answer is a node in the graph ranked according to its relevance and importance to the query. We developed techniques that improved the authority flow based search on data graphs by creating a framework to explain and reformulate them taking in to consideration user preferences and feedback. We also applied the proposed graph search techniques for Information Discovery over biological databases. Our algorithms were experimentally evaluated for performance and quality. The quality of our method was compared to current approaches by using user surveys.

TABLE OF CONTENTS

CHAPTER PAGE

1 INTRODUCTION 1

2 RESEARCH SIGNIFICANCE 5

3 RELATED WORK 7
3.1 Keyword Search on Data Graphs 7
3.2 IR Ranking 8
3.3 Link based Semantics 9
3.4 Document Summarization & Web Search 10
3.5 Relevance Feedback and Query Reformulation 11

4 FRAMEWORK & PROBLEM DEFINITIONS 13
4.1 Data Model 13
4.1.1 Web graph 13
4.1.2 Page graph 13
4.1.3 Data graph 15
4.1.4 Schema graph 16
4.1.5 Authority Transfer Schema Graph 17
4.1.6 Authority Transfer Data Graph 19
4.2 Problem definitions 21
4.2.1 Definition 1 (Minimal Total Web Spanning Tree). 21
4.2.2 Definition 2 (Minimal Total Page Spanning Tree). 21
4.2.3 Definition 3 (Search Result). 23

5 OVERVIEW AND ALGORITHMS 24
5.1 Web Search 25
5.1.1 Building Page Graphs 25
5.1.2 Query-Specific Document Summarization 29
5.1.3 Search using Composed Pages 33
5.1.4 Experimental Results 41
5.2 Authority Flow-Based Graph Search 46
5.2.1 Explaining Query Results 47
5.2.2 Query Reformulation 54
5.2.3 Experimental Results 58
5.3 Graph Information Discovery (GID) 64
5.3.1 GID Query Language 68
5.3.2 Related Research 73
5.3.3 Algebra for GID 76
5.3.4 GID Soft Filters computed by Authority Flow 80
5.3.5 GID Optimizer and Execution 85
5.3.6 Experimental Results 88

5.4 Comparing Top-*k* XML Lists 95
5.4.1 XML Lists Distance based on Total Mapping (XLDTM) 98
5.4.2 Computing *XLDTM* 103
5.4.3 Experimental Results 105

6 CONCLUSIONS 110

LIST OF REFERENCES 111

VITA 118

LIST OF TABLES

TABLE PAGE

Table 1: Top-3 search results for query - Graduate Research Scholarships. 24

Table 2: Real & Synthetic Datasets. 41

Table 3: Average Top-5 search result ratings for 10 queries. 42

Table 4: Average summary ratings for documents. 43

Table 5: Average summary ratings for Queries 1 and 2 in DUC topics. 44

Table 6: Average summary ratings for Queries 3 and 4 in DUC topics. 44

Table 7: Queries used for documents 45

Table 8: Real and Synthetic Datasets 58

Table 9: Physical Implementation of GID Algebra Operators. 87

Table 10: Datasets 89

Table 11: XML Datasets 105

LIST OF FIGURES

FIGURE PAGE

Figure 1: Sample Web pages from www.fiu.edu. 14

Figure 2: A page graph of Page 1 in Figure 1. 15

Figure 3: A subset of the DBLP graph. 16

Figure 4: The DBLP schema graph. 16

Figure 5: DBLP authority transfer schema graph. 16

Figure 6: A subset of the Biological data graph. 17

Figure 7: Subset of Schema Graph for a Biological Dataset. 18

Figure 8: Authority Transfer Schema Graph for Biological Database. 18

Figure 9: The DBLP Authority transfer data graph. 19

Figure 10: Authority transfer data graph for Biological database. 19

Figure 11: The Minimal Total Web Spanning Trees of Web graph in Figure 1 for query - Graduate Research Scholarships. 20

Figure 12: The Minimal Total Page Spanning Trees of Pages 2 and 4 in Figure 1 for query - Graduate Research Scholarships. 22

Figure 13: Top summary of web page 1 of Figure 1 for query - research scholarships. .. 29

Figure 14: Top-1 Enumeration Algorithm. 31

Figure 15: Top-1 Expanding Search Algorithm. 33

Figure 16: Composed Page for Search Result #1 for query - Graduate Research Scholarships. 35

Figure 17: Preprocessing Algorithm. 38

Figure 18: Heuristic Top-k Expanding Search Algorithm. 40

Figure 19: The DBLP Authority transfer data graph annotated with authority flows for query - OLAP. 48

Figure 20: Intuition behind flow adjustment. 49

Figure 21: Algorithm to Compute Flows in Explaining Subgraph. 52

Figure 22: Explaining Subgraph for Range Queries in OLAP paper in Figure 9. 53

Figure 23: Average Precision for different calibration parameters. 60

Figure 24 : Training of the Authority Transfer Rates. 61

Figure 25: Average Precision using structure-only reformulation with Cf=0.5. 61

Figure 26: Training of the Authority Transfer Rates. 62

Figure 27: DBLPcomplete Execution. 63

Figure 28: Sample semantic query evaluation. 73

Figure 29: Execution plan for query q1 79

Figure 30: Layered Graph. 82

Figure 31: Approximate Single Authority-Flow Soft Filter. 83

Figure 32: Performance experiments of Path-Length-Bound Technique. 91

Figure 33: Quality Experiments of Path-Length-Bound Technique. 91

Figure 34: Performance experiments of Graph-Sampling Technique. 93

Figure 35: Quality Experiments of Graph-Sampling Technique. 93

Figure 36: Top-3 trees for query - Ullman Database. 96

Figure 37: Algorithm for computing XLDTM 103

Figure 38: XLDTM Experiments on DBLP Dataset. 106

Figure 39: XLDTM Experiments on NASA Dataset. 108

Figure 40: Performance Experiments on NASA dataset 110

1 INTRODUCTION

Graph-structured databases are widely prevalent, and the problem of effective search and retrieval from such graphs has been receiving much attention recently. For example, the Web can be naturally viewed as a graph [PBMW98, Kle99]. Likewise, a relational database can be viewed as a graph where tuples are modeled as vertices connected via foreign-key relationships [BNH+02], and a XML database can be represented as a graph with XML elements as nodes and containment or ID-IDREF edges as hyperlinks [GSBS03, CMKS03]. Keyword search querying has emerged as one of the most effective paradigms for information discovery, especially over HTML documents in the World Wide Web [PBMW98, Kle99, LCVA01]. One of the key advantages of keyword search querying is its simplicity – users do not have to learn a complex query language, and can issue queries without any prior knowledge about the structure of the underlying data. Since the keyword search query interface is very flexible, queries may not always be precise and can potentially return a large number of query results, especially in large document collections. Consequently, an important requirement for keyword search is to rank the query results so that the most relevant results appear first. Recently, the problem of keyword search over relational [HGP03, HP02, ACD02, BHP04] and XML [GSBS03, HPB03, CMKS03] databases has received much attention.

The goal of this thesis is to develop techniques for user-friendly, high quality and efficient searching of graph structured databases. Several ranked search methods on data graphs have been studied in the recent years. A data graph is a graph $G(V, E)$, where V is

a set of vertices, and E is a set of edges between the vertices. The graph could be either weighted or un-weighted. Given a top-*k* keyword search query on a graph and some ranking function, a *keyword proximity search* finds the top-*k* answers where each answer is a substructure of the graph containing all query keywords. Conceptually, the problem may be defined as follows. Given a keyword query *Q* as a set of keywords, a search result is a tree *R* which is a sub-graph of *G* such that every keyword is contained in at least one vertex of *R*, and we cannot remove any node from *R* and still have a tree. The score[1] of R is defined as the sum of the weights of all edges in *R*. Given a graph G(V,E), a keyword query *Q*, and an integer *k*, we are interested in retrieving the *k* search results with the smallest scores. When $k = 1$, the keyword proximity search problem has been shown to be equivalent to the Group Steiner problem [Rei89], which is NP-complete. There have been efforts to approximate the Group Steiner tree problem in the theory community [Ihl90,GKR00]. In the database community, past research has focused on fast heuristic solutions for the keyword proximity search problem for general values of *k* [BNH+02, GSVM98, LCVA01]. We apply keyword proximity search on the web and the page graph of web documents to find top-k answers that satisfy user's information need and increase user satisfaction.

Another effective ranking mechanism applied on data graphs is the authority flow based ranking mechanism. Given a top-*k* keyword search query on a graph, an authority-flow based search finds the top-*k* answers where each answer is a node in the graph ranked according to its relevance and importance to the query. This technique was first

[1] This (or an equivalent) definition of score has been commonly used in earlier works [GSVM98, LCVA01, BNH+02, ACD02, HP02] – informally, this measure favors "tighter" trees.

applied on the web [PBMW98] and later over databases [BHP04] and XML[GSBS03]. In the context of the Web, PageRank [PBMW98] is used to compute a global ranking of the pages based on the hyperlink structure. ObjectRank [BHP04] applies the idea of authority flow on a data graph, where nodes represent entities like tuples, and edges represent associations like primary-to-foreign keys. In contrast to PageRank, ObjectRank provides query-specific ranking by using the query-specific nodes as the authority source (called base set). Another key feature of ObjectRank, as explained below, is that different edge types carry different amounts of authority. The Hubs of Knowledge project [SIY06] applies the PageRank algorithm on a query-dependent subgraph of the original biological graph. Raschid et al. [RWL+06] apply PageRank and ObjectRank to answer navigational queries on biological data. Conceptually, the ranking is produced in the following way: Myriads of random surfers are initially found at the objects containing the keyword "OLAP", which we call the base set, and then they traverse the database graph. In particular, at any time step, a random surfer is found at a node and either (i) makes a move to an adjacent node by traversing an edge, or (ii) jumps randomly to an "OLAP" node without following any of the links. The probability that a particular traversal happens depends on multiple factors, including the type of the edge. These factors are depicted in an authority transfer schema graph. Figure 5 illustrates the authority transfer schema graph used by the ObjectRank project [BHP04]. Assuming the probability that the surfer moves back to an "OLAP" node is 15% (damping factor–random jump probability [PBMW98]), the collective probability to move to a referenced paper is up to 85% · 70% (70% is the authority transfer rate of the citation edge as we explain below), and so on. As is the case with the PageRank algorithm as well, as time goes on, the

expected percentage of surfers at each node v converges to a limit $r(v)$. Intuitively, this limit is the ObjectRank of the node.

We develop techniques to improve the authority flow based search on data graphs by creating a framework to explain and reformulate them taking in to consideration user preferences and feedback. Querying large biological data collections in a flexible and efficient way is a research problem which we plan to explore. Our goal is to apply those techniques taking in to consideration the domain specifics. The specific goals of this thesis are as follows:

1. Improve Web Search Results: Propose and demonstrate a technique that given a keyword query, on-the-fly generates new pages, called composed pages that satisfy the user's information needs and improves user satisfaction. Propose and demonstrate novel algorithms for query-specific web page summarization. Specifically, given a web graph and a keyword query, generate a set of pages, called composed pages that will satisfy the user's information need. Also, given a document and a keyword query, generate a query-specific summary that best describes the document content in a concise manner.
2. Improve Authority Flow based Graph Search: Create a framework and provide algorithms to explain query results and reformulate authority flow queries based on the user's feedback. Specifically, given a top authority flow query search result for a data graph, find a best way explain why or how the top result got its current score. Also, devise efficient query reformulation algorithms to reformulate the authority flow based keyword query.

3. Provide a Flexible and Efficient Querying and Ranking Framework for Hyperlinked Databases: Propose a flexible and extensible framework for querying over large hyperlinked data collections. Specifically, create a flexible and extensible framework for efficiently querying large hyperlinked data sources.
4. Compare Top-k XML Lists: Present distance measures for computing the distance between two ranked lists of XML subtrees where all subtrees from the first list are mapped to subtrees in the second. Unfortunately, previous distance measures are not suitable for ranked lists of subtrees since they do not account for the possible overlap between the returned subtrees. That is, two subtrees differing by a single node would be considered separate objects.

The rest of the dissertation is organized as follows. Section 2 presents the significance of the research. Section 3 describes the related work. Section 4 describes framework and problem definitions. Section 5 presents the algorithms. Section 6 describes the conclusions. Finally, we present the list of references.

2 RESEARCH SIGNIFICANCE

The significance of this research is as follows:

1. Search engine industry is huge. Smallest improvement can result in millions of revenues. The composed pages technique is a novel web search technique. There is a possibility of commercialization of the idea.

2. Explaining and reformulating authority flow keyword queries have the possibility to adapt the ranking mechanism according to user's feedback, which offers new operational areas for this ranking method. The presented ideas improve authority flow ranking methods and make them usable in a broader application area.
3. An increasing amount of data is stored in biological sources, like Entrez Gene, PubMed, and OMIM. Entities of the sources are interconnected through semantic links, created manually or automatically (e.g., using BLAST). As the complexity and size of such databases increases, there is a need for flexible and efficient methods to discover information. We propose a novel extensible query language for biological databases, which is simple to use, yet expressive enough for most query needs.
4. As the use of electronic medical records becomes more widespread, so does the need to search and provide effective information discovery on them. Information discovery methods will allow practitioners and other healthcare stakeholders to locate relevant pieces of information in the growing corpus of available EMRs. The success of Web search engines has shown that keyword queries are a useful tool for locating relevant information in an intuitive and effective manner. The proposed method to apply authority flow ranking techniques considering the domain specifics have applications in creating search environments in hospitals for various users like researcher, physician, pharmacist, nurse, respiratory therapist, physical therapist and so on.

3 RELATED WORK

3.1 Keyword Search on Data Graphs

For both the document summarization as well as the web search problem, when the page graphs are already created and a query arrives, the system searches the page graphs (also the web graph) for sub-trees that contain all (or a subset of) query keywords. This problem has been studied by the database and graph-algorithms communities. In particular, recent work [ACD02, BNH+02, GSVM98, GSB+03, HGP03, HP02, KPC+05, KS06] has addressed the problem of free-form keyword search on structured and semi-structured data. These works follow various techniques to overcome the NP-completeness of the Group Steiner problem, to which the keyword proximity search problems can be reduced. Li et al. [LCVA01] tackle the problem of proximity search on the Web, which is viewed as a graph of hyperlinked pages. They use of the concept of information unit, which can be viewed as a logical Web document consisting of multiple physical pages.

Goldman et al. [GSVM98] use precomputation to minimize the runtime cost. BANKS [BNH+02] views the database as a graph and proposes algorithms to approximate the Group Steiner Tree problem. We consider and experimentally evaluate modifications of these algorithms in this work. XRANK [GSB+03] works on XML trees, which simplifies the problem. [ACD02, HGP03, HP02] perform keyword search on relational databases and exploit the schema properties to achieve efficient execution.

Finally, notice that Buneman et al. [BDFS03] view the problem of adding structure to unstructured data from a completely different angle: how to define a schema to describe a labeled graph (e.g., an XML document).

3.2 IR Ranking

In creating the document graph and computing the node weights, we adopt ranking principles from the Information Retrieval community. Various methods for weighting terms have been developed [Sin01]. The most widely used are the Okapi (Equation 1) and the pivoted normalization weighting, which are based on the *tf-idf* principle.

$$\sum_{t \in Q,d} \ln \frac{N - df + 0.5}{df + 0.5} \cdot \frac{(k_1 + 1)tf}{(k_1(1-b) + b\frac{dl}{avdl}) + tf} \cdot \frac{(k_3 + 1)qtf}{k_3 + qtf} \quad (1)$$

tf is the term's frequency in document,

qtf is the term's frequency in query,

N is the total number of documents in the collection,

df is the number of documents that contain the term,

dl is the document length (in words),

avdl is the average document length and

k1 (between 1.0–2.0), *b* (usually 0.75), and *k3*(between 0–1000) are constants.

For an overview of modern IR techniques we refer to [Sin01]. Any state-of-the-art IR ranking function is based on the tf-idf principle [Sin01]. The shortcoming of these semantics is that they miss objects that are much related to the keywords, although they do not contain them. The most popular specificity metric in Information Retrieval is the

document length (dl). The relevance information is hidden in the link structure of the data graph which is largely ignored by the traditional IR techniques.

3.3 Link based Semantics

Savoy [Sav92] was the first to use the link-structure of the Web to discover relevant pages. This idea became more popular with PageRank [PBMW98], where a global score is assigned to each Web page. HITS [Kle99] employ mutually dependant computation of two values for each web page: hub value and authority. Balmin et al. [BHP04] introduce the ObjectRank metric. In contrast to PageRank, it is able to find relevant pages that do not contain the keyword, if they are directly pointed by pages that do.

Haveliwala [Hav02] proposes a topic-sensitive PageRank, where the topic-specific PageRanks for each page are precomputed and the PageRank value of the most relevant topic is used for each query. Both works apply to the Web and do not address the unique characteristics of structured databases. Furthermore, they offer no adjusting parameters to calibrate the system according to the specifics of an application.

Recently, the idea of PageRank has been applied to structured databases [GSB+03, HXY03]. XRANK [GSB+03] proposes a way to rank XML elements using the link structure of the database. Furthermore, they introduce a notion similar to ObjectRank transfer edge bounds, to distinguish between containment and IDREF edges. Huang et al. [HXY03] propose a way to rank the tuples of a relational database using PageRank, where connections are determined dynamically by the query workload and not statically by the schema. However, none of these works exploits the link structure to provide

keyword-specific ranking. Furthermore, they ignore the schema semantics when computing the scores.

3.4 Document Summarization & Web Search

A large corpus of work has focused on generating query-independent summaries [AP00,BE97,BM00,GKMC99]. The OCELOT system [BM00] provides the summary of a web page by selecting and arranging the most (query-independent) "important" words of the page. Amitay and Paris [AP00] propose a new fully automatic pseudo-summarization technique for Web pages, where the anchor text of hyperlinked pages is used to construct summaries. [BE97] uses lexical chains for text summarization.

The majority of systems participating in the past Document Understanding Conference [DUC05] (a large scale summarization evaluation effort sponsored by the United States government), and the Text Summarization Challenge [FO01] are extraction based. Extraction-based automatic text summarization systems extract parts of original documents and output the results as summaries [CKS03,Edm69,GKMC99,HL00]. Other systems based on information extraction [RM98] and discourse analysis [Mar99] also exist but they are not yet usable for general-domain summarization. However these works do not exploit the inherent structure of the document and mostly focus on query-independent summaries. In this work (as in [VH06]) we also show the semantic connections between the extracted fragments.

White et al. [WRJ02], Tombros and Sanderson [TS98] and Goldstein et al. [GKMC99] create query-dependent summaries using a sentence extraction model in which the documents (web pages) are broken up into their component sentences and

scored according to factors such as their position. A number of the highest-scoring sentences are then chosen as the summary. [AP97,Hea94,SSMB97] select the best passage of a document as its summary. However, these works ignore possible semantic connections between the sentences or the possibility that linking a relevant set of text fragments will provide a better summary. Radev et al. [RFZ98] provide a technique for multi-document summarization used to cluster the results of a web keyword query. [ER04,MT04] provide a technique to rank sentences based on their similarity with other sentences across multiple documents and then provide a summary with the top ranked sentences. However, their methods are query-independent in contrast to our work.

The idea of splitting a Web page to fragments has been used by Cai et al. [CHWM04] and Song et al. [SLWM04], where they extract query-independent rankings for the fragments, for the purpose of improving the performance of web search. Cai et al. [CHWM04] partition a web page into blocks using the vision-based page segmentation algorithm. Song et al. [SLWM04] provide learning algorithms for block importance. Finally, all major Web search engines generate query-specific snippets of the returned results. Although their algorithms are not published, we observed that they simply extract some of the query keywords and their surrounding words. Recently, some of these companies made available tools to provide the same search and snippet functionality on a user's desktop [GD07,MD07].

3.5 Relevance Feedback and Query Reformulation

Salton and Buckley [SB90] introduced the idea of using relevance feedback for improving search performance. Relevance feedback covers a range of techniques

intended to improve a user's query and facilitate retrieval of information relevant to a user's information need. In [BSA94, BSA+95], they showed that query expansion and query term reweighting are essential to Relevance Feedback. For a detailed survey of relevance feedback methods we refer to [RL03, Har92]. The basic approach of term selection, term reweighing and query expansion [Efth93,Har88,MSB98, SVR83,SB95,KF06,XC96,LJ01,HC93] using terms drawn from the relevant documents works well for traditional IR which is content-based. For link-based metrics like ObjectRank [BHP04] this yields poor results. Hence, we need link-based (structure-based) relevance feedback methods.

Nie et al. [NZW+05] and Ararwal et al. [ACA06] present query-independent techniques to assign popularity propagation factor values (similar to the authority flow rates of ObjectRank) to Web objects, given an optimal object ranking. Our structure-based reformulation technique, which is query and feedback-specific, is inspired by these works. A recent work [VB06] on relevance feedback is based on web-graph distance metrics. The basic idea, which is similar to our content-based reformulation technique, is that relevant pages tend to point to other relevance pages, while irrelevant pages are pointed to by other irrelevant pages. Another recent study on relevance propagation over the web [QLZ+05] propose site-based propagation models that out-perform hyperlink-based models. Another recent work [SZ05] describes active feedback algorithms that help to choose documents for relevance feedback so that the system can learn most from the feedback.

4 FRAMEWORK & PROBLEM DEFINITIONS

4.1 Data Model

4.1.1 Web graph: Let $D=\{d_1,d_2,\ldots,d_n\}$ be a set of Web pages $d_1,d_2,\ldots,d_n$. Also let $size(d_i)$ be the length of d_i in number of words. Term frequency $tf(d,w)$ of term (word) w in a Web page d is the number of occurrences of w in d. Inverse document frequency $idf(w,D)$ is the inverse of the number of Web pages containing term w in them.

The *Web graph* $G_W(V_W,E_W)$ of a set of Web pages $d_1,d_2,\ldots,d_n$ is defined as follows:

- A node $v_i \in V_W$, is created for each Web page d_i in D.
- An (undirected) edge $e(u,v) \in E_W$ is added between nodes $u,v \in V_W$ if there is a hyperlink between u and v.

An example of a web graph is shown in Figure 1.We view the Web graph as undirected since an association between pages occurs along both directions of a hyperlink.

4.1.2 Page graph: In contrast to previous works in Web search [Kle99, LCVA01, PBMW98], we go beyond the page granularity. To do so, we view each page as a set of text fragments connected through semantic associations.

A key component in our work is the *page graph* $G_d(V_d,E_d)$ of a Web page d which is defined as follows:

- d is split into a set of non-overlapping text fragments and each fragment is represented by a node $v \in V_d$. A text fragment corresponding to a node v is denoted as $t(v)$.

- An undirected, weighted edge $e(u,v) \in E_d$ is added between nodes $u,v \in V_d$ if there is an association (further discussed later) between $t(u)$ and $t(v)$ in d.

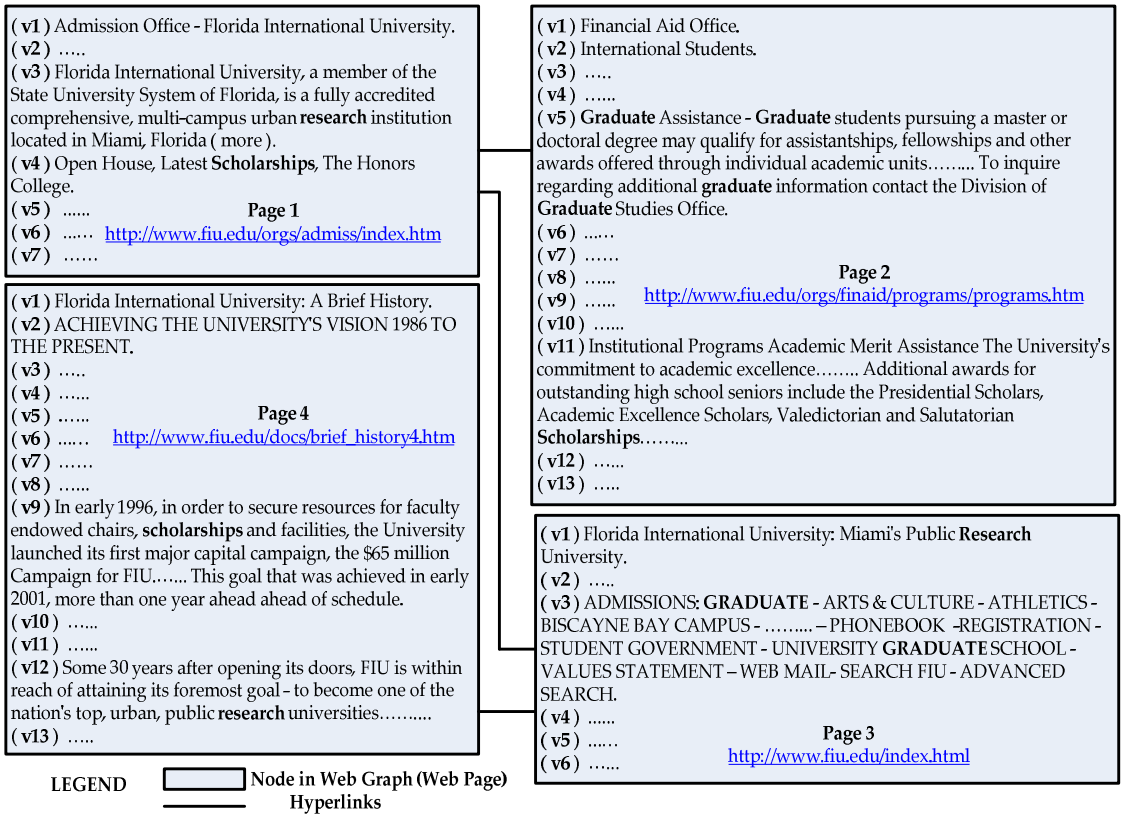


Figure 1: Sample Web pages from www.fiu.edu.

Figure 2 shows the page graph of Page 1 in Figure 1. The process of building page graphs is explained later. The page graph is equivalent to the document graph in [VH06]. Notice that there are many ways to define the page graph for a Web page. In this work we exploit the HTML tags to split the page into text fragments, and edges are added when the text fragments are associated through common (or related) words. The semantic association between the nodes is used to compute the edge weights (query-independent)

while the relevance of a node to the query is used to define the node weight (query-dependent). Note that the Web graph now becomes a graph of page graphs.

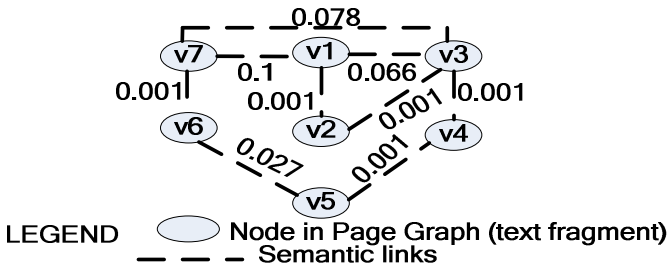


Figure 2: A page graph of Page 1 in Figure 1.

4.1.3 Data graph: We view a database as a labeled graph, which is a model that captures both relational and XML databases. The *data graph* $D(V_D,E_D)$ is a labeled directed graph where every node v has a label $\lambda(v)$ and a set of keywords. For example, the node "ICDE 1997" of Figure 3 has label "Year" and the set of keywords {''ICDE'', ''1997'', ''Birmingham''}. Each node represents an *object* of the database and may have a sub-structure. Without loss of generality, ObjectRank assumes that each node has a tuple of attribute name/attribute value pairs. For example, the "Year" nodes of Figure 3 have name, year and location attributes. Notice that the keywords appearing in the attribute values comprise the set of keywords associated with the node. One may assume richer semantics by including the metadata of a node in the set of keywords. For example, the metadata "Forum", "Year", "Location" could be included in the keywords of a node. A subset of a biological data graph is shown in Figure 6.

Each node v has a role $\lambda(v)$. For instance, the ICDE conference node in Figure 3 has role "conference". Each edge e from u to v is labeled with its role $\lambda(e)$ (we overload λ) and represents a relationship between u and v. For example, every "paper" to "paper"

edge of Figure 3 has the label "cites". When the role is evident and uniquely defined from the labels of *u* and *v*, we omit the edge label. For simplicity we will assume that there are no parallel edges and we will often denote an edge *e* from *u* to *v* as "*u*→*v*". The data graph can represent relational [ACD02, HP02] and XML [HPB03, GSB+03] databases, as well as the Web [PBMW98], although we repeat that the Web is out of the scope of this work.

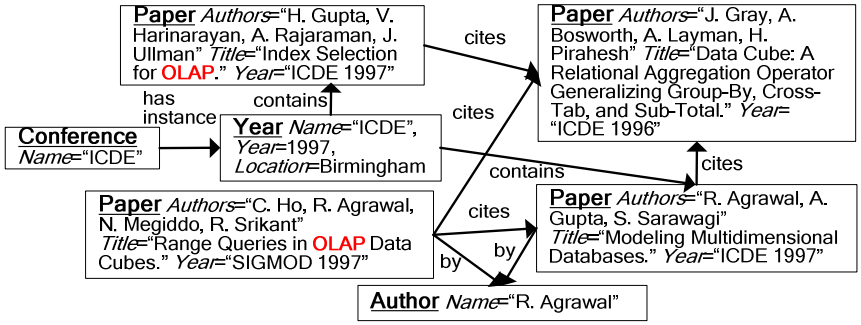


Figure 3: A subset of the DBLP graph.

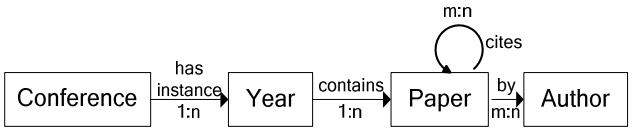


Figure 4: The DBLP schema graph.

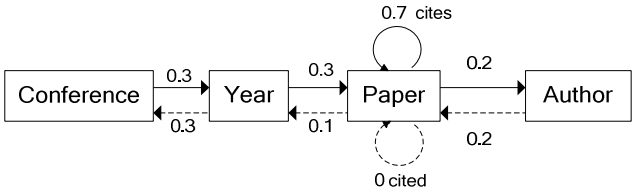


Figure 5: DBLP authority transfer schema graph.

4.1.4 Schema graph: The *schema graph* $G(V_G,E_G)$ (Figures dblp_schema and bio_schema) is a directed graph that describes the structure of *D*. Every node has an

associated label. Each edge is labeled with a role, which may be omitted, as discussed above for data graph edge labels. We say that a data graph $D(V_D,E_D)$ *conforms* to a schema graph $G(V_G,E_G)$ if there is a unique assignment μ of data-graph nodes to schema-graph nodes and a consistent assignment of edges such that: (1) for every node $v \in V_D$ there is a node $\mu(v) \in V_G$ such that $\lambda(v) = \lambda(\mu(v))$; (2) for every edge $e \in E_D$ from node u to node v there is an edge $\mu(e) \in E_G$ that goes from $\mu(u)$ to $\mu(v)$ and $\lambda(e) = \lambda(\mu(e))$.

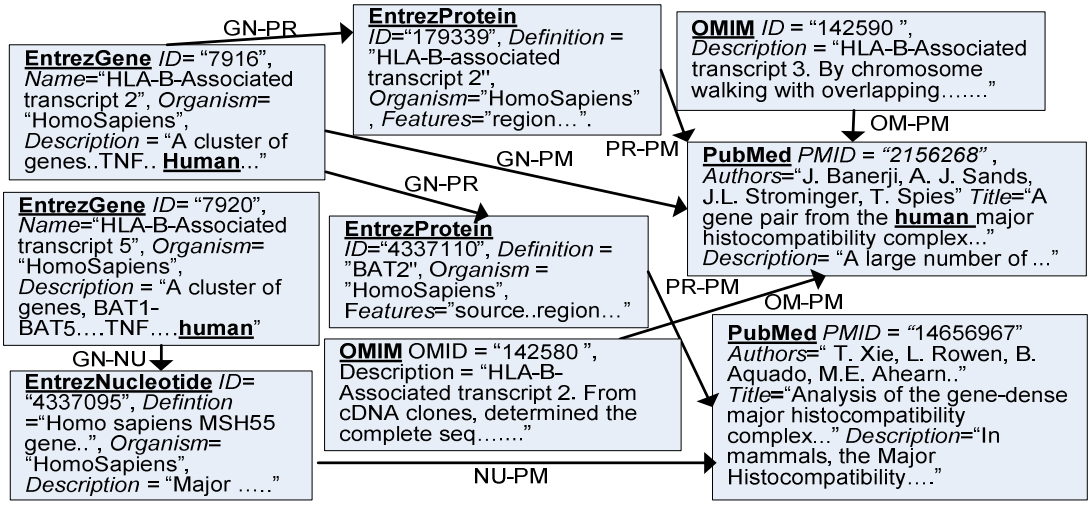


Figure 6: A subset of the Biological data graph.

4.1.5 Authority Transfer Schema Graph: From the schema graph $G(V_G,E_G)$, we create the authority transfer schema graph $G^A(V_G,E^A)$ to reflect the authority flow through the edges of the graph. In particular, for each edge $e_G = (u \rightarrow v)$ of E_G, two *authority transfer edges*, $e_G^f = (u \rightarrow v)$ and $e_G^b = (v \rightarrow u)$ are created. The two edges carry the label of the schema graph edge and, in addition, each one is annotated with a (potentially different) authority transfer rate - $\alpha(e_G^f)$ and $\alpha(e_G^b)$ respectively. We say that a data graph *conforms* to an authority transfer schema graph if it conforms to the corresponding

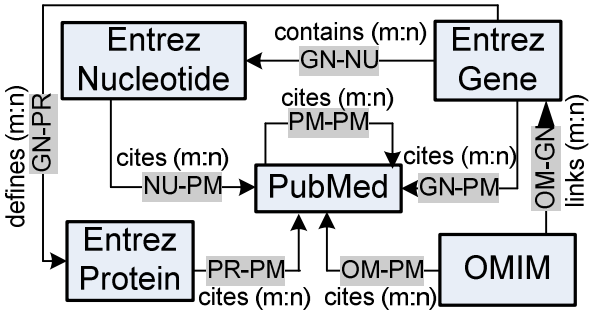


Figure 7: Subset of Schema Graph for a Biological Dataset.

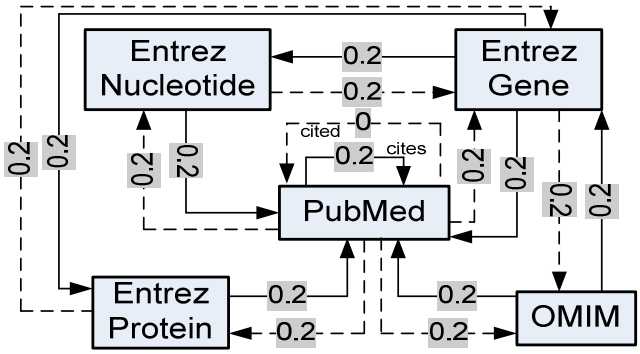


Figure 8: Authority Transfer Schema Graph for Biological Database.

schema graph. (Notice that the authority transfer schema graph has all the information of the original schema graph.) In Balmin at el. [BHP04] the authority transfer rates for each edge type was assigned manually by a domain expert on a trial and error basis. In contrast, our techniques allow this task to be done automatically based on the user's feedback as we explain in later sections.

Figure 5 shows the authority transfer schema graph that corresponds to the schema graph of Figure 4 (the edge labels are omitted), while Figure 8 shows the authority transfer schema graph that corresponds to the schema graph of Figure 7 (the edge labels are omitted). The motivation for defining two edges for each edge of the

schema graph is that authority potentially flows in both directions and not only in the direction that appears in the schema. For example, a paper passes its authority to its authors and vice versa. Notice however, that the authority flow in each direction (defined by the authority transfer rate) may not be the same. For example, a paper that is cited by important papers is clearly important but citing important papers does not make a paper important.

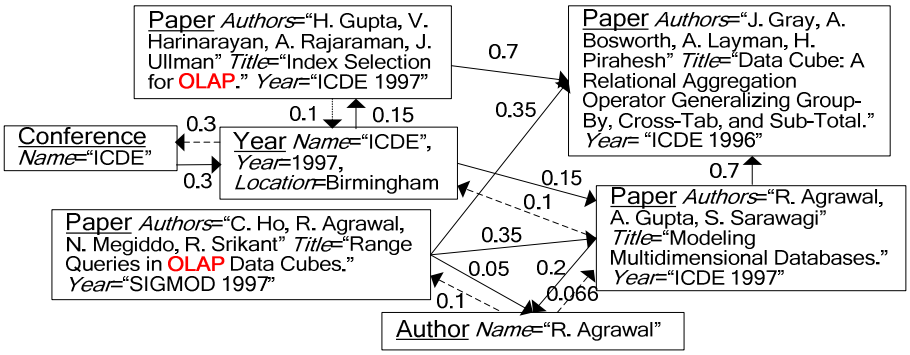


Figure 9: The DBLP Authority transfer data graph.

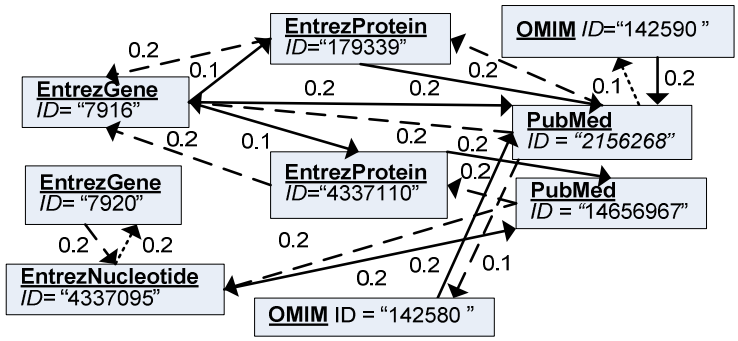


Figure 10: Authority transfer data graph for Biological database.

4.1.6 Authority Transfer Data Graph: Given a data graph $D(V_D,E_D)$ that conforms to an authority transfer schema graph $G^A(V_G,E^A)$, we can derive an authority transfer data graph $D^A(V_D, E_D^A)$ as follows. For every edge $e = (u \rightarrow v) \in E_D$ the authority transfer data

graph has two edges $e^f = (u \rightarrow v)$ and $e^b = (v \rightarrow u)$. The edges e^f and e^b are annotated with authority transfer rates $\alpha(e^f)$ and $\alpha(e^b)$. Assuming that e^f is of type e_G^f, then

$$\alpha(e^f) = \begin{cases} \dfrac{\alpha(e_G^f)}{OutDeg(u, e_G^f)}, & if OutDeg(u, e_G^f) > 0 \\ 0, & if OutDeg(u, e_G^f) = 0 \end{cases} \qquad (2)$$

where $OutDeg(u, e_G^f)$ is the number of outgoing edges from u, of type e_G^f. The authority transfer rate $\alpha(e^b)$ is defined similarly. Figure 9 illustrates the authority transfer data graph that corresponds to the data graph of Figure 3 and the authority transfer schema graph of Figure 5.

Each edge is annotated with its authority transfer rate. Note that the edge between "Range Queries in OLAP" paper and author "Agrawal" is labeled 0.05 as the paper has three other authors not shown in Figure 9. Notice that the sum of authority transfer rates of the outgoing edges of a node u of type $\mu(u)$ in the authority transfer data graph may be less than the sum of authority transfer rates of the outgoing edges of $\mu(u)$ in the authority transfer schema graph, if u does not have all types of outgoing edges. Figure 10 illustrates the authority transfer data graph that corresponds to the data graph of Figure 6 and the authority transfer schema graph of Figure 8.

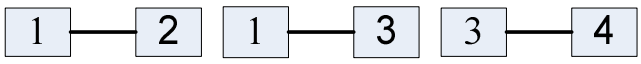


Figure 11: The Minimal Total Web Spanning Trees of Web graph in Figure 1 for query - Graduate Research Scholarships.

4.2 Problem definitions

A keyword query Q is a set of keywords $Q=\{w_1,\ldots,w_m\}$. Before defining the result of a keyword query we need a few more definitions.

4.2.1 Definition 1 (Minimal Total Web Spanning Tree). *Given a Web graph $G_W(V_W,E_W)$, a minimal total Web spanning tree of G_W with respect to a keyword query $Q=\{w_1,\ldots,w_m\}$ is a sub-tree T of G_W that is both:*

- *Total: every keyword $w \in Q$ is contained in at least one node (page) of T.*
- *Minimal: we cannot remove any node from T and still have a total sub-tree.* □

Figure 11 shows the minimal total spanning trees for the query "Graduate Research Scholarships" on the web graph of Figure 1. A result of a keyword query Q at the page granularity is a minimal total Web spanning tree T. We go one step further in order to improve the user's experience and locate the specific parts of each Web page in T that are relevant to Q. For that, we need the following definition.

4.2.2 Definition 2 (Minimal Total Page Spanning Tree). *Given a page graph $G_d(V_d,E_d)$ for a Web page d and a set of keywords $Q_i \subseteq Q$ ($Q_i=Q$ for query-specific summarization), a minimal total page spanning tree p of G_d is a sub-tree of G_d that is both:*

- *Total: every keyword $w \in Q_i$ is contained in at least one node of p.*
- *Minimal: we cannot remove any node from p and still have a total sub-tree.* □

Figure 12 shows two minimal page spanning trees for Pages 2 and 4 respectively for the query “Graduate Research Scholarships”. In both cases *v2* is a Steiner node, i.e., it does not contain any query keyword in it, but is helpful in forming a minimal total spanning tree for the pages as it has semantic links to the nodes that contain the keywords.

There is a subtle difference in the page spanning tree computation for our two different applications - searching using composed pages and query-specific summarization. For query-specific summarization of a web page we compute the page spanning tree that contains all the keywords in Q. For the composed pages application, for single-page results we compute the page spanning tree for Q, while for multi-page results we compute them for subsets of Q (see Definition 3). Note that for Steiner nodes, Q_i is empty. In this case p is an empty tree, which we represent by just displaying the *title* of the page in our system.

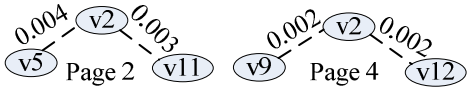


Figure 12: The Minimal Total Page Spanning Trees of Pages 2 and 4 in Figure 1 for query - Graduate Research Scholarships.

A minimal total Web spanning tree T is “refined” by finding a minimal total page spanning tree p for each of the Web pages $d \in T$ as formally explained in Definition 3. Henceforth we omit the words “minimal total” for brevity if it is clear from the context when referring to minimal total Web spanning trees or page spanning trees. The size of a Web or page spanning tree is the number of edges it contains.

4.2.3 Definition 3 (Search Result). *Given a Web graph $G_W(V_W,E_W)$, page graphs for each Web page in G_W, and keyword query $Q=\{w_1,\ldots,w_m\}$, a search result R is a minimal total Web spanning tree T with nodes (pages) $d_1,..,d_z$, along with a minimal total page spanning tree for each d_i with respect to a subset Q_i of Q. Each page d_i is assigned a subset Q_i of Q (d_i must contain all keywords in Q_i although it may contain more keywords of Q than Q_i) such that $Q_i \cap Q_j = \varnothing$ for every $i \neq j$, and $Q_1 \cup \ldots \cup Q_z = Q$.* □

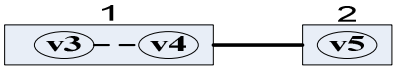
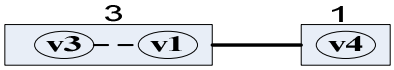
For example, Table 1 shows the Top-3 search results for the query "Graduate Research Scholarships". The Web spanning tree 3—1 gives rise to two search results. Page 3 contains keywords "graduate" and "research" and Page 1 contains "research" and "scholarships", that is, keyword "research" appears in both pages. One search result is computed with subsets Q_1 = {graduate, research} for Page 3 and Q_2 = {scholarships} for Page 1, while the other with Q_1 = {graduate} for Page 3 and Q_2 = {research, scholarships} for Page 1. We only return the best search result for each Web spanning tree to the user as shown in Table 1.

We are now ready to formally define the two problems addressed in this work. The scoring of search results and summaries trees is presented in later sections. Smaller scores correspond to higher ranking.

Problem 1 (Top-k Search Results). *Given a Web graph G_W, the page graphs for all pages in G_W, and a keyword query Q, find the k search results R with minimum Score(R).* □

Problem 2 (Query-Specific Summarization). *Given a document $d \in D$ and its page graph G_d, and a keyword query Q, find the best summary, i.e., the minimal total spanning tree with minimum score.* □

Table 1: Top-3 search results for query - Graduate Research Scholarships.

Rank	Score	Search Results
1	12.50	
2	101.60	
3	209.89	

Notice that typically a single summary per page is required and hence Problem 2 is a top-1 problem. Notice that the totality property implies that we use *conjunctive* query semantics (AND). Applying OR semantics to Problem 2 is straightforward, as we just replace *Q* by *Q′*, where *Q′* is the set of query keywords contained in the page. Applying OR semantics to Problem 1 is unintuitive since the primary purpose of the composed pages approach is to produce complete (total) answers to the user.

5 OVERVIEW AND ALGORITHMS

In this section we present various algorithms used in our system. In section 5.1, we present the algorithms to compute query-specific summarization and composed pages. Note that the algorithms used in the query-specific summarization problem are also used

as a component of the composed pages problem. The pre-computation requirements are also the same. In section 5.2, we present algorithms to improve the authority flow-based graph search by providing a way to explain query results and also provide algorithms for query reformulation.

5.1 Web Search

5.1.1 Building Page Graphs

The page graph $G_d(V_d,E_d)$ of a page $d \in D$ is constructed as follows [VH05,VH06,VHL06,VHL08]. First we parse d and split it into text fragments using parsing delimiters (e.g., <p>,
 tags). Each text fragment becomes a node in the page graph. A weighted undirected edge is added to the page graph between two nodes if they either correspond to adjacent text fragments in the text or they are semantically associated. The weight of an edge denotes the association degree of the association.

There are many possible ways to define the association degree between two text fragments. In this work we consider two fragments to be associated if they share common words (excluding stop words) and the degree of association is calculated by an adaptation of traditional IR term weighting formulas [Sin01], as described below. We also consider a thesaurus to enhance the word matching capability of the system. In future versions of our system we will consider using WordNet and Latent Semantic Indexing (LSI) techniques to improve the quality of the edge weights. To avoid dealing with a highly interconnected graph, which would lead to slower execution times and higher maintenance cost, we only add edges with weights above a threshold. Also notice that the

edge weights are query-independent, so they can be pre-computed. Q is only used in assigning weights to the nodes of G_d.

The following input parameters are required during the pre-computation stage to construct the page graph:

1. *Threshold for edge weights*. Only edges with weights not below *threshold* will be created in the page graph. The choice of the threshold is a tradeoff between performance and quality, since a zero threshold would build a dense graph which would increase the processing time, while a higher threshold would decrease the quality of results by not including enough edges.

2. *Parsing Delimiters*. Parsing delimiters are used to split the Web page into text fragments. Typical choices are the <p> (paragraph) tag (each text fragment corresponds to a paragraph) or the
 (each text fragment is a sentence). Other tags that could be surrounding a possible text fragment are the <table> tag, <ul>, <ol> tags and so on. For all these tags the text between the opening and closing counterparts constitute a text fragment. In this way we found a set of tags that when used as delimiters lead to paragraphs that are typically short and leads to more compact page graphs. For plain text documents, typical choices are newline characters (each text fragment corresponds to a paragraph) or periods (each text fragment corresponds to a sentence).

3. *Maximum Text Fragment Size.* This is used in cases where a fragment is too long which would lead to large nodes (text fragments) and hence large summaries. Users typically desire concise and short summaries.

After parsing the page and creating the graph nodes (text fragments), for each pair of nodes u,v we compute the association degree between them, that is, the score (weight) $EScore(e)$ of the edge $e(u,v)$. If $EScore(e) \geq threshold$, then e is added to E_d. The score of edge $e(u,v)$ where nodes u, v have text fragments $t(u)$, $t(v)$ respectively is:

$$EScore(e) = \frac{\sum_{w \in (t(u) \cap t(v))} \left((tf(t(u), w) + tf(t(v), w)) \cdot idf(w) \right)}{size(t(u)) + size(t(v))} \quad (3)$$

where $tf(d,w)$ is the number of occurrences of w in d, $idf(w,D)$ is the inverse of the number of pages containing w, and $size(d)$ is the size of the page (in number of words). That is, for every word w appearing in both text fragments we add a quantity proportional to the $tf \cdot idf$ score of w. Notice that stop words are ignored. Furthermore, we use thesaurus and stemmer (we rely on Oracle interMedia [OI07]) to match words that are related. The sum is divided by the sum of the lengths of the text fragments in the same way as the document length (dl) is used in traditional IR formulas.

Edges between adjacent fragments: We consider adjacent fragment edges as a special case because two adjacent fragments are semantically related because of their close proximity. Furthermore, linking the adjacent nodes ensures the connectivity of the page graph. We use the following formula, which ensures that there is always an edge between nodes with adjacent text fragments:

$$EScore(e) = max(EScore(e), threshold) \quad (4)$$

The calculation of the edge weights concludes the query-independent part of the page graph creation. Next, when a query Q arrives, the nodes in V_d are assigned query-

dependent weights according to their relevance to *Q*. In particular, we assign to each node *v* corresponding to a text fragment *t*(*v*) node score *NScore*(*v*) defined by the Okapi formula [Sin01] (Equation 1). In order to accelerate this step of assigning node scores we build a full-text index on the set *D* of pages. The details of this index are out of the scope of this paper.

Ranking of Page Spanning Trees

In this section we present our ranking framework for page spanning trees. Recall that the top page spanning tree is the query-specific summary for Problem 2. Given the page graph G_d of page *d* and a query *Q*, a page spanning tree *p* is assigned a score *Score*(*p*) by combining the scores of the nodes $v \in p$ and the edges $e \in p$.

$$Score(p) = a \sum_{edgee \in p} \frac{1}{EScore(e)} + b \frac{1}{\sum_{nodev \in p} NScore(v)} \quad (5)$$

where *a* and *b* are constants discussed below. *EScore(e)* is the score of edge *e* using Equation 4, *NScore*(*v*) is the score of node *v* using Equation 1.

Intuitively, if *p* is larger (has more edges) then its score should degrade (increase) since larger trees denote looser semantic connections [ACD02,BNH+02,HP02,HPB03]. This is the reason we take the sum of the inverse of the edge scores in Equation 5. Furthermore, if more nodes of *p* are relevant to *Q*, the score should be improved (decreased). Hence, we take the inverse of the sum of the node scores.

Constants *a* and *b* are used to calibrate the importance of the size of the summary (in number of edges) versus the amount of relevant information contained. In particular,

higher *a* values boost the score of smaller and tightly connected summaries, whereas higher *b* values benefit summaries with more relevant content (i.e., containing nodes with high score with respect to the query). Notice that *a* and *b* can also be viewed as adjusting parameters for the query-independent and dependent parts of the scoring function respectively. We use a=1 and b=0.5 in our system, which we have found to produce high-quality answers.

5.1.2 Query-Specific Document Summarization

This section tackles Problem 2 [VH05,VH06,VHL08]. Given a query *Q* and a page graph G_d for a page *d*, the query-specific summary is the page spanning tree *p* of the G_d with minimum *Score*(*p*), according to Equation 5.

The extraction of the most relevant pieces of information from a web page using the notion of the page spanning tree has another application (side product), in addition to being a component in creating composed pages. In particular, it is used to perform *query-specific summarization* of web pages. The most popular use of query-specific summarization today is the snippets displayed for each of the page results of Web search engines. We show how the query-specific summaries corresponding to page spanning trees have better quality than current approaches.

Florida International University, a member of the State University System of Florida, is a fully accredited comprehensive, multi-campus urban **research** institution located in Miami, Florida (more)
└Open House, Latest **Scholarships**, Honors College

Figure 13: Top summary of web page 1 of Figure 1 for query - research scholarships.

Example: *For the web page 1 of Figure 1 and the keyword query "Research Scholarships", the top summary v3-v4 is shown in Figure 13. The top summary is the top spanning tree of the page graph of page 1 shown in Figure 2. Nodes v3 and v4 are associated because they are adjacent in the text (stronger associations are assigned when the nodes have common words as explained below in the text).* □

For both Problems 1 or 2, we need to solve a variant of the Group Steiner Tree problem, which is referred to as keyword proximity search problem [BNH+02, GSVM98] and is defined as follows: *Given a weighted data graph G(V, E), a keyword query Q which is a set of keywords, and an integer k, find the k minimum-weight sub-trees of G such that every keyword in Q is contained in at least one vertex of the sub-tree, and we cannot remove any node from it and still have a tree.*

When k = 1, the keyword proximity search problem has been shown to be equivalent to the Group Steiner problem, which is NP-complete. The keyword proximity search problem is slightly more complex since the groups of nodes are not disjoint, in contrast to the Group Steiner Problem, which is defined as follows:

Given an undirected, connected, and weighted graph G=(V, E); and given a family $R=\{R_1,\ldots R_k\}$ *of disjoint groups of vertices, where* R_i *is a subset of V, find a minimum-cost tree T that contains at least one vertex from each group* R_i*. Since the weights of the graph are non-negative, the solution is a tree-structure.*

This section presents two algorithms adapted from BANKS [BNH+02] to compute the top query-specific summary: the enumeration and the

expanding search algorithms. The algorithms return a top-1 summary for a Web page *d*, given its page graph G_d and a query *Q*. The reason we employ top-1 summary algorithms is that typically the user only requests a single summary for a document, as in the case of snippets in Web search engine results.

Top-1 Enumeration Algorithm: This algorithm, which is abbreviated as *Top-1-MTPST-Enumeration* (Top-1-Minimal-TotalPageSpanningTree-Enumeration), is shown in Figure 14. First, we find all combinations of nodes in G_d that are minimal (no node is redundant) and total (collectively contain all keywords in *Q*). Then, for each combination we create

```
Top-1-MTPST-Enumeration (Page Graph G_d, Query Q, Quality parameter ω)
1. Results ←∅; /*stores summaries*/
2. Find all nodes in G_d that contain some keyword of Q; /*use full-text
   index*/
3. Find all minimal combinations of nodes that collectively contain all
   keywords in Q;
4. For each minimal node combination C do {
5.    Create closure graph G_c that contains only the nodes in C;
6.    Find all possible spanning trees S of G_c;
7.    Calculate the score of each spanning tree in S using Equation 4
      by using shortest path weights between any two nodes;
8.    Pick the spanning tree p with the minimum score;
9.    Replace the edges u~v in p with their pre-computed shortest
      paths u~u_1~…~u_k~v; /* i.e., we are adding the Steiner nodes.*/
10.   Trim p to make it a minimal total spanning tree;
11.   Recalculate the score of p using Equation 5 and add p to Results;
12.   ω--;
13.   If(ω==0) Return the top ranked summary in Results; }
```

Figure 14: Top-1 Enumeration Algorithm.

a complete graph G_c (called *closure graph*) that contains all nodes in the combination and all-pairs of edges between them with weight equal to their pre-computed shortest-path distance. We then calculate all possible spanning trees in G_c, and compute their scores using Equation 5 and so on (see Figure 14 for more details). This algorithm accepts a

quality parameter ω. Higher values of ω yield higher quality results. Intuitively this parameter decides the number of different summaries that are considered before we pick the best one, given that this is an NP-complete problem.

Top-1 Expanding Search Algorithm: The basic idea is that an expanding area is created for each keyword node (node that contains a query keyword) of G_d and we start from the nodes that contain the query keywords and progressively expand them according to a shortest-paths algorithm until we find all minimal total spanning trees. In particular, the algorithm (Figure 15) finds (using the pre-computed full-text index) all the nodes that match some keywords in the query and starts expanding them incrementally. We call the sub-graph created from each keyword node *v*, expanding area of *v*. At each iteration, we expand each expanding area in parallel by adding all adjacent edges (later we discuss heuristics of expansion) to the expanding area of the previous iteration. A result (summary) is generated when a set of expanding areas meet at a common point (node) and form a minimal total page spanning tree for *Q*.

We use the precomputed all-pairs shortest paths data to efficiently grow the expanding area. That is, we only consider the edges that are contained in a shortest path from the current node *v* to any other node *u* that contains additional query keywords than *v*. When two or more expanding areas meet we check for possible new summaries. If a summary is found, it is trimmed to become minimal and its score is calculated using Equation 5.

```
Top-1-MTPST-ExpandingSearch(Page graph G_d, Query Q, Quality
parameter ω)
1. Results ←∅;  /*stores summaries*/
2. Find all nodes N={N_1,…,N_m} that contain the keywords in Q and
   create expanding areas for each; /*N_i has the nodes that contain
   w_i*/
3. Repeat until each expanding area spans the    entire graph G {
4.    For each node v in N do {
5.     Add to the expanding area of v the minimum-score adjacent edge
       from the (precomputed) shortest paths starting at v and ending
       at a node in N not containing the same keywords as v;
6.     Check for new results (summaries);/*i.e., trees that contain
       a node from each of N_1,…,N_m */
7.     Trim summaries to make them minimal;
8.     Calculate the score of each summary p using Equation 5 and
       store in Results;
9.     ω--;
10.    If(ω==0) Return the top ranked summary in Results;}}
```

Figure 15: Top-1 Expanding Search Algorithm.

5.1.3 Search using Composed Pages

This section tackles Problem 1 [VHL06,VHL08]. In this Section we explain how a search result (Definition 3) is ranked and discusses how a composed page is constructed for a search result.

Ranking search results

Recall that a *search result R* is a Web spanning tree *T* where each page *d* in *T* is represented by its page spanning tree *p*. Clearly there is no optimal ranking function since it is possible to come up with different ranking functions for different domains or specific queries. In this work we adopt principles well-accepted in previous works on ranking Web pages [Kle99,LCVA01,PBMW98] and trees of data [ACD02,BNH+02,GSVM98,GSBS03,HP02,VH06].

The *first ranking principle* we adopt [LCVA01] is that search results involving fewer pages are ranked higher. Intuitively, if a search result is larger (has more edges) then its score should degrade (increase) since larger trees denote looser semantic connections. Hence, search results are primarily ranked by the (inverse of the) size of their Web spanning tree. Recall that by Definition 3, all search results contain all query keywords.

Within search results with the same size of Web spanning tree, we rank according to the scores of the involved page spanning trees, computed by Equation 5. Note that the first ranking principle also applies in ranking individual page spanning trees as expressed in Equation 5, that is, page spanning trees with smaller size are ranked higher.

What is left, is to define how the scores of the constituting page spanning trees computed by Equation 5, are combined to compute the overall score of a search result. Again, we do not claim that we have the optimal combining function, but we rely on previous work to define the next principle. The *second ranking principle* is that the scores of the page spanning trees are combined using a monotone combining function to compute the score of the search result. Notice that we already used another variant of this principle in Equation 5, where the scores of the nodes and edges are combined using a monotone function.

To incorporate the global importance of the pages used in constructing a search result, we use their PageRank [PBMW98] values. Equation 6 computes the score of a search

result R given the scores of its page spanning trees p, where we chose summation as our monotone combining function.

$$Score(R) = \sum_{p \in R} \frac{Score(p)}{PR(p)} \qquad (6)$$

where $PR(p)$ is the PageRank score of page d that contains the page spanning tree p.

Figure 16: Composed Page for Search Result #1 for query - Graduate Research Scholarships.

Composed Pages

Our technique has the following key steps: During the preprocessing stage, for each web page we create a labeled, weighted graph, called the *page graph*, by splitting the page to a set of *text fragments* (graph nodes) and computing the semantic associations between them (graph edges). Then, at query time, given a set of keywords, we first find a tree, called *web spanning tree*, of hyperlinked pages that collectively contain all the query keywords. Then we perform keyword proximity search on the each page's page graph to discover how the keywords contained in the page are associated with each other. For each page in the web spanning tree we extract a *page spanning tree* that contains a subset of the query keywords. The page spanning trees of the pages of the web spanning tree are appropriately combined into a composed page, which is returned to the user. As we will

explain later, smaller web spanning trees are preferable and hence single-page results, as created by current Web search engines for AND semantics are ranked higher.

Example: *Figure 1 shows a Web graph extracted from the www.fiu.edu Web site. The hyperlinks between pages are depicted in the Web graph as edges. The nodes in the graph represent the Web pages. Figure 2 shows the page graph of Page 1 in Figure 1. As denoted in Figure 1, Page 1 is split into 7 text fragments v1…v7, using the newline delimiter, and each one is represented by a node in the page graph. The edges denote semantic associations. Table 1 shows the Top-3 search results (composed pages) for the query "Graduate Research Scholarships". We represent the nodes of a web spanning tree using rectangles and the nodes of a page spanning tree using circles. Hyperlinks are solid lines, while the semantic links within in a page graph are dotted lines. The page spanning trees represent the most "relevant pieces" of each page.* □

Note that a key assumption we make in this paper is that hyperlinked pages are associated to each other. This is a reasonable assumption. Furthermore, each result should be composed of pages associated to each other to have a cohesive meaning. Hence, we only consider hyperlinked pages in building web spanning tree.

A composed page is a dynamic page created on-the-fly by stitching together pieces from other pages. Given a query *Q*, a composed page is a representation of a search result, as defined in Definition 3, in a Web page format. The score of a composed page is the score of the corresponding search result defined by Equation 6. The key

requirements in constructing a composed page are the following: First, display the tree-structured (more specifically tree of trees) search result in a page format. Second, allow users to easily navigate to the original pages that were used to construct the composed page. Figure 16 shows the composed page constructed for the Search Result #1 of Table 1. A composed page for a search result is constructed by displaying links to all pages in its Web spanning tree along with the text fragments of the page spanning trees. The page spanning trees are displayed in an unordered list format that depicts their structure. A sub-bulleted list denotes the parent-child relationship in the page spanning tree of text fragments.

Figure 17 describes the preprocessing algorithm. Before any query arrives we pre-compute and store the following:

- *The page graph for each page*. In particular, we parse the HTML documents based on the tags and compute the edge weights. The parameters described in Section 3 are taken as input and page graphs are built accordingly.
- *PageRank values* of each page by executing the PageRank algorithm [PBMW98].
- *A full-text index* to efficiently locate the pages and specifically the text fragments that contain the keywords and calculate their query-specific score.
- In order to boost the performance of the algorithms, *the all-pairs shortest paths* between the nodes of the page graph G_d of every page *d*. Note that the inverse of the edge weights is used since larger edge weights denote tighter association in our setting.

```
Preprocess (Web Graph G_w, Parsing Delimiters P, Threshold τ, Maximum Fragment size sz)
1.  For each web page (node) d in G_w do {
 /* create and store page graph G_d for d*/
2.    Parse d and split it into text fragments with maximum size
      sz using the delimiters in P;
3.    Create a node for each text fragment and add it to the page
      graph, G_d of d;
4.    For every pair of nodes in G_d find if they are semantically
      related by calculating the edge weight using Equation 1 and
      add it to G_d if the edge weight ≥ τ;
5.    For every pair of adjacent nodes, build an edge e with
      weight equivalent to max(Escore(e), τ) according to Equation
      2;/*in close proximity as explained*/
6.      Find All-pairs shortest path using Floyd Warshall's
      algorithm using the inverse of each edge's weight;}
7. Compute and store the PageRank values of all  pages (nodes) in
  G_w; /* compute PageRank values;  build full-text index*/
8. For each keyword w locate and store all pages
   in D that contain w; /*Stemming is used in
   this step. Stop words are ignored*/
```

Figure 17: Preprocessing Algorithm.

This algorithm is an adaptation of the Top-1 expanding search algorithm. It also uses the *Top-1-MTPST-ExpandingSearch* method as a subroutine to compute the page spanning trees of the pages in a Web spanning tree. We adapt expanding search and not the naïve enumeration algorithm since the former is shown to perform better. The key differences from the algorithm of Figure 15 are the following. First, *Heuristic-Top-k-Expanding-Search* (Figure 18) operates on Web graphs instead of page graphs, and hence produces web spanning trees instead of page spanning trees. Second, we introduce the following heuristic based on Equation 6, which is our ranking function. In particular, we first expand towards pages *d* with highest *HeuristicWeight* value as defined by:

$$HeuristicWeight(d) = PR(d) * IRScore(d) \qquad (7)$$

where *d* is a Web page, *PR* its PageRank value, and *IRScore(d)* its Information Retrieval score for *Q*. The *PR(d)* component of Equation 7 is intuitive since it also appears in the

ranking equation (Equation 6). The *IRScore(d)* component is a heuristic estimate of the *Score*(*p*) component of Equation 6, where *p* is the page spanning tree for page *d*.

The intuition is that a page with high IR score for *Q* is also expected to have page spanning trees with high score for *Q*. We use the full-text indexer to compute *IRScore(d)*. Finally, notice that *Heuristic-Top-k-Expanding-Search* algorithm has two steps: first it computes the Web spanning trees, and for each one of them it computes the top search results by computing the corresponding page spanning trees for its pages (*getTopSearchResult* method). The following are the key steps of the algorithm involved in computing the top-*k* search results for a query *Q*.

- Compute a minimal total Web spanning tree, *WST* given the web graph G_w and query *Q*.
- Then compute the best search result for *WST*, given the page graphs of each page in *WST* and the query *Q* by considering all possible combinations of keyword assignments to the pages of *WST*.

The above steps are repeated until *k* search results are computed. The g*etTopSearchResult* method takes as input a web spanning tree and the page graphs of the constituent pages and returns the best search result after evaluating all possible search results. It uses the *Top-1-MTPST-ExpandingSearch* method to compute the top page spanning trees corresponding to the query.

```
Heuristic-Top-k-Expanding-Search(Web graph G_w, Page graphs PG ={G_d1, G_d2 … G_dn}, Keyword query Q = {w_1,…,w_m})
1.   Results ← 0; /* result count */
2.   Find all keyword nodes KN in G_w using the full text index; /*nodes that
match some keyword in Q*/
3.   Let Z_j be the set of nodes of G_w that contain w_j;
4.   Let L_j be the set of expanding areas corresponding to the root nodes in
     Z_j;
5.   Let buffer(i) be an array ordered by score to buffer search results
     containing i pages;
6.   For each node(page)d contained in Z_1∩Z_2∩…∩Z_m do {/*single-page search
     results*/
7.      TSR ← getTopSearchResult(d,{G_d},Q);
8.      Insert TSR into buffer(1);/* Insert TSR into the ordered buffer of
        single page search results */
9.      Results++;}
10.   While (Results < k) {
11.         For j in 1...m do {
12.         For each expanding area L in L_j do {
13.         Expand the expanding area L, with a node v having the maximum
            HeuristicWeight; /* Equation 6*/
14.         Join v to all previously expanded nodes u generated by the
            expanding areas L_s, s≠j;
            /* By "join" we mean find all instances of v as an end node in
            the already expanded nodes. */
15.       For each web spanning tree WST generated by the join {
16.        Trim useless leaves to make it minimal;
17.        TSR ← getTopSearchResult(WST,{G_d1, G_d2… G_dz}, Q);
18.        Insert TSR in to buffer(length(TSR)); / * length(TSR) equals
           number of pages in TSR */
19.        Results++; If(Results = k){ Output results in buffer and
           return; }}}}}
MODULE: getTopSearchResult(Web spanning tree WST, Page graphs WPG = {G_d1, G_d2
… G_dz} of WST, Keyword query Q = {w_1,…,w_m})
1.   SearchResults ←∅; /*stores search results*/
2.   Find the set of possible partitions PQ of Q as per Definition 3;
3.   For each partition {Q1,…,Qz} of the keywords in PQ do{
4.     For each page d_i in WST do {
5.     PSP_i←ϕ;
6.     If(Q_i≠ϕ){
7.          PSP_i ← Top-1-MTPST-ExpandingSearch(G_di,Q_i,ω); }} /* Q_i is the
           subset of Q assigned to page d_i, ω is the quality factor*/
8.          Create a search result R with each PSP_i and WST;/*if PSP_i = ϕ we
            use the title of page d_i  (this corresponds to the Steiner node
            which has no keywords in it)*/
9.          Compute Score(R) using Equation 6 and add R to SearchResults;}
10.  Return the top ranked search result in SearchResults;
```

Figure 18: Heuristic Top-k Expanding Search Algorithm.

5.1.4 Experimental Results

To evaluate the quality of the results of our approach for Problems 1 and 2, we conducted three surveys, one for Problem 1 and two for Problem 2. The subjects of the survey are twenty students (of all levels and various majors) at Florida International University (FIU), who were not involved in the project. In these surveys the users were asked to evaluate the results based on their quality.

Datasets: We use two real datasets (Table 2). FIU1 is a hyperlinked set of 25,108 Web pages (nodes) crawled from the *fiu.edu* domain, connected through 137,929 hyperlinks (edges) used for performance evaluation. FIU2 is a subset of the web pages available in *fiu.edu* domain used for quality evaluation, which offers faster response times and more focused results that are easier to compare.

Table 2: Real & Synthetic Datasets.

Name	*#nodes* (Web pages)	*#edges* (Hyperlinks)	*Size (MB)*
FIU1	25,108	137,929	4564
FIU2	6,054	45,405	115

We used FIU2 for our user surveys. The participants were asked to evaluate the quality of the search results with respect to ten queries. We chose both long and medium sized queries. For each query, users were asked to rate their satisfaction for the Top-5 search results produced from the Heuristic Top-*k* Expanding Search algorithm, and for the results produced by Google. We chose the first 5 results from Google that are included in the subset of crawled FIU web pages. The Google query was constrained to pages using the "site: fiu.edu" condition. Each participant was asked to assign a score between 1 and

5 to each alternative query answer, where 5 denote the highest user satisfaction. The results of the survey prove the superiority of our approach, as shown in Table 3.

Table 3: Average Top-5 search result ratings for 10 queries.

Keyword Queries	Google Search	Heuristic Expanding Search
Undergraduate Housing safety	2.06	3.41
Graduate financial aid regulations	2.41	3.59
Computer Science Internship opportunities	2.88	3.65
Campus Safety requirement regulations	2.24	3.35
Biomedical Research fellowship eligibility	1.24	3.35
Undergraduate Summer athletics accomplishments	2.25	4.5
Physics alumni achievements	3.25	3.00
Electrical transfer student eligibility	2.66	4.66
Freshman internship opportunities	1.66	4.66
Mechanical Graduate admission policies	1.66	4.66
Average Rating	2.44	3.88

To evaluate the quality of our query-specific summaries we created two user surveys on a DUC and a Web dataset as explained below. The size of a result was also taken into consideration by the participants – a longer result carries more information but is less desirable. Each participant was asked to compare the summaries and rank them, assigning a score of 1 to 5, according to their quality for the corresponding query. A rank of 5 (1) represents a summary that is most (least) descriptive.

Comparison with DUC dataset

The dataset used in this survey consists of twenty documents and four queries taken from the DUC 2005 dataset [DUC05] as shown in Table 5 and 6. We compare our summaries with DUC Peer summaries for quality. DUC peers are human and automatic summaries

used in quality evaluation. We compared our summaries against the DUC peers with highest linguistic quality. Unfortunately, most of the summaries in the DUC datasets are query-independent and the few query-dependent ones are multi-document. Hence, in order to compare our work to that of DUC we used the following method to extract single-document summaries from query-dependent multi-document summaries for a set of twenty documents over four topics. The sentences that have been extracted from a document *d* to construct the multi document summary are viewed as *d*'s single-document summary for the query/topic. Notice that the DUC summaries are created by extracting whole sentences from documents.

Table 4: Average summary ratings for documents.

Docs	**Keyword Queries**														
	Google Desktop Summary					MSN Desktop Summary					Top-1 Expanding Summary				
	Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*
D1	2.33	2.00	3.00	1.67	2.00	2.33	2.00	0.67	1.67	3.00	4.87	4.33	4.93	4.67	4.00
D2	3.67	3.33	2.67	2.67	1.67	3.67	3.00	3.00	3.00	1.00	3.67	3.33	4.00	4.00	3.67
D3	1.60	1.60	2.00	1.60	2.00	1.60	1.00	1.80	2.20	1.20	4.00	4.20	4.00	3.60	3.40
D4	1.00	1.33	0.66	1.33	2.33	2.66	2.00	1.33	1.66	1.33	3.66	3.66	4.00	4.00	3.33
D5	2.50	3.00	2.50	1.00	3.00	1.50	1.50	1.50	2.00	3.50	4.00	3.50	4.00	4.00	3.50
D6	1.00	1.50	1.50	2.50	1.00	2.00	2.50	1.50	3.50	2.00	4.00	4.50	4.00	2.50	4.00
D7	3.00	1.00	3.00	1.00	1.00	1.50	2.50	1.50	2.50	2.00	3.00	4.00	3.00	4.50	4.50
Average Rating	1.97					2.00					3.89				

The results of the survey prove the superiority of our approach, as shown in Table 5 and 6. Our method of combining extracted sentences using semantic connections in the form of Steiner trees leads to higher user satisfaction than the traditional sentence extraction methods. In particular, the Steiner sentences in summaries provide coherency in the aggregation of the keyword-containing-sentences.

Table 5: Average summary ratings for Queries 1 and 2 in DUC topics.

Query 1 (*International Organized Crime*) DUC Topic ID: d301i			Query 2 (*Women in Parliaments*) DUC Topic ID: d321f		
Doc. ID	DUC Peer	Top-1 Expanding	Doc. ID	DUC Peer	Top-1 Expanding
FT941-3237	2.33	4.66	FT921-7786	4.00	2.50
FT944-8297	2.50	3.33	FT922-190	2.00	4.00
FT931-3563	2.83	3.00	FT921-937	2.00	4.33
FT943-16477	4.00	4.17	FT922-13353	2.83	4.17
FT943-16238	3.67	3.67	FT921-74	2.33	3.67
Average	3.06	3.77	Average	2.63	3.73

Table 6: Average summary ratings for Queries 3 and 4 in DUC topics.

Query 3 (*Drugs Mental Illness*) DUC Topic ID: d383j			Query 4 (*Stolen Art Recovered*) DUC Topic ID: d422c		
Doc. ID	DUC Peer	Top-1 Expanding	Doc. ID	DUC Peer	Top-1 Expanding
FT933-4868	2.00	4.33	LA051889-0110	4.00	3.00
FT942-16465	1.00	5.00	FT911-5359	2.00	3.00
LA090389-0060	1.66	4.33	LA070990-0048	2.33	4.33
FT922-715	1.00	4.33	LA032090-0091	3.00	3.66
LA111290-0137	1.66	4.33	FT923-1946	4.33	3.00
Average	1.46	4.46	Average	3.13	3.40

Comparison with Google and MSN Desktop

The dataset used in this survey consists of seven news documents taken from the technology section of *cnn.com*. The participants were asked to evaluate the quality of the summaries of the seven documents with respect to five queries each (35 queries in total). We chose queries where keywords appear both close and far from each other. For each query-document pair, three summaries are displayed corresponding to (a) the result of the

Top-1 expanding search algorithm, (b) Google Desktop's summary, and (c) MSN Desktop's summary. Summaries (b) and (c) were created by indexing the two documents in our desktop and then submitting the five queries to the Desktop engines.

The summaries are the snippets output for these documents. In order to compare apples to apples, we chose queries for which the length of the summaries produced by all three methods are similar, since clearly it is not fair to compare summaries of different lengths as some people favor conciseness while others the amount of information.

In this survey we set constant *a* to 1 and *b* to 0.5 in Equation 5, which we found to produce higher-quality summaries. Notice that by increasing the value of constant *a*, we favor short results, while by increasing constant *b* we favor longer and more informative results. Hence, by setting *a* to 1 and *b* to 0.5 we favor shorter summaries, which have similar size to the ones produced by Google and MSN Desktop. This makes their comparison fairer.

Table 7: Queries used for documents.

Query #	Document *D1*	Document *D2*
1	Microsoft worm protection	IT Research awards
2	Anti-virus protection	Algorithms development Research
3	Recovering worm deleted files	Software projects
4	Worm affected agencies	Large research grants
5	Deleted computer software	Computer network security project

The results of the survey, which show the superiority of our approach, are presented in Table 4, while the queries are shown in Table 7 (only 8 queries are shown while the remaining 25 are omitted due to space constraints). Notice that Google and MSN Desktop systems do not always include all keywords in the summary when they are

more than two and have big distances between them. In contrast, our approach always finds a meaningful way to connect them.

5.2 Authority Flow-Based Graph Search

In this section we first define a query and describe a modified version of ObjectRank originally presented in [BHP04], called ObjectRank2. The modification to the original definition is that the nodes of the base set are weighted. The weights are computed using IR techniques for the original query and using query expansion techniques for subsequent queries [VH05,VH06,VHL06,VHL08].

Keyword Query. A keyword query Q is defined as a tuple of keywords $Q=[t_1,\ldots,t_m]$. To incorporate weighing in the base set, we define the query vector as follows. For each query $Q=[t_1,\ldots,t_m]$ we define a *query vector* $\boldsymbol{Q}=[w_1,\ldots,w_m]$ where w_i is the weight of the query keyword t_i. The initial query vector for a query is $[1,\ldots,1]$, since we assume that the query term weights are all 1. These weights change during the query expansion stage. The answer to $\boldsymbol{Q}$ is a list of objects with descending ObjectRank2 scores with respect to $\boldsymbol{Q}$.

ObjectRank2 is computed as follows on the authority transfer data graph $D^A(V_D, E_D^A)$. A surfer starts from a node (database object) v_i of the base set of V_D and at each step, he/she follows an edge with probability d or gets bored and jumps to a node in the base set with probability $1 - d$. The ObjectRank2 value of v_i is the probability that at a given point in time, the surfer is at v_i. The query base set $S(Q)$ (from now on referred to simply as base set when the keyword is implied) is the set of nodes/objects that contain at least one keyword in Q. In contrast to the original ObjectRank [BHP04], the random

surfer jumps to different nodes of the base set with different probabilities. This probability for a node v is proportional to the IR score $IRScore(v,\boldsymbol{Q})$ of the node(a node is also viewed as a document–we overload symbol v in this case) given the query vector $\boldsymbol{Q}$.

$$IRScore(v,\boldsymbol{Q}) = \boldsymbol{v}\cdot\boldsymbol{Q} \qquad (8)$$

where "·" denotes the dot product operator, $\boldsymbol{v}=[W(v,t_1),\ldots,W(v,t_m)]$ is the document vector for v, and $W(v,t)$ is the IR weight of term t for document v. $W(v,t)$ is defined using well studied traditional IR formulas like BM25 [RW94] or Okapi [Sin01].

We normalize the IR scores of the nodes in the base set to sum to one, since they represent probabilities. The ObjectRank2 scores vector $\boldsymbol{r}^Q = [r^Q(v_1),\ldots,r^Q(v_n)]^T$ given query vector $\boldsymbol{Q}$, where $n=|V_D|$, is defined as follows:

$$\mathrm{r}^{\mathrm{Q}} = dA\mathrm{r}^{\mathrm{Q}} + \frac{(1-d)}{|S(Q)|}s \qquad (9)$$

where A is a $n \times n$ matrix with $A_{ij} = \alpha(e)$ if there is an edge $e(\mathrm{v_j} \rightarrow \mathrm{v_i})$ in E_D^A and 0 otherwise, d is the damping factor which controls the base set importance, and $s = [s_1, \ldots s_{\mathrm{i}} \ldots, \mathrm{s}_n]^T$ is the base set vector, where $s_i = IRScore(v_i,\boldsymbol{Q})$ if $v_i \in S(Q)$ and $s_{\mathrm{i}}= 0$ otherwise. Note that the only difference to ObjectRank is the definition of the s_i's which were 0 or 1 in [BHP04].

5.2.1 Explaining Query Results

In this section we tackle the problem of explaining a query result [VHR08]. For instance, as discussed in Section 1, the "Data Cube" paper in Figure 3 (see Figure 9 for corresponding authority transfer data graph) is ranked high for the query "OLAP". What

is the best way to explain to the user why this paper, referred to as the *target object*, received a high rank? This problem is even more critical in complex biological databases.

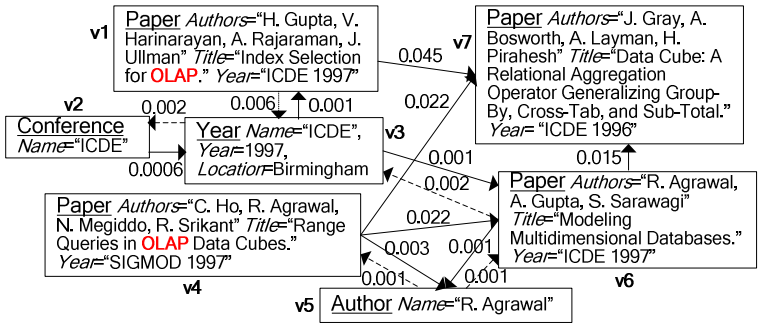


Figure 19: The DBLP Authority transfer data graph annotated with authority flows for query - OLAP.

Intuitively, we want to show to the user the paths in the authority transfer data graph D^A that authority traversed to reach the target object *v*, starting from the nodes in the base set *S*(*Q*). For that, we create an *explaining subgraph* G_v^Q of D^A that contains all edges that transfer authority to *v* given *Q*, and every edge in G_v^Q is annotated with the amount of authority that flows on this edge and eventually reaches *v*.

We create G_v^Q in two stages:

(i) *Construction stage:* G_v^Q contains all nodes and edges of D^A that are part of a directed path going from the base set *S*(*Q*) to *v*. That is, G_v^Q contains all edges that can potentially carry authority flow to *v*.

(ii) *Flow adjustment stage:* We compute the *explaining authority flows* on the edges of G_v^Q. The explaining authority flow *Flow*(*e*) of an edge *e* is the amount of authority flow that is transferred through *e* and eventually reaches *v*, on D^A for *Q*.

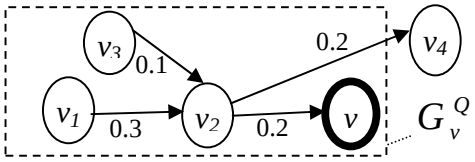


Figure 20: Intuition behind flow adjustment.

The construction stage is straightforward and is achieved as follows: We first construct the temporary subgraph D_v, starting from the target node v and traversing edges of D^A following the edges in the opposite direction in a breadth first manner (depth first would also work) until no more edges can be traversed. Then, we start from the authority sources (base set nodes) of D_v and traverse the edges of D_v in the forward direction until no more edges can be traversed. All nodes and edges traversed in the forward stage are added to the explaining sub graph G_v^Q.

The flow adjustment stage is more challenging because we have to adjust the "original" edge authority flows for Q to subtract the authority flow not reaching to v. For instance, in Figure 20 we must subtract from the edge flows the amount that will eventually "leak" out of G_v^Q through $v_2 \rightarrow v_4$. By "original" flows we refer to the authority flows at convergence state in D^A for ObjectRank2 execution for query Q. The original flow for edge $v_i \rightarrow v_j$ is:

$$Flow_0(v_i \rightarrow v_j) = d \cdot \alpha(v_i \rightarrow v_j) \cdot r^Q(v_i) \qquad (10)$$

where $\alpha(v_i \rightarrow v_j)$ is the authority transfer rate of edge $e = (v_i \rightarrow v_j)$ in D^A according to Equation 2.

Figure 19 illustrates the original authority flows for d = 0.85 and query Q=["OLAP"], on the authority transfer data graph of Figure 5. The computed ObjectRank2 scores vector $\boldsymbol{r}^Q$ = [0.076, 0.002, 0.009, 0.076, 0.017, 0.025, 0.083]^T, after 5 iterations. It is more intuitive to view the problem as adjusting the edge flows instead of adjusting the node scores, although the adjusted node scores can be easily computed given the edge flows in the end. One could think of simply reducing the flow on an incoming edge $v_i \rightarrow v_j$ of G_v^Q proportionally to the ratio of the outgoing flow of v_j going outside G_v^Q. However, this approach will fail if there are cycles in G_v^Q, since adjusting the flow of an edge can have a ripple effect. Hence, an iterative method is used. In particular, for every node u, with the exception of the target node v, we iteratively reduce its incoming flows proportionally to the flow going from u towards nodes outside of G_v^Q. We do not adjust the incoming flows of the target node v, as the purpose of the explaining subgraph is to explain to the user the total authority that v receives from other nodes in D^A. We assume all edges are bidirectional (arbitrarily small flow rates can be assigned to direction of small importance) to guarantee convergence as proved in the extended version [VHR07].

For instance, for the explaining subgraph in Figure 20 with target node v, where we assume $d=1$ (i.e., nodes pass all their authority to their neighbors) and all edges are of the same type, we adjust the original edge flows of $v_1 \rightarrow v_2$ and $v_3 \rightarrow v_2$ as follows: Half of the flow going through these edges goes through $v_2 \rightarrow v$ and half through $v_2 \rightarrow v_4$. Since $v_2 \rightarrow v_4$ is outside G_v^Q, we cut the flows of $v_1 \rightarrow v_2$ and $v_3 \rightarrow v_2$ to half, i.e., to 0.15 and 0.05

respectively. This process is repeated iteratively for all edges in G_v^Q until the computation converges. Note that the flow on edges $v_i \rightarrow v$, i.e., edges that end at v, are not adjusted.

Details of adjustment stage: The details of the adjusting algorithm are as follows: For each node v_k in G_v^Q, let $O(v_k)$ be the summation of all outgoing flows of v_k in G_v^Q and $I(v_k)$ be the summation of all incoming flows of v_k in G_v^Q (we consider all incoming edges in G_v^Q and not D^A since Observation 1 below shows that both are equal). It is

$$\mathrm{I}(v_k) = \sum_{(v_j, v_k) \in G_v^Q} Flow(v_j \rightarrow v_k) \qquad (11a)$$

$$O(v_k) = \sum_{(v_k, v_j) \in G_v^Q} Flow(v_k \rightarrow v_j) \qquad (11b)$$

Observation 1: *There is no incoming edge $v_i \rightarrow v_j$ with non-zero authority flow, where v_j is in G_v^Q but v_i is outside G_v^Q. If such en edge existed, it would have been included to G_v^Q during the construction stage.* □

As mentioned before, our goal is to compute the factor $h(v_k)$ by which the incoming flow $I(v_k)$ of each node v_k must be reduced to be consistent with the reduced outgoing flow $O(v_k)$ of v_k in G_v^Q. It is:

$$Flow(v_j \rightarrow v_k) = h(v_k) \cdot Flow_0(v_j \rightarrow v_k) \qquad (12)$$

Intuitively, this factor $h(v_k)$ is computed by the ratio of $r^Q{}'(v_k)$ and $r^Q(v_k)$ which are the ObjectRank score of v_k in G_v^Q (the "original" score) and D^A respectively. Hence, for a node v_k:

$$r^Q{}'(v_k) = \frac{O(v_k)}{d} \qquad (13)$$

```
Explain-ObjectRank(Target Object v, Graph D^A, Base Set
S(Q)={s_1,…,s_n},Threshold T) {
      /*Construction Stage */
1)    Create a temporary subgraph D_v by executing breadth-
      first search on D^A with v as the root node,
      traversing edges in opposite direction;
2)    Create explaining subgraph, G_v^Q by executing breadth-
      first search on D_v with the nodes in base set S(Q) as
      root nodes, traversing edges in right direction;
      /*Flow Adjustment Stage */
3)    For each edge v_i->v_j in G_v^Q,compute Flow_0(v_i->v_j) using
      Equation 10;
4)    For each node v_k in G_v^Q set h(v_k)=1;
5)    While not converged do
            For each node v_k in G_v^Q except v do
                  Compute h(v_k) using Equation 15;
6)                Update the Flow of each edge in G_v^Q using
                  Equation 12;
7)Return G_v^Q ;
```

Figure 21: Algorithm to Compute Flows in Explaining Subgraph.

$$\mathrm{h}(v_k)=\frac{r^{Q'}(v_k)}{r^{Q}(v_k)} \qquad (14)$$

Combining Equations 10, 11b, 12, 13 and 14, we get the following fixpoint equation for the computation of $h(v_k)$. (For the intermediate steps and more details see [VHR07].)

$$\mathrm{h}(v_k)=\frac{\sum_{(v_k,v_j)\in G_v^Q}\left(h(v_j)\cdot Flow_0(v_k,v_j)\right)}{d\cdot r^{Q}(v_k)}$$

We rewrite this equation using Equation 10:

$$\mathrm{h}(v_k)=\frac{\sum_{(v_k,v_j)\in G_v^Q}\left(h(v_j)\cdot d\cdot \alpha(v_k\rightarrow v_j).r^{Q}(v_k)\right)}{d*r^{Q}(v_k)}$$

which then becomes

$$h(v_k) = \frac{d \cdot r^Q(v_k) \cdot \sum_{(v_k, v_j) \in G_v^Q} \left(h(v_j) \cdot \alpha(v_k \rightarrow v_j) \right)}{d \cdot r^Q(v_k)}$$

and finally,

$$h(v_k) = \sum_{(v_k, v_j) \in G_v^Q} \left(h(v_j) \cdot \alpha(v_k \rightarrow v_j) \right) \qquad (15)$$

Observation 2: *The "original" ObjectRank2 scores are not used in computing the reduction factor $h(v_k)$.* □

The iterative computation of Equation 10 on the explaining subgraph converges [VHR07].

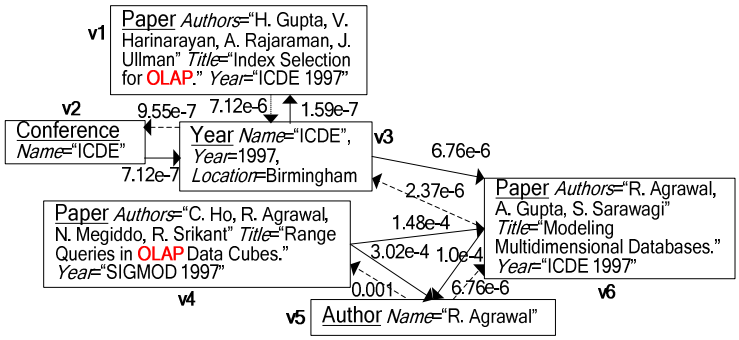


Figure 22: Explaining Subgraph for Range Queries in OLAP paper in Figure 9.

Example. *Figure 22 shows the explaining subgraph for Q=["OLAP"] and target object v4 after 5 iterations of Equation 15. Note that the "Data Cube" paper* (*see Figure 9*) *is not in* G_v^Q, *since there is no path from that paper to v4. Notice that the incoming flows of the target object v4 are the same as the original ones of Figure 19. The computed reduction factors after 5 iterations are as follows: h(v1)=1.59e-4, h(v2)=4.77e-4, h(v3)=0.0011, h(v4)=1.0, h(v5)=0.1006 and h(v6)=0.0067. Note that h(v4) is 1 as v4 is*

the target object which implies that its incoming flow from v5 is not adjusted as shown in Figure 14. □

The explaining subgraph G_v^Q can be very large which would make its generation slow and its display to the user, impossible. Hence, in practice we limit the radius of G_v^Q to *L* (longer paths are generally unintuitive [CQ69] and carry less authority) and only keep the paths with high authority flow. We apply these techniques in our online demo. We have found that a relatively small *L* (e.g., *L=3*) value is adequate to effectively explain a result and produce useful reformulations. Figure 21 presents the Flow adjustment algorithm.

Theorem 1: *Iteratively computing Equation 15 on the explaining subgraph converges.*

Proof: The fixpoint computation of Equation 15 is equivalent to the PageRank computation, if we replace incoming by outgoing edges and remove the damping factor. The PageRank computation has been shown to converge if the graph is aperiodic and irreducible [MR95]. The former is generally satisfied, whereas the latter is satisfied for connected graphs. The explaining subgraph is connected due to its construction method – all nodes are connected to the target node. To guarantee convergence, we always consider a non-zero reverse direction edge type for every edge type. Furthermore, there are no flow sinks [BP98] since there is a path from every node to the target node. □

5.2.2 Query Reformulation

Query reformulation [VHR08] using relevance feedback has been well studied in traditional IR [SB90, RL03, Efth93, BSA+95, Har88], where query expansion has been

the dominant strategy. That is, keywords are added to the original query according to the user's feedback. Such techniques are not adequate for ObjectRank2, since they ignore the link-structure of the graph which plays a key role in the ranking. For instance, if the user selects the "Range Queries in OLAP" paper in Figure 9 as a relevant object, what is the best way to reformulate the query using this paper (referred as *feedback object*)? The explaining subgraph described in previous Section is a key structure for query reformulation since *a "vote" of the user for feedback object v can be viewed as "vote" of the user for the explaining subgraph* G_v^Q *of v*.

Overview of process: First, the system computes the top-*k* objects with the highest ObjectRank2 values. The user marks a result object *v* (we extend to multiple objects in [VHR07]) as relevant − user's click-through could be used to implicitly derive such markings. Then the explaining subgraph G_v^Q of *v* is computed. Based on the content and link-structure of G_v^Q we reformulate the initial query. In particular, the *Content-based* component of the reformulation is inspired by traditional query expansion ideas and leads to a query expansion; whereas the *Structure-based* component adjusts the authority transfer rates of the authority transfer schema graph based on the edge types in G_v^Q. The two reformulation components can be combined.

Content-based Reformulation

According to traditional reformulation techniques, the terms in the feedback object *v* (viewed as a document) should be added, appropriately weighted, to the original query. However, due to the nature of authority flow ranking, we extend this idea to also include

terms in the objects that transfer high authority to *v*. These objects are the nodes of the explaining graph G_v^Q. The weight of an expansion term *t* is proportional to the flow that the nodes that contain *t* pass to *v*, that is, the outgoing flow of these nodes in G_v^Q.

A term *t* is weighted according to its distance from *v* and the amount of authority it transfers to *v*, as shown in Equation 16. The authority flow a node transfers to *v* is its outgoing flow in the explaining graph G_v^Q.

$$w^f(t) = \sum_{v_k \in G_v^Q \wedge t \in v_k} \left((C_d)^{D(v_k,v)} \cdot \sum_{(v_k,v_j) \in G_v^Q} Flow(v_k \rightarrow v_j) \right) \quad (16)$$

where $0 \leq C_d \leq 1$ is the *decay factor* (in the spirit of XRANK [GSB+03]) which is typically set to 0.5, and $D(v_k,v)$ is the distance (length in number of edges) of v_k from *v*. Note that if v_k is *v*, then we use $_d \cdot \sum_{(v_j,v_k) \in G_v^Q} Flow(v_j \rightarrow v_k)$ instead of $\sum_{(v_k,v_j) \in G_v^Q} Flow(v_k \rightarrow v_j)$, since the outgoing flow of *v* is not specified in G_v^Q. We select the top-*s* terms *Z* with highest weight (ignoring stop words) and add them, after normalizing them as explained below, to the original query vector $\boldsymbol{Q}_0$. The reformulated query vector $\boldsymbol{Q}_i$ at iteration *i* is defined as

$$\mathrm{Q}_i = \begin{cases} \mathrm{Q}_{i-1} + C_e \cdot \sum_{t \in Z} w^f(t) \cdot \mathrm{t}, & i > 1 \\ \mathrm{Q}_0, & i = 0 \end{cases} \quad (17)$$

where **t** is the vector of term *t* (as in the vector space model [Sin01]), and $0 \leq C_e \leq 1$ is the *expansion factor*, typically 0.5, used to scale the weights of new terms (as well as new weights of old terms) with respect to the terms present in current query vector. Normalization issues are discussed in [VHR07].

Example. *Consider the authority transfer data graph of Figure 9, query Q=["OLAP"], and feedback object, v is the "Range Queries in OLAP" paper. The explaining subgraph* G_v^Q (*Figure 14*) *is created. Using Equation 16, and assuming* C_d *and* C_e *are 0.5, the top-5 new terms are olap*(*1.0*), *cubes*(*0.99*), *range*(*0.99*), *multidimensional*(*0.05*) *and modeling*(*0.05*). *Note that the terms in the feedback object* (*target object of* G_v^Q) *generally get a higher weight due to the decay factor* C_d. *The reformulated query vector* ***Q*** *computed by Equation 17 is* [*olap, cubes, range, multidimensional, modeling*] = [*2.0, 0.99, 0.99, 0.05, 0.05*].□

Structure-based Reformulation

The structure-based reformulation adjusts the authority transfer rates based on the explaining subgraph G_v^Q. Intuitively, if edges of an edge type e_G carry large authority in G_v^Q then the user probably believes e_G is an important edge type for the query. We boost the authority transfer rate of each edge type present in G_v^Q according to the authority it transfers (to the feedback object *v*). The reformulated authority transfer rate $\alpha'(e_G)$ of edge type e_G is computed by,

$$\alpha'(e_G) = \left(1 + C_f \cdot \sum_{(v_k, v_j) \in G_v^Q \wedge (v_k, v_j) \text{ has type } e_G} Flow(v_k \rightarrow v_j)\right) \cdot \alpha(e_G) \quad (18)$$

where $0 \leq C_f \leq 1$ is the *authority transfer rate adjustment factor*, typically set to 0.5, used to scale the authority transfer rates with respect to their previous values, $\alpha(e_G)$ is the

previous authority flow rate of edge type e_G. Normalization issues are discussed in [VHR07].

Example. *The authority transfer rates of the original query are [PP,PP',PA,AP,CY,YC,YP,PY] = [0.7,0.0,0.2,0.2,0.3,0.3,0.3,0.1]. Using Equation 18 and the normalization process, the reformulated authority transfer rates are [0.67,0.0,0.24,0.16,0.24,0.24,0.24,0.08]. Notice that the transfer rates of PA and AP edge types are increased and decreased respectively as they carry greater and lesser authority to the feedback object respectively.* □

5.2.3 Experimental Results

We experimentally evaluate our algorithms in terms of quality and performance. This section is organized as follows: First we briefly describe the datasets used for evaluation and then present the user surveys and the performance experiments respectively.

Datasets: We use two real datasets (Table 8). DBLPcomplete and DBLPtop are the complete DBLP dataset and a databases-related subset respectively. We shredded the downloaded DBLP file into the relational schema.

Table 8: Real and Synthetic Datasets.

Name	*#nodes*	*#edges*	*Size*(*MB*)
DBLPcomplete	876,110	4,166,626	3950
DBLPtop	22,653	166,960	136

User Surveys

We used DBLPtop for our user surveys and not DBLPcomplete since on-the-fly ObjectRank2 executions on the latter are slow and survey subjects would be irritated. The

first phase was conducted at Florida International University (FIU) involving five professors and PhD students from the database lab, who were not involved with the project. The goal of this survey was to compare content-based, structure-based, and content & structure-based reformulations. The result was that structure-based reformulation is superior. The second phase focused on structure-based reformulation and involved 10 FIU and outside (including IBM TJ Watson and Almaden) database researchers, not involved in the project. In both phases we also measure the capability of our system to discover the authority transfer rates set by a domain expert.

Internal Survey. The residual collection method [RL03, SB90] can be summarized as follows: All objects seen by the user or marked as relevant are removed from the collection and both the initial and all reformulated queries are evaluated using the residual collection. We use the average precision as the evaluation measure. Note that the recall is the same as the precision in our case since we limit the output results to *k*. We report the survey results for 4 relevance feedback iterations and for the following 3 settings: i) Content-Only reformulation (C_f=0&C_e=0.2), ii) Content & Structure-based reformulation (C_f =0.5& C_e =0.2) and iii) Structure-Only reformulation (C_f =0.5& C_e =0). (We have found that these values of C_f and C_e are appropriate for this dataset.) The decay factor C_d is set to 0.5. We use $L=3$ to limit the size of the explaining subgraph as explained. We initialize the authority transfer rates of each edge type to 0.3. Figure 23 shows the survey results. We see that the structure-only reformulation performs the best. Content-based reformulation is not effective in our setting because the users are domain experts and hence know the right keywords, i.e., traditional query expansion is not

effective. Note that in a different domain the results could vary. Next we evaluate the effectiveness of structure-based reformulation to automatically train the authority transfer rates of the DBLP authority transfer schema graph and compare the learned weights to the ones of [BHP04], which we view as ground truth. The rates there

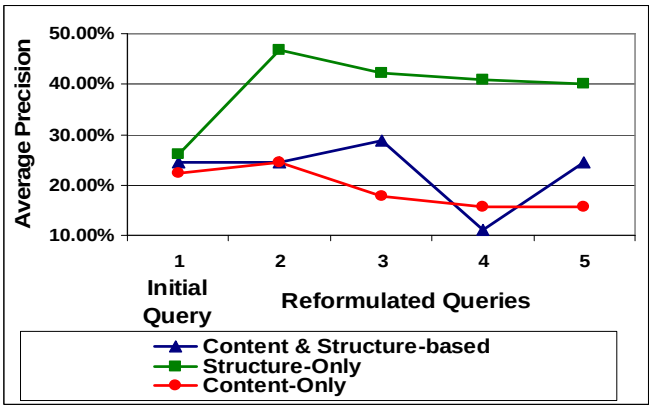


Figure 23: Average Precision for different calibration parameters.

were assigned manually by domain experts in a trial and error manner. We start by setting the transfer rates of all edge types to 0.3. We again limit the length of paths of the explaining graph with $L=3$. Let UserVector[PP,PP′,PA,AP,CY,YC,YP,PY] be the authority rates vector. It is initialized to [0.3,0.3,0.3,0.3,0.3,0.3,0.3,0.3]. The ground truth ObjVector is [0.7,0.0,0.2,0.2,0.3,0.3,0.3,0.1]. At each iteration we compute the current UserVector produced by the reformulation and compute the cosine similarity cos(ObjVector,UserVector). Figure 24 shows the cosine similarity training curves for 4 users averaged over 5 queries each for a different value of C_f (C_e is always 0). We see that the cosine similarity initially increases with the number of iterations and then decreases due to overfitting. Larger C_f values lead to faster peak, since the adjustment of the rates is less smooth.

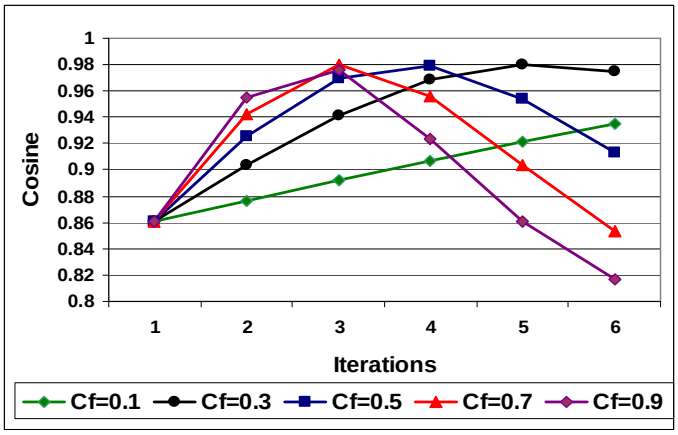


Figure 24 : Training of the Authority Transfer Rates.

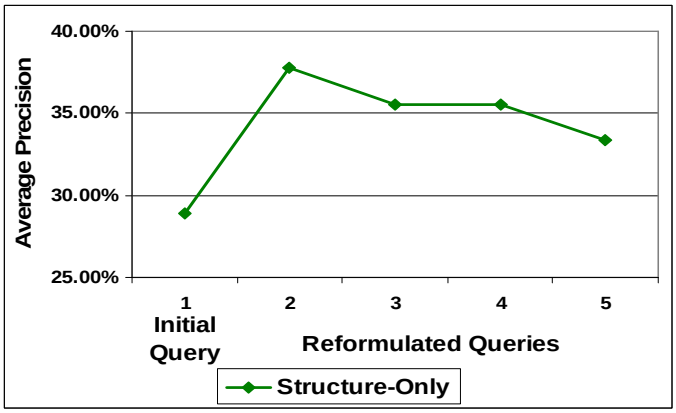


Figure 25: Average Precision using structure-only reformulation with Cf=0.5.

ObjectRank2 vs. ObjectRank: We also conducted a survey comparing the quality of ObjectRank2 with ObjectRank [BHP04]. We found that ObjectRank2 is only slightly better by 3%. The reason is the ObjectRank also uses something equivalent to the idf of our IR function: they weigh the ObjectRank values for multi-keyword queries according to the size of the base set. However, we believe that ObjectRank2 will be superior in datasets with longer text descriptions.

External Survey. We conducted an external survey operating on DBLPtop using only structure-based reformulation as it was found to be the best, in the internal survey. Figure 25 shows the average precision curve for 5 iterations averaged over 20 queries by 10 users (2 queries per user). Figure 26 shows the authority transfer rate training curves for the external survey which are similar to those in the internal survey.

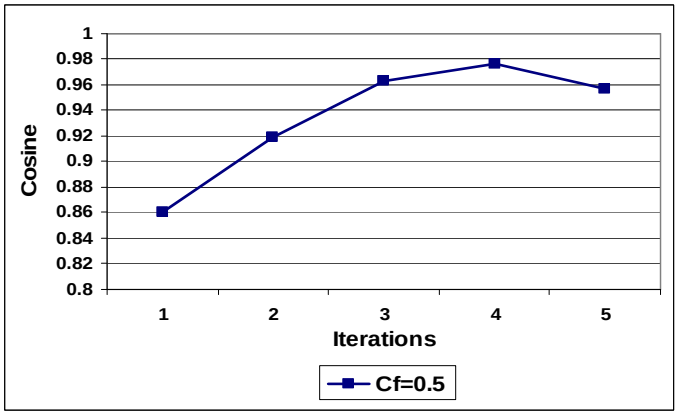
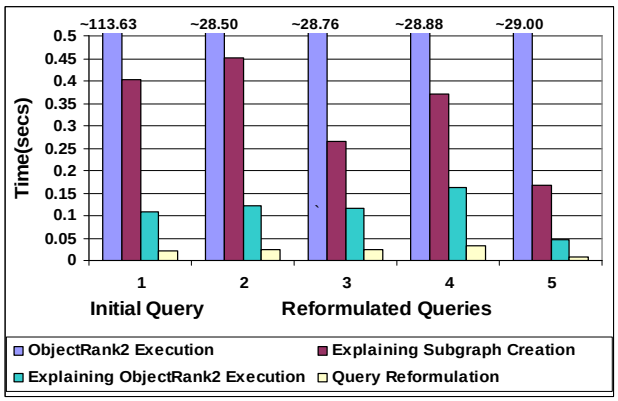


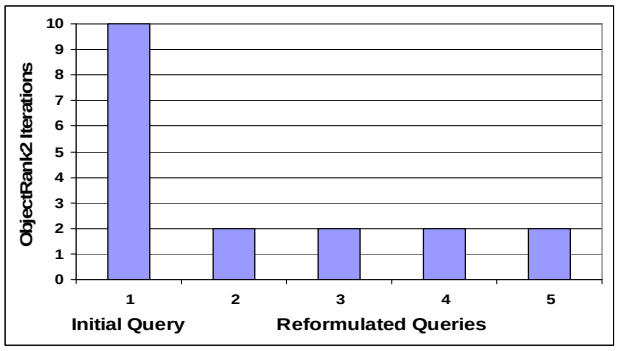
Figure 26: Training of the Authority Transfer Rates.

Performance Experiments: To evaluate the performance of our algorithms, we conducted experiments on DBLPcomplete. We used a linux machine with Power 4+ 1.7GHz processor and 20GB of RAM. The total execution time is measured for various stages: (a) computing the top-*k* objects for the initial or reformulated query, (b) creating the explaining subgraph, (c) executing the explaining ObjectRank2 on the explaining subgraph, and (d) creating the reformulated query. As in [BHP04], for the initial user query, we initialize every node in D^A with their global ObjectRank values, to achieve faster convergence. Then, for the first reformulated query we use the ObjectRank values of the initial query and so on. The intuition is that the ObjectRank values of the newly reformulated query are expected to be close to the ones obtained by the previous query.

Figure 27(a) shows the execution times for the various components of the process: execute the query (first bar), and create the reformulated query (last three bars) at each user feedback and reformulation iteration. We use $L=3$ as the radius of the explaining subgraph, and convergence threshold 0.0001. Figure 27(b) shows the number of ObjectRank2 iterations for the initial and the reformulated queries over the whole graph. Clearly, using the previous scores as initial values accelerates the convergence of ObjectRank2.



(a): Query and Reformulation Times.



(b): ObjectRank2 iterations.

Figure 27: DBLPcomplete Execution.

The ObjectRank2 execution times for DBLPcomplete is clearly too long for exploratory searching. This can be addressed in one of the following ways: use faster hardware, precompute ObjectRank2 values as in [BHP04], or define focused subsets like DBLPtop. The ObjectRank2 execution times for these datasets are about 2 seconds for the initial query and less than 1 sec for the subsequent reformulated queries (graphs omitted due to space constraints).

5.3 Graph Information Discovery (GID)

There has been an explosion of hyperlinked data in many domains, e.g., the biological Web. Expressive query languages and effective ranking techniques are required to convert this data into browsable knowledge. We propose the Graph Information Discovery (GID) framework [VHR+09] to support sophisticated user queries on a rich web of annotated and hyperlinked data entries, where query answers need to be ranked in terms of some customized ranking criteria, e.g., PageRank or ObjectRank. GID has a data model that includes a schema graph and a data graph, and an intuitive query interface. The GID framework allows users to easily formulate queries consisting of sequences of hard filters (selection predicates) and soft filters (ranking criteria); it can also be combined with other specialized graph query languages to enhance their ranking capabilities. GID queries have a well-defined semantics and are implemented by a set of physical operators, each of which produces a ranked result graph. We discuss rewriting opportunities to provide an efficient evaluation of GID queries. Soft filters are a key feature of GID and they are implemented using authority flow ranking techniques; these are query dependent rankings and are expensive to compute at runtime. We present

approximate optimization techniques for GID soft filter queries based on the properties of random walks, and using novel path-length-bound and graph-sampling approximation techniques. We experimentally validate our optimization techniques on large biological and bibliographic datasets. Our techniques can produce high quality (Top K) answers with a savings of up to an order of magnitude, in comparison to the evaluation time for the exact solution.

Consider a rich web of annotated data entries (objects) in Internet accessible sources with hyperlinks to entries in other sources. Examples include the biological Web, GIS datasets and their metadata, bibliographic data sources, healthcare data, desktop files and Intranets. Such graphs have significant differences from the general Web graph. Each of the data entries or documents contains some specific typed knowledge, e.g., information on genes and proteins for the biological Web. Thus, this graph has an underlying schema graph. Users of such *typed webs* want answers to queries that are meaningful to them and go beyond traditional Information Retrieval (IR) keyword queries. These users have sophisticated information needs, which require both customization and personalization, when ranking query results. For example, a biologist may only want to retrieve protein data entries from SwissProt, or she may be interested in discovering the associations between a particular drug and a disease by following the links among publications that are linked to proteins and vice versa.

The challenges to query answering in this rich web of entities include supporting users to retrieve meaningful answers, given the user's preferences, rather than just

retrieving relevant data entries. The Graph Information Discovery (GID) framework must support a simple yet flexible query interface where a user can easily pose a complex query. Ranking of answers must reflect the semantics of this rich Web and the user's personal perspective. GID queries must be interactive and support the exploratory discovery process. Hence, they must support formal semantics so that queries can be optimized and evaluated efficiently.

The limitations of many prior solutions are that they typically converge on the extremes of query complexity, i.e., plain keyword or complex queries, with few solutions in between, or they fail to consider ranking. Web search [PBMW98, Hav02, FLW+06, NDQ06, RPB06] employs excellent ranking techniques but have limited search capability. The keyword search paradigm of Web search has also been adapted to structured databases [ACD02, HP02, BNH+02]. On the other hand, there are a variety of extensions of SQL for Web graphs (WebSQL [MMM97], W3QL [KS95], WebOQL [AA98], StruQL[FFL+97, FLM98]) and RDF graphs (SPARQL [SPQL]). However, none of these languages provide customized ranking techniques. The approach in [RG03] is an excellent start towards incorporating ranking in structured Web queries. They provide an underlying algebra and optimization; however, they do not support an interface that allows users (scientists in the case of the scientific Web) to intuitively write useful complex queries, nor do they support powerful ranking techniques like authority flow based ranking. NAGA [KSI+08] implements reasoning tasks on RDFS documents, and supports complex queries and ranking. NAGA targets typed graphs of facts and labeled relationships that may be expensive to create and keep up-to-date. It does not support

query-customized ranking. That is, a fixed confidence-based ranking function is applied to the final results. In contrast, GID allows the user to specify what ranking. mechanism (if any) should be used for each leg of the query. Furthermore, NAGA uses expensive reasoning algorithms, which may not scale to very large datasets like PubMed, whereas GID relies on a suite of scalable approximation and optimization techniques. We show that our framework can complement such prior research and extend it with support for sophisticated queries and ranking.

This section addresses the challenges of expressing and answering sophisticated user queries on typed graphs. We focus on a web of annotated data entries from biological data sources for our running examples and experiments. However, the generic GID framework is applicable in multiple domains; we use bibliographic data as a second evaluation domain in our experiments. The GID framework has the following features and capabilities:

- Given a typed graph, GID provides a user interface to specify a combination of hard and soft filters; the latter incorporate ranking in an intuitive manner. GID emulates domain graph query languages such as lgOR, *lgPR* [RWL+06] and filter queries in PubMed [PM07]. GID can be combined with more general graph languages to support complex queries.
- Filters are implemented by an underlying closed algebra of physical operators. Each operator produces a ranked graph and GID operators can be combined. The properties of the operators are used to determine the relevant query rewriting rules.

- GID soft filters are implemented using authority flow based ranking; they are query dependent and must be computed at runtime. Two novel approximation techniques are studied in order to achieve interactive query response times. One is a path-length-bound technique, where only paths of limited length are considered. The second is a graph-sampling approximation technique, where sampling over a Bayesian network is used to create sampled graphs and estimate the ranking scores.
- GID queries were evaluated on biological and bibliographic datasets. We show that our approximation methods achieve execution time reductions of up to an order of magnitude, with negligible degradation of the Top-*k* answer's quality (in comparison to the exact ranking). This allows GID to support an exploratory framework.

5.3.1 GID Query Language

The intuition of the GID framework is the application of a sequence of hard and soft filters. A filter generally takes as input a ranked graph and outputs a ranked subgraph of the input graph. A hard filter is used to eliminate some nodes in a Boolean manner whereas a soft filter provides ranking.

GID Query Syntax: Given a data graph *DG* and a schema graph *SG*, a query q is a sequence $q=[r_1>\ldots>r_m]$ of filters r_i. We use the ">" symbol to denote a total order between the filters and this represents a pipelining of the output of one filter as input to the next. The results of a query, which are usually (see exception below) the nodes of the graph output by the last filter, are referred to as *target objects*.

A query may also specify the number *k* of the requested top-*k* results. A filter *r*={*R,N,S*} is the following 3-tuple:

(1) The selection condition *R* as follows:

- A keywords Boolean (OR, AND, NOT) expression *E*, e.g., *Keywords = "cancer" AND "breast"*.
- An attribute value pair *av*, e.g., *title* = "*A comparative…*"
- A type *T*, e.g., *Type* = {*EntrezGene*}.
- A Path expression *P*, e.g., *Path = EntrezGene* /*PubMed* or *Path = EntrezGene* [*Keywords = "tnf"*] / *PubMed* [*author="Michael*"].

(2) A Boolean *N;* the value=true means that *r* is negated.

(3) A Boolean *S;* a value=true means that *r* is soft.

GID does not support soft filters (*S=true*), where *R* is a path expression, or negated soft filters (*N=true* and *S=true)* since the semantics are unintuitive. Path expression *P* may contain types, unidirectional single step navigational operators (/), multi-step navigational operators (//), and type wildcards (*). Notice that "*Path*", "*Keywords*" and "*Type*" are reserved words in GID. GID does not support a combination of selection conditions (keyword expression, attribute value pair, type or path expression) within a single filter, in order to simplify the implementation and optimization process.

***Example*:** *A biologist's exploration is as follows: Starting from genes in Entrez Gene she follows links to Entrez Protein and then to PubMed; her target objects are a set of papers*

in PubMed. She wants to rank these papers by their importance/relevance to the word "human". The following expresses her needs:

q_1 = [{*Path = EntrezGene/EntrezProtein/PubMed, false, false*}

> {*Keywords="human", false, true*}

> {*Type = PubMed, false, false*}].

The first hard filter creates a subgraph of paths from genes in Entrez to proteins to PubMed publications. The second, soft filter provides a "goodness" ranking (*to be discussed below*) *with respect to the keyword "human", and the last, hard filter identifies the "target objects" - publications from PubMed – in the result.* □

The most simple and intuitive GID query for novice users is to specify a set of hard filters $\{r_1,\ldots,r_t\}$ and a single soft filter r_s. This can have a default interpretation of $q = \{r_1,\ldots,r_t\} > r_s$ or as $q = r_s > \{r_1,\ldots,r_t\}$ depending on the application semantics. The specific ordering of the hard filters $\{r_1,\ldots,r_t\}$ is not important as long as they do not include Path filters.

Target Objects: As mentioned above, we assume by default that all the objects of the resulting subgraph of the query are output to the user. Alternatively, the $ sign is used to select a more fine-grained group of target objects. For instance, q_2 = [{*Path* = $*EntrezGene*$*/EntrezProtein, false, false*}] returns all EntrezGene objects that point to an EntrezProtein object.

GID Query Semantics: To define the semantics of GID queries, we first define a *score assignment function, Score* for a data graph $DG(V_D,E_D)$ to be a mapping of nodes $v \in V_D$ to real values $Score(v)$ in [0,1]. A *unit score assignment, $Score_{unit}$,* assigns $Score_{unit}(v)=1$ to every $v \in V_D$. The input of a filter r is a pair $(G_{in},Score_{in})$ of a data graph G_{in} and a

scores assignment $Score_{in}$ for G_{in}. Similarly, the output is a pair (G_{out},$Score_{out}$), where G_{out} is a subgraph of G_{in}. Applying the filter is as follows: $r(G_{in},Score_{in})=(G_{out},Score_{out})$. Given a GID query $q=[r_1>r_2>\ldots>r_{m-1}>r_m]$ on the data graph $DG=(V_D,E_D)$ the result $(G_R,Score_R)$ of q is $r_m(r_{m-1}(\ldots(r_2(r_1(DG, Score_{unit}))\ldots))$.

During query evaluation, filters are applied in the order indicated in the query. Note that the unit score assignment is used for the first filter r_1. Alternative initial scores are possible, e.g., the global score of a node computed by a method like PageRank [PBMW98]. Each filter may change the scores of the data graph. This may also eliminate nodes and edges as explained next. Applying filter r on graph DG is as follows:

- Each v in DG is assigned a score $Score(v)$ in [0.0,1.0].
- When node v is assigned $Score(v)=0$, then the node and its incident edges are removed. For example, applying r = {*Keywords="human", false, false*} removes all nodes and incident edges in graph G_{in} that do not contain the keyword "human" to create G_{out} .

Given the result $(DG_R,Score_R)$ of q, where $DG_R=(V_R,E_R)$, GID will display a list of the nodes v of V_R ranked by decreasing $Score_R(v)$ values.

Hard filters are used to eliminate nodes (and their incident edges) of G_{in}. The filter is evaluated as a Boolean and may assign score 0 to some nodes. The score is unchanged for the rest of the nodes. Consider the following filter r={*R,false,false*}:

1. If R is a keyword expression E (or simply a keyword), $Score_{out}(v)=0$ if v does not satisfy E, else $Score_{out}(v) = Score_{in}(v)$.

2. If R is a attribute value pair av, then $Score_{out}(v)=0$ if node v does not satisfy av, else $Score_{out}(v) = Score_{in}(v)$.
3. If R is a type T, then $Score_{out}(v)=0$ if v is not of type T, else $Score_{out}(v)=Score_{in}(v)$.
4. If R is a path P, then $Score_{out}(v)=0$ for nodes not contained in a path of type P, else $Score_{out}(v)=Score_{in}(v)$.

The opposite scores are assigned if $r=\{R,true,false\}$.

Soft filters rank a result subgraph and are inherently fuzzy. Suppose R is a keyword w or keyword expression E, then, applying r results in the following score:

$Score_{out}(v)=f(Score_{in}(v),Score_r(v))$ where $0 \leq Score_r(v) \leq 1$ is the score assigned to v by r. $Score_r(v)$ shows how "good" v is, given the graph G_{in}. GID does not specify the exact semantics or computation of these scores $Score_r(v)$ for soft filters. Various approaches are possible including authority flow, IR scoring [Sin01], path count [Katz53], keyword proximity [GSVG98, HPB03], minimum distance from the keyword nodes and so on.

Note that $Score_r(v)$ must be positive (non-zero) and must not depend on the input score assignment $Score_{in}(v)$. This important assumption, the *non-pruning order-free assumption* for soft filters, is needed to obtain useful rewriting axioms. This assumption is reasonable to implement since a small epsilon value can be assigned to nodes instead of 0 if they are completely irrelevant to R. We use a *combining function f* (e.g., product or min). In principle, any combining function may be used. However, a monotone function is usually more intuitive and also allows pipelining and fast computation of the top results [FLN01]. In order to maintain the $Score(v)$ in [0.0,1.0], we normalize the $Score(v)$ after application of each filter.

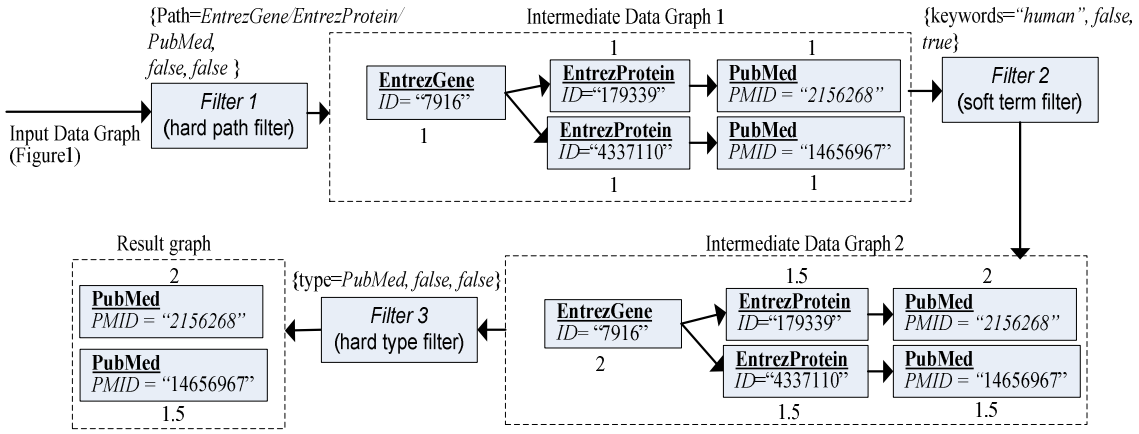


Figure 28: Sample semantic query evaluation.

***Example* (*cont'd*):** *Figure 28 shows the query evaluation of query q_1 given the input data graph DG of Figure 6. We assume initial unit scores assignment $Score_{unit}$. We also assume a simple soft filter scoring function with $Score_r(v)=0.5$ if a node does not contains the term and $Score_r(v)=1$ otherwise. The combining function f is summation.* □

5.3.2 Related Research

Meeting target user needs: We interviewed biomedical domain experts and examined popular search tools. When asked to describe the selection of target objects (results) that are documents in PubMed, these users chose progressive filtering of the objects; see PubMed filter queries [PM07]. They also requested simple navigational paths. PubMed supports filters in a limited manner; users can select a set of predefined filters (hard filters in our terminology), e.g., filter the publications that cite MEDLINEplus articles. In [VHR08], we conducted user experiments that show the benefits of soft filters for this domain. We note that the real test of the GID framework will be a friendly graphical user interface and user evaluation studies; this is included in our future work.

A second aspect of user needs is the richness of the data model. The GID model is much simpler compared to RDF, yet it can capture much of the knowledge used by a scientist in the process of literature based discovery (LBD) on the Web. NAGA [KSI+08] has a similar labeled directed multi-graph data model. However, they may have significant overhead in determining the confidence of facts and relationships of the RDFS graph.

A third aspect of user needs is personalized ranking. NAGA does not support query-customized ranking. That is, a fixed ranking function is applied to the final results, based on confidence-based edge weights that reflect the estimated accuracy of the extraction process and trust in the source. In contrast, GID allows the user to specify what ranking mechanism (if any) should be used for each leg of the query. GID supports authority flow based ranking and the authority weights can be personalized. This is well suited to scientists whose value for specific domain knowledge may vary depending on the task.

Expressive power: GID is clearly more powerful than the current PubMed language which only supports hard filters and has no ranking capability. Research by Raghavan and Garcia-Molina [RG03] studies an expressive graph algebra and query operators. The GID language can support the “linear” plans of this algebra. The “tree” plans were not considered since they cannot be supported by a simple user language. While users wanted navigation, they did not express a need for general join operations, recursion, etc. as found in [RG03]. GID soft filters are more general than the ranking operators in [RG03]. GID soft filters are evaluated against the whole input subgraph (e.g., ObjectRank) instead

of just relying on the properties of each individual node as is done in [RG03]. This property is the key to intuitive GID user query interface.

Example: This example shows that the GID query language allows expressing complex queries in an intuitive way; no query language was proposed in [RG03]. Consider the following sample query from [RG03]: “Generate a list of universities with whom Stanford researchers working on ‘Mobile networking’ collaborate”. A sequence of instructions corresponding to this query is presented in [RG03]: *Let S be a weighted set consisting of all the pages in the stanford.edu domain that contain the phrase ’Mobile networking’. The weight of a page in S is equal to the normalized sum of its PageRank and text search ranks. Compute R, the set of all the “.edu” domains* (*except stanford. edu*) *to which pages in S point. For each domain in R, assign a weight equal to the sum of the weights of all the pages in S that point to that domain. List the top-10 domains in R in descending order of their weights* [RG03]. Creating the algebraic execution plan for this query (Figure 8 of [RG03]) requires significant training.

In contrast, the hard and soft filters of GID can express this query in the following sequential and straightforward manner: [{*Keywords*="",*false*,*true*}>{*IRFilter*("*Mobile Networking*"), *false, true*} > {*Path*=*Webpage*[*URL*="*stanford.edu*" AND *Keywords* = "*Mobile networking*"]/$*Webpage*[*URL*=".*edu*" AND *URL* ≠ "*stanford.edu*"]$, *false, false*}> {*URL*="*stanford.edu*", *false, true*}].

For this query, we first initialize the graph nodes with global PageRank scores (empty keywords expression in first soft filter). For computing the textrank (IRscores), we need to introduce the IR soft filter. The *combining function*, *f* is summation that adds textranks and pageranks. Notice that the last filter is a soft filter that computes the final scores for each web page and outputs the non-Stanford.edu *pages* in descending score order. We assume that this attribute-constrained soft filter uses the scores of the nodes in the input graph as the weights in the base set for the authority flow execution algorithm.

There has been significant work on query languages for the Web and search engines ranging from keywords based languages to query languages for semi-structured data, to graph query languages. For users who require general query language features to write complex queries, the GID operators and ranking semantics can be incorporated in a straightforward manner into a language such as SPARQL. Alternatively, more complex path expressions or other relational operators can be incorporated into the GID language. NAGA too can express complex queries and can support a powerful inference mechanism; however, this may not scale well to large graphs and an interactive discovery process.

5.3.3 Algebra for GID

We present a closed algebra where the algebraic operators have a one-to-one correspondence to the filters. A binary *Combine* operator is introduced to combine scores. Each (unary) operator, with the exception of *Combine*, accepts as input a pair of data graph and score assignment (*DG*, *Score*) and produces the pair (*DG'*, *Score'*), where $DG=(V_D,E_D)$ and $DG'=(V_D',E_D')$. Further, $V_D' \subseteq V_D$ and $E_D' \subseteq E_D$.

Operators

1. *HardExp*(*DG*,*Score*,*E*) $\rightarrow$ (*DG′*,*Score′*) where *E* is a Boolean expression over keywords, such that, $V_D' = \{v \mid v \in V_D \text{ and } satisfy(v,E)\}$, $E_D' = \{e=(u,v) \mid e \in E_D \text{ and } u,v \in V_D'\}$ and the Boolean predicate *satisfy*(.,.) is defined by induction over *E* as follows:
   - If *E* is a term, *satisfy*(*v*,*E*)=*true* if *v* contains the term *E*, *false* otherwise.
   - If *E*=*E1 Op E2*, *satisfy*(*v*,*E*)=*satisfy*(*v*,*E1*) *Op satisfy*(*v*,*E2*).
   - If *E* = not (*E1*), *satisfy*(*v*,*E*)= *not*(*satisfy*(*v*,*E1*)). The score of each node $v \in V_D'$ remains the same, i.e., *Score′*(*v*)=*Score*(*v*).
2. *HardAttribute*(*DG*,*Score*,*av*) $\rightarrow$ (*DG′*,*Score′*) where *av* is an attribute value pair, such that, $V_D' = \{v \mid v \in V_D \text{ and } satisfy(v,av)\}$, $E_D' = \{e=(u,v) \mid e \in E_D \text{ and } u,v \in V_D'\}$ and the Boolean predicate *satisfy*(*v*,*E*)=*true* if *v* contains the corresponding value for the attribute specified, *false* otherwise. Notice that we overload the satisfy predicate.
3. *HardType*(*DG*,*Score*,*T*)$\rightarrow$(*DG′*,*Score′*) where *T* is a set of types (nodes of the schema graph), $V_D' = \{v \mid v \in V_D \text{ and } \exists t \in T \text{ and } v \in t\}$, $E_D' = \{e=(u,v) \mid e \in E_D \text{ and } u,v \in V_D'\}$. The score of each node $v \in V_D'$ remains the same, i.e., *Score′*(*v*)=*Score*(*v*).
4. *HardPath*(*DG*,*Score*,*P*)$\rightarrow$(*DG′*,*Score′*) where *P* is a path expression, $V_D' = \{v \mid v \in V_D \text{ and } satisfyPath(v,P)\}$, $E_D' = \{e=(u,v) \mid e \in E_D \text{ and } u,v \in V_D'\}$, the Boolean predicate *satisfyPath*(*v*,*P*) is *true* if *v* is part of a path *p* that satisfies *P*; *false*

otherwise. The score of each node $v \in V_D'$ remains the same, i.e., $Score'(v)=Score(v)$.

5. *SoftExp*(*DG, Score, E, ScoreFunction*) → (*DG', Score'*) where *E* is a Boolean expression over keywords, and *ScoreFunction* is a function such that, given *E* and *DG*, maps each node *v* to a score *ScoreFunction*(*DG,E,v*) in [0.0,1.0] ((0.0,1.0] given the non-pruning assumption for soft filters). Alternatives for *ScoreFunction* include ObjectRank, path count, MinDistance, keyword proximity and so on. The score for E is computed as follows:

- If *E=E1 OR E2*, *ScoreFunction*(*DG,E,v*) = *ScoreFunction*(*DG,E1,v*)+ *ScoreFunction*(*DG,E2,v*).
- If *E=E1 AND E2*, *ScoreFunction*(*DG,E,v*) = *ScoreFunction*(*DG,E1,v*) . *ScoreFunction*(*DG,E2,v*).
- If *E=not*(*E1*), *ScoreFunction*(*DG,E,v*) = 1 – *ScoreFunction*(*DG,E1,v*).
- If *E* is a term *w*, *ScoreFunction*(*DG,E,v*) = *ScoreFunction*(*DG,w,v*).

Once *ScoreFunction* is executed, the scores *Score'*(*v*) of the nodes in *DG* are updated as follows: *Score'*(*v*) = *ScoreFunction*(*DG,E,v*). Note that *Score'*(*v*) is the $Score_r(v)$, that is, the score assigned by the soft filter. This score will then be combined with the previous nodes scores *Score(v)* using the *Combine* operator below.

6. *Combine*(*DG1,Score1,DG2,Score2,f*) → (*DG',Score'*) where *f*(*score1,score2*) is a combining function like product. For every node in the union of *DG1* and *DG2*, *Score*(*v*) = f(*Score1*(*v*),*Score2*(*v*)). *Given* $DG_1=(V_{D1},E_{D1})$ *and* $DG_2=(V_{D2},E_{D2})$, *the*

*g*raph $DG' = (V_D' , E_D')$ *is defined as f*ollows: $V_D' = \{v \mid v \in V_{D1} \cup V_{D2}$ *and* $Score'(v) > 0.0\}$, $E_D' = \{e=(u,v) \mid e \in E_{D1} \cup E_{D2}$ *and* $u,v \in V_D'\}$.

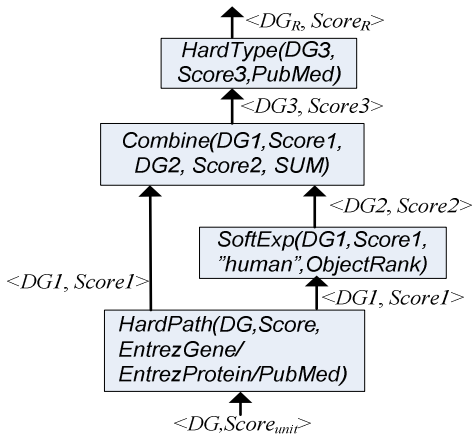


Figure 29: Execution plan for query q1

***Example* (*cont'd*):** *Figure 29 shows an execution plan for query q1. We use f(.,.)=SUM(.,.) as the combining function (other combining functions are possible as explained above) and ObjectRank as the ScoreFunction .* □

Axioms: In this section we present the rewriting rules for GID queries, assuming any implementation for the soft filters, i.e., any definition of *ScoreFunction***.** These rules will be applied together with the approximations. Consider the following theorems (without proof):

Theorem 2: *Let H_i, H_j be hard filters and S_i, S_j be soft filters. The following properties hold:*

1. The *commutative property of non-path hard filters* $H_i > H_j \Leftrightarrow H_j > H_i$.
2. The *commutative property of soft filters* $S_i > S_j \Leftrightarrow S_j > S_i$.
3. The *idempotence property of hard filters* $H_i > H_i \Leftrightarrow H_i$ □

The proof is straightforward and relies on the following: The soft filters are non-pruning and always assign a non-zero score. The combining function *f* which combines the scores of a soft filter with the current scores is commutative (e.g., product, sum, max).

Theorem 3: *The rewritings of Theorem 2 can be applied to any subsequence of a query.*□

For example, if $Q = S_1 > H_1 > H_2 > S_2$ where H_i and S_j are hard and soft filters respectively, then using the commutative property of hard filters we can rewrite Q as $S_1 > H_2 > H_1 > S_2$.

5.3.4 GID Soft Filters computed by Authority Flow

GID soft filters will typically be the most expensive operators since the popular authority-flow based ranking techniques used by most soft filters are well known to be expensive for relatively large data graphs. PageRank [PBMW98] and ObjectRank [BHP04], rely on pre-computing and indexing global or keyword-specific rankings. Given that the GID framework is meant to be interactive and exploratory, we aggressively optimize the evaluation of authority-flow soft filters. We first provide an overview of some ranking metrics. We then discuss two approximation techniques.

Layered Graph ObjectRank (lgOR): The class of GID queries with a hard path filter followed by a soft term filter is very useful and expressive. [RWL+06] proposed the lgOR ranking, a variant of ObjectRank, to answer such queries. These queries apply authority flow ranking on an acyclic directed *layered graph* produced by the hard path filter.

Example: *Consider the following GID query:* [{*Path = EntrezGene/EntrezProtein/$PubMed$, false, false* } > {*Keywords="aging" OR "cancer"*), *false, true*}]. *First, the hard filter creates a layered graph of paths satisfying the path expression EntrezGene/EntrezProtein /PubMed* (*Figure 30*). *A layered graph is a DAG comprised of layers; each layer has data entries of one or more types, which have only edges to data entries in the next layer of the graph. The data entries in the last layer, which are returned by the query, are called the target objects. For simplicity we assume that each layer is composed of data entries of one type. Next, the soft filter executes ObjectRank on the layered graph for the keyword expression "aging" OR "cancer". The target objects* (*PubMed objects*) *are ranked according to their ObjectRank value.* □

A key point of lgOR is that the authority flows between objects in the layered graph are only determined by the scores of the parents of each object in the previous layer of the graph, and the incoming authority transfer rates. lgOR is defined as follows: The ranking vector R of the target objects in the last layer of the layered graph $RG=(V_{lg},E_{lg})$ of k layers is defined by a transition matrix $A_{\lg}$ and an initial ranking vector R^{ini}:

$$R = A_{\lg}{}^{k-1} R^{ini} = (\prod_{l=1}^{k-1} A_{\lg}) R^{ini} \qquad (19)$$

The transition matrix is $A_{\lg}$, where, $\alpha_{lg}(e)$ is the authority transfer rate of edge e between nodes u and v of type U and V, respectively, in adjacent layers p and q. The

O*utDeg*(*u*,*V*), the outdegree of node *u* to nodes of the type *V*, is limited to nodes and edges in the layered graph as follows:

$$A_{\lg}[u,v] = \begin{cases} \alpha_{\lg}(e), & \text{if} \quad e = (u \to v) \in E_{\lg} \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

We present two techniques to achieve fast, high quality approximate rankings. Each of these two techniques is more effective in different settings. The *path-length-bound* technique considers paths with an upper bound on the length, in computing authority flow. The approximation is effective in evaluating a single authority-flow soft filter and can be applied to a sequence of soft filters. The *graph-sampling* technique probabilistically selects a subset of the paths using a Bayesian network. It is applied to approximating lgOR queries (introduced in [RWL+06]), which are equivalent to a hard path hard filter followed by an authority-flow soft filter. This approximation is indispensable when the data graph is large. In both techniques, the complexity of evaluating a query is reduced, by minimizing the number of nodes visited during query execution time.

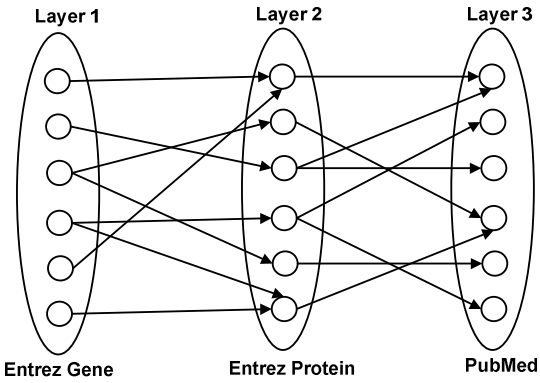


Figure 30: Layered Graph.

Approximate a Soft Filter with Path-Length-Bound Technique*:* A *path-length-bound* technique is applied to approximate the evaluation of an authority-flow soft filter. The key idea is to evaluate ObjectRank on a subgraph $TDG'(V_{TD}',E_{TD}')$ of TDG (V_{TD},E_{TD}). TDG' is created by first selecting all nodes $V_{TD}', \subseteq V_{TD}$ with distance up to M from the base set (the nodes that contain the keywords of the soft filter), where M is the *radius constant*, usually set to a number between 2 and 4 in our datasets. We add the edges $E_{TD}' \subseteq E_{TD}$ that connect nodes in V_{TD}'. Figure 31 shows the detailed steps of this optimization.

In order to guarantee interactive response times, we start with path length M=1 and progressively increase it to improve the results quality, until the user is satisfied with the current results' quality. To further accelerate the execution, we reuse the ObjectRank values of the previous iteration. Note that this algorithm is applicable for a sequence of soft queries, by merging their base sets (node weights are added if ObjectRank2 [VHR08] is used, which has weighted base set).

```
1. Let q=[r_s] be a query composed of a single soft filter r_s
2. Let w be the keyword expression of r_s.
3. Initialize TDG' with the set of nodes in TDG satisfying w.
4. Repeat until user is satisfied with current results'
   quality {
5.     Do one step of breadth-first search in TDG' and add each
       newly accessed node.
6.     Exit loop, if no new nodes are added.
7.     Execute ObjectRank on TDG'.
8.     Output top-k objects. }
```

Figure 31: Approximate Single Authority-Flow Soft Filter.

Approximate lgOR: {Hard Path Filter} > {Soft Filter} with a Graph-Sampling Technique*:* A graph-sampling technique can be applied to approximate lgOR on a query comprising a hard path filter followed by a soft filter. Given a layered graph

$RG=(V_{lg},E_{lg})$, the problem of approximating lgOR for RG is reduced to estimating a subgraph RG' of RG, so that with high confidence (at least δ) the relative error of computing an approximation of lgOR in RG' is ε. First, a set $\{RG^1,\ldots,RG^m\}$ of independent and identically distributed subgraphs of RG is generated. Then, RG' is computed as the union of the m subgraphs. Each RG^i is generated using a Direct Sampling technique over a Bayesian network [RN03] that encodes all the navigational information encoded in RG and in the transition matrix A_{lg}. Finally, an approximation of lgOR is computed in RG'.

A Bayesian network $BN=(V_B,E_B)$ is built as follows:

- BN and RG are homomorphically equivalent, i.e., there is a mapping f: $V_B \rightarrow V_{lg}$, such that, $(f(u),f(v)) \in E_{lg}$ iff $(u,v) \in E_B$.
- Nodes in V_B correspond to discrete random variables that represent if a node is visited or not, i.e., $V_B = \{X \mid X$ takes the value 1 (true) if the node X is visited and 0 (false) otherwise$\}$.
- Each node X of V_B has a conditional probability distribution:

$$\Pr(X \mid Parents(X)) = \sum_{j=1}^{n} (\alpha(f(Y_j), f(X)) \cdot Y_j) \qquad (21)$$

where, Y_j is the value of the random variable that represents the *j-th* parent of the node X in the previous layer of the network, and $\alpha(f(Y_j),f(X))$ corresponds to the authority transfer rate of edge $(f(Y_j),f(X))$ in the layered graph, and is seen as the probability to move from Y_j to node X in the network. Thus, the conditional probability distribution of a node X represents the collective probability that X is visited by a random surfer, which starts from the objects in the first layer of the layered graph. Finally, the probability of the

nodes in the first layer of the network corresponds to a score that indicates how good each object is with respect to the keywords in the original query.

Direct Sampling is performed using the Bayesian Network and the topological ordering of the layered graph to generate each subgraph RG^i. Once an iteration i of the Direct Sampling is finalized, the sampled layered graph $RG^i=(V^i_{lg},E^i_{lg})$ is created. The conditional probability of each node in the last layer of each subgraph RG^i corresponds to an approximated value of lgOR. After all the subgraphs $RG^1,\ldots,RG^m$ are computed, an estimate RG' is obtained as the union of these m subgraphs. The approximation of lgOR in the graph RG' is computed as the average of the approximated lgOR values of target objects in the subgraphs $RG^1,\ldots,RG^m$. To achieve an estimate RG' so that the confidence level in the relative error ε of computing an approximation of lgOR in RG' is at least δ, the Chernoff-Hoeffding's bound yields an upper bound on the number of times the Direct Sampling process needs to be evaluated, i.e., an upper bound on the size m of $\{RG^1,\ldots,RG^m\}$.

5.3.5 GID Optimizer and Execution

We present an overview of the GID optimizer and execution engine, to illustrate how the rewriting rules and the approximation techniques are applied together to achieve interactive response times for GID queries. ObjectRank is used to implement the soft filters. The GID system works on top of relational DBMS, which stores the data graph.

Precomputation: Precomputation is required to achieve exact and timely query answering. (1) We build an ObjectRank index which stores the ObjectRank score for

each pair of a keyword and an object. A threshold is used to avoid storing objects with very small scores. (2) Full-text indexes are created for all text attributes and keyword, as well as indexes on the primary keys of the relations. However, if the query does not allow the use of pre-computed structures (e.g., the soft filter follows a hard filter), then the approximation techniques are employed.

Query time: The GID optimizer accepts an input GID query and produces an execution plan. In particular, the following rewritings are possible:

1. Select a physical implementation for each GID algebra operator. Table 9 shows the available physical operators for the GID algebra operators. Note that the path-length approximation is identified as a possible implementation for *SoftExp*.
2. Change the order of operators using the rewriting potential of the axioms.
3. Insert the *Combine* operator to support each *SoftExp* operator.
4. Replace a subsequence of operators with an equivalent "superoperator". Only one such superoperator is currently implemented as shown in the last line of Table 9. It replaces (*HardPath> SoftExp)* and is implemented using the graph-sampling approximation.

Note that we only consider linear plans in this version of GID optimizer. This is a natural choice given the linear nature of execution of GID operators. We will relax this restriction as more capabilities are added to the GID algebra.

We use some rules-of-thumb as indicated in the last column of Table 9 to determine which physical operator is preferred by the optimizer for each algebraic operator. Again, fine-tuning will be conducted in future versions in order to avoid using an index for non-

selective hard filters. Also note that the Graph-Sampling algorithm is always used for *HardPath>SoftExp* subsequences. When re-ordering hard filters, we first apply the more-selective filters (if these statistics are known). In the future, we plan to integrate our GID optimizer with the relational cost-based optimizer to make better decisions.

Table 9: Physical Implementation of GID Algebra Operators.

Algebra Operator	Physical Operator	Requirements/Conditions for Selecting
HardExp	Index Lookup	Full-Text Index Available/ Always if available
	On-the-fly	None
HardPath	Index Lookup (not supported currently)	Path Indexes Available/ Always if available
	On-the-fly	None
HardType	Table Scan	Separate objects table for each type/ Always if available
	On-the-fly	None
HardAttribute	Index Lookup	B+-tree index on this attribute available/ Always if available
	On-the-fly	None
SoftExp	ObjectRank index lookup	ObjectRank index available. Should be First filter of query/Always if available
	Path-Length-Bound Approximation (Progressively increase path length)	None
Combine	On-the-fly	None
HardPath > SoftExp	Graph-Sampling	None/Always used for this sequence of operators

We illustrate how the optimizer creates a plan for three key template queries involving the expensive soft filters.

a. If the query begins with a keyword *SoftExp*, the precomputed ObjectRank index is used to evaluate the filter. For instance, for query {*Keywords="TP53", false, true*} > {*Path = EntrezGene/PubMed, false, false*}, the precomputed ObjectRank index of keyword "*TP53*" is used to evaluate the soft filter.

b. If the query starts with a *HardPath* filter followed by a keyword SoftExp filter, e.g., {*Path = EntrezProtein/PubMed, false, false*} > {*Keywords =“cancer”, false, true},* we replace this subsequence with the superoperator and introduce the *Combine* operator. Our experiments will show that this superoperator and the graph-sampling approximation are essential when the data graph is large.

c. If a hard filter (excluding a *HardPath* filter) is followed by a keyword SoftExp filter, *e.g.*, {*Keywords = “TP53”, false, false*}> {*Keywords =“cancer”, false, true*} - then we apply the path-length-bound technique. We start with path length *M*=1 and progressively increase it to improve the result quality.

Clearly, it is not always possible to compute accurate results in interactive time for some complex queries, e.g., for a long alternating sequence of hard/soft filters. However, such queries are typically unintuitive.

5.3.6 Experimental Results

Our experiments focus on the evaluation time performance and the quality of producing approximate answers in the interactive GID framework. We do not compare with other systems. The framework of [RG03] is not targeted for online computation. They report on the evaluation times for an *exact* computation (in a warehouse environment) and the execution times that they report are in many hundreds of seconds. Other graph query languages, e.g., SPARQL, do not provide the sophisticated ranking which is the key to GID framework and so the comparison would not be meaningful.

Table 10: Datasets

Name	#nodes	#edges	Size (MB)
DS7	699,199	3,533,756	2,189
DBLP	876,110	4,166,626	3,950
DS3	28,351,615	10,014,869	5,978

Datasets: We use three real datasets (Table 10). DS3 and DS7 are two biological datasets while DBLP is a bibliographical dataset. The biological datasets were created following an experimental protocol that start from annotated gene records in public Web accessible sources, and follow hyperlinks, to reach publications in PubMed. A subset of the schema of DS3 and DS7 is in Figure 7. DS7 follows less hyperlinks and visits less sources; hence it creates a smaller graph. We use the larger graph DS3 to experiment with the graph-sampling approximation. We shredded the downloaded DBLP file [DBLP09] into the relational schema shown in Figure 4.

Evaluation Metrics: We evaluate both quality and performance.

(1) The quality of the ranking is with respect to the exact ranking. For the approximation techniques presented we measure the quality of the approximation using a normalized top-*k* Spearman's rho with ties [FKM+04, FKM+06, FKS03]. Let σ_1 and σ_2 be 2 top-*k* lists. The set of results in ties is called a *bucket*. The ranked list of results, then can be viewed as ranked buckets B_1, B_2,….,B_n. The position of bucket B_i, denoted $pos(B_i)$ is the average location within bucket B_i. We assign $\sigma(x)=pos(B)$ where $\sigma(x)$ is the rank of result, x and B is the bucket of x. ρ is

the Spearman's rho metric, which is a normalized distance measure that lies in the interval [0, 1] defined as follows:

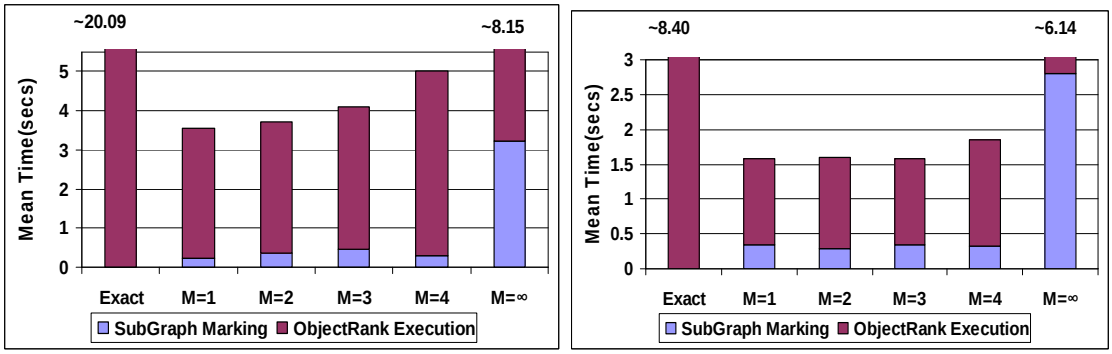
$$\rho(\sigma_1, \sigma_2) = \frac{\left(\sum_{i=1}^{k} |\sigma_1(i) - \sigma_2(i)|^2 \right)^{1/2}}{\left(k * (k+1) * (2k+1) / 3\right)^{1/2}} \quad (22)$$

where we use k+1 as the penalty constant [FKS03]. Note that the denominator of Equation 22 is used for normalization.

(2) We also report on runtime performance. The experiments were evaluated on a Solaris machine with Sparcv9 1281 MHz processor and 16GB of RAM. All algorithms were implemented in Java (JDK version 1.5.0_12). Oracle DBMS (version 10g Enterprise Edition Release 10.2.0.1.0) was used to store the database and JDBC was used to connect to the database system. We report on the execution time for successive iterations of the approximation algorithm.

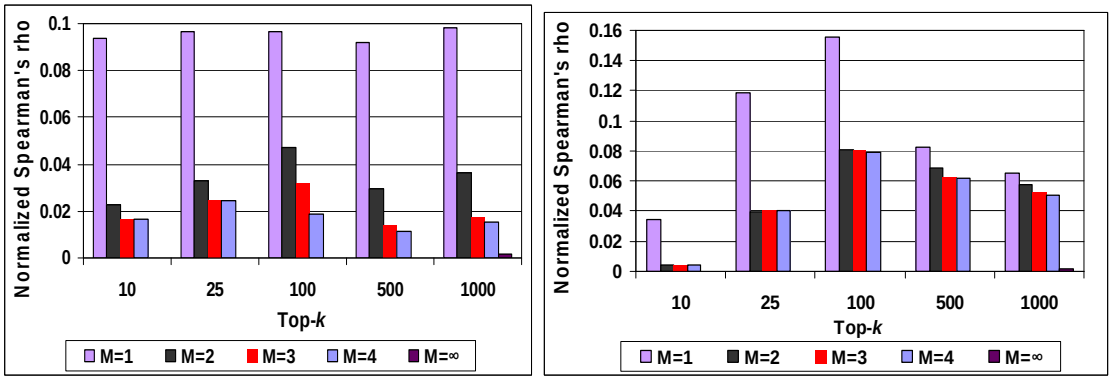
Evaluate Path-Length-Bound Technique: We evaluate the effectiveness of the path-length-bound optimization technique described on query template (c) as follows: *Hard Filter* > *Keyword Soft Filter*. We conducted these experiments on the DS7 and DBLP datasets. We did not use DS3 because this approximation technique was not scalable to the large DS3 dataset, as the value of the *radius constant, M,* increased. The entire data graph is loaded into memory. The database is then consulted only to find the base set (with their IR scores using oracle intermedia *contains*()) of each query. We optimize the query execution by avoiding the explicit creation of a subgraph. To do this, we reuse the original DBLP or DS7 database graph (already in memory) and mark the nodes in the

subgraph using a Boolean. For example, we mark all nodes that are part of the subgraph "*true*" while the rest are marked "*false*". Then we execute the path-length-bound approximation of ObjectRank using only those nodes and edges that are part of the subgraph.



(a): DBLP Execution (b): DS7 Execution

Figure 32: Performance experiments of Path-Length-Bound Technique.



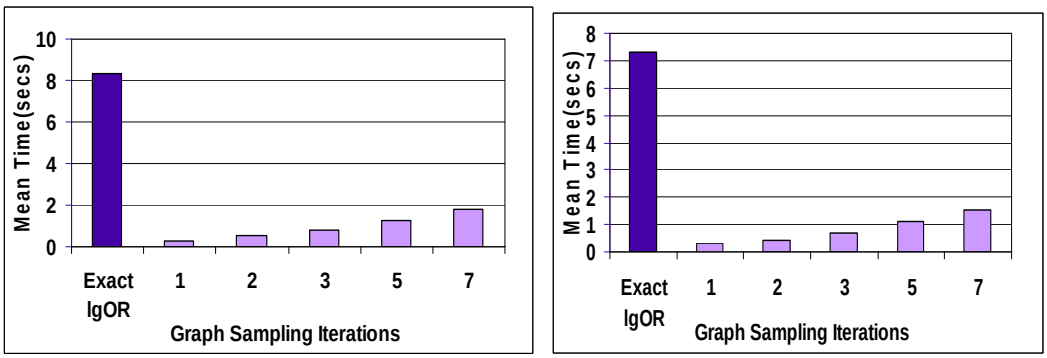
(a): DBLP Execution (b): DS7 Execution

Figure 33: Quality Experiments of Path-Length-Bound Technique.

The total execution time is measured for the following stages: (i) creating the subgraph for the keyword hard filter and (ii) executing the keyword soft filter (ObjectRank) on the subgraph. Figures ptlen_per(a) ptlen_per(b) show the execution time averaged over 20 queries, for the DBLP and DS7 datasets respectively, for increasing

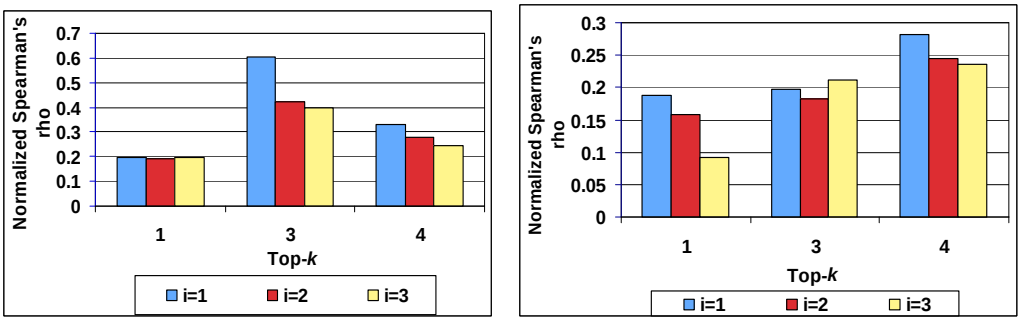
values of the radius constant, *M*, and a convergence threshold of 0.0001. To provide a baseline, we compare our execution time with the exact solution - the original ObjectRank algorithm executed over the data subgraph after application of the hard filer. This is equivalent to setting *M* to ∞. Note the significant execution time for the exact solution (over 20 seconds) for DBLP when compared to DS7 dataset is due to its larger size and high connectivity.

We note that in the GID exploratory framework, we can iteratively provide answers to users. Thus, for *M* values of 1 and 2, we can provide answers after a relatively short delay (in Figure 32 each bar for varying *M*=1, 2, 3, 4 represents the delay time while *M*=∞ represents the total execution time). Figures 33(a) and 33(b) show the quality of the results using the top-*k* Spearman's rho metric for the DBLP and DS7 datasets, respectively. Each group of results is for varying *top-k* and each bar is for varying *M*. As the radius constant *M* increases, the performance degrades and the quality improves (lower value of Spearman's rho metric) since a larger subgraph is used for ObjectRank execution. There is clearly a trade-off; for lower *M* we have lower delay but also lower quality. Notice that in both datasets, for *M*=2, we achieve a good tradeoff of quality and performance (higher quality for a relatively shorter delay time), when compared to *M*=1, 3, or 4. There is a small improvement in quality (lower value of Spearman's rho metric) for Top-500 and Top-1000 in both datasets. This is because of the large number of ties towards the end of these top-*k* lists.



(a) DS3 Execution. **(b): DBLP Execution.**

Figure 34: Performance experiments of Graph-Sampling Technique.



(a) DS3 Execution. **(b): DBLP Execution.**

Figure 35: Quality Experiments of Graph-Sampling Technique.

Evaluate Graph-Sampling Technique: We evaluate the effectiveness of the approximate lgOR metric using the Bayesian network and graph-sampling on the DS3 and DBLP datasets. (DS7 results are similar and omitted). We consider 30 queries of the query template (b). The sample queries for DS3 are as follows: {*Path = EntrezGene/*/PubMed, false, false*} > *Keyword Soft Filter.*

A key success factor in sampling is to reach the *golden objects*. For these queries, we identified the *golden objects* as the objects in PubMed whose normalized score was greater than some threshold. To compute the exact lgOR metric for a given query, the entire layered graph is loaded in memory. The database is contacted to construct the

layered graph and to find the base set of the query. Then, the lgOR is computed by traversing the whole layered graph. To compute the graph-sampling for a given query, the entire layered graph is also loaded into main memory to build the Bayesian network. Then, the approximated lgOR is computed by following the direct sampling method in which a node in the network is visited depending on the conditional probability distribution of the node. Assuming that *golden objects* have a relatively high probability of being visited during the sampling, we optimize the query execution by avoiding traversing the whole layered graph and visiting only nodes that conduce to the *golden objects* of the query.

Figure 34(a) reports the average execution time over 30 top-*k* queries in DS3 and Figure 34(b) reports time over 30 queries in DBLP. Graph-sampling is executed for $i = 1$ to 7 iterations where i corresponds to the number of sampled layered graphs RG^i. The total execution time corresponds to the time of creating the layered graph and the base set and computing approximate lgOR on the layered graph. We first observe that despite DS3 being a very large dataset, the execution times of approximate lgOR range from 1 to 2 seconds and show up to an order of magnitude improvement over the exact computation. This improvement suggests that this sampling method will be the key to success of the GID exploratory framework. These savings are maintained over additional iterations, in particular for the large dataset DS3. The savings for the smaller DBLP dataset are also significant after multiple iterations.

Figure 35 reports the normalized Spearman's rho for the queries in DS3 and DBLP. We group the queries into three groups of ten queries according to the number of *golden objects* whose normalized score is greater or equal than 0.7. The Top-1 group

comprised of queries with one golden object; the Top-3 group with three golden objects and Top-4 group with four golden objects. We report on the average normalized Spearman's rho values over 10 queries of each group. As can be seen, the graph-sampling technique is able to rank the top-*k* objects in the sampled layered graphs RG^i in an order close to the exact solution. These results indicate that the graph-sampling technique successfully achieves our optimization goal of minimizing the number of visited nodes during query execution time.

5.4 Comparing Top-*k* XML Lists

Systems that produce ranked lists of results are abundant. For instance, Web search engines return ranked lists of Web pages. To compare the lists produced by different systems, Fagin et al. [FKM+04, FKM+06, FKS03] present distance measures for top-k lists that extend the traditional distance measures for permutations of objects, like Kendall tau [FKS03] and Spearman's Footrule [FKS03].

In addition to ranking whole objects (e.g., Web pages), there is an increasing number of systems, including XRANK [GSBS03], XSEarch [CMKS03] and XKeyword [CMKS03] that provide keyword search on XML or other semi-structured data, and produce ranked lists of XML subtrees. In addition, XML lists distance measures can also be applied to rank-aware extensions of XPath and XQuery. Furthermore, these measures are needed for XML lists aggregation, where the results from several XML search engines can be aggregated to find the best top-*k* list for the given lists. Clearly, there is a need to have

measures to compare the results of such systems among each other or against the user's ideal list of results.

Unfortunately, previous distance measures are not suitable for ranked lists of subtrees since they do not account for the possible overlap between the returned subtrees. That is, two subtrees differing by a single node would be considered completely different objects. Figure 36 shows two top-3 lists of subtrees produced by two imaginary XML keyword proximity search algorithms. Trees Ta_2 and Tb_3 only differ by a single node but this is ignored by object-level distance measures.

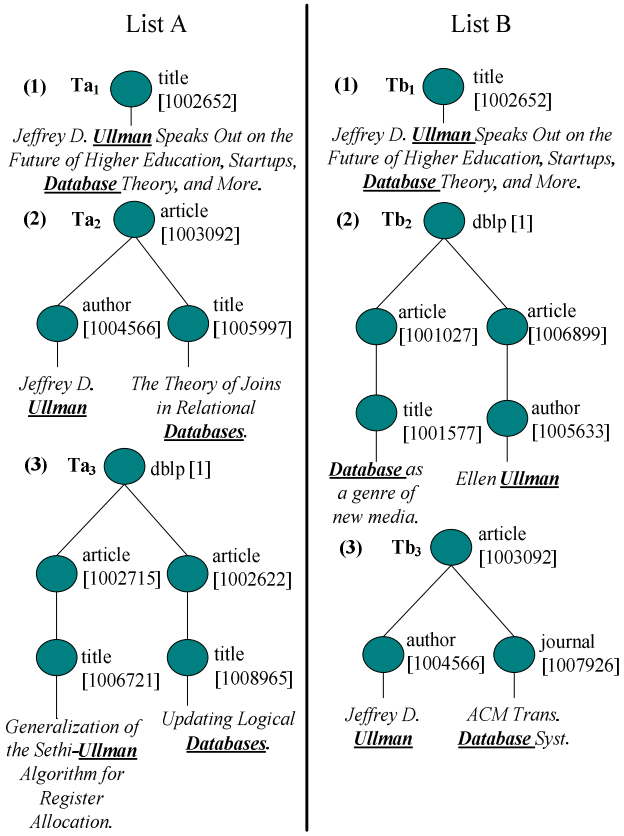


Figure 36: Top-3 trees for query - Ullman Database.

We present the first distance measures for ranked lists of subtrees. In particular, the distance measures consist of two components: the tree similarity component and the

position distance component. The former captures the similarity between the structures of the returned subtrees, while the latter captures the distance of the subtrees in the two lists, similarly to previous object-level distance measures [FKM+06].

Intuitively, our distance measures work in two phases. In the first phase, they find the optimal (closest) mapping between the two top-*k* lists of subtrees, where the distance between a pair of subtrees is computed using one of the approaches proposed in previous works, including tree edit distance [Bil03], tree alignment distance [Bil03], Fourier transform-based similarity, entropy-based similarity, tag similarity, and path shingle similarity. The cost of the optimal mapping between the two lists of subtrees represents the tree similarity component. Next, we compute the position distance component given the optimal mapping, using one of the previously proposed techniques on measuring the distance between top-*k* permutations [FKM+04, FKM+06, FKS03].

The goal of this work is to define and compute the distance between two lists *La*, *Lb* of XML trees, $La=Ta_1, Ta_2,\ldots Ta_k$ and $Lb= Tb_1, Tb_2,\ldots, Tb_k$, where Tx_i are XML trees. Often, as is the case with XML proximity search systems, all Ta_i, Tb_j are included (obtained by a sequence of deletes) in a tree *Ti* of a collection *D=T1,..,Tn*. However, this property is not important in our definitions.

Note that for the case of complete lists (permutations) of subtrees where each subtree appears in both lists, the problem is reduced to the permutations distance

problem.. However, this case is not practical since XML search engines return different XML trees. Hence, we focus on top-*k* lists.

A total mapping *f* from *La* to *Lb* is a bijection from *La* to *Lb*. Hence, tree Ta_i is mapped to $Tb_j=f(Ta_i)$. *N* is the set of all possible total mappings, *f* from *La* to *Lb*. Let *TS*(*T1*,*T2*) be the tree similarity between two trees *T1*, *T2*. *TS* can be the tree edit distance or another measure. *TS* is normalized in [0,1].

5.4.1 XML Lists Distance based on Total Mapping (XLDTM)

We present our first measure for the distance between two top-*k* lists of XML trees. The key intuition is that we extend previous list distance measures that only consider exact mappings between the objects of the two lists to also consider approximate mappings. In particular, we first compute the closest pair-wise mappings between the XML trees from the two lists and then view these mappings as exact mappings and apply list permutation distance measures.

Assuming *k* elements in each XML list, *XLDTM* is defined as follows. First we define the total mapping similarity distance $MSD^T(La,Lb,f)$ between *La* and *Lb* for a total mapping *f* as

$$MSD^T(La, Lb, f) = \frac{\sum_{i=1..k} TS(Ta_i, f(Ta_i))}{k}$$

That is, MSD^T is a measure of how "tight" the total mapping *f* is. Notice that $MSD^T(La,Lb,f)$ takes values in [0,1], since *TS* is also in [0,1] and we divide by *k*.

We next define the minimum total mapping $fmin^T$ as the total mapping between *La* and *Lb* with minimum $MSD^T(La,Lb,f)$. It is,

$$fmin^T = argmin_f\, MSD^T(La,Lb,f)$$

that is, $argmin_f$ is the *f* that minimizes MSD^T.

Given $fmin^T$, we define the *minimum total mapping similarity distance*,

$$MinMSD^T(La,Lb) = MSD^T(La,Lb,fmin^T)$$

Definition: The *XLDTM* between XML lists *La*, *Lb* has two components:

a. The XML similarity component $MinMSD^T(La,Lb)$.

b. The total mapping position distance component $PD^T(La,Lb,fmin^T)$, which is also referred as the position component in this section. PD^T is defined using one of the well known metrics on permutations as discussed below. PD^T is in [0, 1].

It is

$$XLDTM(La,Lb) = a \cdot MinMSD^T(La,Lb) + b \cdot PD^T(La,Lb,fmin^T)$$

where *a*, *b* are the XML similarity and position component constants respectively. *a*, *b* adjust the relative importance of the two components. Notice that *XLDTM*(*La*,*Lb*) is in [0,2] since $MinMSD^T(La,Lb)$ and $PD^{T(}La,Lb,fmin^T)$ are in [0,1] and constants *a* and *b* are in [0,1].□

We choose $fmin^T$ to minimize the XML similarity component and not the whole *XLDTM*, because we believe it is more intuitive to compute the distance component based on the tightest XML similarity mapping rather than mixing the two components. Note that other functions can be used to combine the contribution of the two components, as we discuss below.

Measures for XML Similarity component, $MinMSD^T(La,Lb)$: The tree similarity, *TS* which is used to compute $MinMSD^T(La, Lb)$ can be any of the tree or XML similarity measures.

Measures for Position component, $PD^T(La, Lb, fmin^T)$: Note that list permutation distance metrics (not top-*k* list distance measures) are used in *XLDTM*. Given the mapping $fmin^T$, we naturally extend the Spearman's footrule distance and Kendall tau distance for permutations with ties [FKM+04, FKM+06, FKS03] as follows:

Position distance (PD^{TF}) based on Spearman's footrule metric for permutations, is given by:

$$PD^{TF}(La,Lb,fmin^T) = \sum_{i=1}^{k} \left| pos_{La}(Ta_i) - pos_{Lb}(fmin^T(Ta_i)) \right|$$

where $pos_{La}(Ta_i)$ is the position of tree Ta_i in list *La*. This formula is extended as follows to consider ties. A set of trees with the same score is called a *bucket*. The ranked list of results can be then viewed as ranked list of buckets $B_1, B_2,\ldots,B_n$. The position of bucket B_i, denoted $pos(B_i)$ is the average result location within bucket B_i. We assign $pos_{La}(Ta_i) = pos(B(Ta_i))$ where $B(Ta_i)$ is the bucket of Ta_i.

Position distance (PD^{TK}) based on Kendall tau metric for permutations considering ties, is given by:

$$PD^{TK}(La,Lb,fmin^T) = \sum_{\{i,j\} \in P} \overline{K}_{i,j}(La,Lb')$$

where *Lb'* is constructed from list *Lb* when element Tb_j is replaced by $Ta_i = (fmin^T)^{-1}(Tb_j)$, that is, $Tb_j = fmin^T(Ta_i)$. That is, we assume that an element Ta_i in *La* and

its corresponding element Tb_j in *Lb* are the same. Hence, we just have *k* distinct elements {1, 2, … ,*k*} in both lists, and the problem of computing $PD^{TK}(La, Lb, fmin^T)$ of the two XML lists is same as computing the Kendall Tau metric of two permutations. *P* is the set of all unordered pairs of the *k* distinct elements.

If there are two mappings $fmin1^T$ and $fmin2^T$ that have equal MSD^T, i.e., $MSD^T(La,Lb,fmin1^T) = MSD^T(La,Lb,fmin2^T)$, then we compute *PD* for both and in the end pick the one that gives the smallest *PD*.

Hence, there are two variants of *XLDTM*:

$XLDTM^F(La,Lb)= a \cdot MinMSD^T(La,Lb)+b \cdot PD^{TF}(La,Lb,fmin^T)$

$XLDTM^K(La,Lb) = a \cdot MinMSD^T(La,Lb)+b \cdot PD^{TK}(La,Lb,fmin^T)$

Example: Consider the top-3 lists *La* and *Lb* in Figure 36. We will illustrate the steps involved in computing $XLDTM^F(La, Lb)$ and $XLDTM^K(La, Lb)$. In this example, we use tree edit distance, *TED* as the tree similarity measure, *TS*. We first compute the XML similarity component by finding all possible total mappings, $N= \{f_1, f_2, f_3, f_4, f_5, f_6\}$:

$f_1(Ta_1)=Tb_1,\ f_1(Ta_2)=Tb_2,\ f_1(Ta_3)=Tb_3$

$f_2(Ta_1)=Tb_3,\ f_2(Ta_2)=Tb_2,\ f_2(Ta_3)=Tb_1$

$f_3(Ta_1)=Tb_2,\ f_3(Ta_2)=Tb_1,\ f_3(Ta_3)=Tb_3$

$f_4(Ta_1)=Tb_1,\ f_4(Ta_2)=Tb_3,\ f_4(Ta_3)=Tb_2$

$f_5(Ta_1)=Tb_3,\ f_5(Ta_2)=Tb_1,\ f_5(Ta_3)=Tb_2$

$f_6(Ta_1)=Tb_2,\ f_6(Ta_2)=Tb_3,\ f_6(Ta_3)=Tb_1$

The normalized tree edit distance between each pair of trees in *La* and *Lb* is given by the following matrix:

$$\begin{array}{c} \\ Ta_1 \\ Ta_2 \\ Ta_3 \end{array} \begin{array}{c} \begin{array}{ccc} Tb_1 & Tb_2 & Tb_3 \end{array} \\ \begin{bmatrix} 0.00 & 0.78 & 0.71 \\ 0.71 & 0.58 & 0.20 \\ 0.78 & 0.43 & 0.58 \end{bmatrix} \end{array}$$

The total mapping similarity distance of each total mapping in *N* is calculated by as follows:

$$MSD^T(La, Lb, f_1) = (0.00+0.58+0.58)/3 = 1.16/3 = 0.38$$

$$MSD^T(La, Lb, f_2) = (0.71+0.58+0.78)/3 = 2.07/3 = 0.69$$

$$MSD^T(La, Lb, f_3) = (0.78+0.71+0.58)/3 = 2.07/3 = 0.69$$

$$MSD^T(La, Lb, f_4) = (0.00+0.20+0.43)/3 = 0.63/3 = 0.21$$

$$MSD^T(La, Lb, f_5) = (0.71+0.71+0.43)/3 = 0.63/3 = 0.62$$

$$MSD^T(La, Lb, f_6) = (0.78+0.20+0.78)/3 = 0.63/3 = 0.59$$

Hence, f_4 is the mapping with the minimum mapping distance. It is $minMSD^T(La, Lb) = MSD^T(La, Lb, f_4) = 0.21$.

The normalized Spearman's footrule position component is $PD^{TF}(La,Lb,f_4)=2.0/4.0=0.5$. Hence, $XLDTM^F(La,Lb) = 0.21+0.5 = 0.71$ (assuming a=1 and b=1). If the position distance is calculated using normalized Kendall tau, then $PD^{TK}(La,Lb,f_4)=1.0/3.0=0.33$ and $XLDTM^K(La,Lb) = 0.21+0.33 = 0.54$ (assuming a=1 and b=1). The difference in the two scores is due to inherent differences between the Spearman's footrule and Kendall tau metrics. □

5.4.2 Computing *XLDTM*

In this section, we describe efficient algorithms to compute *XLDTM* given two XML top-*k* lists.

Naive approach: *XLDTM* for any two top-*k* XML lists *La* and *Lb* is computed as follows. First, the set *N* of all possible total mappings from *La* to *Lb* is computed. Then, for each total mapping *f* in *N*, we compute the total mapping similarity distance, $MSD^T(La,Lb,f)$, and then find the minimum mapping $fmin^T$. If we find more than one mapping with the same minimum mapping similarity distance we break the tie by computing the position distance, $PD^T(La,Lb,fmin^T)$ for each of them and in the end pick the one that gives smaller PD^T. Then, we compute $XLDTM(La,Lb)$.

Compute-*XLDTM* (XML List $La=\{Ta_1,Ta_2,\ldots,Ta_k\}$, XML List $Lb=\{Tb_1,Tb_2,\ldots,Tb_k\}$, constants *a* and *b*):

1. Let $S[k, k]$ be a 2-D array that stores the tree similarity measures between every pair of XML trees (one from each List);
2. For *i* in 1...*k* do {
3. For *j* in 1...*k* do {
4. Compute $TS(Ta_i,Tb_j)$;
5. Normalize $TS(Ta_i,Tb_j)$;
6. $S[i, j] \leftarrow TS(Ta_i,Tb_j)$; } }
7. Let $assignment_m[k,2]$ be a 2-D array that stores the m^{th} $fmin^T$ with the minimum mapping distance;
8. *assignment* ← *Ext-Hungarian-Algorithm*(*S*, "*min*");
9. For each $fmin^T$ compute the normalized position distance, $PD^T(La,Lb,fmin^T)$ for Spearman's footrule or for Kendall Tau;
10. Select $fmin^T$ with the minimum position distance;
11. Compute *XLDTM*;

Figure 37: Algorithm for computing XLDTM

Overview of our algorithm: Instead of computing the set N of all possible total mappings and then selecting the minimum mapping $fmin^T$, we precompute the tree similarity measure of each tree pair across the two lists, build a bipartite graph, and apply a minimum cost perfect matching algorithm to compute all minimum mappings $fmin^T$. Figure 37 presents the algorithm.

Algorithm details: The following high-level steps of execution explain the algorithm in detail:

1. Precompute the tree similarity $TS(Ta_i, Tb_j)$ between every pair of XML trees, one from each list La and Lb. There are k^2 such pairs, hence the complexity of this step is $k^2 \cdot Cost(TS(Ta_i, Tb_j))$ where $Cost(TS(Ta_i, Tb_j))$ is the complexity of computing the tree similarity between the two trees Ta_i and Tb_j.
2. Create a weighted complete bipartite graph $G(C, P, W)$ as follows. The first set of nodes $C = \{1, 2…., k\}$ denote the set of elements in XML list La. The second set of nodes $P = \{1, 2…., k\}$ denote the set of elements in XML list Lb. The weight $W(i, j) = TS(Ta_i, Tb_j)$.
3. Execute a minimum cost perfect matching algorithm on $G(C, P, W)$ to compute $fmin^T$. We use the Hungarian algorithm. Notice that, in our case we use an extended version of the Hungarian algorithm that outputs the set of all $fmin^T$ with the same minimum mapping similarity distance, $minMSD^T$. Then, for each $fmin^T$ we compute the position distance $PD^T(La,Lb,fmin^T)$ and pick the one with the least PD^T. Finally, *XLDTM* is computed for Spearman's footrule and Kendall tau position component respectively. The complexity of the Extended Hungarian

algorithm is O(k^3), which is the same as the original Hungarian algorithm. Total Complexity of the algorithm is O(k^2.*Cost*(*TS*(Ta_i, Tb_j))+ k^3).

5.4.3 Experimental Results

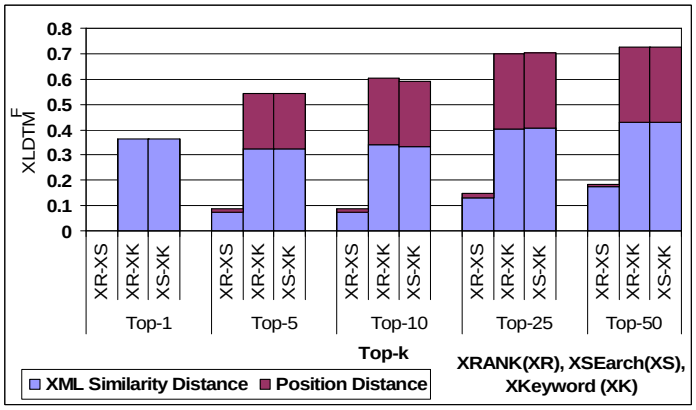
In this section we experimentally evaluate the measures presented in the previous sections by comparing three popular XML keyword search algorithms. We use tree edit distance as the XML similarity measure.

Table 11: XML Datasets

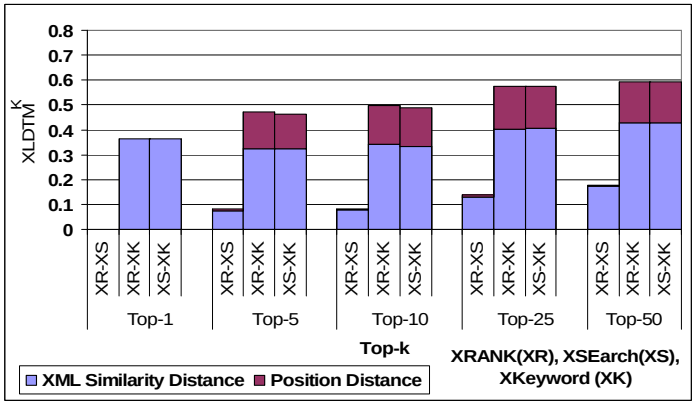
DATASET	NUMBER OF ELEMENTS	AVERAGE DEPTH	MAXIMUM DEPTH
DBLP	7 ,137, 933	1.90	5
NASA	791,923	5.58	8

Datasets: We use two real datasets: the DBLP dataset and the NASA XML dataset available at [NSD08]. Table 11 summarizes their characteristics. We implemented the following XML keyword proximity search systems: XRANK [GSBS03], XSEarch [CMKS03] and XKeyword [HPB03]. These three algorithms take as input a corpus of XML documents and a keyword query, and return as output an ordered list of XML fragments that satisfy the query by containing all the keywords. All three algorithms favor minimal and compact subtrees that satisfy the query, but use different ranking functions and pruning rules. In particular, while XKeyword ranks its answers by the size of the resulting subtree, XRANK and XSEARCH also utilize Information Retrieval (IR) score functions based on *tf·idf*. XSEarch prunes result paths that repeat the same tag in internal nodes, while XRANK prunes results if there is a more specific result in the same

element. Also, XRANK returns whole subtrees while XSEarch and XKeyword return paths.



(a) Average $XLDTM^F$ vs. Top-k



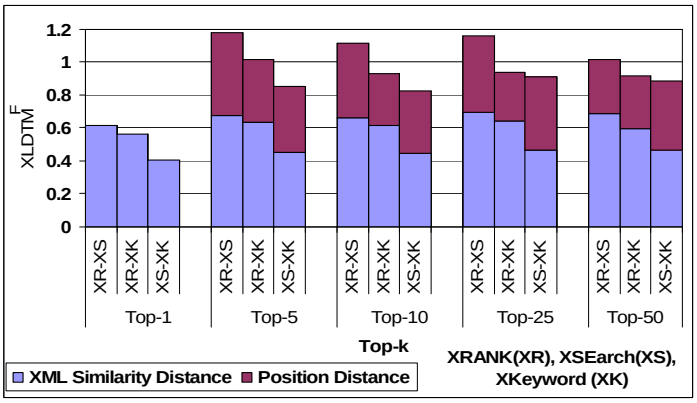
(b) Average $XLDTM^K$ vs. Top-k

Figure 38: XLDTM Experiments on DBLP Dataset.

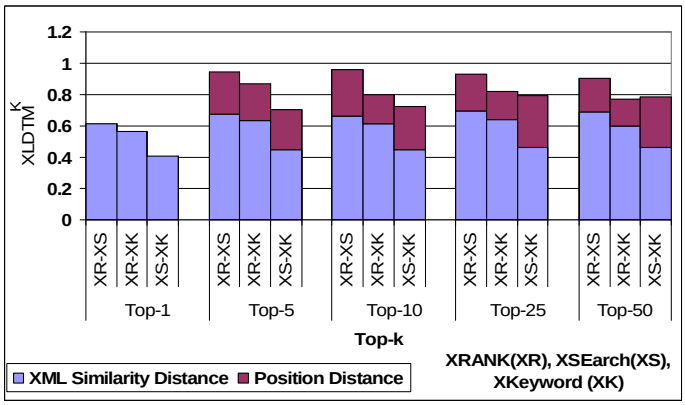
In our implementation, we used the IR score provided by the CONTAINSTABLE function of Microsoft SQL Server 2000 to compute the IR components of both XRANK and XSEARCH ranking functions. The experiments were performed on a PC with an Intel Pentium Core 2 Duo, 2.00 GHz processor, 2GB RAM, running Windows Vista Business. All algorithms were developed in Java (JDK version 1.6.0_06), use the

Document Object Model (DOM) for XML parsing and navigation, and Microsoft SQLServer 2000 for the persistent storage of indexes. The tree similarity (*TS*) measure we use in our experiments is the dynamic programming algorithm by Zhang and Shasha [34] which computes the tree-edit-distance between ordered trees [Bil03] whose complexity is $Cost(TED(Ta_i,Tb_j)) = O(|Ta_i||Tb_j| \cdot \min(leaves(Ta_i), depth(Ta_i)) \cdot \min(leaves(Tb_j), depth(Tb_j))$. We refer to a detailed survey of tree edit distance algorithms [Bil03]. We report average XML Lists Distance values over many experiments on the two datasets.

Figures 38(a) and 38(b) show the total distances (split into the two components) between the result lists produced by the three search algorithms on the DBLP dataset averaged over 50 two-keyword queries, using $XLDTM^F$ and $XLDTM^K$, respectively. The queries used include: "*artificial intelligence*", "*xml indexing*", "*text mining*", "*image retrieval*", "*OLAP mining*". Notice that the distance increases as *k* increases because as the trees get larger, the results become more disparate due to the pruning rules of the algorithms that go in effect for larger trees. As mentioned before, XKeyword ranks its answers by the size of the resulting subtree, while XRANK and XSEARCH also utilize Information Retrieval (IR) score functions based on *tf·idf*. The reason that XKeyword has large distance to the other two rankings is that it does not have an IR component in its ranking function. Hence, when multiple trees have the same size, they are ranked arbitrarily. XRANK and XSEarch have smaller distance between them because their rankings are more similar given that the results were mostly single-node trees.



(a) Average $XLDTM^F$ vs. Top-k



(b) Average $XLDTM^K$ vs. Top-k

Figure 39: XLDTM Experiments on NASA Dataset.

Figure 39 repeats the set of experiments of Figure 38 on the NASA dataset. Some sample two-keyword queries used in these experiments are: *"arcminutes magnitude"*, *"astrographic motion"*, *"equinox culmination"*, *"photographic wavelengths"*, *"oxford zone"*. Some important observations on the results of NASA dataset are (a) Distance between XML lists is generally larger for NASA dataset because of its larger depth. (b) In contrast to Figure 38, XSEarch and XKeyword have the smallest distance because both

algorithms return paths as result. This factor was less important in Figure 38 because most results were single-node. In contrast, XRANK has large distance to the other two rankings because it returns whole subtree as result. (c) XRANK is very close to XSEarch in DBLP, but very far in NASA dataset. The reason is that the XRANK and XSEarch pruning conditions are very rare for very shallow subtrees (DBLP) but more frequent for deeper subtrees (NASA dataset). The latter also leads to unpredictable fluctuations to the distances for increasing *k*, in contrast to the linear increase in the DBLP dataset. In both datasets, notice that the XML Similarity distance contributes the most to the total distance. This shows that the main difference of these three algorithms comes more from how they define a result and less on how they rank them.

We also present performance results on the deeper NASA dataset. Figure 40 shows the average execution time to compute *XLDTM* for various values of *k*, over the same 50 two-keyword queries used in the distance experiments. As expected, the average execution time increases superlinearly as *k* increases because there are more results in the top-*k* lists under comparison. Notice that the execution times are different for the three pairs of search algorithms. The reason is that XRANK produces the largest size of results as it returns whole XML elements, while XKeyword produces concise results by returning paths. XSEarch produces results of intermediate size by returning paths like XKeyword but has different pruning rules. Thus, the execution times of XRANK vs. XSEarch are the highest, while XSEarch vs. XKeyword is the lowest.

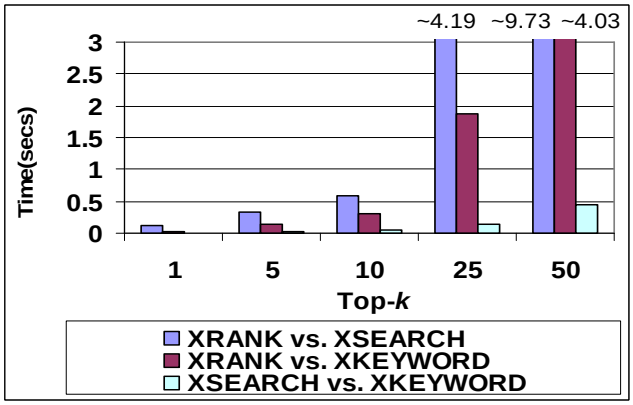


Figure 40: Performance Experiments on NASA dataset

6 CONCLUSIONS

This dissertation presents novel techniques and methods to provide user-friendly, high quality and efficient searching of graph structured databases. In [VH05,VH06,VHL06,VHL08] we propose and demonstrate a technique that given a keyword query, on-the-fly generates new pages, called composed pages that satisfy the user's information needs and improves user satisfaction. In [VHR08] we create a framework and provide algorithms to explain query results and reformulate authority flow queries based on the user's feedback. In a recent work, we propose a flexible and extensible framework for querying over large hyperlinked data collections [VHR+09]. We also devise methods to automatically compare top-*k* XML lists.

LIST OF REFERENCES

[ACA06] A. Agarwal, S. Chakrabarti, and S. Aggarwal. Learning to rank networked entities. ACM SIGKDD 2006

[ACD02] S. Agrawal, S. Chaudhuri and G. Das: "DBXplorer: A System for Keyword-Based Search Over Relational Databases", IEEE ICDE, 2002.

[AP00] E. Amitay and C. Paris: "Automatically Summarizing Web Sites -Is there any way around it?" CIKM, 2000.

[AA98] G. Arocena, A. Mendelzon: WebOQL: Restructuring documents, databases and webs. ICDE 1998.

[AP97] J.Abracos and G. Pereira-Lopes: "Statistical methods for retrieving most significant paragraphs in newspaper articles", ACL/EACL Workshop on Intelligent Scalable Text Summarization, 1997.

[Bil03] P Bille. A survey on tree edit distance and related problems, Theoretical Computer Science, 2005.

[BDFS03] P. Buneman, S. Davidson, M. Fernandez, D. Suciu "Adding Structure to Unstructured Data". ICDM, 2003

[BE97] R. Barzilay and M. Elhadad: "Using lexical chains for text summarization", ISTS, 1997.

[BHP04] A. Balmin, V. Hristidis and Y. Papakonstantinou: "Authority-Based Keyword Queries in Databases using ObjectRank". VLDB 2004.

[BM00] A. L. Berger and V. O. Mittal: "OCELOT: A System for summarizing web pages", ACM SIGIR, 2000.

[BNH+02] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti and S.Sudarshan: "Keyword Searching and Browsing in Databases using BANKS", IEEE ICDE, 2002.

[BSA+95] C. Buckley, G. Salton, J. Allan and A. Singhal. Automatic query expansion using SMART. TREC-3. NIST special publication 500-225. pp 69-80. 1995.

[BSA94] C. Buckley, G. Salton and J. Allan. The effect of adding relevance information in a relevance feedback environment. ACM SIGIR 1994.

[CHWM04] D. Cai, X. He, J.Wen and W.Ma: "Block-level Link Analysis", ACM SIGIR, 2004.

[CKS03] H.H. Chen, J.J. Kuo and T.C. Su: “Clustering and Visualization in a Multi-Lingual Multi- Document Summarization System”, ECIR, 2003.

[CQ69] A. Collins and M, Quillian, Retrieval Time From Semantzc Memory. J. of Verbal Learning and Verbal Behaviour, Vol 8, pp 240-247, 1969.

[CMKS03] S. Cohen, J. Mamou, Y. Kanza and Y. Sagiv. “XSEarch: A Semantic Search Engine for XML”, VLDB, 2003.

[DBLP09] http://dblp.uni-trier.de/xml/

[DUC05] Document Understanding Conference, http://duc.nist.gov, 2005.

[Edm69] H.P. Edmundson: “New Methods in Automatic Abstracting”, ACM Journal, vol.16, no.2, pp. 264-285, 1969.

[Efth93] E. N. Efthimiadis. Interactive query expansion: A user-centered evaluation of ranking algorithms for interactive query expansion. ACM SIGIR. pp 146-159. 1993.

[ER04] G. Erkan and D.R. Radev: “Lexrank: Graph-based centrality as salience in text summarization”, JAIR, vol.22, pp 457-479, 2004.

[FFL+97] M. Fernandez, D. Florescu, A. Levy, D. Suciu: A query language for a web site management system. SIGMOD Record 1997.

[FLM98] D. Florescu, A. Y. Levy and A. O. Mendelzon: “Database techniques for the World-Wide Web: A survey”. SIGMOD Record, 1998.

[FKM+04] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, E. Vee: “Comparing and Aggregating rankings with Ties”. PODS, 2004.

[FKM+06] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, E. Vee: “Comparing Partial Rankings”. SIDMA, 2006, vol. 20, No. 3.

[FKS03] R. Fagin, R. Kumar, D. Sivakumar: “Comparing Top-k lists”. SODA, 2003.

[FLN01] R. Fagin, A. Lotem, M. Naor: Optimal Aggregation Algorithms for Middleware. PODS,2001.

[FLW+06] G. Feng, T.Y. Liu, Y. Wang, Y. Bao, Z. Ma, X. Zhang and W.Y. Ma: “AggregateRank: Bringing order to web sites”. SIGIR, 2006.

[FO01] T. Fukusima and M. Okumura: “Text Summarization Challenge Text Summarization Evaluation in Japan”, WAS, 2001.

[GD07] Google Desktop search, http://desktop.google.com/,2007.

[GKMC99] J. Goldstein, M. Kantrowitz, V. Mittal and J. Carbonell: “Summarizing text documents: Sentence selection and evaluation metrics”, ACM SIGIR, 1999.

[GKR00] N. Garg, G. Konjevod and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem.Journal of Algorithms, Volume 37 , Issue 1 (October 2000), Pages: 66 – 84.

[GSBS03] L. Guo, F. Shao, C. Botev and J. Shanmugasundaram: “XRANK: Ranked Keyword Search over XML Documents”, ACM SIGMOD, 2003.

[GSVM98] R. Goldman, N. Shivakumar, S. Venkatasubramanian and H. Garcia-Molina: “Proximity Search in Databases”. VLDB, 1998.

[HC93] D. Haines and W.B. Croft. Relevance feedback and inference networks. ACM SIGIR 1993.

[Har88] D. Harman. Towards interactive query expansion. ACM SIGIR pp 321-331. Grenoble. 1988.

[Har92] D. Harman. Relevance feedback and other query modification techniques. Information retrieval: data structures and algorithms, Prentice-Hall Inc, 1992.

[Hav02] T. Haveliwala: “Topic-Sensitive PageRank”. WWW, 2002.

[Hea94] M.A. Hearst: “Using categories to provide context for full-text retrieval results”, In Proceedings of the RIAO, 1994.

[HGP03] V. Hristidis, L. Gravano and Y. Papakonstantinou: “Efficient IR-Style Keyword Search over Relational Databases”, VLDB, 2003.

[HL00] E. Hovy and C.Y. Lin: “The automated acquisition of topic signatures for text summarization”, ICCL, 2000.

[HP02] V. Hristidis and Y. Papakonstantinou: “DISCOVER: Keyword Search in Relational Databases”, VLDB, 2002.

[HPB03] V. Hristidis, Y. Papakonstantinou and A. Balmin: “Keyword Proximity Search on XML Graphs”, IEEE ICDE, 2003.

[HXY03] A. Huang, Q. Xue, and J. Yang. TupleRank and Implicit Relationship Discovery in Relational Databases. WAIM, 2003.

[HVP06] H. Hwang, V. Hristidis, and Y. Papakonstantinou. ObjectRank: A System for Authority-based Search on Databases. Demo at SIGMOD, 2006.

[Ihl90] E. Ihler. Bounds on the quality of approximate solutions to the Group Steiner Problem. Lecture Notes In Computer Science; Vol. 484, Proceedings of the 16rd International Workshop on Graph-Theoretic Concepts in Computer Science table of contents, Pages: 109 - 118, 1990.

[JW03] G. Jeh, J. Widom: “Scaling Personalized Web Search”. WWW, 2003.

[KF06] D. Kelly and X. Fu. Elicitation of term relevance feedback: an investigation of term source and context. ACM SIGIR 2006.

[Kle99] J. Kleinberg: “Authoritative Sources in a Hyperlinked Environment”, ACM Journal, vol.46, no.5, pp. 604-632, 1999.

[KPC+05] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai and H. Karambelkar: Bidirectional Expansion for Keyword Search on Graph Databases”, VLDB, 2005.

[KS95] D. Konopnicki, O. Shmueli: W3QS: A query system for the World Wide Web. VLDB 1995.

[KS06] B. Kimelfeld and Y. Sagiv: “Finding and Approximating top-k answers in Keyword Proximity Search”, PODS, 2006.

[KSI+08] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, G. Weikum: NAGA: Searching and Ranking Knowledge. ICDE 2008: 953-962.

[LCVA01] W.S. Li, K. S. Candan, Q. Vu, and D. Agrawal: Retrieving and Organizing Web Pages by "Information Unit". WWW, 2001

[LJ01] A. M. Lam-Adesina and G.J.F. Jones. Applying summarization techniques for term selection in relevance feedback. ACM SIGIR 2001.

[Mar99] D. Marcu: “Discourse trees are good indicators of importance in text”, Advances in Automatic Text Summarization, 1999.

[MD07] MSN Desktop search, http://toolbar.msn.com/, 2007.

[MMM97] A. Mendelzon, G. Mihalia, T. Milo: Querying the World Wide Web. Journal on Digital Libraries 1(1):54-67, 1997.

[MSB98] M. Mitra, A. Singhal and C. Buckley. Improving automatic query expansion. ACM SIGIR pp 206-214. 1998.

[MT04] R. Mihalcea and P. Tarau: “TextRank: Bringing Order into Texts”, EMNLP 2004.

[NDQ06] L. Nie, B. D. Davison, X. Qi: “Topical link analysis for web search”. SIGIR, 2006.

[NSD08] University of Washington Computer Science and Engineering. XML Data Repository. http://www.cs.washington.edu/research/xmldatasets/www/repository.html. 2008.

[NZW+05] Z. Nie, Y. Zhang, J. Wen, and W. Ma. Object-level ranking: Bringing order to Web objects. WWW 2005.

[OI07] Oracle interMedia : http://www.oracle.com/technology/products/intermedia, 2007.

[PBMW98] L. Page, S. Brin, R. Motwani and T. Winograd: “The pagerank citation ranking: Bringing order to the web”, Technical report, Stanford University, 1998.

[PM07] http://www.ncbi.nlm.nih.gov/sites/entrez, 2007.

[QLZ+05] T.Qin, T. Liu, X. Zhang, Z. Chen and W.A study of relevance propagation for web search. ACM SIGIR 2005.

[Rei89] P. W. G. Reich. Beyond Steiner’s Problem: A VLSI Oriented Generalization. Workshop on Graph-Theoretic Concepts in Computer Science, 1989.

[RFZ01] D.R.Radev, W. Fan and Z. Zhang: “WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System”, NAACL Workshop on Automatic Summarization, 2001.

[RG03] S. Raghavan, H. Garcia-Molina: “Complex Queries over Web Repositories”. VLDB, 2003.

[RM98] D.R. Radev and K.R. McKeown: “Generating Natural Language Summaries from Multiple On-line Sources. Computational Linguistics”, vol.24, no.3, pp. 470-500, 1998.

[RN03] S. Russell and P.Norvig: “Artificial Intelligence: A modern approach. Second Edition. Princeton Hall. 2003.

[RPB06] M. Richardson, A. Prakash and E. Brill: “Beyond PageRank: machine learning for static ranking”. WWW, 2006.

[RWL+06] L. Raschid, Y. Wu, W.J. Lee, M.E. Vidal, P. Tsaparas, P. Srinivasan, A.K. Sehgal: “Ranking target objects of navigational queries”. ACM WIDM, 2006.

[RW94] S. E. Robertson and S Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. SIGIR 1994.

[RL03] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems . The Knowledge Engineering Review. 94-145. vol 18, issue 2. 2003.

[SPQL] SPARQL: Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/

[Sav92] J. Savoy. Bayesian inference networks and spreading activation in hypertext systems. Information Processing and Management, 28(3):389–406, 1992.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science. 41. 4. pp 288-297. 1990.

[SB95] G. Salton and C. Buckley. Optimization of relevance feedback weights. ACM SIGIR 1995.

[Sin01] A. Singhal: “Modern Information Retrieval: A Brief Overview”. Google, IEEE Data Eng. Bull, 2001.

[SIY06] P. Shafer, T. Isganitis, G. Yona. “Hubs of knowledge: using the functional link structure in Biozon to mine for biologically significant entities”. BMC Bioinformatics, 2006, Feb 15;7:71.

[SLWM04] R. Song, H. Liu, J. Wen and W. Ma: “Learning Block Importance Models for Web Pages”, WWW 2004.

[SSMB97] G. Salton , A. Singhal , M. Mitra and C. Buckley: “Automatic text structuring and summarization”, Information Processing and Management, vol.33,no.2,pp.193-207, 1997.

[SVR83] A. Smeaton and C. J. van Rijsbergen. The retrieval effects of query expansion on a feedback document retrieval system. The Computer Journal. 26. 3. pp 239-246. 1983.

[SZ05] X. Shen and C. Zhai. Active feedback in ad hoc information retrieval. ACM SIGIR 2005.

[TS98] A. Tombros and M. Sanderson: “Advantages of Query Biased Summaries in Information Retrieval”, ACM SIGIR 1998.

[VB06] S. Vassilvitskii and E. Bill. Using web-graph distance for relevance feedback in web search. ACM SIGIR 2006.

[VGS02] N. Vanetik, E. Gudes and S.E. Shimony: “Computing frequent graph patterns from semistructured data”. IEEE ICDM, 2002.

[VHR08] R. Varadarajan, V. Hristidis, L. Raschid: “Explaining and Reformulating Authority Flow Queries”. IEEE ICDE, 2008.

[VHR+09] R. Varadarajan, V. Hristidis, L. Raschid, M.E. Vidal, L. lbanez and H. Rodriguez: “Flexible and Efficient Querying and Ranking on Hyperlinked Data Sources”. EDBT, 2009.

[VHR07] R. Varadarajan, V. Hristidis, L. Raschid: “Explaining and Reformulating Authority Flow Queries”. Extended version. (http://dbir.cs.fiu.edu/ObjectRankReformulation/)

[VH06] R. Varadarajan and V. Hristidis: “A System for Query-Specific Document Summarization”, ACM CIKM, 2006.

[VHL06] R. Varadarajan, V. Hristidis and T. Li: “Searching the Web Using Composed Pages”, Poster at ACM SIGIR, 2006.

[VHL08] R. Varadarajan, V. Hristidis and T. Li: “Beyond Single-Page Web Search Results”, IEEE TKDE, 2008.

[VH05] R. Varadarajan and V. Hristidis: “Structure-Based Query-Specific Document Summarization”, Poster at ACM CIKM 2005.

[WRJ02] R. W. White, I. Ruthven and J. M. Jose: “Finding Relevant Documents using Top Ranking Sentences: An Evaluation of Two Alternative Schemes”, ACM SIGIR 2002.

[XC96] J.Xu and W.B. Croft. Query expansion using local and global document analysis. ACM SIGIR 1996.

[YH03] X. Yan and J. Han: “CloseGraph: mining closed frequent graph patterns”. ACM SIGKDD, 2003.

VITA

RAMAKRISHNA R. VARADARAJAN

EDUCATION

Doctoral Candidate in Computer Science, Florida International University (2004 – 2009)
School of Computing & Information Sciences, Miami.
CGPA: 3.97/4.0.

M.S. in Computer Science, Florida International University (2004 – 2006)
School of Computing & Information Sciences, Miami.
CGPA: 3.95/4.0

B.E. in Computer Science & Engineering, University of Madras, India(2000–2004)
First Class Honors (Ranked 5^{th} – awarded University Medal)

RESEARCH EXPERIENCE

Graduate Research Assistant, Florida International University (2005 – 2009)
Databases & Systems Research Laboratory (DSRL),
School of Computing & Information Sciences, Miami.

Intern at IBM India Research Lab (May – August 2008)

TEACHING EXPERIENCE

Undergraduate Course Instructor, Florida International University (Spring 2008)
School of Computing & Information Sciences, Miami
Course Title: Introduction to Programming in Java

Teaching Assistant, Florida International University (2006 – 2008)
School of Computing & Information Sciences, Miami
Courses: Data Structures and Principles of DBMS

Lab Assistant, Florida International University (2004 – 2006)
School of Computing & Information Sciences, Miami
Labs: Operating Systems, Computer Data Analysis, Introduction to Micro-computers and Computer Applications for Business.

AWARDS, HONORS AND FELLOWSHIPS

Dissertation Year Fellowship, Florida International University (FIU), 2008-2009.

Presidential Fellowship, School of Computing and Information Sciences(SCIS), FIU.

Outstanding Graduate Research Award, SCIS, FIU, 2007.

Excellence Award, SCIS, FIU, 2006-2007.

Student & New Researcher Travel Support, SIGIR 2006.

Graduate Committee Travel award for IEEE ICDE 2008 & ACM CIKM 2006, SCIS, FIU.

Travel award for IEEE ICDE 2008 & SIGIR 2006, Graduate Student Association - FIU.

University Gold Medal and Shield from University of Madras, Chennai, India. Program: B.E Computer Science and Engineering. Rank: 5th in the University (First Class Honors) - Percentage Score: 90%.

PUBLICATIONS

Ramakrishna Varadarajan, Vagelis Hristidis, Louiqa Raschid, Maria-Esther Vidal, Luis lbanez and Hector Rodriguez-Drumond: "Flexible and Efficient Querying and Ranking on Hyperlinked Data Sources", Extending Database Technology (EDBT) 2009, Saint-Petersburg, Russia. (Acceptance rate – 32.50% Impact factor – 0.90).

Ramakrishna Varadarajan, Vagelis Hristidis and Louiqa Raschid: "Explaining and Reformulating Authority Flow Queries", IEEE 24th International Conference on Data Engineering (ICDE) 2008, Cancun, Mexico. (Acceptance rate – 19% Impact factor – 0.97).

Ramakrishna Varadarajan and Vagelis Hristidis: "A System for Query-specific Document Summarization", ACM 15th Conference on Information and Knowledge Management (CIKM) 2006, Arlington, VA, pages 622-631. (Acceptance rate – 15% Impact factor – 0.90).

Ramakrishna Varadarajan, Vagelis Hristidis and Tao Li: "Searching the Web using Composed Pages" (poster paper), ACM SIGIR Conference on Research and Development on Information Retrieval 2006, Seattle, WA, pages 713-714. (Acceptance rate – 37% Impact factor – 0.94).

Ramakrishna Varadarajan and Vagelis Hristidis: "Structure-Based Query Specific Document Summarization" (poster paper), ACM 14th Conference on Information and Knowledge Management (CIKM) 2005, Bremen, Germany.

Ramakrishna Varadarajan, Vagelis Hristidis and Tao Li: "Beyond Single-Page Web Search Results", IEEE Transactions on Knowledge and Data Engineering (TKDE) 2008.