Effective Searching in Structured Data

A Thesis
Presented to
the Faculty of the School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the requirements for the Degree
Master of Science (Computer Science)
by
Joel Coffman
May 2009
Abstract

Keyword search is the mechanism of choice for information discovery and retrieval due to the enormous success of Internet search engines. In fact, nearly half of Internet users perform at least one search daily. The keyword search paradigm regretfully does not extend to similar forms of content, particularly semi-structured and relational data. Searching structured content is difficult because structured data challenges important assumptions of scoring models developed for unstructured text. First, a complete document may not provide the correct granularity for search results—it may be too coarse (that is, containing irrelevant information) or too fine (that is, lacking context). Second, the relationships among data must be considered, for the correct interpretation of structured data depends on understanding its relationships. Searching relational data is particularly difficult due to the data normalization process that guides the design of relational databases. Although a number of systems have been proposed to remedy this situation, none have met with considerable success. The search techniques described in this thesis apply to both semi-structured and relational data. The key contributions include an intuitive scoring model for scoring related information and a novel hybrid indexing scheme. The intuitive scoring model allows reuse of existing weighting models developed for unstructured text collections; reusing these models is critical due to the extensive testing they have already seen. Also included in the scoring model is 1) an innovative score normalization that blurs the distinction between unstructured text and information connected by relationships and 2) a factor that recalibrates the raw score produced by the weighting model so that the final score adheres to users' expectations. The hybrid indexing scheme practically eliminates data redundancy by exploiting the full-text indexing capabilities of the underlying relational database. A small data graph stored in main memory allows efficient graph traversals to identify related information. Evaluation on two widely different datasets demonstrates the effectiveness of the techniques.
Approval Sheet

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science (Computer Science)

________________________________________________________________________

Joel Coffman

This thesis has been read and approved by the Examining Committee:

________________________________________________________________________

Alfred C. Weaver, Thesis Adviser

________________________________________________________________________

Worthy N. Martin, Committee Chair

________________________________________________________________________

James C. French

Accepted for the School of Engineering and Applied Science:

________________________________________________________________________

Dean, School of Engineering and Applied Science
May 2009
To Mother and Dad
Acknowledgements

To be upfront, I could undoubtedly fill another book with the number of people who deserve recognition, but I must content myself just mentioning a few. First, I must thank my adviser, Dr. Alfred C. Weaver, for his patience and guidance throughout the past two years. Without his encouragement and leadership, I would not be here now. Special thanks to the National Technology Alliance and the National Geospatial-Intelligence Agency for supporting this work, and I would be remiss not to mention Bob Dorsey for his ideas and feedback. Andrew, Colleen, Claire, Dan, and all of the other occupants of Olsson 227, thank you for your help, encouragement, and, on occasion, distracting conversations that keep grad school entertaining.

Mother and Dad, thank you for the love, support, and encouragement you have given me. I know you have spent countless hours in prayer on my behalf, and I do not doubt that your prayers have carried me through both the difficult and the easy times. Andrew, thank you for always being there for me and setting such a godly example as an older brother. Although I may never know everyone else’s name who has prayed for me during the past two years, this thesis is evidence of your investment in my life and the power of prayer. Praise be to the LORD who deserves all glory and honor and praise! He gives me the strength to face each tomorrow, and without him, I could do nothing.

Praise be to the God and Father of our Lord Jesus Christ! In his great mercy he has given us new birth into a living hope through the resurrection of Jesus Christ from the dead, and into an inheritance that can never perish, spoil or fade […]. (1 Peter 1:3–4)
## Contents

### Abstract

### Acknowledgements

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
</tbody>
</table>

### Symbols

### 1 Introduction

1.1 Keyword Search

1.2 Crosspoint

1.2.1 Requirements

1.2.2 Service-Oriented Architecture

1.3 Contributions

1.4 Organization

### 2 Related Work

2.1 Information Retrieval

2.2 Search Engines for Semi-structured Data

2.3 Relational Search Engines

2.4 Searching Data Graphs

### 3 Searching Structured Data

3.1 Formal Problem Description

3.1.1 Data Graph

3.1.2 Queries and Answers

3.1.3 Semantics

3.2 Scoring Answers

3.2.1 Problems with Existing Solutions

3.2.2 Tuples

3.2.3 Related Tuples

3.3 Framework

3.3.1 External Index

3.3.2 Search Algorithm

---

v
## Contents

4 Evaluation ................................. 48
   4.1 IR Effectiveness .......................... 48
   4.2 Datasets and Queries ....................... 49
   4.3 Results .................................. 53
      4.3.1 IMDb .................................. 54
      4.3.2 WikipediaCD ............................. 56
   4.4 Parameter Values ........................... 58

5 Conclusions and Future Work .......... 61
   5.1 Conclusions ................................ 61
   5.2 Future Work ................................ 62

Bibliography ................................ 65

Acronyms .................................... 70

Glossary ..................................... 71
# List of Tables

1.1 Crosspoint’s database schema .......................................................... 13
4.1 IMDb schema ................................................................................. 50
4.2 IMDb queries ................................................................................ 51
4.3 WikipediaCD schema ...................................................................... 52
4.4 WikipediaCD queries ...................................................................... 52
List of Figures

1.1 Crosspoint RFIs ......................................................... 7
1.2 Crosspoint SME profile .................................................. 8
1.3 Crosspoint team .......................................................... 10
1.4 Crosspoint search results ............................................... 11
1.5 Crosspoint’s SOA .......................................................... 12

3.1 Movie database data graph ............................................. 30
3.2 Tree-structured search results ......................................... 32
3.3 Minimum spanning tree and plot synopsis search result .......... 35

4.1 Authority Transfer Graphs .............................................. 53
4.2 IMDb Top-1 Relevant results ........................................... 54
4.3 IMDb R-Precision .......................................................... 55
4.4 IMDb MAP ................................................................. 56
4.5 WikipediaCD Top-1 Relevant results .................................. 57
4.6 WikipediaCD Reciprocal Rank ......................................... 58
4.7 WikipediaCD R-Precision ................................................ 59
4.8 WikipediaCD MAP graph ............................................... 60
Symbols

*avgdl* **average document length** The average length of documents in the complete document collection.

*df* **document frequency** The number of documents, measured across the entire document collection, that contain the specified term.

*dl* **document length** The length of a given document.

*idf* **inverse document frequency** A factor common to many information retrieval weighting methods, inverse document frequency awards documents that contain uncommon terms that are present in the search query.

*ndl* **normalized document length** Since longer documents contain more terms than shorter documents, modern IR weighting methods normalize document length to counteract this effect.

*ntf* **normalized term frequency** Using raw *tf* in weighting methods is suboptimal since users expect documents containing more query terms to rank higher than documents containing fewer; to this end, the *tf* factor is dampened, e.g., a logarithmic *tf* factor.

*tf* **term frequency** The frequency of a given term in a document.
Chapter 1

Introduction

In the 1945 article “As We May Think,” Vannevar Bush [1] envisioned the “memex,” a device capable of storing an individual’s private library and providing instant access to its information. A user creates trails through the material (books, articles, personal correspondence, etc.) where each trail expresses an association among the content. Trails enable selection by association instead of using partial indexes over the title, author, or subject of each work contained in the library. While short of the associative trails imagined by Bush, later work did obviate the partial indexes for written material. Advances in computer systems allowed automatic indexing and retrieval of documents using their complete text. H.P. Luhn [2] laid the groundwork for modern information retrieval by proposing words as the atomic unit to index documents and by using statistical methods to compare search queries to the indexed documents.

Information retrieval (IR) saw many improvements—especially indexing and scoring documents—over the next thirty years. The explosive growth of the Web throughout the 1990s introduced new challenges as its size and rate of growth are unprecedented in human history. In 2008, Google reported 1 trillion unique URLs in its index [3]. Online content has revolutionized the ways people interact with and obtain information. This chapter describes keyword search, which is now the preferred method of information discovery and retrieval, and the difficulty in extending the keyword search paradigm to structured data sources. The Crosspoint system (introduced in section 1.2) serves as a case study to demonstrate how a relational search engine may be used in a variety of contexts, including expert-finding. The thesis of this work is simple: instead of creating new weighting models that specifically target structured data and lack exhaustive evaluation, effective retrieval of structured data can proceed using existing IR weighting models and an intuitive scoring
model, both of which mitigate the immediate importance of extensive user studies. The search techniques developed in this paper are appropriate both to structured data sources and to specialized applications, which are illustrated by expert-finding in the Crosspoint system.

1.1 Keyword Search

Internet search engines have made keyword search the preferred method for users to discover and to retrieve data [4]. Few search engines provide an alternative to the common query box and results list [5] even though Border [6] reports at least three distinct forms of information needs. In spite of the lack of specialized user interfaces, Pew Internet Project memos reveal that nearly half of all Internet users use a search engine daily [7] and that 84% of Internet users have used a search engine at some point [8]. The wild success of keyword search stems from what it does not require—namely, a special query language or knowledge of the underlying structure of the data.

Existing Internet search engines crawl the static web and index page content. The process is similar to indexing a normal document corpus, albeit on a much larger scale. The static content alone is on the order of 400 terabytes [9]. In spite of exhaustive coverage (eventually every URL encountered will be crawled), search engines cannot handle dynamic web content. Dynamic content, which is served on-demand from various data repositories (e.g., relational databases), is not crawled because it normally requires authorization to view. Bergman [10] estimates that the “deep web,” which serves the dynamic content, is several orders of magnitude larger than the static web. Today’s largest data warehouses, which are responsible for financial activity, inventories, and personal information, now contain petabytes of information [11]. Users cannot be expected to learn the schema and (potentially specialized) query languages of these different data sources, and the success of the keyword search paradigm suggests that it would provide an ideal way for naive users to access the information stored in these repositories.

Transitioning the keyword search paradigm to structured content is difficult. Both semi-structured and structured\(^1\) data question a basic premise of IR: what is the correct granularity of query results? For example, one common format for semi-structured data is the eXtensible Markup Language (XML). Because XML is normally structured hierarchically, returning an entire document as a search result is often inappropriate:

\(^1\)This paper uses the terms structured data and relational data interchangeably. Structured data does differ from structured documents, which are a form of semi-structured data. For example, a conference paper is a structured document because it contains well-defined sections—abstract, introduction, etc. For additional details regarding structured documents, see section 3.2.2.
better result might be a single element. Relational data further complicates matters, for the relational model imposes additional structure on information. Database normalization protects against data anomalies (that is, the data must be consistent and correct) by eliminating redundancy. The process normally introduces additional relations and relationships to link the data. Hence, relationships among data must be considered because a single data source (i.e., a tuple) is unlikely to match many query keywords. Coalescing a number of data sources creates a “virtual document,” which serves as a better result for search queries. In addition, the virtual documents may be ranked using traditional IR scoring formulas, which may provide counter-intuitive results when documents match few keywords.

Modern relational databases also provide full text search capabilities. Database vendors have extended SQL to allow the specification of full text search conditions, but users must still know a query language and the schema before writing queries. In contrast, the construction of virtual documents allows a user to search using the familiar keyword search paradigm. The database’s underlying indexes provide exact and partial matches for a given query. These fragments of results are joined before a ranked list of results is returned to the user. In essence, this solution maintains the desirable qualities of keyword search at the cost of increased execution time when compared with pure SQL.

In recent years, a number of researchers began extending the keyword search paradigm to semi-structured and relational data. None of these approaches saw widespread adoption. Existing search engines appear to be good enough for most tasks. Independently developed search engines handle content requiring authorization, or companies purchase solutions from other companies specializing in search technology (e.g., Google’s Search Appliance [12]). The extension of the keyword search paradigm to a variety of sources is the primary goal of this work. Developing a search engine for relational data is the first step in this direction. The proposed search techniques neither require \textit{a priori} knowledge of the data sources nor continue to rely on the assumption that individual documents provide the correct granularity for search results.

Existing IR scoring formulas form the basis of the proposed search techniques. This foundation ensures the search engine will handle unstructured documents as well as structured documents. The extension of the IR scoring formulas is natural: existing parameters are unchanged, and unstructured data sources receive a score identical to that of the original formula. Because databases already provide efficient indexing, external indexes are largely unnecessary. Not creating external indexes prevents data redundancy, and indexes are always up-to-date. Existing standards (e.g., SQL and JDBC) allow the search engine to interface with the numerous relational database systems on the market with a minimum of effort. In fact, only non-standard
1.2 Crosspoint: A Web-Based Collaboration System

Collaboration has been well-studied, and a variety of collaboration tools exist. These tools range from stand-alone solutions (e.g., email, chat, instant message, whiteboards, and videoconferencing) to complete environments that provide their own integrated communication tools (e.g., Microsoft Groove and IBM Lotus Sametime). The first group of products clearly possesses an Achilles’ heel: they operate as independent entities with limited mechanisms for discovery, access, and interaction among users. Essentially, these stand-alone tools assume that the user already knows the identities of those with whom to collaborate and require the user to identify collaborators by name or by group affiliation. Oftentimes, users know the desired expertise or experience of potential collaboration partners, but these tools do not provide a means for creating collaboration sessions using this information. Even the complete collaboration environments do not promote themselves as expert-finding systems, which are capable of recommending the best collaborators for a given problem.

The problem of identifying collaborators when solving a problem is not esoteric—it applies to many fields. For example, analysts in the intelligence world are often called upon to answer questions outside of their area of requisite expertise. New analysts initially experience great difficulty because they lack a network of domain experts, all of whom contribute to the eventual solution. The lack of personal contacts limits the effectiveness of the new analysts, even in situations where these analysts are better qualified than more experienced analysts who merely draw on a larger network of specialists. The medical profession provides another example of this phenomenon. Telemedicine and triage centers are increasingly common. In the former, patients describe their symptoms via telephone (or another collaboration tool) for the purpose of consultation or even remote medical procedures. Triage centers evaluate patients based on the severity of their condition, and treatment follows a specific pattern to ensure as many patients as possible are treated. In either case, it is paramount that patients receive the right diagnosis, which in certain cases warrants referral to or additional consultation with a specialist. Both of these objectives are hampered by a limited network of personal contacts. For example, the triage nurse might send a patient to a doctor specializing in general medicine instead of recognizing that another doctor—whose exact expertise is unknown to the nurse—actually specializes in an area related to the patient’s injury. Even doctors who are closely connected
by an organization (i.e., those working at the same hospital) may not know the specialties of all of their associates, especially when the organization is large or the turnover rate is high. Both of these examples demonstrate that the problem of identifying collaborators spans multiple disciplines and that the problem is of great importance.

Crosspoint provides a user-friendly environment for problem holders and experts alike. Problems are described from the problem holder’s perspective, and the problem holder also identifies the characteristics of the desired outcome. For the intelligence community, an example request might resemble the following:

Troops under my command are penned down in a village due to heavy fire from the surrounding hills. Intelligence reports did not indicate militant activity in the area so the fire may be friendly, coming from another U.N. Peacekeeping team. I need to know if friendlies are in the area, and if so, I need an open communication channel with them.

The analyst assigned to this problem must first identify the relevant details in the request and perhaps ask additional questions to clarify the situation (e.g., Where are the troops? Which village is referred to by the commander?). Next, the analyst must assemble a collaboration team to address the problem and to provide a prompt reply to the request. To assist in the formation of the collaboration team, Crosspoint’s query service locates the experts possessing the required skills and expertise. The analyst undoubtedly needs to communicate with additional troop commanders to identify the locations of all friendly troops in the area. In addition, the analyst should identify other analysts who have recently handled problems in the same geographic region: who authored the intelligence reports about the dearth of militant activity? These different search criteria may be prioritized to increase the relevance of search responses. For example, the analyst receives no benefit from search results regarding militant activity in other geographic locations.

User-specified profiles, which allow experts to reveal details regarding their personal expertise, facilitate the search process.

Many of these individual issues are already present in a variety of commonly used social networking applications. Information sharing, collegiality, group formation, and sense of shared purpose are commonplace in online social networking, but few existing tools address the range of issues that Crosspoint targets. Crosspoint’s goal is simple: simplify collaboration by identifying potential collaborators through a user-friendly environment and expert-finding tools. To facilitate rapid adoption, Crosspoint is based upon a Service-Oriented Architecture and is implemented using web services. Both design decisions avoid the creation of yet another “stovepipe” or independent application. The remainder of this section describes the specific
requirements and architecture of the Crosspoint system.

1.2.1 Requirements

Crosspoint’s design reflects the lessons learned from an extensive study of existing social networking sites. From an initial list of 25, eight—eBay, eHarmony, MySpace, Facebook, Elance, del.icio.us, YouTube, and Wikipedia—were selected for deeper study. These sites exemplify the fact that a commercially successful social networking site represents a win-win proposition between the user and the provider: both the user and the provider must gain something of value for the site to succeed and grow. The study itself falls outside the scope of this work, but Crosspoint’s requirements reflect many of its findings. As evidenced by recent reports, the impact of social networks is expanding at a rapid rate. In the past four years, the number of adults maintaining an online profile quadrupled [13], and within the past year, the percentage increase of Internet users accessing a social network or blog site grew at twice the rate of any other common online activity [14]. In fact, social networking and blogging are now more prevalent than email! These findings all confirm the importance of social collaboration to today’s society.

**Request for Information Service**  Users identify their problems using an electronic form, the Request for Information (RFI). RFIs identify the problem holder and permit a plain text specification of the problem description, the type of result needed, the importance of the request, and the deadline for a response. In a nod to the success of YouTube, Crosspoint provides a simple way for users to include multimedia content (e.g., pictures, audio, video, or other files) when describing a problem. Each RFI is time-stamped upon submission to the Crosspoint database. Analysts view and modify RFIs to improve their specificity following interaction between the analyst and the problem holder.

**Subject Matter Expert Profiles**  Subject Matter Experts (SMEs) are identified by their electronic profiles. Crosspoint displays user profiles using a standardized interface in an effort to simplify the process of locating relevant details about potential collaborators. Profiles permit specification of an expert’s identity, resources, specialization, and experience. Experts identify their specialization at two levels: a general characterization of the specialty chosen from an established ontology and a more detailed specification provided using plain text. The description of the SME’s experience is also provided using plain text. Although there is always the risk that profiles maintained by individuals will become outdated, a recent study [13] of online
Figure 1.1: An example listing of RFIs submitted using Crosspoint. RFIs are sorted by priority and then by date to ensure the most urgent requests are handled first.

Social networking sites show that more than one-third of users visit their profile daily and more than 75% visit it at least once a week. Visiting a profile does not necessarily imply that the profile is up-to-date, but...
Figure 1.2: An illustrative SME profile from Crosspoint. The current user (in this case John Ryan) may modify his profile to ensure it contains up-to-date and relevant information. Other users do not have the ability to alter John Ryan’s profile via the edit and delete links.

It seems reasonable that users deriving value from the site (as evidenced by their frequent logins) will be motivated to keep their personal information updated. Automated systems for maintaining profiles (e.g.,

<table>
<thead>
<tr>
<th>Personal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Last name</td>
<td>Ryan</td>
</tr>
<tr>
<td>First name</td>
<td>John</td>
</tr>
<tr>
<td>Middle name</td>
<td>Patrick</td>
</tr>
<tr>
<td>Organization</td>
<td>Central Intelligence Agency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contact</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Annapolis, MD</td>
</tr>
<tr>
<td>Telephone</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:jack.ryan@cia.web">jack.ryan@cia.web</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject Matter Expert</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Specialization</td>
<td>Tactics</td>
</tr>
<tr>
<td>Detail</td>
<td>Marine Lieutenant (retired), B.S. Economics from Boston College, Certified Public Accountant (CPA), broker for Merrill Lynch, Ph.D. History from Georgetown University</td>
</tr>
<tr>
<td>Resources</td>
<td>contacts in both military and civilian world</td>
</tr>
<tr>
<td>Experience</td>
<td>briefly worked for the Center for Strategic and International Studies; professor of history, U.S. Naval Academy; consultant for CIA (employed by MITRE corporation)</td>
</tr>
</tbody>
</table>
harvesting content from email or chat logs as is done in the SmallBlue system [15]) is undoubtedly the most effective way to counter the risk of outdated profile information.

Like Crosspoint, both eHarmony and Facebook solicit personal information (some of which might be considered private), but Facebook allows users to omit information from their profile. For example, the “political views” field may be left blank so the individual’s profile page will not divulge this information. Crosspoint adopts this approach because Crosspoint allows users to search the database directly to locate collaborators. If information is available to the search engine but not to all other users (e.g., an individual’s affiliation with the CIA is marked as private so other users cannot see it when viewing the profile page), search results could subtly leak the private information (e.g., the query “affiliation:CIA”).

**Teams**  When a single individual does not possess all of the information necessary to respond to a problem, the analyst forms a team to facilitate collaboration and to derive the problem solution. The same team should handle recurrent information needs because the team members possess the most current information regarding the problem and already have working relationships. Even if the information need is not recurrent, a similar problem may arise at a future point, and Crosspoint’s search engine should identify the original team members as ideal collaborators.

**Feedback**  eBay’s unprecedented success with online auctions stems from a simple concept—trust. Trust enables buyers to overcome their initial reluctance to send the seller money without assurance that the seller will fulfill his end of the contract. Electronic reputations serve as a surrogate for trustworthiness in these interactions. Collaboration also requires cooperation and trust among participants and often reflects offline organizational and social contexts [16]. Crosspoint adopts eBay’s strategy of allowing users to rate their personal interactions. When SMEs participate in a team, they provide feedback on their collaborators. The feedback ratings form each SME’s electronic reputation within the Crosspoint community.

**Search Engine**  The key element of Crosspoint is the operation and functionality of its search engine. Analysts submit search queries to identify other SMEs who can contribute to the problem solution. Keyword search is the preferred query method because the ubiquity of the Internet guarantees that people already have experience with keyword search and do not require training to use it effectively. The user interface is modeled after the Google search interface.
Chapter 1  |  Introduction

1.2.2 Service-Oriented Architecture (SOA)

Crosspoint’s adherence to a SOA ensures that any authorized web client can access its services. The system architecture is shown in figure 1.5, and the following paragraphs describe each component in detail.

User  Any individuals who use the Crosspoint system, including problem holders, analysts, and subject matter experts, are classified as users. All users must register before using the system. When a user submits an RFI, contact information is inferred from the user’s profile, which limits the amount of additional detail that must be entered. The limited information required to complete the registration process (i.e., name,
Figure 1.4: Sample results for the query “oil fire.” Results are either teams (e.g., “Oil fire (CAT)”), RFIs (e.g., “Oil fire: Al Anbar, Iraq”), or SMEs (e.g., “Tracy Harrison”). A brief snippet of each result’s information is provided; clicking the link will display the team, RFI, or SME profile page as appropriate.

... (continued from previous page)

... email, and password) is no more than the detail required to contact the user when the problem solution is identified.
**User interface**  The web-based user interface (UI) provides a standardized portal to view information stored within the Crosspoint database and also serves as the gateway to the SOA service engine and search engine. The database stores all information pertinent to Crosspoint including user contact information, SME profiles, RFIs (both active and archived), and collaboration teams. Before gaining access to the information, users must first authenticate. For simplicity, Crosspoint handles authentication using email addresses and passwords, but many other methods of authentication are clearly feasible.

**SOA Service Engine**  The SOA service engine intermediates between the UI and external collaboration tools (e.g., Microsoft Groove or IBM Lotus Sametime). External collaboration environments often provide web-based “lightweight” clients which can be invoked through an ordinary web browser. The SOA service engine maintains separate modules for each external collaboration tool; each module specifies how to create and launch a collaboration session. Naturally, users can opt to store login information for each collaboration tool so that their information will not have to be reentered each time a collaboration session is launched.

**Search Engine**  Crosspoint’s architecture defies the integration of an existing search engine for two reasons. First, Crosspoint’s content requires authorization to view, and each page is dynamic, which reduces the effectiveness of third-party search appliances. Second, modern search engines implicitly enforce strict AND semantics to limit the number of results returned to users, but conjunctive queries fail to return any results when all search terms are not present, leaving the analyst with no search results at all and no indication of
1.3 Contributions

The core contributions of this work stem from an extensive review of similar systems previously developed by researchers. As evidenced by the review of related work in chapter 2, these systems provide a wide variety of alternative designs for indexing data and for scoring search results. A number of previous works have modified existing IR scoring functions to adapt them to specific types of data (most commonly relational). Such solutions should be avoided for at least two reasons. First, proper evaluation of scoring functions requires a significant number of experiments executed on standardized test collections. Unfortunately, there are no standard datasets for evaluating these search engines, which results in ad-hoc experiments on a variety of datasets and limits the ability to compare systems. Second, no published work about keyword search in structured data incorporates formal user studies to determine the true effectiveness of the proposed techniques. User studies are essential for any system designed to extend beyond the academic community. The major contributions presented in this thesis are as follows:

<table>
<thead>
<tr>
<th>Table 1.1: The schema of the Crosspoint database. Bold text denotes the primary key of each table, foreign keys are displayed in italics, and full text indexes are built over underlined attributes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
</tr>
<tr>
<td>User</td>
</tr>
<tr>
<td>SME</td>
</tr>
<tr>
<td>RFI</td>
</tr>
<tr>
<td>Team</td>
</tr>
<tr>
<td>Affiliation</td>
</tr>
<tr>
<td>Feedback</td>
</tr>
</tbody>
</table>

how to modify the query to correct this problem. The search techniques described in this thesis overcome both of these difficulties.

Database As previously stated, the Crosspoint database contains user contact information, SME profiles, RFIs, and collaboration teams. The only requirement is that the relational database supports full text search, which is essential for efficient searching. To encourage portability, the prototype system uses PostgreSQL. Core database functionality is abstracted into modules to allow the system to interface with other relational database platforms on the market (e.g., MySQL, Oracle, and DB2).
• The design of the search engine benefits from previous insights as it incorporates and improves the best approaches among the numerous alternatives. Unlike many previous efforts, the search engine is not limited either to semi-structured data or to relational data; instead, more general insights allow the search engine to handle both types of content (although the discussion in this paper focuses almost exclusively on relational data).

• One of the most ignored parts of previous work is the display of search results. While the solution presented in this thesis is not perfect, it mitigates many of the problems easily identified with similar systems and represents a starting point for future work in the visualization of search results.

• This work uses an intuitive scoring model, which allows existing IR scoring formulas to be used with relational data. The scoring model minimizes alterations to the original scoring functions, which have already undergone extensive testing. Given that the IR weighting models developed for unstructured text have already seen both extensive testing on standard test collections and extensive user studies to determine their effectiveness (e.g., at the Text REtreival Conference (TREC)), the approach reduces the importance of extensive evaluation, at least for the initial system.\(^2\)

• A novel scoring normalization blurs the distinction between structured and unstructured content so the search engine easily adapts to different data types. The two evaluation datasets illustrate the versatility of the search engine in this regard.

• The overhead of the search index has been almost completely ignored in previous efforts. The hybrid index proposed in this thesis minimizes the duplication of data retrieved from the underlying database. An in-memory data graph allows efficient access to determine the relationships among tuples, and because keywords are not tracked explicitly by the hybrid index, the index remains up-to-date far longer than comparable approaches. Previous systems either assume efficient indexing by the underlying relational database system (e.g., systems exclusively targeting relational databases) or do not consider information duplication and the complexities of keeping the index synchronized with the underlying data.

\(^2\)Unlike previous works, which are silent on this matter, extensive user studies are identified as a critical part of future work for the search engine.
1.4 Organization

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of related work. Basic IR concepts are covered first before turning attention to keyword search engines for XML and relational databases. The chapter concludes with a survey of other search engines that target keyword search in arbitrary data graphs. The design and implementation of the search engine is provided in chapter 3. A formal model—robust enough to handle both semi-structured and relational data—is introduced along with precise definitions for query answers and query semantics. The scoring model builds upon these definitions and is based on an intuitive physical model. The implementation section covers the design of specific data structures and the algorithms used by the search engine. An evaluation of the effectiveness of the search engine is given in chapter 4. The evaluation covers two dramatically different datasets—Internet Movie Database (IMDb) and WikipediaCD—to illustrate the wide applicability of the scoring model. Traditional IR evaluation metrics show the search engine’s admirable performance. Finally, chapter 5 contains conclusions and outlines future work.
Chapter 2

Related Work

According to Weikum [17], the separation between databases and IR systems is historical, stemming from radically different target users. Database systems provide exact results to specific queries as quickly as possible. In contrast, a basic premise of IR systems is that users’ queries are approximate, to be reformulated until the desired information is discovered. Hence, database systems process queries using strict logical (i.e., boolean) predicates while IR systems see queries as ranking tasks. When searching for information, the benefits of keyword search cannot be understated—even experienced SQL users discover information more quickly with an unstructured keyword search than with SQL [18]. Within the past decade, researchers published a variety of approaches for extending keyword search to semi-structured and relational data. To understand their techniques, this chapter opens with a brief review of IR concepts. Next, a number of search engines specific to XML (semi-structured) data are described before examining keyword search engines explicitly targeting relational databases. Graph-structured techniques, which are applicable to both types of data, are covered last.

2.1 Information Retrieval

As mentioned briefly in chapter 1, Vannevar Bush [1] was the first to imagine automatic information retrieval. H.P. Luhn [2] laid the groundwork for statistical analyses to compare search queries with document content by suggesting words as the basic indexing unit. In the 1960s, the development of the SMART system [19] allowed IR researchers to experiment with competing ideas for improving search quality, and the Cranfield
Information Retrieval

tests [20] provided the first formal evaluation methodology for IR systems. The following two decades saw a number of advances across IR systems, but researchers questioned whether the models would scale to large text collections. In 1992, the National Institute of Standards and Technology (NIST) founded TREC to provide standardized evaluation methods using large-scale text collections. As a result, IR system effectiveness approximately doubled within the next six years [21]. The IR community uses a number of different techniques and models in their systems. This section provides a brief overview of the most influential developments in IR systems and web search, especially those pertaining to this work. Singhal’s overview [22] provides more complete coverage of the material.

Term weighting lies at the heart of modern IR scoring formulas. Scoring formulas typically have three major factors, which follow from an intuitive notion of scoring documents. Term frequency, $tf$, is the frequency of a given term in a document. Many instances of a term in a document generally imply relevance. IR models do not provide a technical definition of a term—the systems or models are free to define a term as individual words or phrases. Some systems use stemming—conflating the various forms of a word to a common root—in an attempt to improve search effectiveness. For example, fire, fires, fired, and firing all conflate to the root fire. Document length, $dl$, is the second major factor in modern IR scoring formulas. It may be measured in a variety of ways, including size in bytes, number of unique terms, or total number of terms. The last factor, document frequency, $df$, is the number of documents in the collection that contain a specified term. Common terms are not good discriminators between documents and are unlikely to reflect a document’s content. Many IR systems do not index stop words, which are the most commonly occurring words in a language. In English, the list typically includes a, an, the, in, of, etc. Scoring formulas often include $N$, the total number of documents in the collection. Term frequency and document length are usually normalized ($ntf$ and $ndl$) in scoring formulas, and because common terms are not good discriminators of content, inverse document frequency, $idf$, reflects this property. In the formulas described below, $t$ refers to a term, $Q$ to the search query, $D$ to a document in the collection, and $qtf$ to the term’s frequency in the query.

The vector space model measures the similarity between a query vector and a document vector. Terms become independent dimensions in these vectors. Both the cosine of the angle and dot-product between the two vectors are commonly used to measure similarity. Pivoted normalization weighting [23] is a common
Chapter 2  |  Related Work

scoring function derived from the vector space model:

\[
\sum_{t \in Q \cap D} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s \cdot \left(\frac{dl}{\text{avg}dl}\right)} \cdot qt \cdot \ln \left(\frac{N + 1}{df}\right)
\]  

(2.1)

where \( s \) is a constant.

In probabilistic models, documents are ranked in decreasing probability of their relevance to a query. In an ideal system, the probability of each document satisfying the query’s information need is known exactly. In reality, this information is not available to the system—it must be estimated. Okapi weighting [24] is the most prevalent scoring function from the probabilistic retrieval model:

\[
\sum_{t \in Q \cap D} \frac{(k_1 + 1)tf}{k_1(1 - b) + b \left(\frac{dl}{\text{avg}dl}\right) + tf} \cdot \frac{(k_3 + 1)qt}{k_3 + qt} \cdot \ln \left(\frac{N - df + 0.5}{df + 0.5}\right)
\]

(2.2)

where \( k_1, k_3, \) and \( b \) are all constants.

Fang et al. [25] formalize the six constraints that IR scoring formulas should satisfy. Term frequency constraints are intuitive: documents containing a query term more frequently should score higher than documents containing the term less frequently, and the marginal benefit of additional instances of query terms is decreasing (i.e., the score increase for the 3rd instance of the query term is less than the increase for the 2nd). Term discrimination favors documents containing terms that occur less frequently in the collection. Length normalization penalizes long documents but avoids over-penalizing documents (i.e., one composed of concatenating a shorter document with itself multiple times). Term frequency and document length interact to form the final constraint: a document with additional instances of a query term should score higher than a shorter document as long as their difference in length is not too great. Interestingly, no existing IR scoring formula satisfies the aforementioned intuitive constraints. Pivoted normalization scoring is sensitive to its tuning parameter, \( s \), whose value (determined experimentally) should be less than 0.4. (Singhal [22] suggests \( s \) be fixed at 0.2.) Higher values violate the second length normalization constraint by over-penalizing large documents. Okapi scoring satisfies all constraints as long as the \( \text{idf} \) factor is non-negative. A modified Okapi formula, which replaces the original \( \text{idf} \) factor with the pivoted-normalization’s \( \text{idf} \) factor, unconditionally satisfies all but the term discrimination constraint. Experimental results confirm the formal analysis and suggest that pivoted-normalization scoring actually outperforms Okapi scoring when \( s \leq 0.2 \).

IR systems are typically evaluated using recall and precision. Recall is the proportion of relevant doc-
2.1 Information Retrieval

uments returned; precision is the proportion of returned documents that are relevant. Both precision and recall require relevance judgments for each document in the collection and do not consider the task of ranking documents. To put it differently, these metrics are derived from set theory and do not consider a document’s position in a list of results. Large studies typically use pooled relevance judgments as the gold standard for evaluation because only users can determine if a given document satisfies the query’s underlying information need. Because improvements to one measure are likely to impact the other negatively, a number of metrics have been suggested for combining these two concepts. Average precision measures the precision at different recall points, and 11-point precision and recall measures precision at 11 specific recall values (0, 0.1, \ldots, 1.0). Section 4.1 presents additional measures of effectiveness that are used to evaluate this work.

The growth of the web has spurred a great deal of work investigating search trends and user expectations for search results. When designing the V-Twin system [26], Rose and Stevens examined the query logs of a web prototype and found that users tend to submit short queries containing three words or fewer. Query logs from other search providers confirmed the trend. Overall, 87% of queries contained one to three terms. For short queries, users expect documents containing all terms to rank higher than documents containing a subset of the search terms. Users do not want strict AND semantics; they simply want documents containing more search terms to rank ahead of documents containing fewer. Rose and Stevens introduce a function that transitions from ranking according to coordination level to a traditional IR weighting function as query length increases. The transition function ensures that the ranking of results adheres to users’ expectations.

Rose and Stevens’s findings confirm the earlier work of Wilkinson et al. [27] who theorized that ad-hoc queries submitted to online text retrieval systems would fail to produce satisfactory results using the common IR scoring functions. The reason behind the degradation in effectiveness is simple: TREC queries contained, on average, 77 terms while ad-hoc queries only include 2–10 terms. For these short, ad-hoc queries, the cosine measure of document similarity performs very poorly. In contrast, Okapi scoring, which incorporates a form of coordination matching, favors documents containing all query terms.

Jansen et al. [28] provided further corroboration of web search trends by examining a set of more than 50,000 queries submitted to the Excite search engine. While IR systems normally receive queries containing 7–15 search terms, the queries submitted to Excite contained an average of 2.35 terms (although the Excite search engine limited queries to ten terms). Eighty percent of searches used three or fewer keywords, and most users did not bother examining any search results beyond the first page. Later work by Wolfram et al. [29] and Spink et al. [30] confirmed the trends.
In response to these findings, Clarke et al. [31] proposed a new ranking technique—\textit{cover density ranking}. Cover density ranking first orders documents by coordination level and then by term proximity. This ranking scheme does not directly incorporate the traditional IR factors (term frequency, document length, and document frequency). While Clarke et al. do not envision cover density ranking being applied to queries longer than three words, its independence from collection-wide statistics and efficient ranking make it ideal for distributed or parallel systems.

### 2.2 Search Engines for Semi-structured Data

Wilkinson [32] was among the first to investigate search effectiveness for hierarchically-structured documents. He found that document retrieval is not negatively impacted when statistics from the document’s constituent sections are used in lieu of statistics from the entire document, but the converse does not hold—relevant sections of a document cannot be discovered using the entire document’s statistics. Thus, collecting section statistics is critical when complete documents are too coarse-grained for search results. Combining section and document statistics is naturally the most effective way to rank documents.

Initial XML query languages retrieved documents either by structure or by content; little work focused on integrating the two techniques. Fuhr and Großjohann [33] extended the XML query language (XQL) with term weighting, relevance ordered results, vague predicates, and semantic relativism (i.e., information might be an attribute or element) in adherence with their \textit{structured document retrieval principle}: “A system should always retrieve the most specific part of a document answering a query.” Their query language, XIRQL, requires an extended \textit{document type definition (DTD)} that identifies the granularity for indexing content and that enables vague predicates and semantic relativism. XIRQL’s complex query syntax makes it unsuitable for naive users; Fuhr and Großjohann envision users constructing queries via a predefined template. In practice, the extended DTD precludes XIRQL from handling existing data sources.

XKeyword [34] ports a search engine for relational data (see DISCOVER [35], page 22) to XML databases. System administrators define target objects, which are the minimal set of nodes that compose meaningful answers. A presentation graph groups target objects into networks to display the search results. The presentation graph hides redundancy among the results by focusing attention on the structure of results instead of individual nodes. Users may drill-down into the graph to regain the hidden details. Using a relational database to store XML fragments and to enable efficient execution plans forces system administrators to
select the best decomposition and clustering strategy for each XML database.

Guo et al. [36] present a variation of the PageRank algorithm [37] that is specifically adapted for XML. XRANK handles both intra- and inter-document XML references, which are specified by the ref and xlink attributes respectively. Their approach considers result specificity (more specific results should rank higher than less specific results), keyword proximity in the results, and reference awareness (i.e., the prestige of references). A modification to traditional inverted lists allows efficient execution without making the indexes redundant.\footnote{In a naive implementation, a parent element's index duplicates the information found in the indexes of its children.} Regrettably, XRANK supports only conjunctive semantics, and the quality of the ranking function is not measured quantitatively.

In an effort to give users fine-grained control over search results, Cohen et al. [38] create a new query syntax for XSEarch. Queries take the form \texttt{l:k, l:}, or \texttt{:k} where \texttt{l} is a label (attribute or element name) and \texttt{k} is a keyword. Interconnection semantics identify meaningful relationships among the XML elements. For example, two elements are meaningfully related if they share a common ancestor. Interconnection semantics significantly improve the precision of search results, but the actual scoring formula, a variant of \textit{tf–idf} scoring (a simple scoring scheme that incorporates an unnormalized term frequency factor and an inverse document frequency factor), involves a number of additional parameters. Later work [39] supports XML references and enables automatic derivation of relationships even without a complete schema for the data.

Schema-Free XQuery [40] imbues XQuery [41] with a notion similar to the interconnection semantics proposed by Cohen et al. [38]. The meaningful lowest common ancestor structure is more robust than interconnection semantics since the latter relies on exact matches for tag (element or attribute) names, which are dependent on a specific DTD. Like XIRQL, the complicated syntax makes Schema-Free XQuery unsuitable for naive users.

TopX Search [42] provides efficient top-\(k\) query processing by extending the family of threshold algorithms [43] but does not support XML references or links. Because random accesses to check tag matches are expensive (in terms of I/O), a minimum probe and a heuristic approach reduce this cost. A database indexes the document structure to provide further support for efficient execution.

All of these approaches redefine query answers: a complete XML document is too coarse a result for search queries, an informal definition that is more eloquently described by Fuhr and Großjohann’s structured document retrieval principle [33]. Wilkinson’s work [32] first suggested indexing structured documents by their sections, and all of these later approaches adopt this strategy. XSEarch [38] simplifies the query syntax
to enable users to construct highly-expressive queries with a minimum of effort. In particular, the query syntax blurs the distinction between attributes and elements, an important feature because users should not be expected to know the structure of the data. XKeyword’s novel user interface is the first to address the challenge of graphically displaying search results and even handles the repeated-information problem later described by Golenberg et al. \[44\] (see page 28 for the definition of the repeated-information problem). Although a number of systems propose new data structures and algorithms to increase search efficiency, few quantitative results are provided for search effectiveness. In particular, no research includes an extensive survey of the competing approaches to determine the tradeoffs of each.

### 2.3 Relational Search Engines

Mragyati \[45\] was an early attempt to extend the keyword search paradigm to relational databases. The system decomposes keyword queries into search trees that are translated into SQL statements and executed on the underlying database. A hierarchical display of results enables users to drill down to discover additional information. Sarga and Jain do not address scalability or performance issues, though.

In DBXplorer, Agrawal et al. \[46\] propose integrating keyword search and relational databases even without the database supporting full text indexes. Instead, a symbol table, which is analogous to an inverted list, maps keywords to database tuples. When given a keyword query, the system first identifies the tuples containing each query keyword. If no tuple contains all query keywords, DBXplorer constructs SQL join expressions to identify combinations of tuples—linked by foreign key relationships—containing all of the keywords. The system ranks results by increasing number of joins. Evaluation of the system only compares the alternatives for the symbol table design.

DISCOVER \[35\] provides keyword search in a relational database by reusing the database’s underlying full text indexing capabilities. Query answers are minimal networks of tuples that collectively contain all query keywords—no tuple may be removed without losing at least one query keyword (i.e., strict AND semantics). Like DBXplorer, answers are ranked by the size of the network. To construct the networks of tuples, a master index first identifies the relations containing query keywords. A breadth-first search of the schema graph generates all possible networks of tuples. Because the size of the networks of tuples is only data-bound (a non-trivial schema does not allow the value to be bounded statically), a parameter limits the maximum size of all of the networks. Multiple networks of tuples may share common join subexpressions
so Hristidis et al. present a number of greedy algorithms for reusing common expressions (determining the optimal execution plan is an NP-complete problem).

Hristidis et al. [47] later improve DISCOVER by adding support for disjunctive queries and by integrating IR ranking functions to score results. A variant of pivoted-normalization weighting scores individual tuples; to calculate the score of an entire tuple tree (i.e., DISCOVER’s network of tuples), the scores of the individual tuples are summed and then divided by the size of the tree. The combination function is monotonic and allows efficient execution based upon the family of threshold algorithms proposed by Fagin et al. [43]. Evaluation considers the proposed execution algorithms and the DBLP database. In the end, a hybrid algorithm, which processes conjunctive and disjunctive queries with different execution algorithms, provides the best performance.

DbSurfer [48] extends traditional Internet search engines by incorporating a database reader to include content stored in relational databases and a trail-finding system to discover relationships among web pages. Database content is retrieved via a servlet, which constructs a virtual web page for each database tuple and replaces foreign keys with hyperlinks to related content. Trail finding involves scoring each web page, following the links from those that score highest, and removing information redundant within that trail. DbSurfer presents the results of the best trail algorithm as trees of tuples.

ObjectRank [49] extends PageRank [37] to perform keyword search within a relational database. Like PageRank, random surfers start at tuples containing search keywords and iteratively move to related tuples or return to one of the starting tuples. An authority transfer graph specifies the likelihood that a random surfer will move to a related tuple. As an example, consider the Digital Bibliography & Library Project (DBLP) database. Citations reflect a natural flow of authority among papers, but citing influential papers does not make a paper important. Rather, a paper’s importance is based on the number of influential works citing it. In ObjectRank, a tuple’s score incorporates global importance (i.e., pure PageRank) and keyword-specific rank. Since ObjectRank iterates until rankings converge, Balmin et al. introduce a number of performance optimizations. DAGs (e.g., bibliographic databases sorted by date) converge in a single pass because the graph may be sorted topologically. An almost-DAG (i.e., a graph with a small number of backedges) may be efficiently scored by first ignoring backedges and later re-introducing them into the computation. The number of iterations required until convergence may also be reduced by preprocessing the graph and storing the ranks of vertices that score above some threshold. The preprocessing requires a significant amount of time but allows keyword searches to complete quickly.
Su and Widom [18] present the EKSO (Efficient Keyword Search through Offline Indexing) system, which limits the work performed at search time by extensively using the full text capabilities of the underlying database. The EKSO system concatenates the string attributes of related tuples—starting from a root relation—to form virtual documents, which the database indexes. A breath-first traversal with no backtracking identifies related tuples (foreign keys are considered bidirectional). Because the EKSO system does not export the virtual documents from the database, triggers may automatically update the virtual documents following an update to any tuple the virtual document references. The greatest downside of Su and Widom’s approach is the size of the index—for the evaluation datasets, the indexes increase the size of the database by a factor of 2–8. In addition, the triggers that keep the indexes up-to-date cannot handle more than 1 update per second so databases with a large number of updates will suffer from out-of-date full text indexes.

Liu et al. [50] build on the framework of Hristidis et al. [47] but examine search effectiveness and ignore performance issues. They add four database-specific scoring normalizations to pivoted normalization scoring to reflect its new context (that is, scoring relational data). The resulting scoring formula bears only a superficial resemblance to the original. Liu et al. handle schema names (e.g., relation or attribute names) in the query by assigning all result trees that contain the schema name a positive weight, even when none of the tree’s tuples contain the term as the value of an attribute. In addition, phrase ranking elevates the scores of results that match exact query phrases. A lyrics database containing more than 177,000 songs provides the only evaluation dataset; the queries used for evaluation were obtained from a commercial search engine’s logs. In contrast with other studies of web queries [26, 28, 29, 30, 31, 51, 52], query length varies between 2 and 20 keywords with the average number of keywords being 6.7. According to the evaluation, their normalizations outperform Google searches by 16.3% and Hristidis et al.‘s previous work by 77.4%. Nevertheless, it is unclear whether the normalizations and modified scoring formula apply to all relational contexts.

Hwang et al. [53] critique the original ObjectRank implementation following a series of user surveys. Several calibration parameters caused the original version of ObjectRank to prefer general answers instead of specific results. For example, a widely cited paper about databases might be the first result for a query about transactions and concurrency control instead of a paper dealing with only these two topics. The revised ranking scheme introduces inverse ObjectRank, a keyword-specific measure of the specificity of nodes to each query keyword. Hwang et al. also introduce keyword weights so frequent keywords are not favored, which is
similar to the *idf* factor present in IR weighting formulas. An ontology graph, which relates domain-specific knowledge about common keywords, further improves the quality of search results via automatic relevance feedback.

The SPARK system [54] corrects a scoring deficiency present in Hristidis *et al.*'s previous work [47], which used a linear combination of the scores of individual tuples to score an entire result tree. The general problem, which pertains to structured documents, was first noted by Robertson *et al.* [55]. Like the work of Liu *et al.*, SPARK modifies the original IR weighting function both to reflect its new relational context and to enable a number of performance optimizations. In addition, Luo *et al.* add a completeness factor so results containing all query keywords rank higher than results containing only a subset of the query keywords. Efficient processing of the non-monotonic scoring function requires a skyline sweep algorithm [56]. (Hristidis *et al.*'s scoring function is monotonic, a property required for the family of threshold algorithms.) Evaluation shows SPARK outperforming the previous work of Hristidis *et al.* and Liu *et al.* even though efficient execution requires estimating the values of a number of factors in the ranking formula.

From its inauspicious start, the extension of the keyword search paradigm to relational databases has generated a significant amount of research. Most systems follow DBXplorer [46] and DISCOVER’s [35] lead by redefining query answers as trees of tuples. A number of DISCOVER’s algorithms, including its approach to generating networks of tuples, are directly incorporated into later work. The process of generating all possible networks of tuples is an exponential time algorithm whose running time is restricted in practice by arbitrarily limiting the maximum size of the networks. Hence, none of the systems using this algorithm will find all meaningful answers for queries. Liu *et al.* [50] and Luo *et al.* [54] propose significant modifications to pivoted normalization weighting. Although their evaluations highlight the positive impact of the changes, it is unclear whether or not the modifications apply in all contexts, especially to a mixture of relational and unstructured data.

### 2.4 Searching Data Graphs

Goldman *et al.* [57] was the first to view keyword search in relational databases as an instance of a more general problem. **Keyword search in data graphs** is a paradigm rich enough to support both semi-structured and relational content. In Goldman *et al.*’s work, the database is a collection of objects related by some distance function. Queries consist of “find” and “near” objects; the former specifies results, the latter
keywords. The major difficulty for query execution is computing the distances among objects in the database. Computing shortest paths at runtime requires an exorbitant number of disk seeks (under the assumption that the data graph does not fit within memory). Precomputing the shortest paths alleviates the problem, but existing algorithms (e.g., the Floyd-Warshall algorithm [58, 59]) fail to provide a means for early termination. By partitioning the graph around “hubs” so partitions are not connected without traveling through one of the “hub” vertices, the shortest path between any two vertices may be efficiently computed without a significant storage penalty. Search engine administrators may further tune performance by specifying clustering rules, i.e., what objects should be clustered together.

DataSpot [60] enables keyword search on relational database content by constructing a hyperbase. The hyperbase is a graph whose nodes are database relations, tuples, fields, and values and whose edges denote relationships among the nodes. Query answers are subsets of the hyperbase. Because data is completely exported from the underlying data sources, DataSpot easily handles content from heterogeneous databases. Design and implementation details are limited due to DataSpot’s commercial status.

BANKS (Browsing ANd Keyword Searching) [61] provides a proximity search engine and a framework for browsing the contents of a relational database. The database is modeled as a directed graph, and query answers are rooted, directed trees that contain all query keywords. Relationships between tuples—as defined by the schema’s foreign keys—form forward edges in the directed graph. Backward edges whose weights are proportional to the number of related tuples augment the forward edges and ensure a directed path exists between any two vertices. Furthermore, the backward edges mitigate the impacts of “hubs” that connect many nodes. The score of a query answer is the combination of edge and node weights (nodes are weighted using a primitive form of PageRank). Because the group Steiner problem is NP-complete, the backward expanding search algorithm provides an efficient heuristic to guide searches. Duplicate detection filters isomorphic answer trees. In addition to supporting keyword search, BANKS can also dynamically generate a web interface for browsing the contents of the relational database. Foreign keys become hyperlinks to referenced tuples, and the primary key links to tuples that reference it. As others show, the backward expanding search heuristic is not guaranteed to find answer trees of minimum weight [62] or to generate answers efficiently [44].

In a twist on Internet keyword search, Li et al. [63] define an information unit as a logical web document that may contain multiple physical pages. Thus, a query answer is defined as a set of web pages covering all query keywords. Their system ranks results in increasing order of cost, where the cost of an answer
is the aggregation of its edge costs (i.e., the group Steiner tree problem). For a web search engine, top-

$k$ retrieval is obviously of paramount importance, a feature not inherent in most Steiner tree approximation algorithms. Two algorithms are presented for progressive enumeration of query answers. The minimum edge-based strategy grows existing minimum spanning trees (MSTs) by adding an edge of least weight. The balanced MST strategy examines the existing MSTs and adds the edge that minimizes total growth (among all of the potential MSTs). The former is not guaranteed to generate results in increasing rank; the latter is still suboptimal since MSTs only approximate Steiner trees.

Kacholia et al. [64] extend BANKS with an improved search heuristic, bidirectional search. Rather than starting at leaf nodes and working forward to the root of the answer tree, bidirectional search enables the search to start at both leaf and potential root nodes. Nodes are visited according to spreading activation so the algorithm explores nodes with fewer neighbors (and a lower search time) first. The revised search algorithm’s performance is evaluated using three datasets, but the evaluation fails to include search effectiveness.

Ding et al. [62] propose a dynamic programming algorithm for identifying the minimum group Steiner tree in time exponential in the number of search keywords. Following identification of the optimal solution, their approach generates additional solutions in approximate order (only the first answer is guaranteed to be optimal). The approach produces results in similar time but with lower cost (i.e., the weight of the Steiner trees) than bidirectional search [64].

BLINKS [65] extends Kacholia et al.’s work [64] by creating a bi-level index designed to facilitate query processing. Unlike previous work, query answers minimize tree height instead of weight. Essentially, the index stores keyword-node lists containing the distance from the keyword to every node of the graph and an inverse index called a node-keyword map. Because the space required to store the index is quadratic in the number of graph vertices, the graph is partitioned and then indexed. The intra-block indexes are defined as previously described but must be augmented by another index that handles the distances among the individual partitions. BLINKS provides an order-of-magnitude performance increase over bidirectional expanding search. Unfortunately, this approach introduces significant redundancy by explicitly tracking which nodes contain each keyword.

Golenberg et al. [44] present a keyword proximity search algorithm that satisfies the following three constraints: searches must provide all relevant results, they must be efficient, and the ordering of results must be correlated with the actual ranks of the results. The second criterion, efficiency, is the delay between outputting consecutive answers. A polynomial delay is the accepted upper-bound for efficient search engines.
although previous works \cite{62, 64} cannot guarantee such performance. The third requirement may be unintuitive, but since keyword proximity search may be reduced into the group Steiner tree problem, the ordering of search results must be relaxed to meet the second criterion. Golenberg et al. define a query answer as a non-redundant subtree containing all query keywords (although OR semantics are also supported); these subtrees are enumerated in increasing order of height. Candidate answers are passed to the ranker before being output. The ranker applies a scoring function (e.g., an IR scoring function) to give answers a baseline score. Then, the ranker penalizes redundant answers, answers similar to previous results, so the user is not confronted by the repeated-information problem (i.e., answers contain the same nodes from the data graph but with different relationships among them). The second phase, penalizing redundant answers, is adaptive; the penalty is dynamically computed based on the previous results already returned to the user.

The numerous search techniques targeted at data graphs all suffer from the intractability of the group Steiner problem. Early work \cite{61, 62, 63, 64} proposed a number of heuristics to provide reasonable search performance. Later researchers \cite{44, 65} realized the weight of an answer tree is correlated with the answer tree’s height, which enables the use of efficient algorithms that are guaranteed to discover all possible results. The greatest downside of some of the proposed indexing techniques is the reliance on explicitly tracking the nodes that contain each keyword. Such an index grows quickly out-of-date when the underlying data changes rapidly. Golenberg et al. \cite{44} were the first to propose a secondary ranking scheme based on traditional IR scoring formulas to reorder results. The importance of this idea should not be understated because it enables both efficient execution and final rankings that are more consistent with users’ expectations.
Chapter 3

Searching Structured Data

As evidenced by the introduction, searching semi-structured and relational data is difficult, and a wide variety of techniques have been proposed for handling both types of data. The graph-structured techniques reviewed in section 2.4 are the most general, applying both to semi-structured and to relational content. Hence, this work adopts a similar formal problem description, which is keyword search in data graphs. A simple definition of query answers is selected from the various alternatives that are present in previous work. The intuitive model for scoring answers allows the use of existing IR weighting models for scoring both individual and related tuples. The chapter concludes with the design of the hybrid index and graph traversal algorithms.

3.1 Formal Problem Description

Although the crux of this work is the extension of the keyword search paradigm to relational databases, this problem is merely an instance of a more general problem—keyword search within data graphs. This section formally defines the problem of keyword search within an arbitrary data graph even though later discussion focuses on a relational setting. Following the formal definitions of data graphs and queries, several competing definitions for query answers and query semantics are examined. An extensive description of the reasoning behind adopting the definitions used in this work follows the definitions.
Figure 3.1: A data graph for a portion of a movie database. Each vertex represents a database tuple. Cast information, containing foreign keys of related tuples, is shown as numbered vertices.

3.1.1 Data Graph

Keyword search in graphs is general enough to encompass both semi-structured and structured data. Even though the former is often tree structured, id references allow the formation of complex data graphs, e.g., a bibliographic database. In the relational model, foreign keys among tuples naturally express the edges of a data graph. Thus, the directed graph $G = (V,E)$ encodes XML elements as vertices or relational tuples as vertices. Within the graph, an edge from $v$ to $w$ denotes a relationship between the underlying elements or tuples. For a data graph expressing XML, edges denote either hierarchical containment (e.g., a conference paper contains a title, abstract, etc.) or arbitrary references (e.g., citations in the bibliographic database). In a relational setting, two tuples are related via foreign key constraints. As an example, consider the graph shown in figure 3.1. The portion of the database¹ shown lists three actors and the character each plays in two movies. Each directed edge is a foreign key in the database schema from cast information to the referenced tuple.

3.1.2 Queries and Answers

A search query, $Q = [k_1, k_2, \ldots, k_n]$, comprises a list of keywords. Query keywords need not be unique because repeating a search term is one method for indicating its importance, which is a property reflected

¹The relational schema for the data is provided in table 4.1.
by the IR scoring functions presented in section 2.1.

The most common definition of a query answer is a tree where each leaf contains at least one keyword and collectively the leaves contain all of the keywords. The answer tree must be minimal: it cannot contain any subtree that also satisfies the original definition. In the case of proximity search, query answers are defined as directed trees where every node is reachable from the root. Systems developed solely for relational databases often do not use this explicit definition, but the SQL expressions that they produce adhere to this concept. ObjectRank [49] employs an alternate definition: query answers take the form of a ranked list of database tuples (i.e., vertices in the data graph). Balmin et al. argue that this definition is appropriate whenever there is a flow of authority between objects. For example, citations in a bibliographic database indicate authoritative papers; when performing a keyword search, certain influential papers may not contain the keywords themselves but are cited by those papers that do contain the search terms. ObjectRank’s definition of query answers seems appropriate for this work for a number of reasons.

First, data-driven applications, both web and stand-alone, have a rich, preexisting framework for viewing data. The framework provides a simple means for navigating among related pieces of information and tags information with user-friendly labels. For example, IMDb pages link actors to the characters they have played and the films in which they appear. MediaWiki sites (e.g., Wikipedia) are known for their extensive cross references that link to related content. Even when such a framework is not existent, a simple browsing framework may be automatically generated without user intervention (as demonstrated by Bhalotia et al. [61]). These existing browsing frameworks already account for peculiarities (e.g., associative relations) introduced by the relational model. Associative relations are the by-product of many-to-many relationships and are rarely of any interest to users. Other relations are present only to guarantee data integrity. For the movie database, the browsing framework is unlikely to reveal the cast relation that associates actors with films. Instead, the browsing framework merely displays the actors who appear in a film and vice versa. The cast relation is a by-product of the relational model, and as such, it should not have equal importance as the entity relations.

Second, the relationships present in search results pose a number of difficulties. Some relationships are trivial and unnecessary to report to users. Consider the data graph shown in figure 3.1. One path between the actors Harrison Ford and Sean Connery passes through the actor tuple; this path represents one relationship between the two people. Ordinary users grasp this relationship implicitly—normally, it need not be restated. For the data graph shown, a more meaningful answer contains the film Indiana Jones.
and the Last Crusade. Moreover, displaying complex relationships among separate pieces of information requires a non-trivial user interface. Several existing systems display answer trees using text, but results may be arbitrarily complex with multiple n-ary intermediate relationships, which makes them difficult to understand at a glance. Another insidious problem is that commonalities among many results may be easily missed, buried within the display of results. As demonstrated by figure 3.2, a number of query results may contain commonalities, a frequent situation in complex data graphs. One expects these tuples to be considered important in the context of the query, but these tuples are easy to overlook. The best approach for calling attention to commonalities is implicit: users process visual cues much more efficiently than scanning text [66]. The complexities of a graphical interface are beyond the scope of this work, but the visualization of search results seems an essential part of future work in this field.

The last reason for defining query answers as individual tuples is the repeated-information problem described by Golenberg et al. [44]. The repeated-information problem stems from the many interconnections found in a complex data graph. If all of the possible connections are enumerated, answers will be very similar to each other, forcing users to view a large number of results to discover a small amount of information. In figure 3.2, the second answer repeats 4 of the 5 meaningful nodes, albeit in a different order. Reporting the
many permutations of candidate answers forces users to sift through a significant number of similar results, possibly rendering the display of the top-k results worthless. On the other hand, if the ranking scheme suppresses some answers—as proposed by Golenberg et al.—there is always the chance that the ranker eliminates the result most relevant to the user. If the ranker suppresses the second answer tree, the user will not know that the two films share a common character. As an alternative, consider elevating the rankings of common vertices. This technique boosts the position of common vertices and implies their importance (and repetition) to users.

To summarize, query answers may be arbitrarily complex and should be presented with familiar semantics (e.g., user-friendly labels for database fields) for user clarity. The presence of a browsing framework easily provides necessary semantics and is designed to handle artifacts introduced by the relational model. Using ObjectRank’s definition of query answers capitalizes on existent functionality—namely, a rich browsing framework. This framework alleviates the complexities of displaying answer trees because connections among related information may be gleaned from the information and links already provided by the application. Finally, the prestige of a tuple (i.e., how often it is referenced) is a factor in the ranking formula so commonalities among multiple results are pushed toward the top of the results lists instead of risking a loss of precision in query answers by eliminating redundancy.

3.1.3 Semantics

Query semantics is a second issue which varies according to approach. Conjunctive queries are typically implicit in proximity search [61, 62, 64] although IR-inspired approaches [47, 50, 54] and the most recent proximity approach [44] also handle disjunctive queries. The distinction stems from an inherent difficulty with proximity search: if a directed path from a root node to all keywords does not exist, different combinations of keywords must be tried in order to produce results (under AND semantics). Clearly trying the many possible subsets is not feasible (a user may submit a lengthy query containing many spurious keywords), and these approaches terminate quickly rather than undertaking a lengthy search process.\(^2\)

Strict AND semantics seem inappropriate for a search engine, especially given the findings of Rose and Stevens [26] and Wilkinson et al. [27]. After all, returning answers containing a subset of the search terms

\(^2\)If every edge in the data graph is augmented by a backwards edge (as BANKS [61] and its successor [66] do) so a directed path always exists between two vertices, the query keywords not present in the graph may be simply discarded. Without backwards edges, though, subsets of the keywords might be unreachable due to edge directionality. In such instances, the query must be evaluated using OR semantics to obtain results.
undoubtedly provides more benefit to users than returning no results at all. Modern IR scoring formulas implicitly use OR semantics, which plays an important part when ranking results. Nevertheless, strict OR semantics is not equivalent to ranking results according to IR scoring formulas, for the latter significantly increase the relevance of results containing keywords that occur infrequently in the collection. As described in section 3.2.2, a smooth transition function from OR to AND semantics gives users complete control over this aspect of ranking results.

3.2 Scoring Answers

A number of previous keyword search systems modify the traditional IR scoring formulas to make them suitable to a relational setting. These systems typically add additional factors to improve search effectiveness, but it is unclear if the modifications apply to all or merely to specific databases. While the inclusion of a state-of-the-art IR scoring formula appears to be a good starting place for effective keyword search in data graphs, modifications to the established formula are ill-advised because proper evaluation requires extensive testing.

3.2.1 Problems with Existing Solutions

Although a number of systems target keyword search in relational databases, XML, or data graphs in general, two significant areas for improvement exist. The first relates to minimum-cost trees, the foundation of proximity search. The second, which affects all of the systems identified in the literature, is potentially more serious because it creates a gap between scoring unstructured documents and their structured counterparts.

Minimum-Cost Trees

Defined strictly, minimum-cost trees reflect no notion of IR scoring functions. Consider searching for keywords \( k_1 \) and \( k_2 \) in a collection of documents \( d_1 = [k_1, \ldots] \), \( d_2 = [k_2, \ldots] \), and \( d_3 = [k_2, k_2, \ldots] \) where \( d_1 \), \( d_2 \), and \( d_3 \) all have equal length. Furthermore, let \( d_1 \) and \( d_2 \) be connected by an edge of weight \( w \) and \( d_1 \) and \( d_3 \) be connected by an edge of weight \( w + \varepsilon \). Ranking query answers strictly by the cost of the tree allows the former to rank higher than the latter. But in the context of IR rankings, it is obvious that \( d_1d_2 \) should score lower than \( d_1d_3 \) because the latter contains more instances of search terms. Golenberg et al. [44] allow a separate ranking engine to sort candidate answers according to IR measures, but this requires \( \varepsilon \) to be small
Raiders of the Lost Ark opens in 1936 with archeology professor Indiana Jones (Harrison Ford) narrowly escaping death while retrieving a golden statue from a South American temple. . . .

Figure 3.3: A minimum spanning tree and plot synopsis relevant to the query “Ark Ford Jones.” Bold text in the plot synopsis indicates **keyword matches**.

enough for both trees to be enumerated within the top-\(k\) results. Other systems fail to reflect this important property of IR scoring functions.

**Size Penalty**

Previous ranking strategies explicitly apply a penalty to search results composed from multiple vertices in the data graph. For relational approaches, the penalty is the raw \([35, 46, 47]\) or normalized \([50, 54]\) size of the join tree; in minimum cost trees, the weight of the edges \([61, 62, 64]\) or height \([44, 65]\) of the tree. This uniform penalty is inappropriate in many instances. Due to data normalization, closely related information is often spread among many tables. Reconstructing the original document—that is, the collection of all text that completely describes a given object—should not incur any penalty; rather the document should be scored as the unified whole. The data were separated to guarantee data integrity, and, if desired, the information could be displayed as a single coherent unit.

As an example, consider searching the data graph shown in figure 3.1. Given the query “Ark Ford Jones,” both relational and proximity based approaches will retrieve a tree similar to the one shown in figure 3.3. Such results should be scored similarly to documents not stored in separate tables. If the database schema expands to include plot synopses, there is a strong likelihood that a plot synopsis (like the one present in figure 3.3) will contain all of the search terms. When this is the case, a size penalty applied because of database normalization will push the synopsis ahead of the previous result. The tree shown in figure 3.3 is preferable because it is the *origin* of the information found in the synopsis.

This problem poses a significant challenge. The naive solution would completely disregard the size of the tree, but then a large tree (e.g., Harrison Ford starred in *Star Wars* with Carrie Fisher who appeared in . . . with Roger Moore who, along with Sean Connery, played the character James Bond\(^3\)) scores identically to the tree shown in figure 3.3. The smaller tree is obviously preferable, and size must be considered when

\(^3\)The ellipsis denotes intentional omission of additional movies or actors. In theory, this answer tree could include every movie and actor in the database.
ranking results. Post-processing the results is one alternative but requires all results be known and sacrifices either efficiency or correctness if approximate ordering is used instead. A key realization is that the preferred tuple tree contains an associative entity linking three entity relations. If information propagates through the associative entity in a single step, it will avoid incurring a size penalty. Each time information crosses an entity relation (e.g., moving past Star Wars in the large tree mentioned previously), the weight of the edge traversed is added to the size of the tree. This approach gives both the small tree and the plot synopsis equal size, and because IR scoring formulas penalize documents containing additional text, the tree will score higher than the plot synopsis.

3.2.2 Tuples

This section describes the scoring functions applied to individual tuples, starting with a baseline IR score and incorporating both a completeness factor (to enforce users’ preferences for coordination matching) and prestige. Handling relationships among tuples (i.e., scoring related tuples) is covered in the next section.

**IR Scoring**

Database tuples are composed of multiple attributes, the importance of which may vary widely. A conference paper contains a great deal of structure (e.g., a title, abstract, and corpus) and often it is beneficial to weight certain portions more heavily than others. A keyword match in the title is a better indicator of relevance than matching the same keyword in the abstract or body text. The problem grows worse when confronted with information from related documents, including conference information. This section examines the problem and one solution for scoring structured documents using ranking formulas designed for unstructured text collections.

An important part of state-of-the-art IR scoring functions is the non-linear term frequency factor. As an example, let document $d_1 = [k_1, k_1, \ldots]$ and document $d_2 = [k_1, k_2, \ldots]$ be two documents in a collection where $d_1$ and $d_2$ have equal length and let the document frequency for the two keywords be equal. Given the query $Q = [k_1]$, document $d_1$ should rank before document $d_2$ because the former contains the keyword twice. If another query $Q' = [k_1, k_2]$ is issued, the rankings should reverse because the marginal benefit of matching two search terms outweighs the marginal benefit of matching one keyword twice [25]. Now consider a different example that includes two related documents (or attributes of a tuple), $d_3 = [k_2, \ldots]$ and $d_4 = [k_2, \ldots]$. Since the documents are related, they could be scored together, but how should the scoring
function be defined? Combining the scores linearly

\[ combine(d_3, d_4) = score(d_3) + score(d_4) \]

allows the combined document to receive a score equal to \( d_2 \)'s for the query \( Q' \) (ignoring the other factors present in the IR scoring function). But this situation is exactly what a non-linear term frequency factor prevents: \( combine(d_3, d_4) \) should receive the same score as \( d_1 \) for the second query because both match a single keyword once. The problem originates with the linear combination of individual scores. A better solution is to construct a new document \( d_3 \cup d_4 \) and use the original scoring function on it. This approach preserves the non-linear impact of term frequencies so \( d_1 \) and \( d_3 \cup d_4 \) receive the same score for the second query. A question arises of how to score the combination if \( d_3 \)'s content is more important than \( d_4 \)'s. For example, the content of a title or abstract of a conference paper is much more significant than the body of the paper in the context of keyword search.

Robertson et al. [55] previously examined this problem and proposed an elegant solution: to weight a portion of a structured document (one composed of multiple unstructured documents like \( d_3 \) and \( d_4 \)) more heavily, include it multiple times when scoring the structured document. For example, if \( d_3 \) was the title of a paper and \( d_4 \) the body and the title has twice the importance as the body, the structured document scores identically to \( d_3 \cup d_3 \cup d_4 \). This satisfies expectations for how documents should be scored: if all parts receive equal weight, they score identically to their concatenation, and weighting one part more heavily is reflected by the term frequencies and by the document length of the structured document. Formally, a structured document contains multiple unstructured fields

\[ d = d_1, d_2, \ldots, d_n \]

but to score it when certain portions have greater weight than others, create the document

\[ d' = \sum_{i=1}^{n} w_i \cdot d_i \]

where \( w_i \) is the weight placed on the \( i \)th field.

This definition naturally extends to scoring tuples. Tuples contain multiple text fields; thus, a tuple is a structured document and certain fields may be weighted more heavily than others. For practical reasons,
a domain expert is in the best position to select default field weights for the data. One immediate question is how to determine which attributes of a tuple should contribute to its score. There are two possible resolutions to this dilemma. First, the structured document formed from a tuple could contain the values of every attribute. Alternatively, the structured document formed from a tuple could contain only attribute values that match at least one search term. This work uses the second definition because it limits the amount of information that must be considered. Another benefit is that the contents of different tuples may vary widely: if the movie database stores a plot synopsis within each movie tuple, the length of the plot synopses will likely vary widely. If plot synopses (even when they do not contain any search terms) are automatically included, an empty plot synopsis could boost one tuple’s score over another’s with a complete description of the movie.

Pivoted normalization weighting [22] provides the baseline (IR) score:

\[ \sum_{t \in Q \cap D} \frac{ntf}{ndl} \cdot qtf \cdot \ln(idf) \] (3.1)

where

\[ ntf = 1 + \ln(1 + \ln(tf)) \]
\[ ndl = (1 - s) + s \cdot \left( \frac{dl}{avgd} \right) \]
\[ idf = \frac{N + 1}{df} \]

and \( qtf \) is the frequency of the term in the query. The length of a document, \( dl \), is the total number of terms contained within that document. The document collection comprises all structured documents (tuples) that contain at least one search term. Pivoted normalization weighting is ideal due to its single tuning parameter, \( s \), for which IR researchers have already identified a good default value (0.2).

**Completeness**

Unfortunately, pivoted normalization alone is not sufficient for expressing users’ preferences. As previously stated, users prefer results containing all query keywords to rank ahead of results containing a subset of the query terms. The \( idf \) factor of most IR scoring formulas heavily favors documents containing terms that occur infrequently in the document collection. Hence, given the keyword query \( Q = [k_1, k_2, k_3] \) where \( k_1 \) and
3.2 | Scoring Answers

$k_2$ occur frequently and $k_3$ occurs very infrequently, a document containing only $k_3$ will score much higher than a document containing only $k_1$ and $k_2$. This differs from user expectations.

Luo et al. [54] add a completeness factor so rankings conform to user preferences. The completeness factor is derived from the extended boolean model for information retrieval [67] and is defined for a document as

$$completeness = 1 - \left( \frac{\sum_{t \in Q \cap D} qtw^p(1 - tw)^p}{\sum_{t \in Q} qtw^p} \right)^\frac{1}{p}$$

(3.2)

where $qtw$ is the weight of the query term ($0 < qtw \leq 1$), $tw$ is the weight of the query term in the document ($0 \leq tw \leq 1$), and $p$ is a real-valued parameter in the range $[1, \infty)$. Since pivoted normalization weighting already provides an IR score, the goal of the completeness factor is to reward documents that contain larger subsets of query terms. The parameter $p$ smoothly transitions from OR to AND semantics as $p$ increases. The proof is simplified by allowing each query term to be weighted equally and the term weight to be boolean (that is, 1 if the term is present; 0 otherwise). When $p = 1$, observe that the completeness factor varies directly with the number of query terms present in the document. For example, a document containing 3 of 3 query terms will score 1 while a document containing 2 of the 3 will score $1 - \frac{1}{3} = \frac{2}{3}$, which conforms to users’ expectations. Now as $p$ increases,

$$\lim_{p \to \infty} completeness = 1 - \left( \frac{\max((1-tw_1),(1-tw_2),\ldots,(1-tw_n))}{\max(qtw_1, qtw_2, \ldots, qtw_n)} \right) = \min(tw_1, tw_2, \ldots, tw_n)$$

which will be 1 only when all query terms are present. Hence, whenever all query terms are not present in the document, the completeness factor will be 0, which conforms to strict AND semantics.

**Prestige**

As suggested by the earlier discussion of query answers, a pivotal factor in ranking is the notion of prestige. Popularized by PageRank [37], prestige boosts the scores of vertices which are highly referenced above those with few incoming edges. Given vague queries (e.g., “Connery” for the movie database), the most preferred answer is likely to be the best known actor (i.e., Sean Connery). In the absence of other information, the
best proxy for popularity is the number of films in which the actor has appeared and well-known characters (e.g., James Bond) that the actor has played. The approximation for popularity is crude but necessary to distinguish a popular actor from an extra who has appeared in a single film. A prestige factor is most important for vague queries; as the query is refined, prestige plays a decreasing role when scoring tuples.

**Complete Scoring Function**

In summary, the scoring function for structured documents (of which database tuples represent an instance) contains three factors. The first is pivoted normalization weighting; the second is a completeness factor, which varies between 0 and 1. The completeness factor recalibrates a document’s score so the score conforms to users’ expectations—namely, documents containing all query terms should be ranked before documents containing only a subset of the terms in the query. Prestige improves rankings for vague queries that are not scored adequately by the other factors.

### 3.2.3 Related Tuples

The final consideration is how to score related tuples, which are associated via foreign keys. The size penalty plays an important role in this issue because connecting related tuples cannot be free or else query answers will not be minimal. Since tuples are scored as structured documents, one approach would be to merge the attributes of two tuples and score them as a single structured document, but this introduces two potential problems, which are illustrated by an example.

Let $t_1$ and $t_2$ be tuples where $t_2$ contains a foreign key to $t_1$, and let one attribute of $t_1$ match a search term while $t_2$ does not match any search terms. Because $t_1$ itself contains the query terms, $t_1$ should rank ahead of both the merged document $t_1t_2$ and $t_2$. Decreasing the weight of $t_1$ in the merged document does not work because that by itself is not a sufficient condition for guaranteeing that $t_1$ scores higher than $t_1t_2$. Regardless of the penalty applied, $t_1$ will always score the same as $t_1t_2$ because all terms (specifically $tf$ and $dl$) in the IR scoring formula are decreased equally. Furthermore, it becomes unclear how to proceed when decreasing an attribute with unit weight: i.e., what does it mean for a document to contain a term a fractional number of times, and how does one model this behavior in regard to document frequency? One possible solution would involve reworking the IR scoring formulas and proving that their desired behavior is unchanged by the modifications, but a dramatically different alternative presents itself.
3.2 | Scoring Answers

Tuples are scored over a series of time steps. Initially, a tuple is scored according to the attributes it contains; tuples not containing any search terms receive a score of 0. Next, a tuple expands to include the information contained in related tuples—both tuples referencing this tuple and any tuples this tuple references (i.e., edge directionality in the data graph is ignored). The process continues until the tuple exhausts sources of related information. Thus, at the first time step, individual tuples are scored as isolated structured documents—no relationships are considered. For each successive time step, the tuple includes additional information specified by the relationships: the structured document representation of the tuple incorporates the related information. An external observer might see the scoring strategy like one sees the concentric rings formed when a pebble is thrown into water. The ripple progressively expands until it encompasses the entire body of water or subsides. Similarly, if the data graph is connected, eventually every vertex will be reached. The alternative (that is, the ripple eventually runs out of energy and subsides) is modeled by introducing a damping factor to control the importance of each successive time step. The damping factor then expresses the user’s preference for how “related” information must be. This intuitive model enjoys two significant benefits. First, scoring tuples becomes identical to scoring a collection of documents where the collection (potentially) changes at each time step. This method of scoring allows the use of the IR formulas previously described without modification. Second, the damping factor bounds the maximum score attainable by a tuple, a precondition for efficient execution algorithms.

The score of a tuple is the sum of its score taken at each of an infinite number of time steps.\(^4\) If each score is normalized relative the maximum score of the time step, each score will be between 0 and 1, and the relative order of the scores remains unchanged. Multiplication of each score by the damping factor produces a geometric series that is guaranteed to converge since the damping factor is bounded by 0 and 1. Formally,

\[
score(t) = \sum_{i=0}^{\infty} score_i(t) \cdot \alpha^i
\]

(3.3)

where \(score_i(t)\) is the score of the tuple \(t\) on the \(i\)th time step and \(\alpha\) is the damping factor. An upper bound on the maximum score of any tuple is then

\[
\max score(t) = \frac{1}{1 - \alpha}
\]

(3.4)

\(^4\)Since the data graph is finite the maximum number of time steps is bounded by the number of vertices in the graph, but the assumption merely affects the stopping conditions of our algorithm and not correctness.
The maximum score provides an early stopping condition, which is critical when the data graph is large.

Without a bound on the maximum score, it would be necessary to completely traverse the entire graph. To understand why, observe that the $idf$ factor of pivoted normalization may vary widely. Given two terms $t_1$ and $t_2$ that are present in documents $d_1$ and $d_2$ respectively where $t_1$ occurs frequently in the collection and $t_2$ occurs infrequently and the lengths of $d_1$ and $d_2$ are identical, note that

$$\lim_{df_1 \to N} \ln \left( \frac{N + 1}{df_1} \right) \approx 0$$

and

$$\lim_{df_2 \to 1} \ln \left( \frac{N + 1}{df_2} \right) \approx \ln N$$

Extending this result to an arbitrary time step of scoring shows the maximal score a tuple may receive cannot be bounded without both normalization and application of the damping factor. Both are essential for efficiently determining the top-$k$ results.

### 3.3 Framework

The design of the external index—that is, the in-memory data graph and additional information stored on disk—is critical to the system. The design of this index is described in detail before turning attention to the implementation of the search algorithm.

#### 3.3.1 External Index

A hybrid indexing scheme provides a host of benefits. First, it minimizes the number of queries that must be sent to the database. Other systems \[35, 47, 50, 54\] targeted at relational databases create complex SQL expressions usually involving multiple joins that can be costly for the database to execute. The hybrid index stores every tuple’s unique key in an external index so queries sent to the database need only retrieve the values of the unique key. No join expressions need to be computed. This approach enables the database to respond very quickly to queries since it has full text indexes on attributes of interest.

The primary concern for an in-memory data graph is size. The amount of data stored in main memory is minimized by assigning every vertex a unique (integer) id. A random access file provides fast access to additional information mapping each vertex id to the original database tuple. Vertex ids are assigned
by retrieving—in a specified order—every tuple from each database table. As a result, the random access file is structured such that all tuples belonging to each table are located together and in ascending order. The format allows binary searches on the random access file to determine the vertex id when provided its corresponding database tuple. The high latency of disk operations makes the approach suboptimal; a faster technique would use a B-tree to minimize the number of disk accesses. On the other hand, a random access file is trivial to create, and even a relatively small cache of the values retrieved by read operations provides significant savings in the number of disk accesses. Even when performing repeated searches on tables containing hundreds of thousands of tuples, the common probes (e.g., $\frac{n}{2}$, $\frac{n}{4}$, $\frac{3n}{8}$ where $n$ is the number of tuples in the table) will be satisfied from the cache.

The only requirement imposed on the underlying data is that every tuple contain a unique key composed of integer fields. Integer fields are required because most database management systems support types with arbitrary length (e.g., text), and a random access file requires records to have a constant size. The unique key is often the table’s primary key, but any unique key of integer fields may be used. If a database table of interest lacks a unique key composed of integers, a synthetic key may be added to the table because adding a unique id field to any table does not impact data normalization.

The index makes no attempt to track which nodes contain search terms. Modern relational database management systems feature full text indexes and can quickly retrieve all tuples containing query keywords. Mapping these tuples to their vertex ids has the same effect as including keywords (and their edges) in the graph but clearly reduces the total amount of memory required. Moreover, a tuple’s unique key is far less likely to change than the text contained within the tuple. Thus, the approach reaps the benefit of reusing the existing functionality of the underlying database and the graph remains up-to-date far longer than it would otherwise, provided that the number of insert and delete operations is small compared to the number of updates.

The cost of maintaining an external index can be high. With regard to memory consumption, a Java implementation cannot compare to a fine-tuned C or C++ implementation that manages its own memory. More significantly, any external index will at some point become outdated, but there are a number of strategies for mitigating this cost. The index can be regularly rebuilt in much the same way as Internet search engines crawl and index the web. Only in rare instances will the most up-to-date information be required, and because of the reliance upon the underlying database’s full text search capabilities, the index can only grow out-of-date with regard to the insertion or deletion of tuples (whose ids are not present in
the graph). If a user knows certain information is present, an alternative method should allow the user to navigate to the desired information using the existing browsing framework. As a second strategy, the index may be kept current by inserting middleware (e.g., a JDBC wrapper) between clients and the database itself. The middleware layer processes all queries and updates the index as required.

### 3.3.2 Search Algorithm

The actual search algorithm consists of three phases. First, the database is queried to retrieve tuples that contain the specified search terms. Next, one step in the graph traversal is executed and the vertices encountered thus far are ranked. If the top-$k$ results can be determined, the algorithm terminates and returns the results to the user; otherwise, the algorithm iterates for another time step.

**Query Database**

The first task is querying the database to retrieve tuples that contain search terms. The search engine maintains a view of the database schema that provides the names of database relations, attributes, and indexes. Naturally, tables with full text indexes are queried for keyword matches. A full text index is not appropriate for certain attributes that do not contain text (e.g., a movie’s year of release). These attributes are likely to be indexed using a different means, perhaps a B-tree. If an index exists on any attribute, the system will test it for strict equality. Exact matches are required for these indexes for two reasons: meaningful results could not be retrieved without them for some types of queries, and the presence of an index ensures efficient query execution. For the first case, consider the query “Sean Connery 1989,” designed to retrieve films released in 1989 that star Sean Connery. Even though *Indiana Jones and the Last Crusade* satisfies the search criteria, apart from the exact match on the year of release, the system lacks the means to link it with this important search term. After retrieving all tuples that contain at least one query keyword, the external index is used to retrieve the tuple’s unique id for the in-memory data graph. The ids as well as the contents of each tuple (i.e., the frequency of each search term in the tuple) are then passed to the graph traversal algorithm.

**Graph Traversal**

According to the illustrative scoring model, each vertex in the data graph progressively incorporates additional information at each time step, just as a pebble tossed into water creates a ripple that progressively...
expands. This scoring intuition is valued over proximity search because the approach rewards tuples that contain additional occurrences of search terms. In the movie database example, Harrison Ford played the character Indiana Jones in two movies named after the title character, which makes Harrison Ford a better match for “Indiana Jones” than River Phoenix who appeared as “Young Indy” in *Indiana Jones and the Last Crusade*. (The latter information is not shown in the data graph.) Minimum-cost trees cannot distinguish between any of the aforementioned relationships—all have equal weight.

A naive implementation for the scoring model creates $|V|$ traversals (where $V$ is the set of vertices in the data graph) and executes them concurrently. As $|V|$ increases, the naive approach becomes impractical. A better solution creates $n$ traversals, one for each tuple that contains a search term. As these traversals cover the graph and encounter additional vertices, each encountered vertex is marked with its distance from the source. This second implementation produces the same results as the first but is much more efficient if keyword matches are rare (i.e., $n \ll |V|$).

Nevertheless, this approach may be further optimized. Given the query “Indiana Jones,” observe that Harrison Ford appears at the same level of each traversal. Although not shown by the example data graph, Harrison Ford has starred in a number of other movies, and each traversal will discover these connections independently. Thus, portions of each traversal will be identical—they merge after a number of time steps. To exploit this feature, the $n$ independent traversals are combined to save the traversals from repeating work. The state of the single traversal is modeled as a sequence of queues, all of which are initially empty. For each vertex that has been encountered, a map relates vertices containing search terms to their distance from the vertex in focus. The traversal is initialized by adding the vertex ids retrieved by querying the full text indexes of the database to the first queue of the sequence. As each queue in the sequence is processed, neighbors are enqueued appropriately into a queue later in the sequence.

Because certain paths through the data graph are not informative, some edges present in the data graph may be weighted more heavily than others. It is true that both Harrison Ford and Sean Connery are actors, but this relationship applies to most men in the database (directors, musical composers, etc. are the exception). A more meaningful response will state that both appeared in *Indiana Jones and the Last Crusade* or played the same character in different films. For this reason, the reverse of the edge from the cast relation to the role relation is weighted more heavily than other edges (the default weight is 1). When this edge is traversed, the target is not enqueued in the next queue of the sequence. Instead, it is placed in the appropriate queue later in the sequence.
As a concrete illustration, let the reverse edge (that is, the edge from actor to cast) have weight 1000. If a user searches the data graph for Harrison Ford and Sean Connery, both encounter the actor vertex on the first level of the traversal. When the actor vertex is dequeued, it will add all of its neighbors to the appropriate queue. Because the edge weight from actor back to other tuples in the cast relation is 1000, the neighboring tuples will be enqueued for processing in the 1001st level of the traversal. This technique enables more meaningful relationships (i.e., those containing films or characters) to be ranked ahead of the superficial actor relationship. The ability to weight certain edges more or less heavily is modeled after ObjectRank's authority transfer graph [49].

This technique is merely an optimization. Postprocessing could penalize these superficial relationships, but the cost of the traversal grows with each time step as additional vertices are encountered. In a large data graph, the optimization postpones processing vertices and saves a considerable amount of execution time. It is important to note that the optimization does not preclude the actor vertex from being deemed relevant to the query. If the user states that roles are relevant answers to a query, the actor tuple will be ranked as highly as Indiana Jones and the Last Crusade. The optimization simply does not allow information to pass through the actor tuple to form non-informative relationships.

Termination

Equation 3.4 provides an upper bound on the maximum score any tuple can receive. Two methods for early termination of the traversal algorithm exist. First, the maximum possible future score that additional levels of the traversal can contribute to any tuple’s final score is maintained. Note that before the first time step, the maximum possible future score is described by equation 3.4. After each time step, the maximum possible future score decreases proportionally to the damping factor. Following each level of the traversal, the accumulated scores are scanned to find the highest scores. If the maximum score exceeds the next largest by at least the maximum possible future score, the maximum result is immediately output. Once the top-\(k\) results have been identified, the algorithm terminates. In certain cases, this approach cannot terminate execution early. Consider searching the example data graph for “Lost Ark Last Crusade.” Both Harrison Ford and the character Indiana Jones are the same distance from each film. Hence, both will receive the same score from these keyword matches and will tie for the tuple with the maximum score. The tie precludes early termination under the conditions given above so a second means for early termination is required. Once

\footnote{Recall that the size penalty ignores associative entities (like cast information) that are artifacts of the relational model.}
the maximum possible future score drops below a threshold value, the top-k results identified thus far are returned. Using this strategy to terminate execution creates an approximate ordering for results but allows users to specify an arbitrarily small error bound.
Chapter 4

Evaluation

Evaluation of IR systems is notoriously difficult, primarily because the definition of relevance is user-dependent. Extensive user studies are the gold standard for measuring search effectiveness, but in their absence, a number of other metrics must be used for evaluation. It cannot be overstated that there is no established document collection for evaluating the effectiveness of search engines that target structured data. Without such a collection, researchers are required to create ad-hoc experiments using a variety of datasets. In addition, the fundamental differences in the definition of query answers (see section 3.1.2) make it infeasible to compare systems that use different definitions. Evaluation of the techniques presented in this thesis uses a number of traditional IR evaluation metrics. While performance guided portions of the design, the primary focus is the extension of IR scoring methodologies for data graphs. Like many previous works, direct comparison with alternate approaches is not included, but the evaluation still shows that the search engine performs admirably when tested against two dramatically different datasets—IMDb and WikipediaCD.

4.1 IR Effectiveness

As stated in section 2.1, precision and recall are the traditional measures of IR effectiveness. Precision is the ratio of the number of relevant documents retrieved to the number of documents retrieved whereas recall is the ratio of the number of relevant documents retrieved to the total number of relevant documents. Neither of these metrics considers the task of ranking results (i.e., more relevant results should appear before less relevant results). Pooled relevance judgments are used to determine which results are relevant
Datasets and Queries

because relevance varies according to each individual user. In the absence of pooled relevance judgments and
given that many queries used in evaluation contained few relevant results (a reflection of the datasets and
specificity of queries), precision and recall are not directly applied to the search engine. Instead, the selected
measurements, which are partially guided by the measures of effectiveness applied to other systems [50, 54],
do not require every relevant result to be known. First, “top-1 relevant” measures the number of queries for
which the first result is relevant to the query. When confronted with a significant number of results, users
are unlikely to view more than the first page of results [28, 51, 52, 68] and direct most of their attention
towards the highest ranked results [68, 69]. In a similar vein, reciprocal rank is the reciprocal of the position
of the first relevant result to the query. Unlike some other studies, this value is not approximated when
analyzing only the top-k results of queries. A common metric that has been popularized by Internet search
rankings, precision at k measures the precision value for the first k results returned by the search engine.
Unfortunately, this metric is inappropriate when there are fewer than k relevant results. For example, if
there are 3 relevant results for a query, an upper bound for precision at 3 is 1.0 while the upper bound for
precision at 10 is 0.3. The drop is counterintuitive given that the search engine returns the relevant results
first in both occasions. A better measure is R-Precision, which is \( \frac{r}{|Rel|} \) where Rel is the set of relevant results
and \( r \) is the number of relevant results identified within the first \(|Rel|\) results. R-Precision considers the
maximum number of relevant results so a perfect score is always 1.0. Average precision is calculated by
averaging the precision value after each relevant result is retrieved. Mean average precision (MAP) averages
this value across a set of queries. MAP is a good metric since it measures precision across multiple recall
levels.

4.2 Datasets and Queries

The first test dataset consists of part of the IMDb. The database’s schema is shown in table 4.1. One of the
primary arguments for defining query answers as tuples is the existence of frameworks for browsing data.
The browsing framework for this dataset is assumed to be structured similarly to the IMDb website. That
is, a movie listing contains cast information (the actors and characters that each actor plays), an actor’s
page displays all films in which he has appeared along with the corresponding character, and viewing a
character’s record shows all actors who have ever played the character and the films in which the actor
appears. In an effort to provide the most uniform evaluation with previous work, the queries are adapted
Table 4.1: The schema for the IMDb dataset. Bold text denotes the **primary key** of each table, foreign keys are displayed in italics, and **full text indexes** are built over underlined attributes. An index (B-tree or full text) exists for every attribute.

from the evaluation of the SPARK system [70]. The database used by SPARK differs in that it includes film genres; queries specifying a particular genre were replaced with a query specifying a particular cast role (e.g., actor, actress, director).\(^1\) In addition, several more queries were replaced because the relevant results could not be identified (i.e., the original information need was unclear from the query or the database lacked tuples containing the specified keywords). For each query, an information need (often generic, i.e., information about a particular actor) was identified; query results were judged relevant if they addressed the information need. In all, 8 of the 22 original queries were modified. The queries are provided in table 4.2.

A query answer is relevant if the browsing framework contains the desired information. For example, there are three relevant answers for the query “Ark Ford Jones:” the film *Raiders of the Lost Ark*, the actor Harrison Ford, and the character Indiana Jones. Viewing any of these three records will cross-reference the related information pertinent to the query. In certain instances, one answer is preferable. The best result for the query “Harrison Ford” is neither a list of the movies in which the actor has appeared nor a list of every character the actor has played—the preferred result is the actor’s tuple. When a subset of the relevant results is preferred, the top-1 result is required to be part of the preferred results. In other words, when calculating the number of top-1 relevant results, the first result must be among the preferred results. The notion of preferred results does not impact the definition of reciprocal rank. Given the query “Indiana Jones,” if the character tuple is ranked second behind *Indiana Jones and the Last Crusade*, the number of top-1 relevant results is 0 but the reciprocal rank is still 1 (because the first result is relevant, albeit less relevant than the second).

The second dataset is a selection of articles from Wikipedia. The selection includes all of the articles from the 2006 Wikipedia CD selection.\(^2\) Current versions of all of the articles were retrieved from the online

---

\(^1\)The original queries did not cover specific roles; i.e., no query specified a particular cast role by including the keywords actor, actress, or director.

4.2 | Datasets and Queries

1. Woody Allen
2. Holy Grail
3. Harry Potter 2002
4. Titanic Jack Dawson
5. Harrison Ford President
6. Space Odyssey director
7. Bourne
8. 2004 Hanks
9. Godfather Marley
10. Godfather Coppola
11. Mathilda Léon
12. Mathilda Léon 1994
13. Miyazaki director
14. Jason Bourne
15. Wachowski Trinity
16. Hamill actor
17. Clockwork Orange Odyssey
18. Godfather Superman
19. Wachowski Trinity Oracle
20. Fache Léon
22. Robbins Giuntoli

Table 4.2: The IMDb queries are largely derived from Luo et al.’s evaluation [54]; unmodified queries are indicated via italics.

encyclopedia. Due to the complexity of the MediaWiki database, it is not shown here in its entirety, but it is available online.\(^3\) Table 4.3 shows the relevant tables of the MediaWiki schema. To exploit cross-references, the pagelinks table is augmented so it contains a foreign key to the referenced page. The existing MediaWiki framework is used for browsing and navigation. A total of 18 queries (which are provided in table 4.4) were constructed using a variety of keywords. Each individual query has a unique information need; relevant Wikipedia articles must answer the information need. Like the definition of relevant for the IMDb dataset, the notion of preferred results applied to the WikipediaCD dataset. The content of the WikipediaCD dataset differs dramatically from the IMDb dataset: full text indexes exist on the titles and text of articles, and all query keywords are often present in multiple articles. Hence, the completeness factor is expected to play a lesser role when searching articles, realizing the article text places significant emphasis on the basis IR scoring formula (pivoted normalization). In addition, there are few relevant results per query when compared to the IMDb queries—few Wikipedia queries have more than 3 relevant results. Although the MediaWiki schema is not highly normalized and therefore not the primary target of the search engine, the schema is

\(^3\)http://www.mediawiki.org/wiki/Manual:Database_layout
Table 4.3: The schema of the WikipediaCD database. Bold text denotes the **primary key** of each table, *foreign keys* are displayed in italics, and *full text indexes* are built over underlined attributes. Due to the complexities of the complete schema, many relations and attributes are omitted; an ellipsis (…) denotes additional attributes that are not shown for each relation. Note the addition of the `pl_to` attribute to the `PageLinks` table. The original schema did not include a foreign key to the cross-referenced page.

<table>
<thead>
<tr>
<th>Relation</th>
<th>attribute₁, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td><code>page_id</code>, <code>page_title</code>, …</td>
</tr>
<tr>
<td>PageLinks</td>
<td><code>pl_from</code>, <code>pl_title</code>, <code>pl_to</code>, …</td>
</tr>
<tr>
<td>Revision</td>
<td><code>rev_id</code>, <code>rev_page</code>, <code>rev_pagecontent_id</code>, <code>rev_user</code>, …</td>
</tr>
<tr>
<td>PageContent</td>
<td><code>id</code>, <code>text</code>, …</td>
</tr>
<tr>
<td>User</td>
<td><code>user_id</code>, …</td>
</tr>
</tbody>
</table>

1. largest planet solar system
2. natural habitat frog
3. natural habitat alligator
4. Boston Tea Party
5. antelope
6. capital Namibia
7. sockeye salmon
8. Sahara desert
9. formation Hawaiian islands
10. Hawaii island volcano
11. relationship slug snail
12. Elizabeth reign
13. Elizabeth date rule
14. oxbow lake
15. volcano glacier
16. geothermal power
17. Darfur conflict
18. infant mortality Ghana

Table 4.4: The queries used for the WikipediaCD evaluation.

fixed, which makes it an ideal candidate for future comparison with other search engines.

One important issue raised in the discussion of the graph traversal is how to handle uninformative relationships among tuples. For example, most men in the IMDb dataset are related by the actor role—that is, they are all actors. In the WikipediaCD dataset, an article may be deemed relevant to another simply because the same user has modified both. To prevent these relationships from associating two dissimilar actors (i.e., those who do not appear in the same movie or do not play the same character in different films) or articles (i.e., those that have no cross-references between them), certain edges in the graph are weighted more heavily than others. This idea is similar to the authority transfer graph used in ObjectRank [49].
Like ObjectRank, the default weights of the graph edges should probably be fixed by a domain expert, but general intuition guides the process. Figure 4.1 shows the authority transfer graphs used for the IMDb and WikipediaCD evaluations. The weights are arbitrary, intended only to penalize uninformative relationships. Adjusting these values will likely impact the effectiveness of the search engine, but the appropriate values are user-specific since the definition of a relevant result is inherently tied to individual users.

4.3 Results

The scoring formula involves several tuning parameters. Pivoted normalization weighting includes a parameter, $s$, which controls the relative importance of the length of the document. The value of $s$ is fixed at 0.2, the value suggested by IR researchers. The second tuning parameter is the damping factor in the graph traversal. The damping factor controls the maximum score a tuple can receive and, in practice, controls how closely tuples must be related. For the experiments, a range of values, from a very heavy damping factor (0.5) to a relatively light damping factor (0.9), were selected. The appropriate value of the damping factor is likely to vary among datasets, but the damping factor merely expresses a user’s preference for how “related” search results should be. A heavy damping factor requires results to be closely related while a light damping factor facilitates information discovery by identifying lesser known relationships to the user. For example, the extreme damping factor 0.0 does not consider any relationships among the data. For this extreme damping factor, a tuple’s score is based entirely on the information it contains, which is equivalent to returning the best matches from the full text indexes maintained by the database. The third tuning parameter is $p$, which is present in the completeness factor. Recall that $p$ transitions from OR semantics ($p = 1.0$) to AND semantics ($p \to \infty$). Previous work [54] found that even a relatively small value of $p$ (2.0)
could—in practice—enforce AND semantics. The parameter $p$ is varied through a similar range of values, 1.0 to 10. As previously noted, the prestige factor serves primarily as a discriminator for vague queries. The weight of the prestige factor is arbitrarily fixed to be 0.1 of the maximum score of any tuple in the traversal (as given by equation 3.4). In addition, a threshold of 0.001 is set for early termination of the search. When the maximum possible future score drops below the threshold, the traversal terminates immediately. The top-5 results were nearly always output before this threshold was reached; the ordering of additional results is approximate, but the final score cannot vary by more than the threshold value. Additional details regarding how to set these parameters in different datasets is presented in section 4.4.

All queries were executed on a workstation running Windows XP with an Intel quad-core CPU running at 2.4 GHz with 4 GB of RAM. This machine hosted the in-memory data graph. The backend database was PostgreSQL 8.3 release candidate 2 compiled for Linux using GCC 4.1.2. The database server ran on a separate workstation using Fedora 7 with an Intel Xeon processor executing at 2.8 GHz with 4 GB of RAM.

### 4.3.1 IMDb

The first IR metric is the number of top-1 relevant results. As demonstrated by figure 4.2, the search engine ranks a preferred result first for no fewer than 15 of the queries. This metric responds positively to
Figure 4.3: R-Precision for the IMDb queries. Higher values are better, and note the y-axis starts at 0.8. (The values for R-Precision may range from 0.0 to 1.0.) Note the significant drop in R-Precision for the heaviest damping factor, especially under OR semantics ($p = 1.0$). The combination of a heavy damping factor and OR semantics favors results containing a subset of the search terms.

decreasing the damping factor though not as much as increasing $p$ to more strongly enforce AND semantics.

The most severe damping factor (0.5) is the only series that does not benefit by increasing $p$. The heavy damping simply does not allow the traversal to encounter vertices that contain all query keywords. The maximum number of top-1 relevant results is 21. Examining the query logs shows that the query “Space Odyssey director” never returns the preferred result (Stanley Kubrick). Although the desired tuple can receive a score that matches the film *2001: A Space Odyssey*, the search engine breaks ties by preferring tuples that directly contain query keywords. Hence, 21 top-1 results is the practical maximum. Reciprocal rank fails to show any meaningful behavior since it is 1.0 for all query and tuning parameter combinations. This is one reason for the distinction between preferred results (which are reflected by the number of top-1 relevant results) and relevant results (which are reflected by reciprocal rank). Although this data point is uninteresting, it is important to note that the first result is always relevant, even when it is not the best result.

The graphs of R-Precision and MAP in figures 4.3 and 4.4 are more telling. Both graphs reveal that moderate damping factors (i.e., 0.65 – 0.8) generally produce the best results regardless of the value of $p$. For heavy damping factors (i.e., 0.5 and 0.65), $p$ corresponds directly with effectiveness. Light damping
factors (i.e., 0.9 and 0.85) show little correlation with $p$. These results are expected for the queries used in evaluation. Luo et al. rank trees of tuples, and for the given queries, answer trees have a moderate size (e.g., 3–5 tuples). As the size of the tree increases, a lighter damping factor is required to identify the relevant results, but no sizable answer trees are required to answer the queries. When joined with the number of top-1 results, the best damping factor is 0.75. Maximizing the number of top-1 relevant results requires $p$ to be at least 5, but $p = 2$ produces the best values for both R-Precision and MAP. Although a damping factor of 0.65 produces good results as measured by R-Precision and MAP, it lags significantly behind in the number of top-1 relevant results, which suggests the more moderate damping factor is better.

4.3.2 WikipediaCD

As shown in figure 4.5, the search engine always ranks a preferred result first for at least 11 of the queries. The number of top-1 results is relatively constant for each damping factor although larger values of $p$ can boost this metric. The reciprocal rank (shown in figure 4.6) is more interesting for this dataset. Particularly for small values of $p$, the heaviest damping factors (e.g., 0.5 and 0.65) are more effective than the lightest damping factor (0.9). Increasing $p$ produces improvement for all damping factors. Presumably, the lightest damping factor (0.9) allows too much information to flow through cross-references, allowing common search

![IMDb Mean Average Precision](image)

Figure 4.4: A graph of MAP for the IMDb queries. Higher values are better. MAP may vary from 0.0 to 1.0 although the y-axis range in this graph is only 0.11. Like R-Precision, a heavy damping factor and OR semantics fail to produce as good of results as the other combinations of tuning parameter values.
terms to overwhelm the more discriminating terms in the search query. Once again, increasing the value of $p$ results in more effective retrieval. Despite lower values, the graphs of reciprocal rank, R-Precision, and MAP in figures 4.6, 4.7, and 4.8 corroborate the IMDb results. A moderate damping value (0.7) is most effective for reciprocal rank and MAP ($p = 5$); 0.75, 0.8, and 0.85 all maximize R-Precision when $p = 1$. For R-Precision, the extreme damping value 0.5 produces the worst results. Although setting the damping factor to 0.5 provides good results as measured by reciprocal rank, it often fails to rank additional relevant results close to the top, especially when $p$ is small. Similar to the IMDb results, moderate damping factors seem most appropriate. For WikipediaCD queries, a single article is usually the desired result, and a light damping factor facilitates information discovery (e.g., it identifies unexpected relationships among the search keywords) rather than returning the “expected” results. Although the relevance of such results is debatable, they are often unintuitive and therefore not included in the set of relevant results.

Note that there is a reduction in search effectiveness for the WikipediaCD dataset. Examination of the query logs suggests two possibilities for the degradation. First, the search engine uses stemming so search terms are not required to match exactly. For example, singular, plural, and possessive forms of search terms are all considered identical. In the context of Wikipedia articles, stemming is important because the actual search term may appear infrequently while other forms of the root word abound. Using stemming results in
a minor gain in search effectiveness but also has the potential of lowering the precision of search results [22]. Stemming is not a factor for IMDb queries because the names of movies, actors, and characters are matched exactly. Second, additional tests show that certain queries are sensitive to pivoted normalization’s tuning parameter \( s \), which is not varied. Decreasing the value of \( s \) to 0.05 and below significantly improves search effectiveness for some queries. Additional testing is required to determine if the default value suggested by IR researchers (0.2) is appropriate for this dataset.

4.4 Parameter Values

Questions may remain about the derivation of the “correct” value for each parameter included in the scoring formula. Doubtless, the best values are determined via extensive user studies for each dataset encountered by the search engine. Such an approach is not feasible for a design that seeks to minimize the effort applied to system deployment. Therefore, intuition forms the basis for determining the value of each parameter in the scoring formula. To clarify, reconsider the earlier discussion of parameter values in light of the experimentation. Pivoted normalization weighting’s parameter, \( s \), is fixed at the default value suggested by IR researchers. The evaluation performed by Fang et al. suggests that any value less than 0.4 is an
4.4 Parameter Values

Figure 4.7: R-Precision for the WikipediaCD queries. Higher scores are better, and the y-axis range is reduced—R-Precision may vary from 0.0 to 1.0—to show greater detail. Unlike the graph of R-Precision for the IMDb queries (figure 4.3), many values are identical due to the few relevant articles for each query. A severe damping factor significantly reduces R-Precision, particularly when OR semantics are enforced.

acceptable value for $s$, but the best value varies according to the dataset [25]. Informal tests using specific queries that scored poorly (particularly WikipediaCD queries) showed significant improvement for smaller values of $s$. Thus, setting $s$ to 0.2 is simply a good starting point for a given system. The second parameter is the damping factor, which controls the amount of information discovery. A light damping factor (e.g., 0.85–0.9) allows the search engine to encounter more graph nodes, expanding the search frontier. If query answers were defined as trees, the light damping factor would tend to create larger trees than a heavier damping factor. In fact, the heaviest damping factor (0.0) only produces trees with a single node. Provided that the queries used during evaluation reflect common, everyday queries, a moderate damping factor seems most appropriate (and is confirmed by the experimental results). The most complex IMDb query results tend to include two actors related by a film or two films related by an actor. Nevertheless, certain queries do not follow this trend and require a very light damping factor to resolve. For example, determining the relationship between two actors who do not appear in the same film (i.e., the first appeared in a movie with a third actor who co-starred with the second actor in a different movie) requires a lighter damping factor. Hence, the appropriate damping factor is query-dependent. Because users prefer results that contain all query keywords to appear before results containing a subset of the search terms, the completeness factor’s $p$
parameter should reflect this preference. Even moderate values of $p$ (2.0–5.0) produce the desired outcome; if a user wants to explicitly issue a query with OR semantics (or more strict AND semantics), the query interface should allow the exact value of $p$ to be specified. Finally, the most intuitive group of parameters is the edge weights present in the authority transfer graph. The authority transfer graph is intended to remove the uninteresting, intuitive relationships from the data graph. As explained earlier, if a user initiates a search containing two names, the user probably already knows that both individuals are actors, actresses, directors, etc. The authority transfer graph suppresses these intuitive relationships to prevent them from cluttering the list of results. The edge weights chosen for evaluation are intended to reflect intuitive understanding and nothing more. In the case of the MediaWiki authority transfer graph, the edge from the `PageLinks` table to the `Page` table has weight 3 because most users will not be interested in results which span multiple pages unless no single page will satisfy the information need. In conclusion, determining the appropriate value for each tuning parameter is not only user-dependent but also query-dependent. There may not be any single best value for any individual user or group of users. General intuition seems a good method for setting parameter values although additional experimentation doubtlessly would fine-tune the result rankings for any dataset.
Chapter 5

Conclusions and Future Work

5.1 Conclusions

The importance of automatic IR has been recognized for more than half a century. As computers gained computational power, IR techniques expanded to include complex indexing and ranking models that maximize the relevance of retrieved documents. The prevalence of Internet keyword searches testifies to the importance of existing indexing and ranking techniques. Nevertheless, these approaches are quickly growing outdated. Indexing and ranking models developed for unstructured documents cannot be immediately applied to relational data—relationships among the data must be considered. As the Internet transitions from static web pages to dynamic web portals, novel techniques must be developed for effective searching. These techniques challenge the traditional approaches for two reasons. First, as unprecedented as the size of the Internet is, the information hidden in the deep web is still orders-of-magnitude larger. These vast repositories demand new indexing techniques due to the infeasibility of duplicating this information in a separate index. Second, the correct granularity of results is difficult to achieve. A large XML document should not be returned as a query result when a single element contains the desired information. In addition, the single element should not necessarily include all of its children—some child elements may have no relevance for the given query. In contrast, an isolated tuple in a relational database often lacks context. Its relationships with other tuples must be understood before it will be considered a relevant answer.

To achieve both of these goals, this thesis introduces novel techniques for indexing and scoring search results for structured data. The hybrid index practically eliminates redundancy with the data stored in
the underlying relational database. Moreover, reusing the relational database’s existing full text indexing capabilities allows the hybrid index to remain up-to-date far longer than possible with a naive indexing scheme that explicitly tracks which tuples contain each search term. Allowing the relational database to manage these indexes further supports efficient execution as the data never needs copied from the database, which grows expensive as the size of the data increases. In addition, the intuitive scoring model enables traditional IR scoring formulas to be used. Because these scoring formulas have already undergone extensive testing, it is highly desirable to reuse them instead of developing new weighting techniques. Reliance upon an existing framework for browsing the data sidesteps the complexities associated with displaying the relationships among search results. The solution is not ideal and, as will be mentioned later, is a critical part of future work in this field.

Typical IR evaluation metrics show that the scoring model performs admirably on two dramatically different datasets. The IMDb dataset’s normalized structure emphasizes the iterative scoring model while the Wikipedia articles stress the baseline IR score and—to a lesser extent—the relationships among data via the cross-references among related articles. The high effectiveness for both datasets suggests wide applicability in that the scoring model transitions easily across normalized and unnormalized content.

The Crosspoint system provides the primary motivating factor for this work. It targets a problem endemic to a diverse range of systems—finding the correct collaboration partners is essential to solving complex problems encountered daily in the real world. Crosspoint provides a simple means for describing the problem and facilitates collaboration by recommending those who possess the desired expertise and availability to answer the underlying information need. The search engine provides a simple interface to achieve these goals by identifying similar problems that have previously been addressed and by recommending collaborators (e.g., those who have previously solved similar problems). This ability plugs a hole present in existing collaboration software, which assumes the user already knows the identities of those with whom he wishes to collaborate.

5.2 Future Work

For future work, three broad avenues exist, starting with performance. The in-memory data graph requires a significant amount of main memory for even moderately sized graphs (i.e., 100,000 vertices and 250,000 edges). Existing open-source graph packages provide a convenient API as well as a number of common graph
5.2 Future Work

algorithms, but the generality provided by these packages comes at a significant price. For larger datasets, the generality must be sacrificed to allow the data graph to remain in memory. Informal tests suggest that a compact array-based implementation reduces the memory footprint by as much as 75%. Additional evaluation is required to determine the tradeoffs between reducing memory utilization and increasing computation time. For even larger datasets (e.g., the entire Wikipedia), it may be necessary to partition the data graph so infrequently used portions remain on disk. Of course, effective partitioning techniques are required to ensure locality among the individual partitions. Existing graph packages assume memory utilization is not an issue and do not provide a means to write individual portions of the graph to disk. Also in the performance vein, more efficient graph traversal algorithms are required to increase search efficiency. The existing search engine is not interactive (that is, requiring < 1 second for keyword searches) for either the IMDb or WikipediaCD dataset. The search time must be reduced before the search engine is considered a viable solution for searching large data repositories. While optimizations provide one avenue for reducing search time, any optimizations are almost inherently tied to the underlying scoring formulas. Alternate scoring formulas (e.g., Okapi scoring) may be desirable for some applications; this issue emphasizes general algorithms that meet the desired performance goals rather than specific optimizations.

The visualization of search results remains another prominent issue. The simplistic ranked list of results is not the quintessential interface. Kaki and Aula [71] show that even a simple modification—filtering results by categories—increases the speed and accuracy of retrieval tasks. Search result interface improvements are not only possible but also essential. Based on the number of different techniques that have been suggested, it is clear that the research community has not yet determined the best method to display search results. In the context of searching structured data, a ranked list cannot capture the relationships among the results, which is perhaps the most important factor for a relational search engine. This inability limits knowledge building by association and domain exploration, both of which are critical components of exploratory search [72]. For this reason, the existing browsing framework is important: it allows users to navigate among the various search results to discover their relationships. As described in section 3.1.2, attempting to display tree-structured search results is fraught with difficulties. Commonalities among the search results are easy to overlook, and the top-\(k\) results may repeat the same nodes, showing different combinations of the relationships among them. Both problems suggest that a more ideal solution exists. As Sebrechts et al. [66] state, humans process visual cues much more acutely than scanning text. Incorporating sophisticated graph visualization techniques into the search results interface addresses the central problem of commonalities in the results.
and furthers both data discovery and exploration. Users should be able to “drill down” on graph nodes and edges to determine the information each contains. Given that graph visualizations are already a common means for visualizing networks, the semantic web, and other systems that contain significant structure, it is possible to incorporate these visualizations into a user interface, but extensive user studies are needed to determine their effectiveness.

Finally, user studies should be conducted to enhance the effectiveness of the search engine. Without established datasets for evaluation, user studies are the gold standard for evaluating effectiveness because the definitions of both relevance and importance of search results are clearly user dependent. In fact, the largest existing user study for a comparable system used 17 Ph.D. students [18], and the evaluation only considered information discovery using keyword search verses SQL. Especially once a visualization engine provides a novel method to view search results, user studies should be used to compare the various search engines already described in the literature. Cutrell and Guan [68] predict a wide range of studies in user interface design for IR systems as these systems adapt to specific retrieval tasks. While it is clear that tree-structured results are less than ideal, it may be the case that users can overcome the difficulties with a modicum of effort. However, only extensive user studies can reveal this fact. The user studies must be conducted on a diverse group of individuals to ensure the benefits of a particular approach are not limited to specific groups of users (e.g., computer scientists). The flexibility of the techniques presented in this paper should show immediate value as the search engine may interface with any relational database, offering the user studies the ability to be specific to an individual’s area of expertise (e.g., medical students could search the PubMed database).
Bibliography


[3] Jesse Alpert and Nissan Hajaj. We knew the web was big.... [http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html](http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html), July 2007.


## Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBLP</strong></td>
<td>Digital Bibliography &amp; Library Project.</td>
<td>23</td>
</tr>
<tr>
<td><strong>DTD</strong></td>
<td>document type definition.</td>
<td>20</td>
</tr>
<tr>
<td><strong>IMDb</strong></td>
<td>Internet Movie Database.</td>
<td>vi–viii, 15, 31, 48–59, 62, 63</td>
</tr>
<tr>
<td><strong>IR</strong></td>
<td>information retrieval.</td>
<td>ix, 1–3, 13–20, 23, 28, 29, 31, 33–41, 48, 51, 53, 58, 61, 62, 64, 71</td>
</tr>
<tr>
<td><strong>MAP</strong></td>
<td>mean average precision.</td>
<td>viii, 49, 56, 60</td>
</tr>
<tr>
<td><strong>MST</strong></td>
<td>minimum spanning tree.</td>
<td>viii, 27</td>
</tr>
<tr>
<td><strong>NIST</strong></td>
<td>National Institute of Standards and Technology.</td>
<td>17</td>
</tr>
<tr>
<td><strong>RFI</strong></td>
<td>Request for Information.</td>
<td>viii, 6, 7</td>
</tr>
<tr>
<td><strong>SME</strong></td>
<td>Subject Matter Expert.</td>
<td>viii, 6, 8</td>
</tr>
<tr>
<td><strong>SOA</strong></td>
<td>Service-Oriented Architecture.</td>
<td>v, viii, 10, 12</td>
</tr>
<tr>
<td><strong>TREC</strong></td>
<td>the Text REtreival Conference.</td>
<td>14, 17</td>
</tr>
<tr>
<td><strong>UI</strong></td>
<td>user interface.</td>
<td>12</td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td>the eXtensible Markup Language.</td>
<td>2, 15, 16, 20, 21, 30, 34, 61</td>
</tr>
</tbody>
</table>
Glossary

average document length  The average length of documents in the complete document collection. 18, 38

document frequency  The number of documents, measured across the entire document collection, that contain the specified term. 17, 18, 38, 42

document length  The length of a given document. 17, 18, 38, 40

inverse document frequency  A factor common to many information retrieval weighting methods, inverse document frequency awards documents that contain uncommon terms that are present in the search query. 17, 18, 21, 25, 38, 42

normalized document length  Since longer documents contain more terms than shorter documents, modern IR weighting methods normalize document length to counteract this effect. 17, 38

normalized term frequency  Using raw $tf$ in weighting methods is suboptimal since users expect documents containing more query terms to rank higher than documents containing fewer; to this end, the $tf$ factor is dampened, e.g., a logarithmic $tf$ factor. 17, 38

term frequency  The frequency of a given term in a document. ix, 17, 18, 21, 38, 40, 71