Toward Practical Relational Keyword Search Systems

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Joel Coffman

May 2012

Abstract

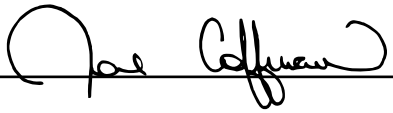
The amount of information in the world is increasing exponentially. Keyword search has proven to be an effective method to discover and retrieve information online as evidenced by the success of Internet search engines. Unfortunately, many common information management systems do not support the familiar keyword search interface that people now expect. Web sites, corporations, and governments all use relational databases to manage information, but keyword search in relational databases is difficult due to data transformations that eliminate redundancy and ensure consistency. Relational keyword search enables users to retrieve information and to explore the relationships among that information all via a familiar interface.

Although a decade has passed since keyword search in databases became a hot topic for academic researchers, little progress has been made in the interim. In particular, no systems have appeared outside the academic community despite a long-standing promise to revolutionize the way people interact with information. This dissertation addresses the challenges inherent in transitioning relational keyword search techniques from the computer science community to practical systems that can be deployed against existing data repositories. A key contribution of this research is an extensive benchmark specifically designed to evaluate relational keyword search techniques. Extensive empirical experiments both identify why existing search techniques cannot handle existing data repositories and identify areas for future research in this field. Improvements to relational keyword search come in the form of two novel ranking schemes that significantly improve search effectiveness. The first explicitly enforces users' preferences regarding the order of search results. The second uses machine learning to weight the various scoring factors that have been proposed to date in the literature, and analyzing their importance indicates a number of factors that can be excluded without sacrificing search effectiveness. This dissertation also examines key issues related to the evaluation of proposed search techniques that derail many existing evaluations from accurately reflecting real-world retrieval tasks. This work bridges the gap between academic research and keyword search techniques that are ready to be deployed in real-world environments.

Approval Sheet

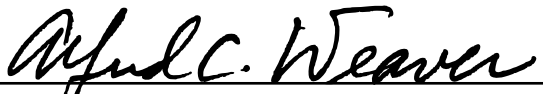
This dissertation is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

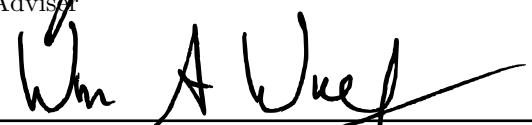


Joel Coffman

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

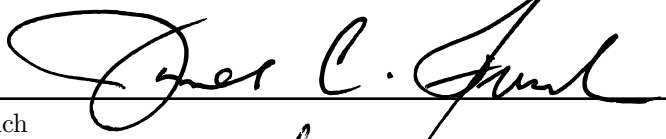


Alfred C. Weaver, Adviser



William A. Wulf, Committee Chair

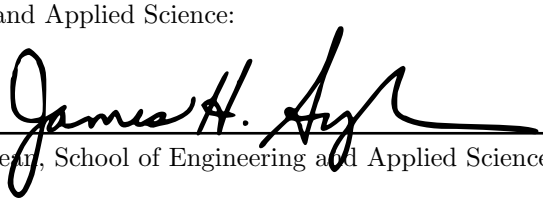
Sang H. Son



James C. French

Ronald D. Williams, Minor Representative

Accepted for the School of Engineering and Applied Science:



James H. Aylor, Dean, School of Engineering and Applied Science

May 2012

In Memory of
Grandma Patty and Grandpa Samuel

"I have fought the good fight, I have finished the race, I have kept the faith."
(2 Timothy 4:7 NIV 1984)

Acknowledgments

Not to us, O LORD, not to us but to your name be the glory, because of your love and faithfulness.

(Psalm 115:1 NIV 1984)

It's difficult sometimes to know who to thank first when you owe so much to others that the LORD brings into your life. Mother and Dad and Andrew, thank you for your constant support, especially during the last five years. Your love and godly example mean so much to me, and your encouragement to persevere during the difficult times has helped to keep me on track. Thank you for your many prayers on my behalf, which I know make a difference in my life. I am most blessed to part of our family. Grandma Janny and Grandpa Bill, thank you for your influence and presence in my life. I value our weekly conversations and the time we spend together. Even though Grandma Patty and Grandpa Samuel are not here to witness this day, I know that they celebrate with me. I miss our conversations and their presence. Rebecca, you are so special to me. I treasure our relationship and the time we've spent together during the past three years.

Thank you to all the graduate students at the University of Virginia who have made my experience here so enjoyable. Claire, Andrew, Colleen, Michelle, Luther, and Dan, thank you for your help and encouragement along the way to this milestone. I will miss all our conversations and the time we've spent together. To the Frey family and Nanny Grace, thank you for adopting me during my time in Charlottesville. John, thank you for the time you chose to invest in me and challenging me to become the man that God desires me to be. I'm so appreciative of your influence and introducing me to midnight hikes. Thank you to Josh, Lindsay, Tim, Deborah, Crystal, Loren, Sunshine, Paul, Sarah, Brooke, Jake, Heather, and all the others at home group who have prayed for me during the past four years. Peter, I miss all the fun times that we had together at Furman, especially playing *FIFA Soccer*. Jimmy, I still cannot forget your challenge to be a "six-step worshipper" in my pursuit of the LORD.

My opportunity to be here today is also the direct result of the many professors and teachers who have invested their time in me. Alf, thank you for your guidance during the past five years and your patience, especially as I've worked on my own research ideas. Thank you for convincing me to come study at UVa and for supporting me from the very beginning of my graduate studies. I certainly would not be here today

without you. Dr. Treu, thank you for insisting that I apply to UVa and your letter of recommendation on my behalf. I also owe you a debt of gratitude for teaching CS 40 during my time at Furman and piquing my interest in databases. Dr. Healy, thank you for introducing me to research and all your wonderful classes at Furman. Mr. Moulton and all the other teachers at Wilson Hall, thank you for laying a firm academic foundation for others to build upon. I'm amazed by how frequently I apply something you all taught me.

Thank you to the many others—many of whom I probably don't even know—who have prayed for me during the past five years. To the LORD belongs all glory and honor and praise: "To him who sits on the throne and to the Lamb be praise and honor and glory and power, for ever and ever!" (Revelation 5:13 NIV 1984). Father, the mercy and grace that you give me so freely is incomprehensible at times. All that I am and have belongs to you.

> Therefore, prepare your minds for action; be self-controlled; set your hope fully on the grace to be given you when Jesus Christ is revealed. As obedient children, do not conform to the evil desires you had when you lived in ignorance. But just as he who called you is holy, so be holy in all you do; for it is written: "Be holy, because I am holy."
>
> [. . .] live your lives as strangers here in reverent fear. For you know that it was not with perishable things such as silver or gold that you were redeemed from the empty way of life handed down to you from your forefathers, but with the precious blood of Christ, a lamb without blemish or defect. Through him you believe in God, who raised him from the dead and glorified him, and so your faith and hope are in God. (1 Peter 1:13–19, 21 NIV 1984)

Contents

List of Tables

List of Figures

List of Symbols

μ **mean** The arithmetic mean of a population or sample.

σ **standard deviation** The standard deviation of a population or sample.

avgdl **average document length** The average length of documents in a 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 that rewards 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

The world is awash with information. The amount of digital information world-wide increases by an order of magnitude every 5 years [GCM+08]. Humanity created an estimated 150 exabytes of data in 2005; by 2010 that number soared to an estimated 1200 exabytes [Eco10]. By 2015, Cisco estimates that Internet traffic will exceed 966 exabytes of information [Cis11].

This increase in information is rapidly outstripping humanity's (and even technology's) capacity to absorb it [GCS+07]. In 2008, it was estimated that American households were bombarded by 3.6 zettabytes of data [BS09], which represents the consumption of approximately 34 gigabytes of data per person each day. The Large Hadron Collider can generate 100 terabytes of data each second [Col08], which is far more than can currently be recorded on any storage medium in the same duration. Between 2006 and 2011, digital information grew at a compound annual rate of 60% [GCM+08], and this growth rate continues to increase! Unfortunately, the sheer quantity of data makes it increasingly difficult to use it effectively. For example, a torrent of scientific publications makes it virtually impossible for medical doctors to stay abreast of the latest research and treatment techniques. It has been estimated that epidemiologists would need to spend 21 hours each day to remain current in just their subfield of medicine [AHE+04]. Whereas a lack of information has been a critical problem in years past, now the most pressing challenge might be filtering the available information to discard the irrelevant.

Even though humans may not be capable of processing the volumes of information just described, computers are ideal for this task. Computers can filter billions of bits of data and return only those that satisfy some need of their user. Computers can spot previously unrecognized patterns in datasets that are far larger than one person could ever fathom processing. Today most individuals already have experience using computers to filter enormous quantities of information: they do it implicitly via Internet search engines.

3,000	3 thousand	images uploaded to Flickr every minute [She10]
50,000	50 thousand	hours of video uploaded to YouTube every day [Wal10]
20,000,000	20 million	videos uploaded to Facebook every month [Law10a]
200,000,000	200 million	Tweets per day on Twitter [Twi11]
30,000,000,000	30 billion	pieces of content shared on Facebook every month [MCB+11]
235,000,000,000,000	235 terabytes	data collected by US Library of Congress in April 2011 [MCB+11]
20,000,000,000,000,000	20 petabytes	data processed by Google each day [DG08]
1,800,000,000,000,000,000,000	1.8 zettabytes	information created and replicated in 2011 [GR11]
50-fold		growth in information managed by enterprise datacenters during next decade [GR11]

Figure 1.1: Size and growth rate of information

The ubiquitous search text box has transformed the way people interact with information. Keyword search is the preferred means for individuals to discover and to retrieve information [Fox02]. Nearly half of all Internet users use a search engine daily [Fal08], performing in excess of 4 billion searches each day [com10]. The success of keyword search stems from what it does not require—namely, a specialized query language or knowledge of the underlying structure of the data. Internet search engines allow people to use the information published online effectively, retrieving only the most relevant web pages from more than 600 million web sites [Net12] containing more than 1 trillion unique URLs [AH07].

Despite the enormous success of Internet search engines making content accessible to users, it is well known that much content available online remains unindexed [AH07]. Web sites serve increasing amounts of data on-demand in response to user requests (e.g., checking a library's holdings to see if a book is available) and to personalize web pages. Information served dynamically in response to user requests cannot be processed by traditional indexing techniques that crawl web sites to discover links to new content. The hidden web is so named because its content is not visible to traditional search engines. Examples include topical databases (e.g., SEC corporate filings, medical databases, patent records) and publications (e.g., library holdings and papers published in conferences). As another example, social networking sites are used by 65% of adults who are online [MZ11], but accessing this content requires authentication, and a myriad of privacy controls restrict what information is available to other users. Nevertheless, these sites provide a gold mine of information that people want to search through including semantically-rich user profiles, relationships among users, and user interactions in the form of postings, chat transcripts, and email correspondence.

Databases traditionally house all the aforementioned sources of data, and while the information technically is searchable via web forms, users must search each data source independently. Altogether the size of these searchable databases is estimated to be several orders of magnitude larger than the static web [Ber01, RGM01, LV03]. Other databases—for example, medical health records—are not publicly available online but are

rapidly becoming critical infrastructure both for businesses and for individuals. The explosive growth of social networks, microblogs, and electronic data in general (e.g., electronic medical records) all contribute to the ever-increasing amount of information stored in databases, information that cannot be indexed by even the most sophisticated retrieval systems in the world today. Novel search techniques are required to access this information efficiently and effectively.

This chapter introduces the problem of keyword search in relational databases. It starts by highlighting two different approaches to managing information and why it is advantageous to combine them. Appendix A explores this material in greater depth, including an overview of the historical differences between databases and information retrieval. The overview of both fields highlights their differences and why integrating them (e.g., by supporting keyword search in databases) is a difficult task. Next, different applications for relational keyword search are presented. A comparison between web search and relational keyword search illustrates the greater power provided by the latter search techniques. This chapter concludes with the key contributions and outline of this dissertation.

1.1 Databases and Information Retrieval

Despite their common objective to manage information, databases (DBs) and information retrieval (IR) systems evolved independently and developed their own unique models to allow users to access information. The need to reconcile these differences is well-known [CRW05, AYCR+05, Wei07], but the desired integration into a single platform has proven elusive. A glance at the original applications driving the development of these systems provides some insight into their continued separation. Databases were designed for business applications such as accounting, payrolls, and inventory management. For these applications, data consistency and precise query processing are paramount. The former in particular is clearly seen in data models that avoid data redundancy. In contrast, IR systems were developed to aid library users. In this context, ranking schemes and user satisfaction are key because many documents may pertain to a given query. Interestingly, the problem eliminated by databases (that is, data redundancy) is inherent to IR. A retrieval system must distinguish the most relevant documents in a set that often contains similar—if not identical—information nuggets pertaining to the query. In a database, core IR techniques (such as ranking) have little place where the focus is on precise queries. For example, databases are used to retrieve a particular customer's account information, find all the customers who recently purchased a particular product, and update the salaries of all employees to give them a cost of living raise. IR queries are poorly defined in contrast to these tasks.

Given their significant differences, why is there so much interest in integrating databases and IR systems? In short, the growth of digital information worldwide is allowing individuals to explore data in ways not

previously possible. Databases provide one way to manage this information, but existing query languages pose a significant barrier to using these resources effectively. Hence, there is a need for systems that enable users to search datasets with complex predicates while ranking results in a meaningful way. From the IR perspective, answering queries with complex predicates and ranking results by their relevance both require sophisticated query optimization to achieve reasonable performance. The first requirement (i.e., efficiently answering queries with complex predicates via extensive query optimization) is an area where databases currently enjoy a considerable advantage. Examples of application areas that would immediately benefit from the integration of database and IR technologies include digital libraries (with metadata about content and user annotations), customer support, personal information management, and health care, all of which currently exhibit tremendous growth.

1.2 Vision

Data warehouses hold massive quantities of information, but accessing this information is typically restricted to experts due to complex query languages. Given the success of Internet search engines in enabling even non-technical users to satisfy their information needs via a keyword search interface, researchers have proposed extending this retrieval paradigm to structured data (i.e., XML and relational data). The structure of relational databases introduces a myriad of research challenges in supporting efficient and effective keyword search. The goal of keyword search in databases is to provide users with a simple interface to discover and to retrieve information. The objective of the work described in this dissertation is to transition existing search techniques from research curiosities into real systems that meet the aforementioned goals. To meet this objective, approaches should be capable of being deployed against previously unseen databases while proffering the same high quality search results that users have come to expect from Internet search engines. This section briefly addresses the reasons why structured query languages are not satisfactory and describe a variety of applications that would benefit from keyword search in relational databases.

Traditional query languages for structured data such as XQuery [BCF+07] (for XML) and SQL [CAE+76] (for relational databases) are notoriously difficult to learn. The challenge of teaching SQL to even undergraduate computer science students is well-documented [Mit98, SOSL04]. The learning curve is sufficiently high that it is unreasonable to expect non-technical users to learn existing query languages, particularly when keyword search interfaces have met with so much success online. Even query by example (QBE) [Zlo75] requires that users have some understanding of the underlying relational schema—e.g., knowing which tables and fields are pertinent to the query. Furthermore, these approaches were not designed to take advantage of the ranking schemes developed by the IR community to rank results. Even expert users who are already

familiar with structured query languages may find it considerably easier to retrieve information using a keyword search interface [SW05]. For example, enterprise databases often contain hundreds or even thousands of interrelated tables [YPS09, Sri10], which presents a steep learning curve when first encountering the schema. Even when the schema is familiar, specifying all these relationships using traditional query languages is cumbersome. In contrast, relational keyword search systems shield their users from the complexities of writing queries, deciding how to rank search results, and requiring detailed knowledge of the database schema.

Applications

Although some potential applications for keyword search in databases have already been mentioned, these examples have not been explored in detail. Four applications are presented in the following paragraphs to illustrate how these systems would improve users' ability to retrieve information. In many instances, keyword search for databases generalizes existing solutions, obviating the current need for developers to create customized solutions for each different database.

Federated Search Databases that provide content to web sites present two challenges to traditional search engines. First, some databases (i.e., the hidden web mentioned previously) are searchable but their content is not linked to statically, which limits search engines' ability to incorporate these data sources. Second, even when the information can be included in a search engine's index, periodically crawling the content (e.g., monthly) inevitably results in outdated information appearing in search results. Real-time search attempts to integrate content immediately into a search engine's index [Cor10], but most real-time search services rely on customized APIs that are specific to each data source. While web sites often provide their own search form to find content on their site, these facilities are typically far inferior to Internet search engines, neither providing the sophisticated ranking schemes that give high-quality results nor supporting the variety of special operators that many users now expect.

Federated search addresses all these issues. In federated search, the results from different search engines (e.g., for each database powering dynamic web sites) are combined to give the appearance of a single system. Suppose that each database advertises a web service for searches. When a user enters a query into an Internet search engine, each web service receives the query, and their results are returned to the search engine to be integrated into the final list of results presented to the user. Such a service is not unlike the nascent real-time search features currently provided by search engines, but a federated approach would standardize APIs and extend support to any database, not just the most popular web sites in use today.

Another application building off this general approach is searching within social networking web sites. Currently, authentication is required before users can access content, and the amount of content visible to

each user is controlled by various privacy settings. A keyword search system for databases can enforce these privacy settings as part of the search process instead of filtering the results that must be restricted due to privacy controls. Assuming that users allow an Internet search engine to execute queries on their behalf (by providing the necessary authentication credentials), even content from social networking sites can be aggregated seamlessly into a single page of results.

IMDb The Internet Movie Database (IMDb)[1] is a web site devoted to movies, television shows, video games, and the fictional characters and cast members appearing in these forms of media. Although its data focuses exclusively on entertainment, IMDb does illustrate the key advantages of keyword search in databases. Moreover, its content is widely accessible, which makes its examples much more intuitive than domain-specific datasets that require some level of expert knowledge. For this reason, many examples provided in this dissertation are based on IMDb although the search techniques are just as applicable to other databases. Even though the following two examples of keyword search in IMDb are entertainment-oriented, many business-oriented examples are strikingly similar, albeit with different data (and relationships among that data).

Milgram popularized the notion of "six degrees of separation" through his small world experiments [Mil67, TM69]. These experiments investigated the average path length in social networks and have since been expanded to professional communities that have well-defined associations among members. For example, researchers in mathematics can be characterized by their "Erdös number" [GI95], which is the number of collaborations (as recorded by shared authorship of peer-reviewed publications) between the individual and Paul Erdös, the most published mathematician in history [New01]. In the entertainment genre, the game "Six Degrees of Kevin Bacon" challenges players to link an individual to the actor Kevin Bacon via the minimum number of films in which each appears. For example, Sean Connery was in *The Hunt for Red October* with Ned Vaughn and Ned Vaughn appeared in *Frost/Nixon* with Kevin Bacon. Because general-purpose search engines are not designed to discover these relationships, one must often use specialized search forms (such as those provided by "The Oracle of Bacon"[2]) to obtain the desired information. These specialized systems provide high-quality results by exploiting domain-specific information, which makes them unsuitable for different databases. Because keyword search in databases explicitly identifies the connections among related bits of data, they are more general than these existing solutions, enabling users to pose more challenging questions (e.g., how are the actors Sean Connery, Harrison Ford, and Kevin Bacon all related?).

[1] http://www.imdb.com/
[2] http://oracleofbacon.org/

Merging information from IMDb with users' viewing habits[3] would allow answering other common questions such as determining which recently-viewed films feature a particular actor or actress. Typical methods to answer this question invariably require users to navigate through the data to identify what is relevant. For example, the user views the cast information for the film being viewed, selects the individual, and manually inspects the list of films in which the individual has appeared to determine which one(s) the user has previously seen. A keyword search interface could automate this task by implicitly ranking the individual's films first by whether or not the user has seen them and then by general popularity.

EMR Electronic medical records (EMRs) are being widely adopted by the healthcare industry as a way to lower costs and to improve the quality of healthcare. In fact, the market for medical clinical information providers is estimated to reach $10 billion (USD) by the end of the decade [MCB+11]. Creatively using existing healthcare data to improve the efficiency and quality of health care could be worth $300 billion each year in the United States alone [MCB+11]. As an example of using existing data more effectively, consider the following scenario for how keyword search in databases can assist doctors in treating patients.

> Several days after a successful heart surgery, a patient is diagnosed with a staph infection. Upon entering the information into the patient's records, the physician is alerted that another patient in the hospital has also been diagnosed with a similar infection. Given the difficulty of treating staph infections—particularly those resistant to antibiotics—identifying the source of the infections is critical. By simply entering both patient's names into a search text box for the hospital's patient database, the physician learns that both patients shared the same recovery room following their surgeries. After moving out of recovery, the patients were on separate floors of the hospital and received care from different doctors and nurses. Hence, the recovery room is the likely origin of the infection.

Although this information is readily available in the patients' records, systematically checking for common relationships is a laborious, time-intensive task. Moreover, there is always the risk that important relationships might be overlooked or the physician might not know the correct question to ask to identify the common source of infection. A keyword search system for the medical records enables a non-technical user (i.e., an individual who does not know structured query languages) to ask complex questions whose answers are critical to the well-being of patients.

National Security Given the amount of information housed in today's data warehouses and government records, keyword search in databases has the potential to revolutionize law enforcement and counter-terrorism,

[3] This information is already used by companies such as Netflix and Hulu to improve recommendations of other films the user might enjoy.

both of which are areas where it can be difficult to extract meaningful information from the flood of available data. For example, the US military collected 24 years worth of video footage from aircraft drones flying over Afghanistan in 2009; drones deployed in 2010 had 10 times as many data streams, and those deployed in 2011 had 30 times as many [Eco10]. Using the available data effectively is now more difficult than collecting it.

Consider a law enforcement official who is warned about a terror cell thought to be operating in the region. Several days later, the agency is alerted to two separate purchases of large quantities of ammonium nitrate by a local farmer. Even though ammonium nitrate is commonly used as fertilizer, it can also be used as an oxidizing agent in explosives. Law enforcement officials would naturally want to determine if the purchase posed any risk—i.e., if the ammonium nitrate might be used to construct a bomb. To determine the risk, law enforcement officials must determine if the farmer has any connection to the terror cell, but identifying the connection can be difficult. Fortunately, keyword search in databases is designed to find the relationships among different entities. By entering the name of the terror cell and the farmer, an analyst is given the following chain of connections, mined from government databases and data warehouses.

- The terror cell is headed by an individual codenamed "Felix."
- A series of fraudulent transactions, which have been attributed to Felix, identify him as visiting Istanbul several weeks prior.
- During this same period, an individual named Reuben Watts visited Istanbul; Reuben has also made and received several phone calls from Istanbul in the interim period.
- Credit card transactions place Reuben Watts and Manuel Vasquez in the same bar three times during the previous month. Moreover, the bar is not conveniently located for either individual.
- Manuel Vasquez was recently hired as a day laborer by the farmer.

The connection between the terror cell and the purchase of the fertilizer suggests that ammonium nitrate might be used to construct a bomb. Investigators can use this information, which would be difficult to discover using existing tools, to prevent a possible incident.

Intelligence databases also fuel another application (which is similar to federated search) for keyword search in databases. When confronted with a problem, analysts often need to know which individuals in their organization have the expertise to solve that problem. This knowledge is built over the course of the analysts' careers, via their personal contacts and professional associations forged from previous collaborations. Similar to social networking sites, government databases that identify analysts' areas of expertise are tightly controlled: the information is not publicly available, which makes it difficult to design tools that allow analysts to use the data effectively. Keyword search techniques for databases can be deployed against these

repositories to provide high-quality search results without knowing the database's schema and without any prior knowledge of the type of data stored in the database.

1.3 Keyword Search in Relational Databases

Keyword search on semi-structured data (e.g., XML) and relational data differs considerably from traditional IR. For instance, the granularity of search results must be reconsidered for both these data sources. An XML document might contain a single element that is pertinent to a given query along with many unrelated elements. The XML dump of the Digital Bibliography & Library Project (DBLP)[4] currently contains more than 1.3 million publications; searching this repository for a particular paper should return only the information about that paper and not the complete bibliography.

Identifying relevant results is further complicated due to the fact that a physical view of the data often does not match a logical view of the information. Relational databases normalize data to eliminate redundancy. Logically related information often appears in different relations, and foreign keys provide the only link between the data. Whenever search queries cross a relationship modeled in the database (i.e., a subset of search terms is present in one tuple and the remaining terms are found in related tuples), the data must be mapped back to a logical view to obtain meaningful search results. Figure 1.2 shows several logical views of information from IMDb where the bottom two logical views combine information about different entities (i.e., people, characters, and films). Furthermore, the implicit assumption of keyword search—that is, the search terms are related—complicates the search process because typically there are many possible relationships between two search terms. This realization leads to tension between the compactness (and consequently performance) and coverage of search results. Based on the logical views shown in Figure 1.2, it is unreasonable to return actors' filmographies when a query mentions only a film title. The filmography view should be returned when it is necessary to answer a query's underlying information need.

Creating coherent search results from discrete tuples is the primary reason that searching relational data is significantly more complex than searching unstructured text. Retrieval systems typically index unstructured text at the same granularity as the desired results (e.g., by documents or sections within documents). This task is impractical for relational data because an index over logical (or materialized) views is often an order of magnitude larger than the original data. For example, Su and Widom [SW05] indexed a subset of the logical views in two databases and found that their index exceeded the size of the original data between two and eight times. Baid *et al.*'s experiments [BRL+10] with EASE [LOF+08] suggest that such an index can

[4] http://dblp.uni-trier.de/

People

Person
Harrison Ford
Sean Connery
Karen Allen
Steven Spielberg

Characters

Character
Indiana Jones
Marion Ravenwood
Professor Henry Jones

Films

Film	year
Raiders of the Lost Ark	1981
Indiana Jones and the Last Crusade	1989

Filmography

Person	Role	Film	year
Karen Allen	actress	*Raiders of the Lost Ark*	1981
Sean Connery	actor	*Indiana Jones and the Last Crusade*	1989
Harrison Ford	actor	*Raiders of the Lost Ark*	1981
		Indiana Jones and the Last Crusade	1989
Steven Spielberg	director	*Raiders of the Lost Ark*	1981
		Indiana Jones and the Last Crusade	1989

Cast

Film	year	Person	Character
Raiders of the Lost Ark	1981	Harrison Ford	Indiana Jones
		Karen Allen	Marion Ravenwood
Indiana Jones and the Last Crusade	1989	Harrison Ford	Indiana Jones
		Sean Connery	Professor Henry Jones

Figure 1.2: Example logical views of relational data. Information is taken from IMDb.

exceed the size of the original data by a factor of twenty! Such excessive overhead (and the precomputation required to create the index) makes it impractical for these approaches to search large data repositories.

Web Search and Relational Keyword Search

Figure 1.3 portrays how IMDb displays the information shown in Figure 1.2. Each group of text represents a separate web page; hyperlinks and their targets are denoted by arrows. One major difference between web search engines and relational keyword search systems is how they handle the hyperlinks or relationships between information. For an Internet search engine, the granularity of search results is fixed as a single web page that contains all the search terms.[5] If all the search terms do not appear on a web page, no results are returned to the user. The hyperlinks between pages serve as a global measure of importance: for example, PageRank [BP98] is commonly used to determine which pages are authoritative. However, the hyperlinks are not used to identify a group of pages that collectively address the user's information need [LCVA02].

In contrast, a relational keyword search system must be capable of finding the relationship(s) among disjoint subsets of search terms. These systems first identify tuples that contain search terms and then use their relationships (and possibly additional tuples not directly referenced by the query) to construct results

[5]Although different (e.g., OR) semantics are possible, most popular search engines require all search terms to be present on the result page.

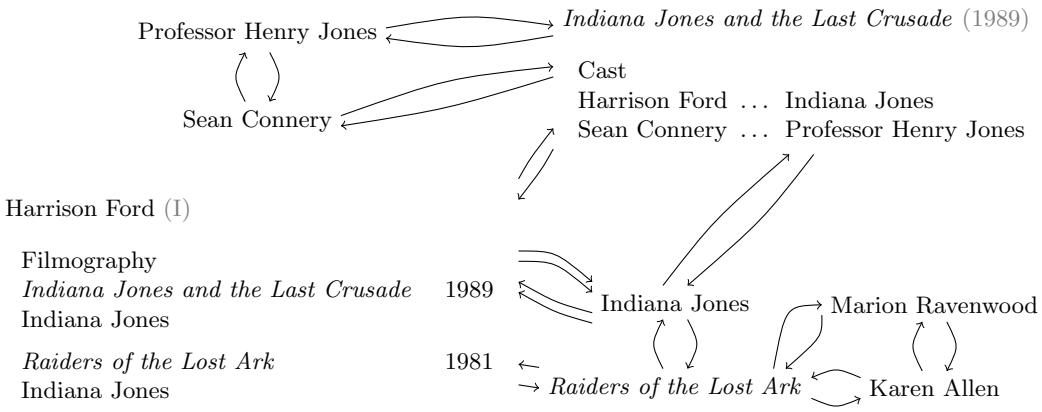
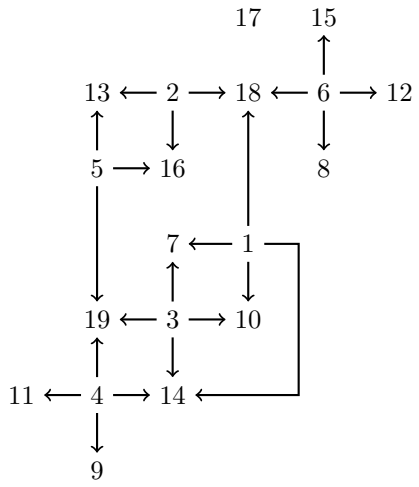


Figure 1.3: Example of IMDb web pages and the hyperlinks among them

dynamically. For these systems, the use of relationships (or links) is not confined to ranking; instead, they are an integral part of the search process, and their presentation may also be essential to the relevance of a result. For example, assume a user wants to know the character that Harrison Ford played in the film *Raiders of the Lost Ark*. Cast information must be included in results or else the user will not be able to determine which character Harrison Ford played. More pointedly, returning individual tuples would fail to identify any results that address the user's underlying information need.

As a concrete example of these differences, consider the keyword query "Harrison Ford Sean Connery," which a user might pose to determine if the two actors have appeared in any films together. Because all the search terms appear on the page about *Indiana Jones and the Last Crusade*, a web search engine will return a meaningful result. Although web search engines perform admirably when an existing web page contains all the user's search terms, they cannot handle more distant relationships such as the relationship between Sean Connery and Karen Allen. Relational keyword search systems are designed to probe the database to identify these complex relationships.

Figure 1.4 presents an abstraction of keyword search in databases for the query "Harrison Ford Sean Connery." On the left is a graph representation of the database where the numbered vertices correspond to database tuples (see also Figure A.2 on page 113). The right side of the figure contains an inverted index for the portion of the database shown. Relational keyword search starts by scanning the inverted index to identify vertices (tuples) that contain the search terms. In this example, vertices 10 and 11 both match disjoint subsets of the search terms. Hence, it is not appropriate to return an individual vertex as a search result because each vertex by itself does not address the relationship between the search terms. Search heuristics are typically used to identify these relationships. In this case, vertices 10 and 11 are related by the path 10–3–**14**–4–11 and by the path 10–3–**19**–4–11. These two relationships can be described loosely as "both are actors" (10–3–**14**–4–11) and "both appeared in the film *Indiana Jones and the Last Crusade*"



Postings	
1981	→ 18
1989	→ 19
actor	→ 14
actress	→ 15
allen	→ 12
and	→ 19
ark	→ 18
cast	→ 1, 2, 3, 4, 5, 6
character	→ 7, 8, 9
composer	→ 17
connery	→ 11
crusade	→ 19
director	→ 16
ford	→ 10
harrison	→ 10
henry	→ 9
id	→ 1, 2, 3, . . . , 19
indiana	→ 7, 19
jones	→ 7, 9, 19
karen	→ 12

Postings	
last	→ 19
lost	→ 18
marion	→ 8
movie	→ 1, 2, 3, 4, 5, 6
name	→ 7, 8, 9, 10, 11, 12, 13
of	→ 18
person	→ 1, 2, 3, 4, 5, 6
professor	→ 9
raiders	→ 18
ravenwood	→ 8
role	→ 1, 2, 3, 4, 5, 6, 14, 15, 16, 17
sean	→ 11
spielberg	→ 13
steven	→ 13
the	→ 18, 19
year	→ 18, 19

Figure 1.4: Example of a relational data graph (left) and its inverted index (right). The information is extracted from IMDb.

(10–3–**19**–4–11). Handling the query about Sean Connery and Karen Allen is similar to this example; the only difference is that more relationships must be considered.

Enumeration in the context of relational keyword search refers to identifying the tuples and relationships that compose meaningful search results. Enumeration is distinct from *ranking*, where the list of results is ordered by each result's estimated relevance to the user's information need. Assuming the user is trying to determine if Harrison Ford and Sean Connery have appeared in any films together, the latter result (both appeared in the film *Indiana Jones and the Last Crusade*) should appear before the former (both are actors). Because the number of possible results for a query is bound only by the size of the database, achieving an acceptable response time demands that systems enumerate search results in an order that is highly correlated with the final ranking of the results.

Research Motivation

The title of this dissertation emphasizes the practicality of relational keyword search techniques. Despite a decade of academic research, no proposed search technique appears ready for real retrieval tasks on real-world databases. One reason for the slow transfer of research technology into deployed tools is the poor evaluations that litter the literature in this area. These evaluations stand in marked contrast to those performed by the IR community. One of the primary goals of this dissertation is to ascertain the current state-of-the-art

through robust empirical evaluation. Experimental results indicate that existing search techniques are not capable of handling real-world retrieval tasks, which underscores the importance of transitioning existing research prototypes toward practical systems.

1.4 Contributions and Outline

Chapter 2 provides additional background about keyword search techniques for relational databases. The three major approaches are briefly described and their respective advantages and disadvantages summarized. Appendix B significantly expands on this material. The chapter also includes an extensive review of the evaluations that appear in the literature for these search techniques. This review and the discrepancies among the evaluations motivate the need to standardize future evaluations to promote continued progress in this field.

Chapter 3 presents the first major contribution of this research: a benchmark (published at CIKM 2010 [CW10a]) specifically designed to evaluate relational keyword search techniques. This benchmark is the first to satisfy calls by the research community to standardize evaluation via published datasets, query workloads, and relevance assessments and obviates the ad hoc evaluation practices previously used by researchers. The benchmark adheres to the best practices established by the IR community for the evaluation of retrieval systems and is publicly available to promote better evaluation of relational keyword search techniques. Other researchers (e.g., Bicer *et al.* [BTN11] and Mass and Sagiv [MS12]) have used this benchmark to improve the evaluation of their work. The second contribution associated with this benchmark is an extensive empirical evaluation of existing keyword search techniques for relational databases. This evaluation includes twice as many search techniques as any previous evaluation reported in the literature. The empirical evaluation addresses both efficiency and effectiveness; for many search techniques, this evaluation is the first that considers both aspects of a deployable system. A major research finding is that many existing results are not reproducible. In particular, experiments using the benchmark indicate that many search techniques scale much worse than suggested by previous evaluations and are comparable with regard to search effectiveness.

The lackluster performance of existing search techniques indicates that significant improvement is possible. Chapter 4 presents two novel schemes that improve the effectiveness of relational keyword search techniques. Structured cover density ranking (presented at KEYS 2010 [CW10b]) provides an alternative to traditional IR scoring functions. It specifically enforces users' expectations regarding the order of search results: results are ranked first by coordination level (i.e., the number of terms in common with the query) and then by a measure of term co-occurrence. Generalizing an existing unstructured ranking scheme to handle structured documents

proves the merit of adapting rather than reinventing prior work by the IR community. More importantly, structured cover density ranking outperforms previous attempts to generalize unstructured scoring functions. In a different vein, a learning-to-rank approach (described at CIKM 2011 [CW11b]) provides an effective way to investigate the importance of factors used to rank search results. Applying ordinal regression to learn the relative importance of proposed ranking factors is far more robust than tuning scoring functions by hand. A major research finding is that several factors with strong anecdotal support in the literature are not actually correlated with relevance. The learning-to-rank scheme significantly outperforms previous work at high recall levels.

The third contribution of this work is an investigation of parameters that might impact the evaluation of keyword search techniques for databases. Investigating these parameters and implicit assumptions illuminates how to continue to improve the evaluation of relational keyword search techniques. Chapter 5 provides an analysis of a search engine log to understand how real users would use relational keyword search systems. This work (presented at WSCD 2012 [CW12]) is the first to propose a model of user interaction with a relational keyword search system, and a classification taxonomy teases apart the intent behind a query from the users' expression of that intent. Comparing real user queries with those being used to evaluate relational keyword search techniques reveals that existing query workloads tend to be skewed away from the most common types of user queries. This research also demonstrates how to generate representative query workloads synthetically to enable more realistic evaluation of relational keyword search techniques.

Chapter 6 gives an overview of related work. Most relational keyword search techniques are omitted from this overview and presented in Appendix B instead. Extensions to the research presented in this dissertation are outlined in Chapter 7. Chapter 8 concludes this dissertation.[6]

[6]Appendix A describes the differences between databases and information retrieval in greater detail than presented in this introduction. Appendix B expands on the background information in Chapter 2 and also describes the various search techniques evaluated in this dissertation.

Chapter 2

Background

Goldman *et al.* [GSVGM98] and Dar *et al.* [DEGP98] both recognized the usefulness of a keyword search system for databases, but significant research in this area did not commence for several years after these early publications. Interestingly, these proposed systems coincide with early formal attempts to integrate DB&IR technologies [FR97, NBY97, Coh98]. This chapter presents an overview of approaches that support keyword search in databases. This chapter also reviews the evaluations of proposed search techniques and motivates the need to standardize future evaluations.

2.1 Keyword Search in Relational Databases

A wide variety of techniques to support keyword search in relational databases have been proposed in the literature. Unfortunately, there is neither a standard problem definition nor a standard definition for valid query results. Each proposed technique tends to adopt definitions that enable different query processing (i.e., enumeration) strategies in an attempt to improve performance. Following the lead of Golenberg *et al.* [GKS08], the problem statement outlined below assumes that techniques should identify all possible answers to a given query. To improve performance, approaches may choose enumeration strategies that omit results. This decision is justifiable because retrieval systems are not evaluated by their completeness (that is, whether or not they identify all possible results) but rather by user satisfaction (that is, how well do the query results address the user's information need). In most cases, the distinctions among different approaches are relatively insignificant.

Problem Statement A relational database is conceptually a graph $G = (V, E)$. Each vertex $v \in V$ corresponds to a tuple in the database. An edge $(u, v) \in E$ denotes each relationship (i.e., foreign key) in the

relational database. Each vertex is decorated with the set of terms (keywords) it contains. A keyword query Q comprises a set of keywords. A result for Q is a tree T that is reduced with respect to $Q' \subseteq Q$;[1] that is, T contains all the keywords of Q' but no proper subtree that also contains all of them.[2] The result tree T is conceptually a single document but may be composed of multiple tuples from the database. Hence, its description focuses on the structure of the document (i.e., hierarchical) but its relevance is determined based on the information that it contains. Because the number of results increases exponentially with the size of the tree, a parameter $T_{\max}$ generally limits the maximum number of edges in or depth of the tree. Results are ranked in decreasing order of their (estimated) relevance to the information need expressed by Q. It is important to remember that a retrieval system's objective is to satisfy the end user's information need, not to enumerate all possible results.

The remainder of this section presents three different approaches to supporting keyword search in relational databases. Schema-based approaches are specific to relational databases. These search techniques reuse the underlying database to the greatest possible extent including query processing. Graph-based approaches are much more general and apply to any data that can be modeled as a graph. These search techniques are applicable to other sources of structured data such as XML. The third approach moves much of the work required for efficient query processing to an offline phase. Possible search results are indexed prior to any searches; this step allows traditional IR techniques to be used for query processing and ranking. Because of the large number of proposed search techniques within each general approach, specific search techniques are only mentioned briefly in this section. Appendix B provides more detail regarding each search technique.

2.1.1 Schema-based Approaches

Schema-based search techniques support keyword search over relational databases via direct execution of SQL commands. These techniques model the relational schema as a graph where vertices are relational tables and edges denote foreign keys between tables. Query processing follows three phases. First, database tuples that contain search terms are identified. Second, candidate networks (SQL expressions) that could relate these tuples are systematically enumerated. Third, these SQL expressions are executed against the database to identify results, which are returned to the user. Because there are many possible ways to relate the search terms, efficient query processing precludes executing all possible SQL expressions. Instead, only the most

[1] Alternative definitions for query results include defining a result as individual tuples [BHP04, HHP06] or as a graph [LOF+08, QYCT09]. The definition given above captures the intent of the search techniques compared in this dissertation even if they use a slightly different definition to improve their efficiency.

[2] It should be acknowledged that forcing all results to be minimal (i.e., every leaf node contains at least one search term) may prevent any relevant results from being identified for some queries. Relaxing the constraint on minimality so related information is included (i.e., to ensure relevant results are identified for the aforementioned queries) is a trivial post-processing step. The given definition skirts a fine line between a crisp problem statement that clearly expresses the essence of the problem and one that is technically correct but obscures the objective of relational keyword search. This definition also agrees with problem statements that appear in the existing literature.

promising SQL expressions are actually executed against the database; the remainder are ignored once the top-k results are known.

DISCOVER [HP02] proposed the general system architecture that most IR approaches follow. Hristidis *et al.* [HGP03] refined DISCOVER by adding support for OR semantics and by including an IR scoring function (pivoted normalization weighting [SCH+99]) to rank results. A monotonic score aggregation function enables efficient execution. Liu *et al.* [LYMC06] propose four additional normalizations for pivoted normalization weighting to adapt it to a relational context. SPARK [LLWZ07a, LWL+11] returns to a non-monotonic score aggregation function for ranking results; the non-monotonic function requires new query processing algorithms. Qin *et al.* [QYC09] investigate query processing to eliminate the middleware layer between search systems and the underlying relational database.

2.1.2 Graph-based Approaches

Graph-based approaches assume the database is modeled as a weighted graph where the weights of edges indicate the importance of relationships. Proximity search strategies attempt to minimize the weight of result trees. This task is a formulation of the group Steiner tree problem [DW71], which is known to be NP-complete [RW90]. In addition to ranking results by their total edge weight, many search techniques also include a prestige (i.e., node weight) factor to prefer results that contain more highly-referenced database tuples. Graph-based search techniques are more general than schema-based approaches, for relational databases, XML, and the Internet can all be modeled as graphs.

None of the graph-based search techniques described in the literature operate on the database itself. Instead, the relational database is explicitly converted to a graph. Tuples become vertices in the graph, and edges denote foreign keys. Most search techniques also distinguish edges induced by foreign keys from "backward" edges, which connect the same vertices but reverse the edge's direction. The addition of the backward edges allow directionality to be considered; weighting the backward edges higher than the "forward" edges discourages the inclusion of common relationships that users would find uninformative. For example, the relationship between Sean Connery, Harrison Ford, and the film *Indiana Jones and the Last Crusade* is considered more important by these search techniques than the relationship between these two individuals and the actor role, which is shared by many people across many different films. Edge weighting and directionality allow these search techniques to suppress common relationships.

BANKS [BHN+02] introduced the backward expanding search heuristic for enumerating results. Bidirectional search [KPC+05] improves the performance of the original algorithm. Ding *et al.* [DYW+07] use dynamic programming—the DPBF algorithm—to identify the minimal group Steiner tree in time exponential

in the number of search terms. BLINKS [HWYY07] uses a bi-level index to improve query performance. Golenberg *et al.* [GKS08] guarantee a polynomial delay when enumerating results but enumerate results by height rather than weight. Dalvi *et al.* [DKS08] investigate keyword search on external memory graphs using a multi-granular graph representation to reduce I/O costs as compared to data structures residing in virtual memory. STAR [KRS+09] approximates the optimal result tree in pseudo-polynomial time, which is shown to outperform several other proximity search heuristics.

2.1.3 Precomputing Results

A downside of the schema- and graph-based approaches is that they perform the majority of work at query time. Although the database tuples (vertices in a data graph) are indexed *a priori*, the relationships among them are discovered as part of the search process. Identifying these relationships is expensive; in the case of proximity search, finding even the best result is an intractable problem. Approaches based on precomputing results moves the discovery of relationships to an offline phase. Essentially, all possible results are materialized prior to any searches. These materialized views are indexed via a traditional inverted index, which is scanned at query time to identify results that satisfy a query.

These search techniques are closest to traditional IR. Most variation among these systems stems from different semantics for query results and how to support these semantics efficiently via indexing and query processing. Figure 1.2 on page 10 illustrates several different logical views over relational data; these logical views are similar to the materialized views created by these search techniques.

EKSO [SW05] creates virtual documents comprised of related database tuples and reuses the database's full text search capabilities to support keyword search on these virtual documents. EASE [LOF+08] proposes a graph index to support efficient searches on unstructured, semi-structured, and relational data where query results are subgraphs whose radius does not exceed a maximum size. The concept of a "tuple unit" [LFZ08] generalizes EKSO's notion of virtual documents; later work [FLW11] allowed search results to span multiple tuple units. CSTree [LFZW11] indexes the distance from search terms to the root of potential result trees and stores this index in a relational database to provide efficient top-k query processing.

2.1.4 Summary

Even without describing existing search techniques in detail, there are several high-level advantages and disadvantages to each approach. The schema-based approaches benefit from the underlying database's query optimization and processing. In addition, if the database supports full text indexes, then no separate index is necessary, and indexes are guaranteed to be consistent with the data. The biggest downside of

Table 2.1: Comparison of general approaches for keyword search in relational databases

	Index		Efficiency		Ranking		
Approach	cons.	size	index	exec.	IR	weight	Notes
Schema	⊕	⊕	⊕	⊖	⊙	⊖	inefficient query processing (especially when ranking via non-monotonic scoring function)
Graph	⊙	⊙	⊙	⊙	⊖	⊕	potential inconsistency between graph and database; IR-style ranking difficult to incorporate
Precompute	⊖	⊖	⊖	⊕	⊕	⊕	likely inconsistency between results and database; index size explosion

Legend

⊕	good	cons.	consistency	IR	IR-style ranking
⊙	fair	exec.	execution time	weight	proximity search
⊖	poor				

the schema-based approaches is their brute-force approach to identify join expressions that might relate tuples. Many of these expressions will fail to produce any results satisfying the query so much work could be wasted. In comparison, the graph-based systems have more directed search heuristics—that is, they make more incremental progress toward generating results. However, these approaches operate on a data graph that is external to the database, which opens the door for inconsistency. For example, a tuple might be deleted from the database, but the corresponding vertex in the data graph may not be removed until much later. Although the data graph can be updated incrementally, no work has investigated this process in detail; in particular, it is unknown how to handle updates efficiently. In addition, all the existing search techniques merely approximate solutions to the group Steiner problem. For the approaches that precompute results, the major disadvantage is an index that is entirely redundant with the database. The size of this index can also exceed the size of the database by an order of magnitude [SW05, BRL+10]. However, answering queries is very fast because the computationally expensive work is completed prior to any searches. Nevertheless, these search techniques only index results up to a predetermined size; they are unable to answer queries where the distance between terms exceeds this limit. Table 2.1 summarizes the advantages and disadvantages among the general approaches.

2.2 Survey of Existing Evaluations

The history of the IR community illustrates the importance of standardized evaluation. Singhal states, "A system for experimentation coupled with good evaluation methodology allowed rapid progress in the field and paved [the] way for many critical developments" [Sin01]. The Cranfield experiment [Cle97] pioneered the system evaluation paradigm that has been successfully used for decades. The Text REtreival Conference

(TREC) testifies to the impact of this paradigm, for search effectiveness *doubled* within six years of its inception [TRE].

The past decade of academic research has led to a number of approaches that extend the keyword search paradigm to relational data, but none of these proposed search techniques are actively used outside the academic community. This transition may not have occurred due to a host of issues that plague research in this field. First, the evaluation of search techniques is typically ad hoc, and results do not seem to generalize to other datasets and query workloads. Baid *et al.* [BRL+10] assert that existing approaches have unpredictable performance, which undermines their usefulness for real-world retrieval tasks. This claim has little support in the existing literature, but the failure for these systems to gain any foothold implies that robust, independent evaluation is necessary. In part, existing performance problems may be obscured by experimental design decisions such as the choice of datasets or the construction of query workloads. Second, many proposed algorithms improve performance but also adopt a new function to score results. Existing evaluations tend to focus on the former while ignoring the latter. Third, extant literature has not dealt with the numerous issues that surround different ranking schemes. For example, although one scoring function might be shown empirically to provide higher quality search results, studies have not investigated which factors in the scoring function are responsible for the improvement. The remainder of this section examines the first two issues in more detail to motivate the creation of a standardized benchmark to evaluate keyword search in relational databases. Issues related to existing scoring functions are presented in Section 4.2.1.

2.2.1 Datasets

Table 2.2 summarizes the datasets appearing in previous evaluations. Although this table suggests some uniformity in evaluation datasets, their schemas and content often differ. Two of the most common datasets (DBLP and IMDb) lack a canonical relational schema, which leads to several different schemas appearing in the literature. The information contained within each dataset also varies (Table 2.3 gives the sizes of the datasets). For example, BANKS-II, BLINKS, and STAR all use a DBLP and IMDb dataset, but only BANKS-II's evaluation includes the entire database. Both BLINKS and STAR use smaller subsets to facilitate comparison with search techniques that assume the data graph fits entirely within main memory. The literature does not address the representativeness of database subsets, which is a serious threat because the choice of a subset can have a profound effect on the experimental results. For example, a subset containing 1% of the original data may be two orders of magnitude easier to search than the original database due to fewer tuples containing search terms.

Table 2.2: Datasets appearing in previous evaluations. Evaluations are ordered by date of publication, from oldest (top) to most recent.

System	DBLP	IMDb	Mondial	MovieLens	TPC-H	Other[a]
DBXplorer					●	○
BANKS	●					○
DISCOVER					●	
DISCOVER-II	○					
BANKS-II	●	○				○
EKSO					●	○
Liu *et al.*						○
DPBF	●			●		
BLINKS	○	○				
SPARK	○	●	●			
EASE	●			●		○
Golenberg *et al.*			●			
BANKS-III	●	○				
STAR	○	○				○
Qin *et al.*	●	●				
SAINT	●			●		
CSTree	●			●		

Legend
- ● identical relational schemas
- ○ different schemas or schemas not provided

[a]This column denotes the presence of additional datasets in the evaluation. None of these datasets are related.

2.2.2 Queries and Relevance Judgments

The query workload is another critical factor in the evaluation of these systems. The trend is for researchers either to create their own queries or to create queries from terms selected randomly from the corpus. The latter strategy is particularly poor for the evaluation of retrieval systems because queries created from randomly-selected terms are unlikely to resemble real user queries [MRS08]. An implicit assumption of keyword search is that all the search terms are meaningfully related, but this assumption is unlikely to hold unless queries are created by hand or sampled from query logs. In addition, many queries selected by researchers subtly reflect their proposed ranking scheme. In Liu *et al.*'s evaluation [LYMC06], every query contained a "schema" term (a search term that matches the name of a database relation or attribute). Matching search terms to the relational schema was not considered in previous work so naturally the proposed ranking scheme outperforms competing approaches.

Coupled with this issue is the number of queries in these workloads. For retrieval tasks, fifty queries is the traditional minimum [Voo02, MRS08] and significantly more may be required to achieve statistical significance [WMZ08]. Unfortunately, no evaluation that uses a realistic query workload satisfies this minimum

Table 2.3: Summary of datasets and query workloads appearing in previous evaluations. Empty cells indicate that the information is not relevant or was not provided and could not be determined from the details reported in the original evaluation. Evaluations are ordered by date of publication, from oldest (top) to most recent.

	Experiment	Dataset		Queries			
System	Dataset	$\lvert V\rvert$	$\lvert E\rvert$	Selection	$\lvert Q\rvert$	$[\![q]\!]$	$\overline{[\![q]\!]}$
DBXplorer	TPC-H			Random	200	1–5	
BANKS	DBLP	100K	300K	Researchers	7		
DISCOVER	TPC-H			Random		2–5	
	TPC-H			Random	200	2–5	
	TPC-H			Random	100	2	2.0
DISCOVER-II	DBLP			Random	100	2	
	DBLP			Random		2–5	
EKSO	TPC-H			Random	50	2–4	
	auction			Random	50	2–4	
	movie			Users			
BANKS-II	DBLP	2M	9M	Random	200	2–7	
Liu *et al.*	lyrics	196K	192K	Search log	50	2–20	6.7
DPBF	DBLP[a]	1.9M		Random	500	2–6	4.0
	MovieLens[a]	1M	1M	Random	500	2–6	4.0
	MovieLens	1M	1M	Random	100	4	4.0
BLINKS	DBLP	409K	591K		60	2–4	3.0
	IMDb[a]	68K	248K		40	2–8	5.0
SPARK	DBLP	882K	1.2M	Researchers	18	2–4	2.7
	IMDb	9.8M	14.8M	Researchers	22	2–3	2.4
	MONDIAL	10K		Researchers	35	2–3	2.2
EASE	DBLife	10K		Researchers	5	4–5	4.6
	DBLP	12M		Researchers	5	2–4	3.2
	MovieLens	1M		Researchers	5	3–4	3.4
	previous 3			Researchers	5	3–4	3.8
Golenberg *et al.*	MONDIAL[a]			Random	36	2–10	6.0
BANKS-III	DBLP	1.77M	8.5M	Researchers	8	2–6	3.5
	IMDb	1.74M	1.94M	Researchers	4	2–3	2.5
STAR	DBLP[a]	15K	150K	Random	180	3,5,7	5.0
	IMDb[a]	30K	80K	Random	180	3,5,7	5.0
	YAGO[a]	1.7M	14M	Random	120	3,6	4.5
Qin *et al.*	DBLP	4.6M	5.6M	Researchers	17	3–5	3.1
	IMDb	15.1M	23.6M	Researchers	20	3–5	3.3
SAINT	DBLP			Researchers	100		
	DBLP			Researchers	20		
	MovieLens	1M	2M	Researchers	100		
CSTree	DBLP			Researchers	100		
	DBLP			Researchers	10		
	MovieLens	1M	2M	Researchers	100		
	MovieLens	1M	2M	Researchers	10		

Legend

- $\lvert Q\rvert$ total number of queries
- $[\![q]\!]$ range in number of query terms
- $\overline{[\![q]\!]}$ mean number of terms per query
- $\lvert V\rvert$ number of vertices in the relational data graph (i.e. tuples)
- $\lvert E\rvert$ number of edges in the relational data graph

[a] The queries are equally partitioned among the number of query terms.

number of information needs. The larger workloads used by many researchers are probably valuable for the performance evaluations that abound in this field, but there is no evidence that these workloads mimic the retrieval tasks that these search techniques purport to address.

Among the evaluations that consider the quality of search results, the definition of relevance is often vague. The evaluation of EASE states, "Answer relevance is judged from discussions of researchers in our database group" [LOF+08]. Such a vague definition does not make the assessment process reproducible by a third-party nor are the relevant results published so other researchers can reuse these relevance assessments. SPARK's evaluation [LLWZ07a, LWL+11] used the following definition: relevant results must 1) contain all query keywords and 2) have the smallest size (presumably of any result satisfying the first criterion). In contrast to this definition, the precedent in IR is clear that relevant results must address the underlying information need and not just contain all search terms [MRS08, Web10]. Moreover, there is little precedent in forcing relevant results to have a minimal size. Previous work in multimedia retrieval [dVKL04] indicates that the total amount of irrelevant information should be minimized, but this goal does not make results with irrelevant information completely useless. Instead, the user's effort increases because the user must separate the desired information from the dross.

Search techniques often perform abnormally well with regard to effectiveness metrics, which may be attributable to non-standard relevance definitions coupled with very general queries. Kacholia *et al.* claim, "The recall was found to be close to 100% for all the cases with an equally high precision at near full recall" [KPC+05]. In the evaluation of EASE [LOF+08], the queries admit a large number of relevant answers, e.g., searching for a movie review written by a college student where any student's review is relevant. Incidently, EASE [LOF+08] reports a precision of 0.9 for 100 retrieved results, which is considerably better than the best scores reported at TREC (roughly 0.25) [Web10].

2.2.3 System Comparison

Table 2.4 shows the evaluations that compare previous work. STAR's evaluation [KRS+09] is the only one that includes more than two other search techniques. Evaluations that include search techniques from more than one general approach (schema-based, graph-based, or precomputing) are rare. Likewise, few evaluations compare the effectiveness of IR-style ranking and proximity search. Lack of cross-evaluation makes it difficult to compare the trade-offs between approaches that vary widely with respect to both query processing and ranking results.

Discrepancies among existing evaluations are prevalent. Table 2.5 lists the mean execution times of search techniques from three evaluations that use DBLP and IMDb datasets. The table rows are search

Table 2.4: Search technique evaluation matrix. Evaluations that compare against other search techniques are listed on the left; the search techniques they compare against appear at the top of the table. If all evaluations were comprehensive, the lower left entries would all be •. Evaluations and search techniques are ordered by date of publication, from oldest (top, left) to most recent.

	BANKS	DISCOVER	DISCOVER-II	EKSO	BANKS-II	Liu *et al.*	DPBF	BLINKS	SPARK	EASE	Golenberg *et al.*	BANKS-III	STAR	Qin *et al.*	SAINT	CSTree
BANKS	–															
DISCOVER		–														
DISCOVER-II			–													
EKSO				–												
BANKS-II	•		○		–											
Liu *et al.*		○	○			–										
DPBF	•				•		–									
BLINKS					•			–								
SPARK			•			○			–							
EASE							•			–						
Golenberg *et al.*											–					
BANKS-III	•		○									–				
STAR	•				•		•	•					–			
Qin *et al.*		•							•					–		
SAINT								•	•						–	
CSTree								•		•						–

Legend
- • exact comparison
- ○ characteristics of system approximated

techniques; the columns are different evaluations of these search techniques. Empty cells indicate that the search technique was not included in that evaluation. According to its authors, BANKS-II "significantly outperforms" BANKS, which is supported by BANKS-II's evaluation [KPC+05]. The most recent evaluation (STAR [KRS+09]) contradicts this claim, particularly for DBLP where BANKS-II performs worse than BANKS. Likewise, BLINKS claims to outperform BANKS-II "by at least an order of magnitude in most cases" [HWYY07], but when evaluated by other researchers (STAR's evaluation [KRS+09]), this significant improvement disappears.

Table 2.5: Example of contradictory runtime performance results that appear in the literature. Values are execution times in seconds (s). Bolded values are the **best** for a particular evaluation.

Dataset	Evaluations					
	DBLP			IMDb		
System	BANKS-II	BLINKS	STAR	BANKS-II	BLINKS	STAR
BANKS	14.8		5.9	5.0		10.6
BANKS-II	**0.7**	44.7	7.9	**0.6**	5.9	6.6
BLINKS		**1.2**	19.1		**0.2**	2.8
STAR			**1.2**			**1.6**

Table 2.6: Example of contradictory search effectiveness results that appear in the literature. Precision @ 1 ($\in [0, 1]$) measures the quality of the highest-ranked result; higher is better. Bolded values are the **best** for a particular evaluation.

	Precision @ 1		
System	Liu *et al.*	SPARK	Xu *et al.*
DISCOVER	0.160		
DISCOVER-II	0.420	0.111	0.000
Liu *et al.*	**0.780**	0.111	
SPARK		**1.000**	0.200
Xu *et al.*			**0.950**

Discrepancies in evaluations of search effectiveness are also common (see also Webber [Web10]). Table 2.6 lists precision @ 1 (that is, the percentage of queries for which the first result is relevant) for three different evaluations that altogether considered five different ranking schemes. While the trend is certainly encouraging—each system appears to at least double retrieval effectiveness—no evaluation produces results that are similar to those previously reported. For example, Liu *et al.* claim to double the effectiveness of DISCOVER-II, but SPARK's evaluation shows that the two techniques perform comparably. While these values are not directly comparable across different collections and information needs [Voo02], the values vary extensively, which suggests that these previous evaluations are not robust.

Tables 2.5 and 2.6 motivate two concerns about existing evaluations. First, the difference in the relative values reported by each evaluation is startling. In Table 2.5, one does not expect STAR's evaluation [KRS+09] to eliminate the *orders of magnitude* performance improvements claimed by BANKS-II [KPC+05] and BLINKS [HWYY07], which is certainly the case on the DBLP dataset. It also seems convenient that the proposed search technique always outperforms the previous work. This trend appears in virtually every evaluation reported in the literature and may suggest that many evaluations are not unbiased measurements of the search technique's performance. In Table 2.6, SPARK's evaluation [LLWZ07a, LWL+11] reports DISCOVER-II and Liu *et al.*'s ranking schemes have comparable performance even though Liu *et al.*'s evaluation claimed to double search effectiveness. These discrepancies are not isolated incidents; careful review of the literature identifies many similar discrepancies between different evaluations. Second, the absolute values that appear in these evaluations appear to be optimistic. As evidenced by Table 2.5, each system claims to provide "interactive" response times (on the order of a few seconds),[3] but this goal seems elusive, met by only the most recent system being evaluated. Evaluations of search effectiveness are similar. Webber reports that the best automatic systems at TREC score around 0.8 for mean reciprocal rank (the reciprocal rank of a system for a query is the reciprocal of the highest ranked relevant results), but the best

[3]BANKS claims that most queries "take about a second to a few seconds" to execute against a bibliographic database [BHN+02].

scores for this metric in previous evaluations [LYMC06, LLWZ07a, XIG09, LWL+11] are considerably higher, including a perfect score of 1.0 in one evaluation. While there is certainly the promise that these systems provide exceptional performance and better search effectiveness than existing solutions, these promises have not been realized by anyone other than the researchers making them.

2.2.4 Summary

In summary, no standardized datasets or query workloads exist for evaluating the performance or effectiveness of existing search techniques. Even among evaluations that use the same dataset, results are not comparable because researchers create random subsets of the original database (see Table 2.2). The past decade of research primarily focuses on performance. Query workloads vary widely across evaluations from large collections of randomly generated queries that may not reflect real user queries to small numbers of more representative queries created by researchers where the number of queries does not meet the accepted minimum for evaluating retrieval systems (Table 2.3). Finally, comparison among search techniques is relatively limited (Table 2.4). Instead of recent experiments validating previous claims, different evaluations currently arrive at very different conclusions (Tables 2.5 and 2.6). These contradictions make it difficult to judge the merits of existing search techniques and suggest that researchers are not evaluating their search techniques reliably. Most search techniques included in this survey have been published in prestigious proceedings (e.g., VLDB, SIGMOD, ICDE), which indicates the unfortunate reality that ad hoc evaluations are an accepted practice rather than aberrations. The experience of the IR community emphasizes the importance of ending this trend, and the following chapter presents a benchmark dedicated to standardizing the evaluation of keyword search in databases.

Chapter 3

A Benchmark for Keyword Search in Databases

The Initiative for the Evaluation of XML retrieval (INEX) workshop [FGKL02] established standardized evaluation procedures for XML retrieval. Despite the similarity of keyword search in semi-structured data and relational data, keyword search techniques for relational databases have not been evaluated at this venue. Perhaps researchers see evaluation forums such as INEX as too expensive to validate experimental system designs, but standardized evaluation is essential for real progress. Researchers from the DB&IR community could create their own evaluation forum, but this strategic step has yet to occur. Until the creation of an evaluation forum, progress will not match that of the larger IR community. In the interim, the community should coalesce behind a standardized set of datasets and queries for evaluating these systems [Web10]. According to Chen *et al.*, "Contributions from the research community are highly demanded for developing comprehensive frameworks for evaluating the retrieval and ranking strategies of keyword search on various structured data models" [CWLL09].

The benchmark described in this chapter enables direct comparison of the efficiency and effectiveness of different search techniques. Considering both efficiency and effectiveness is critical because the myriad of experimental design decisions that confront researchers leads to contradictory results appearing in the literature. Empirical evaluation indicates that many existing search techniques do not provide acceptable performance for realistic retrieval tasks. In particular, memory consumption precludes many search techniques from scaling beyond small datasets with tens of thousands of vertices. Many techniques cannot scale to even moderately-sized datasets that contain more than a million tuples. This evaluation also explores the relationship between execution time and factors varied in previous evaluations, but the analysis reveals that

most of these factors have little impact on performance. These results confirm previous claims [BRL+10] regarding the unacceptable performance of existing search techniques. Evaluation of search effectiveness also contradicts previous evaluations. These findings underscore the importance of standardized evaluation, which will allow continued progress in this field.

3.1 Benchmark Description

Evaluation of a retrieval system requires three items [MRS08]:

1. a collection of documents,
2. information needs, which are expressed as queries, and
3. relevance judgments for each query and document pair.

The document collection for keyword search in databases (i.e., a relational database) differs from traditional IR document collections in that the atomic units for indexing and retrieval (i.e., tuples) often must be combined to answer an information need. This section describes the datasets, queries, and relevance assessments included in the benchmark.

3.1.1 Datasets

Two benchmark datasets are derived from popular websites (IMDb and Wikipedia). The third (MONDIAL) is an ideal counterpart due to its smaller size. Table 3.1 provides summary statistics for all three datasets. In the table, the number of edges is the number of foreign keys in the database.[1] Even though the datasets are relatively small, they are sufficiently challenging for existing search techniques as shown by the experiments presented later in this chapter. Both IMDb and Wikipedia datasets are subsets of the original; both can be scaled up as search techniques continue to improve.

Mondial The MONDIAL dataset [May99] comprises geographical and demographic information from the CIA World Factbook, the *International Atlas*, the TERRA database, and other web sources. A relational version is available online;[2] this version is included in the benchmark. MONDIAL's cyclic relational schema is much more complex than either IMDb or Wikipedia.

IMDb IMDbPY provides an interface to convert IMDb's flat text files into a relational database. Using a third-party tool eliminates any bias in the creation of the schema, which has the potential to affect

[1] Most graph-based search techniques create both "forward" and "backward" edges that are weighted differently to denote the importance of the relationship. The table gives only the number of forward edges for the database. The number of backward edges is comparable to the number of forward edges.

[2] http://dbis.informatik.uni-goettingen.de/Mondial/

Table 3.1: Summary statistics for the datasets included in the evaluation benchmark. While the benchmark includes only the IMDb and Wikipedia subsets, the statistics for the complete databases are provided for comparison.

			in *thousands*		
Dataset	Size (MBs)	Relations	$\|V\|$	$\|E\|$	$\|T\|$
MONDIAL	16	28	17	28	12
IMDb	9262	20	44,303	109,987	21,987
IMDb subset	459	6	1,673	3,037	1,748
Wikipedia	664	42	1,575	1,738	760
Wikipedia subset	391	6	206	392	750

Legend
$|V|$ number of vertices (i.e., tuples)
$|E|$ number of edges in the data graph (i.e., foreign keys)
$|T|$ number of terms in database

experimental results. The initial database contained 20 relations with more than 44 million tuples. Because most graph-based search techniques assume the data graph is small enough to reside entirely within memory, the dataset used in the benchmark is a subset of the original database. While this decision does allow more search techniques to be compared, the subset potentially overstates the runtime performance and effectiveness of search techniques for this dataset.

Wikipedia The third dataset is a selection of articles from Wikipedia. The English Wikipedia contains more than 3 million articles, which makes including all of them infeasible.[3] The subset includes more than 5500 articles chosen for the 2008–2009 Wikipedia Schools DVD, a general purpose encyclopedia, which contains content roughly equal to a traditional 20 volume encyclopedia. Including general content seemed more desirable than having a larger number of articles chosen randomly from the entire corpus. The database subset includes all the tables related to articles and users and augments the `PageLinks` table with an additional foreign key to explicitly indicate referenced pages.

3.1.2 Queries

Fifty information needs is the traditional minimum for evaluating retrieval systems [Voo02, MRS08]. This number of information needs reflects the fact that performance varies widely across queries for the same document collection. Table 2.3 on page 22 shows that other evaluations that use representative queries do not include this number of distinct information needs.[4] In contrast, all the queries included in this benchmark reflect distinct information needs.

[3] The article text alone exceeds 100 GB, and their revision histories require terabytes of disk space.
[4] Liu *et al.* [LYMC06] repeat a number of information needs in their queries.

Table 3.2: Summary statistics for the evaluation queries and results. The results are for the evaluation queries.

	AOL log			Benchmark			Results	
Dataset	$\|Q\|$	$[\![q]\!]$	$\overline{[\![q]\!]}$	$\|Q\|$	$[\![q]\!]$	$\overline{[\![q]\!]}$	$[\![r]\!]$	$\overline{[\![r]\!]}$
MONDIAL				50	1–5	2.04	1–35	5.90
IMDb	101,903	1–96	2.71	50	1–26	3.88	1–35	4.32
Wikipedia	122,956	1–95	2.87	50	1–6	2.66	1–13	3.26
Overall	20,527,863	1–245	2.37	150	1–26	2.86	1–35	4.49

Legend

$|Q|$ total number of queries
$[\![q]\!]$ range in number of query terms
$\overline{[\![q]\!]}$ mean number of terms per query
$[\![r]\!]$ range in number of relevant results per query
$\overline{[\![r]\!]}$ mean number of relevant results per query

Four reasons preclude using user queries extracted from a search engine log. First, many queries are inherently ambiguous. Given the query "Indiana Jones," it is impossible to determine the underlying information need. Does the user want information about the character Indiana Jones or the films named after that title character? Without knowing the user's intent, it is impossible to judge whether the character or a film is the desired result. In contrast, a synthetic query workload[5] based on overt information needs avoids this problem. Second, a large number of queries will likely reflect the limitations of existing search engines—namely, web search engines are not designed to connect disparate pieces of information. Users implicitly adapt to this limitation by submitting few queries (Nandi and Jagadish [NJ09] report less than 2%) that reference multiple database entities. Third, the available search logs provide an insufficient number of user queries for many domain-specific datasets (e.g., DBLP and MONDIAL). Although IMDb and Wikipedia receive many click-throughs, DBLP and MONDIAL click-throughs, for example, are much less common or even non-existent. Fourth, many researchers have recognized the need to create subsets of large datasets (e.g., as was done for IMDb and Wikipedia) for empirical evaluations. When working with a subset of the original database, some queries in the search log may not have relevant results in the subset.

In the absence of user queries, a number of individuals ideally create candidate information needs for an evaluation, and a subset from this pool is actually included. This procedure is used by established evaluation forums (e.g., TREC and INEX) but is impractical for this work given the lack of incentive for others to participate. Instead, a variety of different information needs was created for each dataset; these information needs were based on the query patterns thought to be common for the underlying data.[6]

Table 3.2 provides the statistics of the query workloads and relevant results for each dataset. Five IMDb

[5]Synthetic in this description does not imply unrealistic (e.g., queries constructed by selecting terms at random). Synthetic merely distinguishes these queries, which are constructed by researchers but have well-defined information needs, from the queries captured in search engine logs.

[6]Chapter 5 investigates how similar the benchmark's queries are to real user queries from a search engine log.

queries are outliers because they include an exact quote from a movie. Omitting these queries reduces the maximum number of terms in any query to 7 and the average number of terms per query to 2.91. The statistics for the queries in the benchmark are similar to those reported for web queries [JS06] and an analysis of query lengths from a commercial search engine log [PCT06]. In contrast, the average length of queries used in previous studies (see Table 2.3 on page 22) is almost always greater than the average for web queries.

3.1.3 Assessing Relevance

Relevance is assessed relative to the original information need. Because the information needs are overt, it is possible to execute a number of SQL queries to identify all results satisfying the information need and judge each of these results for relevance. Thus, careful construction of the information needs allows exhaustive relevance judgments for the collection. As is done at TREC, relevance assessments are carried out by a single individual. While using a single assessment as the gold standard does affect the absolute values of effectiveness metrics, it has not been shown to impact the relative effectiveness of the systems under comparison [Voo02].

A common technique used by the IR community is pooling because exhaustive relevance judgments are difficult to obtain for large collections. In this approach, each retrieval system returns its top-k results for a particular query, and relevance is assessed for each document in the union of these results. Exhaustive relevance judgments are preferable to this technique, and not using pooling indicated a number of minor flaws among the implementations of the various search techniques. These flaws were corrected prior to the experiments.

In adherence to the Cranfield paradigm [Cle97], binary relevance assessments are used when judging results. TREC traditionally used binary relevance assessments, and binary relevance assessments appear to have been used by all the previous evaluations reported in Chapter 2. In contrast, INEX distinguishes between highly relevant and partially relevant results. This distinction is good in theory, but it adds considerable complexity to the assessment process and also questions some of the central assumptions of the Cranfield paradigm—namely, all relevant documents are equally desirable. In practice, the notion of relevance, especially for structured data, is extremely subtle, involving novelty and diversity in the results. Clarke *et al.* [CKC+08] provide additional details regarding these issues.

3.2 Experimental Setup

The experiments described in this chapter are intended to provide a robust, independent evaluation of keyword search techniques for relational databases. These experiments should validate the results that already appear in the literature, but this task may not be feasible given the wide variation in previous evaluations (Chapter 2).

If previous results cannot be validated, a new baseline for the performance and effectiveness of these search techniques will be established. The second objective of the experiments is to investigate what effect various parameters (e.g., retrieval depth and maximum result size) have on performance. A detailed study of these parameters is warranted because search techniques that perform well under one set of parameter values may perform poorly for another set of values.

3.2.1 Metrics

Two metrics measure run time performance. The first is execution time, which is the time elapsed from issuing the query until an algorithm terminates. Because there are a large number of potential results for each query, search techniques typically return only the top-k results where k specifies the desired retrieval depth. The second metric is response time, which is the time elapsed from issuing the query until i results have been returned (where $i \leq k$). Because this definition is not well-defined when fewer than k results are retrieved, it is defined for j, where $i < j \leq k$ and i is the number of results retrieved and k is the desired retrieval depth, as the algorithm's execution time. The time required for initialization (e.g., loading a graph representation of the database into memory) is not included when measuring execution time or response time.

System performance should not be measured without also accounting for search effectiveness due to an implicit tradeoff between runtime and the quality of search results. This tradeoff should always be presented because systems typically either improve performance or search quality but rarely both. For example, a very efficient system could return only individual tuples that contain search terms, but these results will not be as relevant as the results discovered via the computationally expensive task of identifying the relationships among these tuples.

Precision is the ratio of relevant results retrieved to the total number of retrieved results. This metric is critical to the evaluation of these systems because different systems have different definitions for query results and not every result that contains all the search terms is actually relevant to the query's underlying information need. Precision @ k (P@k) is the mean precision across multiple queries where the retrieval depth is limited to k results. If fewer than k results are retrieved by a system, the precision value is calculated at the last result. Reciprocal rank is the reciprocal of the position of the highest ranked relevant result for a query. Thus, if the first result is relevant, the reciprocal rank is 1; if the first result is not relevant but the second is, the reciprocal rank is $\frac{1}{2}$; etc. Both P@k and mean reciprocal rank (MRR) tend to be very noisy but indicate the quality of the top-ranked results. Average precision for a query is the average of the precision values calculated after each relevant result is retrieved (and assigning a value of 0 to any relevant results not retrieved). Mean average precision (MAP) averages this single value across information needs to

derive a single measure of quality across different recall levels and information needs. 11-point interpolated average precision is the maximum precision achieved at any recall level greater than or equal to the specified recall level. This metric expresses the intuition that users will always look at one more result if it is relevant; it also summarizes the entire precision-recall curve.

Measuring the completeness of search results returned by the enumeration algorithm is tempting, but only Golenberg *et al.*'s algorithm [GKS08] is proven to be complete (i.e., return all possible results) for a given set of search terms. Furthermore, it is not clear what effect omitting some results may have on a search technique. Unlike recall, which is measured against the set of *relevant* results, omitting a few results may have practically no impact on the effectiveness of the search technique, particularly if the omitted results are highly redundant with others that are enumerated. This issue is not unlike the argument against using set-oriented metrics to evaluate retrieval systems. When there are many possible results, the cognitive limitations of the user result in the user only examining the first few results returned by the system.

3.2.2 Implementations

A number of keyword search techniques described in the literature were reimplemented, and other researchers were also contacted to obtain implementations. Of the search techniques included in these experiments, BANKS, DISCOVER, DISCOVER-II, Liu *et al.*, DPBF, and SPARK were reimplemented from a common codebase, and BANKS-II, BLINKS and STAR were provided by other researchers. All the search techniques were implemented in Java. A host of implementation defects in the implementations provided by other researchers were corrected before running the experiments. To the greatest extent possible, others' implementations also reuse the common codebase to ensure consistent timing when measuring runtime performance.

The implementation of BANKS adheres to its original description except that it queries the database dynamically to identify vertices in the data graph (i.e., tuples) that contain query keywords. DISCOVER's implementation borrows its successor's query processing techniques. Both DISCOVER and DISCOVER-II are executed with the sparse algorithm [HGP03], which generally provides the best performance for queries with AND semantics. Runtime performance was not measured for Liu *et al.*'s ranking scheme or SPARK. The former includes factors that do not lend themselves to efficient performance (i.e., they preclude top-k query processing) while the latter's query processing algorithms are sufficiently complex to dissuade an optimal implementation, which would make runtime comparison unfair for this system. BLINKS's block index [HWYY07] was created using breadth-first partitioning and contains 50 nodes per block.[7] STAR uses

[7]Memory overhead increases when the index stores more nodes per block. As evidenced by the experiments, the significant memory overhead imposed by the bi-level index already prevents BLINKS from completing many queries.

the edge weighting scheme proposed by Ding *et al.* [DYW+07] for undirected graphs.

3.2.3 Experimental Setup

For the experiments, each search technique is executed on a Linux machine running Ubuntu 10.04 with dual 1.6 GHz AMD Opteron 242 processors and 3 GB of RAM. These machines are part of a cluster, but each query runs on a single machine. Each implementation was compiled using `javac` version 1.6 and the implementations run in the Java HotSpot 64-bit server VM. PostgreSQL is the underlying relational database management system (RDBMS).

To avoid excessively long execution times, a timeout of 1 hour was set for each search technique. If the algorithm has not terminated after this time limit, it is denoted as a timeout exception. This threshold seems more than adequate for capturing executions that would complete within a reasonable amount of time. Implementations are allowed to use ≈ 5 GB of virtual memory[8] and the size of results is limited to 5 nodes (tuples). Once a search technique consumes the available physical memory, the operating system's virtual memory manager is responsible for paging data to and from disk. If an algorithm exhausts the total amount of heap memory, it is marked as failing due to excessive memory requirements. Unless otherwise stated, values reported in the experiments are the mean of three different executions of each search technique.

3.3 Experiments

Table 3.3 lists the number of queries executed successfully by each search technique for the datasets and also the number and types of exceptions encountered. Of interest is the number of queries that were not completed successfully. Queries fail due to timeouts (i.e., the algorithm had not terminated after 1 hour of execution time) or exhausting virtual memory. In the table, these exceptions are indicated by "TO" and "VM." Unfortunately, the cause of a search technique's failure is not always apparent, particularly when the system is thrashing due to the use of virtual memory. Severe thrashing prevents graceful cleanup when the time limit expires: it can take a considerable amount of time to page in the error handling code—longer than the 15 additional minutes given the search techniques before the cluster scheduler would kill the job. Likewise, correctly identifying the exhaustion of heap space is challenging because it can be difficult to handle Java's OutOfMemoryError.[9] When the garbage collector cannot free any memory, it may not be possible

[8] This was the maximum amount of memory consistently allocated on the machines without triggering Linux's out-of-memory killer. An incremental garbage collector (enabled by the `-Xincgc` JVM flag) was essential for reasonable performance.

[9] The Java documentation states that "a reasonable application should not try to catch [a virtual machine error]" [Jav11].

Table 3.3: Summary of queries completed and exceptions

(a) Mondial

		✘			
System	✔	TO	VM	?	exec.
BANKS	29	21	—	—	1910.9
DISCOVER	**50**	—	—	—	8.0
DISCOVER-II	**50**	—	—	—	6.5
BANKS-II	**50**	—	—	—	190.2
DPBF	**50**	—	—	—	11.1
BLINKS	**50**	—	—	—	23.6
STAR	**50**	—	—	—	**0.3**

(b) IMDb

		✘			
System	✔	TO	VM	?	exec.
BANKS	7	39	—	4	3239.7
DISCOVER	**50**	—	—	—	227.9
DISCOVER-II	**50**	—	—	—	**201.8**
BANKS-II	—	18	—	32	3604.3
DPBF	5	45	—	—	3399.3
BLINKS	—	—	50	—	—
STAR	—	—	50	—	—

(c) Wikipedia

		✘			
System	✔	TO	VM	?	exec.
BANKS	11	39	—	—	2966.4
DISCOVER	**50**	—	—	—	43.1
DISCOVER-II	**50**	—	—	—	**39.8**
BANKS-II	13	35	—	2	2912.7
DPBF	47	3	—	—	732.1
BLINKS	—	—	50	—	—
STAR	3	—	47	—	22.4

Legend

✔	total queries completed successfully (out of 50), more are better
✘	query failures (out of 50), fewer are better
TO	timeout exceptions (> 1 hour of execution time)
VM	memory exceptions (exhausted virtual memory)
?	either a timeout or memory exception (see accompanying text)
exec.	mean execution time (in seconds) across all queries, lower is better

to execute even the minimal error handling code. Hence, the root cause for some failures (i.e., timeout or memory exception) remains unknown and is indicated in the table by "?".[10]

Most search techniques complete all the MONDIAL queries with mean execution times ranging from less than a second to several hundred seconds. Results for IMDb and Wikipedia are more troubling. Only DISCOVER and DISCOVER-II complete all the IMDb queries, and their mean execution time is several minutes. DPBF comes close to completing the Wikipedia queries but still has several timeout exceptions, and both DISCOVER and DISCOVER-II require in excess of half a minute on average to complete these queries.

To summarize these initial results, existing search techniques provide reasonable performance only on the smallest dataset (MONDIAL). Performance degrades significantly when considering a dataset with hundreds of thousands of tuples (Wikipedia) and becomes unacceptable for a dataset with more than a million tuples (IMDb). The remainder of this section examines runtime performance, memory consumption, and the effectiveness of the various search techniques.

3.3.1 Execution Time

Figure 3.1 shows box plots of the execution times for all queries on each dataset. The box plots confirm the performance trends in Table 3.3 but also illustrate the variation in execution time among different queries. In particular, the range in execution time for a search technique is often several orders of magnitude. Most search techniques also have significant outliers in their execution times; these outliers indicate that the performance of these search heuristics varies considerably due to characteristics of the dataset or queries.

Number of search terms

A number of previous evaluations [HP02, HGP03, KPC+05, DYW+07] report mean execution time for queries that contain different numbers of search terms to show that performance remains acceptable even when queries contain more keywords. Figure 3.2 graphs these values for the different search techniques. Some search techniques fail to complete some queries, which accounts for the omissions in the graph. As evidenced by the graph, queries that contain more search terms require more time to execute on average than queries than contain fewer search terms. The relative performance among the different systems is unchanged from Figure 3.1 although DPBF outperforms the schema-based approaches on queries with only a single term. DPBF's performance falters with additional search terms, which is consistent with its algorithmic analysis—exponential in the number of query terms.

[10] Reducing the amount of virtual memory (e.g., to match the machine's physical memory) would prevent this uncertainty, but some search techniques cannot even search the MONDIAL database with less memory [CW11a].

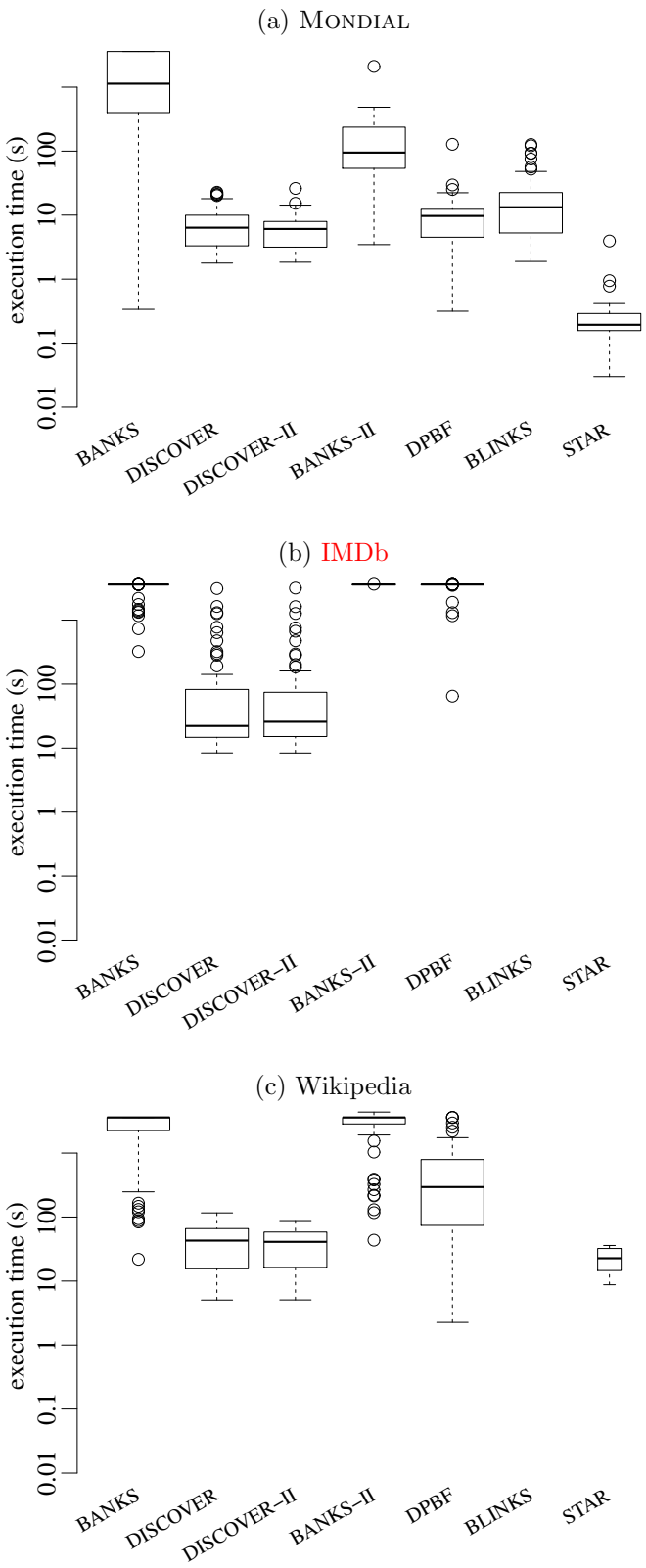


Figure 3.1: Box plots of the execution times for each search technique (lower is better). Note that the y-axis has a log scale. Search techniques are ordered by publication date and the retrieval depth was 100 results.

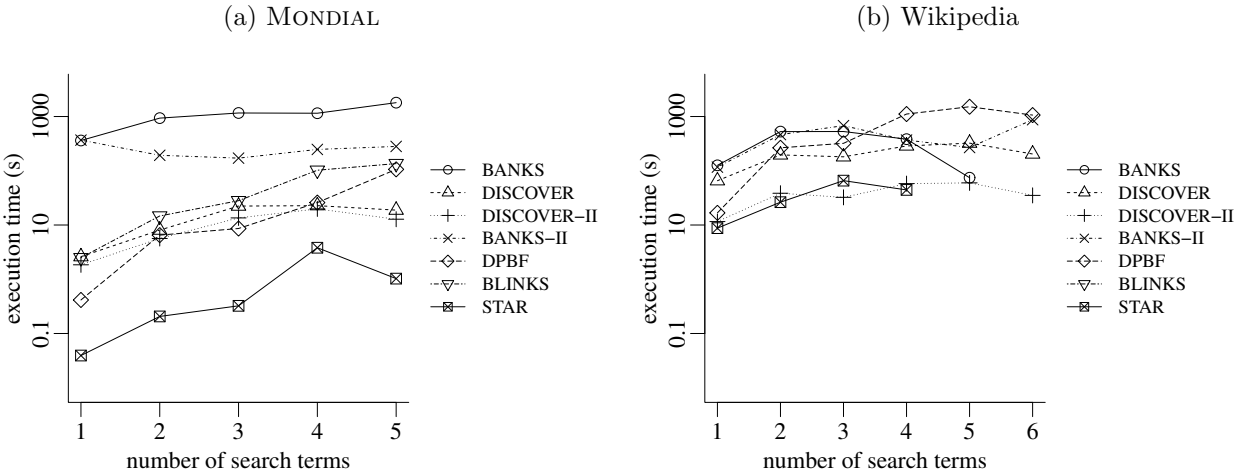


Figure 3.2: Mean execution time vs. query length for Mondial and Wikipedia queries; lower execution times are better, and the retrieval depth was 100 results. Note that the y-axis has a log scale. IMDb is omitted due to the few search techniques that successfully complete its queries.

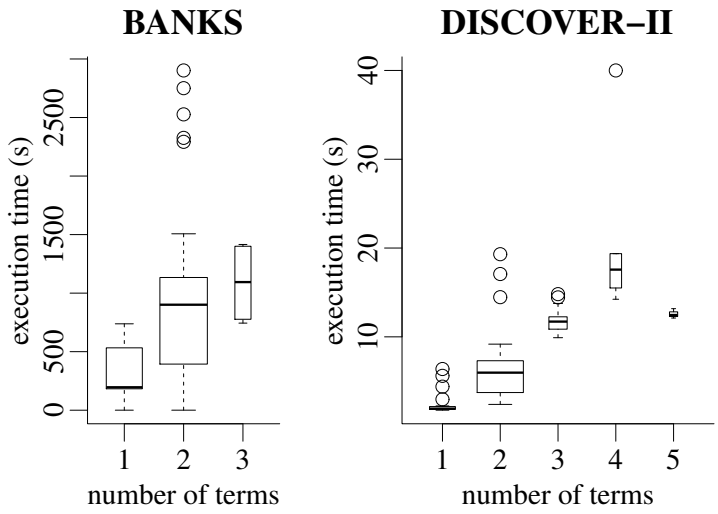


Figure 3.3: Box plots of execution times for BANKS and DISCOVER-II on Mondial with a retrieval depth of 100. Lower execution times are better. The width of the box reflects the number of queries in each sample.

These results are similar to those published in previous evaluations, but using Figure 3.2 as evidence for the efficiency of a particular search technique can be misleading. Figure 3.3 provides box plots of the execution times of BANKS and DISCOVER-II for Mondial queries to illustrate their range in execution times. As evidenced by these graphs, several queries have execution times much higher than the rest. These queries give the system the appearance of unpredictable performance, particularly when the query is similar to another one that completes quickly.

For example, the query "Uzbek Asia" for BANKS has an execution time three times greater than the query "Hutu Africa." DISCOVER-II has similar outliers; the query "Panama Oman" requires 3.5 seconds

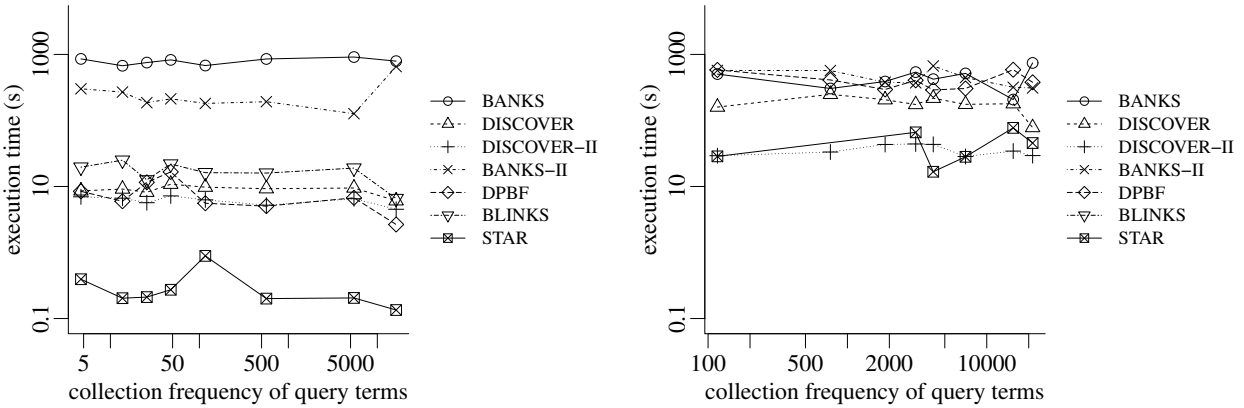


Figure 3.4: Execution time vs. frequency of query terms; the retrieval depth was 100 results. Lower execution times are better. Note that the x-axis and y-axis have a log scales.

to complete even though the query "Libya Australia" completes in less than half that time. From a user's perspective, these queries would be expected to have similar execution times. These outliers (which are even more pronounced for the other datasets) suggest that simply looking at mean execution time for different numbers of query keywords does not reveal the complete performance profile of these systems. Moreover, existing work does not adequately explain the existence of these outliers and how to improve the performance of these queries.

Collection frequency

Figure 3.4 compares mean execution time and the frequency of search terms in the database. IMDb results are comparable to MONDIAL and Wikipedia albeit with significantly higher mean execution times and fewer search techniques completing the queries (hence its omission from the figure). The results are surprising: execution time does not appear to be correlated with the number of tuples containing search terms. This result is unexpected because one expects the time to increase when more tuples (and all their relationships) must be considered. One possible explanation for this phenomenon is that the search space in the interior of the data graph (i.e., the number of vertices that must be explored when searching) is not correlated with the frequency of the keywords in the database.

Retrieval depth

Table 3.4 considers the scalability of the various search techniques at different retrieval depths—10 and 100 results. Overall, there is considerable variation in mean execution time at different retrieval depths. Retrieval depth has practically no impact for DISCOVER and DISCOVER-II whereas execution time is much more sensitive to retrieval depth for the graph-based approaches. While it would be interesting to extend this

Table 3.4: Execution times for different retrieval depths. Lower is better. Bolded values are the **best** for each metric.

(a) Mondial

	execution time (s)		slowdown	
System	$k = 10$	$k = 100$	Δ	%
BANKS	1617.5	1910.9	293.4	18.1
DISCOVER	6.8	8.0	1.2	**17.6**
DISCOVER-II	7.1	6.5	**-0.6**	**-8.5**
BANKS-II	79.1	190.2	111.1	140.5
DPBF	5.4	11.1	5.7	105.6
BLINKS	15.2	23.6	8.4	55.3
STAR	**0.2**	**0.3**	**0.1**	50.0

(b) IMDb

	execution time (s)		slowdown	
System	$k = 10$	$k = 100$	Δ	%
BANKS	2329.2	3239.7	910.5	39.1
DISCOVER	**178.1**	227.9	49.7	27.9
DISCOVER-II	**178.1**	**201.8**	**23.6**	**13.2**
BANKS-II	3604.1	3604.3	0.2	0.0
DPBF	2012.3	3399.3	1387.0	68.9
BLINKS	—	—	—	—
STAR	—	—	—	—

(c) Wikipedia

	execution time (s)		slowdown	
System	$k = 10$	$k = 100$	Δ	%
BANKS	1723.3	2966.4	1243.1	72.1
DISCOVER	40.9	43.1	2.2	5.4
DISCOVER-II	**39.7**	**39.8**	**0.1**	**0.3**
BANKS-II	877.4	2912.7	2035.3	232.0
DPBF	280.8	732.1	451.3	160.7
BLINKS	—	—	—	—
STAR	86.8	22.4	-64.4	-74.2

Legend

k retrieval depth

Δ absolute change in mean execution time, lower is better

% percentage change in mean execution time, lower is better

Table 3.5: Summary of queries completed successfully (higher is better) and execution time (lower is better) for different result sizes. The retrieval depth was 100 results. The results for $T_{\max} = 5$ are copied from Table 3.3. Bolded values are the **best** for the given result size.

(a) Mondial

$T_{\max}$	3		5		7		9	
System	✔	exec.	✔	exec.	✔	exec.	✔	exec.
BANKS	28	1968.1	29	1910.9	32	1708.1	39	1411.4
DISCOVER	**50**	**2.2**	**50**	8.0	40	820.6	15	2489.1
DISCOVER-II	**50**	2.3	**50**	**6.5**	40	837.3	15	2535.6
BANKS-II	**50**	53.8	**50**	190.2	**50**	229.8	**50**	391.7
DPBF	**50**	2.5	**50**	11.1	**50**	**22.2**	**50**	**31.0**

(b) IMDb

$T_{\max}$	3		5		7		9	
System	✔	exec.	✔	exec.	✔	exec.	✔	exec.
BANKS	1	3529.8	7	3239.7	10	3108.8	10	3099.6
DISCOVER	**50**	**12.7**	**50**	227.9	**40**	**963.8**	**31**	**1773.1**
DISCOVER-II	**50**	112.4	**50**	**201.8**	**40**	1122.1	24	2274.4
BANKS-II	0	—	0	3604.3	0	—	0	—
DPBF	24	2176.7	5	3399.3	5	3406.6	5	3398.9

(c) Wikipedia

$T_{\max}$	3		5		7		9	
System	✔	exec.	✔	exec.	✔	exec.	✔	exec.
BANKS	18	2977.4	11	2966.4	15	2614.1	26	2350.9
DISCOVER	**50**	46.1	**50**	43.1	49	308.3	**40**	**1124.8**
DISCOVER-II	**50**	**40.2**	**50**	**39.8**	**50**	**306.7**	37	1779.2
BANKS-II	46	515.6	13	2912.7	11	2912.7	8	3035.5
DPBF	**50**	184.7	47	732.1	38	1274.4	38	1358.0

Legend

✔ queries completed successfully (out of 50), more are better

exec. mean execution time (in seconds) across *all* queries, lower is better

analysis to greater retrieval depths (e.g., 1000 results), the failure for most search techniques to complete the IMDb and Wikipedia queries undermines the value of such experiments.[11]

Result size

Table 3.5 provides the number of successful queries (i.e., queries that did not time out or exhaust virtual memory) and mean execution time when varying the maximum size of result trees (i.e., number of tuples from the underlying database).[12] BLINKS and STAR are not reported in this experiment because neither

[11] Given Mondial's small size, it is not sensible to specify larger retrieval depths because most search techniques already identify all the relevant results. A larger retrieval depth would merely measure how quickly the different approaches exhaust the possible search space. The significant slowdown for the graph-based approaches on the Wikipedia queries suggest that scalability would be an issue for this dataset.

[12] The results in this table are from a single experimental run of each search technique.

provides a parameter to limit the search space to trees of a maximum size (or depth). The table reveals several interesting trends.

First, as result size increases, fewer queries are completed successfully by the search techniques. This result is expected because the number of possible results grows exponentially with the size of allowable results. Second, mean execution time generally increases when the size of the search space increases. The notable exception to these two trends is BANKS where the number of exceptions and mean execution time actually decrease when larger results are permitted. The reason for this behavior is that BANKS's enumeration algorithm returns results in approximate order. These results are buffered in a fixed-size heap, and only after the heap is full (or the search space is exhausted) will BANKS start to output results. Hence, expanding the search space allows BANKS to identify more results than it can when the search space is restricted, and finding more results lowers its total execution time.

The final trend to highlight is the scalability of the schema-based approaches as result size increases. While the schema-based approaches continue to outperform the graph-based search techniques on IMDb and Wikipedia, they falter on Mondial. Despite Mondial having the fewest tuples, it has the most complex schema. DISCOVER and DISCOVER-II's candidate network generation algorithm must enumerate all possible join expressions that contain up to the maximum number of tuples; the total number of join expressions grows exponentially with result size. Hence, Mondial's more complex schema does not allow these approaches to scale as well as the proximity search techniques when the maximum result size is increased. This finding is disappointing, particularly because Mondial's 30-relation schema is still far simpler than enterprise databases, which may contain hundreds or thousands of interrelated tables [YPS09, Sri10].

Response Time

Systems that support top-k query processing need not enumerate all possible results before outputting some to the user. Outputting a small number of results (e.g., 10) allows the user to examine the initial results and to refine the query if these results are not satisfactory.

Table 3.6 gives the mean response time to retrieve the first and tenth query result. Interestingly, the response time for most systems is very close to the total execution time, particularly for $k = 10$. The ratio of response time to the total execution time provided in the table shows that most scoring functions are not good at quickly identifying the best search results. For example, DISCOVER-II identifies the highest ranked search result at the same time as it identifies the tenth ranked result because its bound on the possible score of unseen results falls very rapidly after enumerating more than k results. Although more incremental algorithms exist to enumerate search results, these algorithms did not perform as well as the sparse algorithm used in the experiments. The notable exception to this trend is DPBF, which identifies results the most

Table 3.6: Mean response times to retrieve the top-k query results contrasted with total execution time retrieving 100 results. Lower is better. The **best** values for each metric and value of k are bolded.

(a) Mondial

		$k=1$		$k=10$	
System	exec.	resp.	%	resp.	%
BANKS	1910.9	1681.8	88.0	1778.9	93.1
DISCOVER	8.0	8.0	100.0	8.0	100.0
DISCOVER-II	6.5	6.5	100.0	6.5	100.0
BANKS-II	190.2	54.2	28.5	143.4	75.4
DPBF	11.1	1.9	**17.1**	5.2	**46.8**
BLINKS	23.6	8.4	35.6	14.1	59.7
STAR	**0.3**	**0.3**	100.0	**0.3**	100.0

(b) IMDb

		$k=1$		$k=10$	
System	exec.	resp.	%	resp.	%
BANKS	3239.7	2614.7	80.7	2686.0	82.9
DISCOVER	227.9	227.9	100.0	227.9	100.0
DISCOVER-II	**201.8**	**201.8**	100.0	**201.8**	100.0
BANKS-II	3604.3	3604.3	100.0	3604.3	100.0
DPBF	3399.3	1371.8	**40.4**	2042.1	**60.1**
BLINKS	—	—	—	—	—
STAR	—	—	—	—	—

(c) Wikipedia

		$k=1$		$k=10$	
System	exec.	resp.	%	resp.	%
BANKS	2966.4	2073.7	69.9	2229.9	75.2
DISCOVER	43.1	43.1	100.0	43.1	100.0
DISCOVER-II	**39.8**	**39.8**	100.0	**39.8**	100.0
BANKS-II	2912.7	2635.3	90.5	2755.8	94.6
DPBF	732.1	67.5	**9.2**	251.3	**34.3**
BLINKS	—	—	—	—	—
STAR	22.4	22.4	100.0	22.4	100.0

Legend

- resp. — mean response time (in seconds), lower is better
- exec. — mean total execution time (in seconds) to retrieve 100 results, lower is better
- % — percentage of total execution time, lower is better

incrementally of any search technique. In general, the proximity search systems manage to identify results more incrementally than the schema-based approaches.

Another issue of interest is the overhead required to retrieve additional search results. In other words, how much additional time is spent maintaining enough state to retrieve 100 results instead of just 10? Table 3.7 gives the execution time to retrieve 10 results and the response time to retrieve the first 10 results of 100. The percentage slowdown reveals that the overhead is minimal for most of the search techniques. A minimal

Table 3.7: Comparison of total execution time and response time. Lower values are better. The **best** values for each metric are bolded.

(a) Mondial

System	exec.	resp.	slowdown Δ	slowdown %
BANKS	1617.5	1778.9	161.4	10.0
DISCOVER	6.8	8.0	1.2	17.6
DISCOVER-II	7.1	6.5	**-0.6**	**-8.6**
BANKS-II	79.1	143.4	64.3	81.3
DPBF	5.4	5.2	-0.2	-3.7
BLINKS	15.4	14.1	-1.3	-8.4
STAR	**0.2**	**0.3**	**0.1**	50.0

(b) IMDb

System	exec.	resp.	slowdown Δ	slowdown %
BANKS	2329.2	2686.0	356.8	15.3
DISCOVER	**178.2**	227.9	49.7	27.9
DISCOVER-II	**178.2**	**201.8**	**23.6**	13.2
BANKS-II	3604.1	3604.3	0.2	0.0
DPBF	2012.3	2042.1	29.8	**1.5**
BLINKS	—	—	—	—
STAR	—	—	—	—

(c) Wikipedia

System	exec.	resp.	slowdown Δ	slowdown %
BANKS	1723.3	2229.9	507.8	29.5
DISCOVER	40.9	43.1	10.9	5.4
DISCOVER-II	**39.7**	**39.8**	**0.1**	**0.3**
BANKS-II	877.4	2755.8	1878.4	214.1
DPBF	280.8	251.3	-29.5	-10.5
BLINKS	—	—	—	—
STAR	86.8	22.4	-64.4	-74.2

Legend

exec. total execution time (in seconds) when retrieving 10 results
resp. response time (in seconds) to retrieve top-10 of 100 results

Δ mean absolute difference in time, lower is better
% mean percentage difference in time, lower is better

slowdown is ideal because it indicates that the search techniques should scale gracefully to larger retrieval depths. Nevertheless, the execution and response times remain non-interactive (i.e., requiring more than a few seconds) on all but the MONDIAL dataset.

3.3.2 Memory consumption

To better understand the memory utilization of the systems—particularly the space consumed by an in-memory data graph—each system's memory footprint was measured immediately prior to executing a query. The results are shown in Table 3.8. As evidenced by the table, the schema-based approaches consume very little memory, most of which is used for the database schema. In contrast, the graph-based search techniques require considerably more memory to store their data graph. BLINKS's total memory consumption is particularly high due to its bi-level index, which scales quadratically with the number of nodes per block and is nearly two orders of magnitude larger than the original data graph. STAR is unique in that its implementation [KRS+09] searches from terminals—that is, the data graph includes vertices for each search term (i.e., terminals) and edges from each vertex to the terms that vertex contains. This modification transforms the group Steiner problem into the Steiner problem and accounts for STAR's greater memory consumption for its data graph.

Although conceptually identical, this subtle change in the implementation has important consequences. In particular, the number of additional edges from vertices to terminals (i.e., terms) is considerable. Table 3.9 illustrates the number of edges induced by foreign keys among tuples ($|E_V|$) and the number of edges for the terms ($|E_T|$). As evidenced by the table, the number of keyword edges can exceed the number of database edges by an order of magnitude. Hence, this implementation decision is significant although most papers in the literature do not consider its consequences when designing (and implementing) search techniques.

Table 3.8: Initial memory consumption of each search technique. Smaller values are better.

	MONDIAL		IMDb		Wikipedia	
System	G	Σ	G	Σ	G	Σ
BANKS	9.2	9.3	960.3	960.4	118.5	118.6
DISCOVER	**0.2**	**0.3**	**0.0**	**0.2**	**0.0**	**0.2**
DISCOVER-II	**0.2**	**0.3**	**0.0**	**0.2**	**0.0**	**0.2**
BANKS-II	17.7	22.3	2313.6	3082.0	694.1	1085.4
DPBF	8.0	8.2	849.8	850.0	104.8	104.9
BLINKS	18.5	884.0	—	—	—	—
STAR	50.0	58.5	—	—	—	—

Legend, all values are in MBs
G size of in-memory representation of database
Σ initial memory required by search technique

Table 3.9: Summary statistics for data graphs when terms are included.

		$\lvert E\rvert$			
Dataset	$\lvert V\rvert$	$\lvert E_V\rvert$	$\lvert E_T\rvert$	$\lvert T\rvert$	$\%_{E_T}$
MONDIAL	17	28	154	12	84.5
IMDb	1,673	3,037	30,762	1,748	91.0
Wikipedia	206	392	10,393	750	96.4

Legend, all values are in *thousands*

$\lvert V\rvert$	number of vertices (i.e., tuples)
$\lvert E\rvert$	total number of edges in the data graph
$\lvert T\rvert$	number of unique terms
$\lvert E_V\rvert$	number of edges in data graph ($\lvert E\rvert$ in Table 3.1)
$\lvert E_T\rvert$	number of keyword edges

When compared to the total amount of virtual memory available, the size of the MONDIAL data graphs in particular are quite small, roughly two orders of magnitude smaller than the size of the heap. However, the amount of state maintained by the algorithms (not shown by the table) can be considerable. For example, BANKS's worst-case memory consumption is $O(|Q| \cdot |V|^2)$ where $|Q|$ is the number of terms in the user's query and $|V|$ is the number of vertices in the data graph. It is easy to show that in the worst case BANKS will require in excess of 1 GB of state during a search of the MONDIAL database even when ignoring the overhead of requisite data structures (e.g., linked lists). Hence, searching these data graphs using machines with only a modest amount of available memory (1–4 GB) remains impractical at present.

The amount of space required to store a data graph may prevent these systems from searching other, larger datasets. For example, BANKS requires ≈ 1 GB of memory for the data graph of the IMDb subset; this subset is more than 25 times smaller than the entire database. When coupled with the state it maintains during a search, it is easy to see why BANKS has so many timeout exceptions (due to thrashing) for the IMDb queries. Assuming the size of the data graph scales with the size of the database, BANKS would consume more than 24 GB of memory for its data graph alone for the complete IMDb; this memory consumption rules out executing these search techniques on all but powerful servers. Table 3.8 also reveals that implementation quality varies widely with respect to memory consumption. Although storing data structures (e.g., an inverted index) entirely within main memory is attractive when implementing these search techniques, it does not allow them to scale to larger datasets.

One implementation issue that might significantly impact these systems—particularly the graph-based approaches—is the choice of the graph data structure. All the implementations use the JGraphT library,[13] which is designed to scale to millions of vertices and edges. An empirically-determined lower bound for its memory consumption is $32 \cdot |V| + 56 \cdot |E|$ bytes where $|V|$ is the number of graph vertices and $|E|$ is the number

[13] http://www.jgrapht.org/

of graph edges. In practice, its memory consumption can be significantly higher because it relies on Java's dynamically-sized collections for storage. Kacholia *et al.* [KPC+05] state that the original implementation of BANKS-II requires only $16 \cdot |V| + 8 \cdot |E|$ bytes for its data graph, making it considerably more efficient than the general-purpose graph library. While an array-based implementation is more compact and can provide better performance, it does have downsides when updating the index. Performance issues that arise when updating the data graph have not been the focus of previous work and have not been empirically evaluated for these systems.

3.3.3 Effectiveness

Figure 3.5 shows the mean reciprocal rank for each system for queries where exactly one database tuple is relevant (20, 20, and 15 queries for the respective datasets). Nandi and Jagadish [NJ09] report that these single entity queries are the most common type of query posed to existing search engines. The proximity search systems (BANKS, BANKS-II, DPBF, BLINKS, and STAR) are expected to perform poorly on this task because ranking results by edge weight does not allow these search techniques to distinguish trees containing a single node. Surprisingly, these search techniques perform very well on the Mondial dataset (the best four are all proximity search techniques). BANKS outperforms the IR approaches (DISCOVER-II, Liu *et al.*, and SPARK) on the IMDb dataset. These results—particularly the excellent performance on the Mondial database—counter the original intuition regarding the types of retrieval tasks suited to proximity search techniques. The results for the IR-style ranking schemes are disappointing in view of the excellent scores of the proximity search techniques. Analyzing the results returned for each query sheds some light on the underlying reason: the IR-style ranking schemes prefer larger results that contain additional instances of

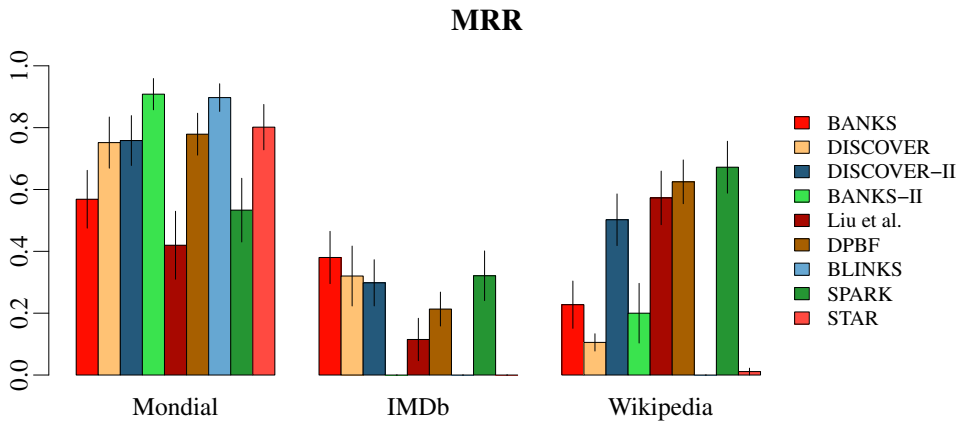


Figure 3.5: MRR ($\in [0, 1]$) measured across the various search techniques and datasets for queries with a single relevant result. The retrieval depth is 100 results. Higher bars (larger values) are better, and the error bars denote the standard error of the mean.

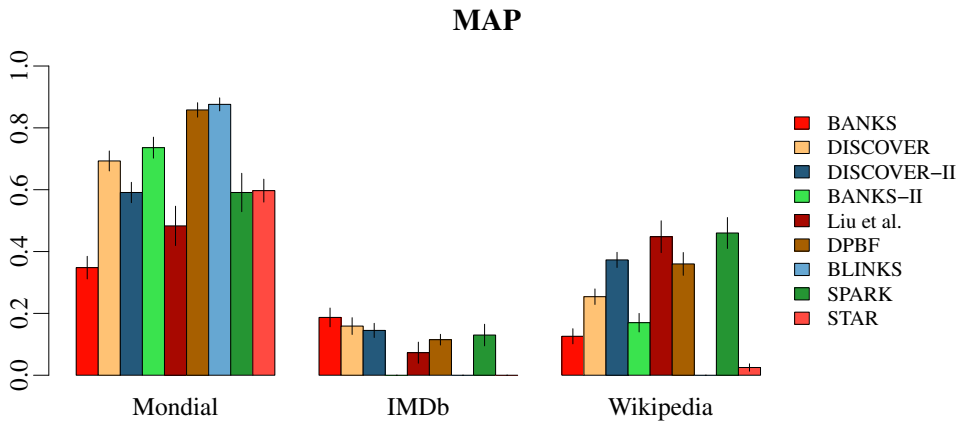


Figure 3.6: MAP ($\in [0, 1]$) measured across the various search techniques and datasets. The retrieval depth is 100 results. Higher bars are better, and the error bars denote the standard error of the mean.

the search terms instead of the smallest result that satisfies the query.

Figure 3.6 shows the MAP scores for the search techniques across all queries and datasets and illustrates several interesting trends. First, the approaches using IR-style ranking schemes (DISCOVER-II, Liu *et al.*, and SPARK) perform similarly due to their common baseline scoring function, pivoted normalization weighting. The different normalizations that each apply to the original scoring function account for their minor differences. Second, the merits of IR-style scoring functions are clearly seen in the Wikipedia results where these approaches are typically twice as effective as the proximity search techniques. Interestingly, DPBF also scores well for the Wikipedia dataset even though its scoring function includes only an edge weight factor. More complex scoring functions that include node weights actually perform worse, with the exception of BANKS for IMDb.[14] Third, search effectiveness drops precipitously as the size of the dataset increases. The drop is most noticeable between MONDIAL and IMDb. The drop also impacts the proximity search techniques more than the IR-style ranking schemes as evidenced by their relative performance compared to the other search techniques. Perhaps this result should not be unexpected because the importance of term weighting and similarity measures (from the IR community) is likely to grow as more tuples contain search terms.

The three IR-style ranking schemes use a modification of pivoted normalization weighting [SCH+99] to rank results; this commonality accounts for their similar performance. The small differences can be attributed to their different normalizations, and the datasets reveal the strengths of each of these normalizations. Due to its harsh size penalty (it prefers small results), DISCOVER-II outperforms Liu *et al.*'s scoring function and SPARK on IMDb because many of the information needs are addressed by a single tuple. Liu *et al.*'s

[14] In the original description of this benchmark [CW10a], DPBF performed particularly poorly for Wikipedia due to an artifact of its authors' implementation. The poor performance led to a hypothesis regarding the importance of node weights in the scoring functions. Reimplementing DPBF significantly improved its search effectiveness, and the inclusion of a node weight factor no longer appears beneficial.

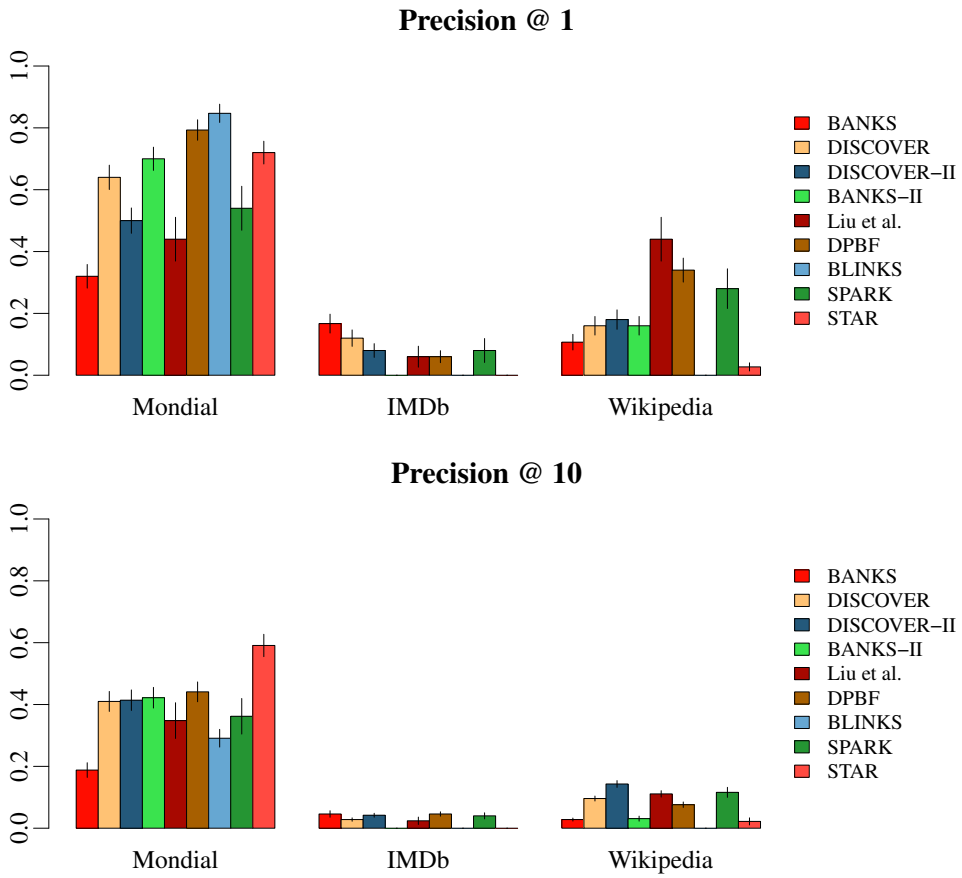


Figure 3.7: Precision @ k ($\in [0, 1]$) measured across the various search techniques and datasets. Higher bars are better.

normalizations do well on the Wikipedia dataset—the mixture of short and long fields (e.g., page titles and article text) plays to its unique normalizations. SPARK does edge the other two ranking schemes for Wikipedia, but its improvement is minimal, which begs the question of whether a theoretically better approach that must estimate parameters for efficient execution is sensible.

The drop in retrieval effectiveness for IMDb across all the search techniques is particularly disturbing because this dataset is the most structured—few fields contain lengthy, unstructured text. In part, the drop in effectiveness undoubtedly mirrors the increased size of the dataset and the original design of pivoted normalization weighting (i.e., very verbose queries for unstructured documents). Larger retrieval depths also significantly reduce search effectiveness. Figure 4.2 graphs precision @ 1 and precision @ 10. P@1 is equivalent to the number of relevant results ranked first by a search technique; this metric has been used in a number of previous evaluations [LYMC06, LLWZ07a, XIG09, LWL+11]. The figure reveals two interesting trends. First, effectiveness drops as the retrieval depth increases although this result might be an artifact of having fewer than ten relevant results for some queries. Second, the values are much lower than those

Table 3.10: 11-point interpolated precision ($\in [0, 1]$) for a subset of the Wikipedia topics. Higher scores are better. BLINKS is omitted due to its failure to complete any Wikipedia queries.

Recall	BANKS	DISCOVER	DISCOVER-II	BANKS-II	Liu *et al.*	DPBF	SPARK	STAR
0.0	0.144	0.388	0.558	0.041	0.975	0.396	0.475	0.033
0.1	0.144	0.388	0.558	0.041	0.925	0.396	0.475	0.033
0.2	0.045	0.229	0.545	0.041	0.565	0.393	0.453	0.033
0.3	0.028	0.196	0.526	0.041	0.360	0.314	0.403	0.000
0.4	0.028	0.196	0.526	0.041	0.247	0.314	0.376	0.000
0.5	0.026	0.158	0.390	0.041	0.156	0.245	0.254	0.000
0.6	0.023	0.092	0.363	0.041	0.118	0.135	0.225	0.000
0.7	0.012	0.031	0.342	0.040	0.025	0.092	0.181	0.000
0.8	0.012	0.031	0.264	0.040	0.013	0.092	0.114	0.000
0.9	0.000	0.000	0.193	0.026	0.000	0.052	0.062	0.000
1.0	0.000	0.000	0.190	0.026	0.000	0.052	0.027	0.000

reported in previous evaluations but more consistent with results from established IR evaluation venues (e.g., TREC). More importantly, the benefits claimed by many search techniques are not present in these results. For example, SPARK doubles the score of P@1 in its original evaluation against DISCOVER-II and Liu *et al.*, but this significant improvement is not seen for this benchmark.

Table 3.10 presents 11-point precision and recall for a subset of Wikipedia topics most similar to those encountered at TREC. The query terms are present in many articles, yet most articles containing the search terms are not relevant. Here, the IR-style scoring functions (particularly Liu *et al.*'s) outperform the proximity search systems. DISCOVER-II, which least modifies pivoted normalization scoring, has the most stable performance across the entire precision-recall curve. In contrast, the effectiveness of the other IR-style scoring functions drops precipitously at higher recall levels.

3.4 Discussion

Unlike many evaluations reported in the literature, these experiments investigate the overall, end-to-end performance of relational keyword search techniques. Hence, the experiments use a smaller, more realistic query workload instead of a larger workload with queries that are unlikely to be representative (e.g., queries created by randomly selecting terms from the dataset).

Overall, these experimental results suggest that considerably more work is necessary before keyword search techniques for relational databases are ready for real-world retrieval tasks. The lack of appreciable difference in search effectiveness suggests that computationally cheap ranking schemes should be used instead of more complex scoring functions that require completely new query processing algorithms (e.g., those proposed by Luo *et al.* [LLWZ07a, LWL+11]). The performance of existing keyword search techniques is also

disappointing, particularly with regard to the number of queries completed successfully (see Table 3.3 on page 35). Given previously published results (see Table 2.5 on page 24), the number of timeout and memory exceptions is especially surprising. Because the greater execution times might only reflect the decision to use larger datasets, the following paragraph focus on two concerns related to memory utilization.

First, no search technique admits to having a large memory requirement. In fact, memory consumption during a search has not been the focus of any previous evaluation. Only two papers [DKS08, KRS+09] have been published in the literature that make allowances for a data graph that does not fit entirely within main memory. Given that most existing evaluations focus on performance, handling large data graphs (i.e., those that do not fit within main memory) should be better-studied. Relying on virtual memory and paging is no panacea to this problem because the operating system's virtual memory manager will induce much more I/O than algorithms designed for large graphs [DKS08]. Kasneci *et al.* [KRS+09] show that storing the graph on disk can also be extremely expensive for algorithms that touch a large number of nodes and edges. Second, these results seriously question the scalability of these techniques. Although the amount of memory used in the experiments (3 GB) is small by today's standards so are the datasets. In particular, the IMDb subset used in the benchmark is more than 25 times smaller than the entire database. Without additional research into high-performance algorithms that maintain a small memory footprint, keyword search techniques will be unable to handle even moderately-sized databases and will never be suitable for large databases like social networks or electronic health records.

Ultimately, a search technique's runtime performance depends a great deal on the characteristics of the underlying dataset because many of these techniques are merely search heuristics, designed to approximate answers to an intractable problem (i.e., the group Steiner tree problem). The results presented in this chapter suggest that researchers should use an existing benchmark to evaluate their work or carefully consider the numerous experimental design issues that arise in the evaluation of these systems to ensure their experimental results are robust.

In terms of overall search effectiveness, the various search techniques vary widely but no existing scheme is always best for search effectiveness. This result contradicts previous evaluations that appear in the literature. Not surprisingly, effectiveness is highest for the smallest dataset. The best systems, DPBF and BLINKS, perform exceedingly well. These scores are considerably higher than those that appear in IR venues (e.g., TREC), which likely reflects the small size of the Mondial database. Search effectiveness falls significantly for larger datasets. Unlike performance, which is generally consistent among systems, search effectiveness differs considerably. For example, DISCOVER-II performs poorly (relative to the other ranking schemes) for Mondial but is comparable to the best search techniques for IMDb and Wikipedia. A ranking scheme that performs well for Mondial queries is not necessarily ideal for other datasets. These results indicate the

importance of balancing performance concerns with a consideration of search effectiveness when designing relational keyword search systems.

In part, this benchmark and empirical evaluation were designed to corroborate the claims of search effectiveness previously presented in the literature. Across all the three datasets, runtime performance and effectiveness are considerably worse than previously reported in other evaluations. While it is known that values for effectiveness metrics cannot be directly compared across different document collections [Voo02], many previous studies appear to have inflated claims of search quality, perhaps due to unreported methodological problems such as tuning their systems on the evaluation queries. For example, Luo *et al.* state [LLWZ07a], "In our experiments, we found that [the following parameter values] yielded good retrieval results for most of the queries." This statement introduces uncertainty about how the system was tuned. Webber [Web10] confirms the trend toward reporting above-average effectiveness scores. The scores of retrieval systems evaluated at TREC and INEX are still considerably lower than those achieved on this benchmark. Perhaps the size of the collections plays a significant role, for effectiveness on the IMDb dataset lags considerably behind both MONDIAL and Wikipedia.

Finally, one additional observation regarding the definition of search results deserves mention. Existing definitions focus on minimality—i.e., each leaf of the result tree must match at least one search term. This assumption does not hold for some queries: for example, Nandi and Jagadish [NJ09] report that "[movie] cast" queries are common for the IMDb. Given that the cast relation is an associative entity in the database, such queries would fail to return any meaningful results. The queries developed for this benchmark explicitly avoid this issue—if anything, these results may over-estimate the effectiveness of these systems for real user queries.

Chapter 4

Ranking Results for Keyword Search in Databases

Given the existing approaches to support keyword search in relational databases, there are two possible areas for novel research contributions. Enumeration algorithms focus on runtime performance. Ranking schemes target search effectiveness. While an "optimal" ranking scheme is unlikely to exist, focusing first on ranking strategies has a major benefit: the enumeration algorithm can be tailored to provide optimal performance for the most effective ranking scheme.[1] In contrast, optimizing the enumeration algorithm first may limit the final ranking scheme (e.g., if certain factors cannot be computed efficiently due to characteristics of the enumeration algorithm).

This chapter proposes two novel ranking schemes to improve the quality of relational keyword search results. The first, structured cover density ranking, extends a ranking scheme for unstructured text. It enforces user preferences regarding how to order search results. The second ranking scheme uses machine learning to weight the factors that have been proposed previously in the literature. Applying machine learning to this problem is much more robust than trying to tune a scoring function by hand. Empirical evaluation using the benchmark described in the previous chapter indicates significant benefits in search effectiveness as compared to previous ranking schemes.

[1] The most effective ranking scheme can be determined via empirical evaluation using a standardized benchmark such as the one described in Chapter 3.

4.1 Cover density ranking

Structured cover density ranking, which is a generalization of an unstructured ranking scheme, provides an effective means to order search results from keyword search on structured data, including XML and relational data. Structured cover density ranking adheres to users' expectations regarding the order of search results—namely, results containing all query keywords should appear before results containing a subset of the search terms. Structured cover density ranking has two advantages over traditional IR similarity measures:

- it is designed for the short, ambiguous queries typically submitted to web search engines, and
- it explicitly adheres to users' preferences regarding ranking by coordination level.[2]

This section describes structured cover density ranking, a novel extension of a ranking scheme originally designed for unstructured text. This work illustrates the benefit of adapting scoring functions from the IR community to rank keyword search results from relational databases. In addition, structured cover density ranking can be used for any structured document (e.g., XML) unlike most scoring functions for relational keyword search that apply only to relational data.

4.1.1 Unstructured documents

Clarke *et al.* [CCT00] proposed cover density ranking in response to the findings of Wilkinson *et al.* [WZSD95] and Rose and Cutting [RC96]. Both previous studies found that users have a strong preference for coordination matching—that is, documents that match more search terms should be ranked ahead of documents matching fewer. In response, both groups introduced a ranking scheme that blended coordination level with a more traditional similarity measure. Cover density ranking goes further by abandoning traditional similarity measures altogether and focusing on ranking documents within coordination levels. Documents that contain all search terms are ranked first, followed by documents that contain all but one search term, etc. Initial ranking of documents by coordination level produces $|Q|$ document sets

$$D_{|Q|}, D_{|Q|-1}, D_{|Q|-2}, \ldots, D_1$$

where $|Q|$ is the number of terms in the user's query. Documents in each successive document set contain one fewer query term than documents in the previous document set. Formally, a document d is in the document set D_i for a particular query if $|\{t|t \in d \text{ and } t \in Q\}| = i$ (that is, i query terms appear in the document). In the final list of results, all documents in D_i appear before all documents in D_j if $i > j$. This ordering follows directly from the definition of coordination matching. Because coordination level provides a gross

[2]The coordination level of a document and query is the number of query terms contained in the document.

Raiders of the Lost Ark *Raiders*[1] *of*[2] *the*[3] *Lost*[4] *Ark*[5] is[6] a[7] 1981[8] American[9] action[10]-adventure[11] film[12] starring[13] Harrison[14] Ford.[15] It[16] pits[17] Indiana[18] Jones[19] (Ford[20]) against[21] a[22] group[23] of[24] Nazis[25] who[26] search[27] for[28] the[29] Ark[30] of[31] the[32] Covenant[33] because[34] Adolf[35] Hitler[36] believes[37] it[38] will[39] make[40] their[41] army[42] invincible.[43]

Indiana Jones and the Last Crusade *Indiana*[1] *Jones*[2] *and*[3] *the*[4] *Last*[5] *Crusade*[6] is[7] a[8] 1989[9] American[10] adventure[11] film.[12] Harrison[13] Ford[14] reprises[15] the[16] title[17] role[18] and[19] Sean[20] Connery[21] plays[22] Indiana[23]'s[24] father,[25] Henry[26] Jones,[27] Sr.[28] In[29] the[30] film,[31] set[32] largely[33] in[34] 1938,[35] Indiana[36] searches[37] for[38] his[39] father,[40] a[41] Holy[42] Grail[43] scholar,[44] who[45] has[46] been[47] kidnapped[48] by[49] Nazis.[50]

Harrison Ford Harrison[1] Ford[2] (born[3] July[4] 13,[5] 1942[6]) is[7] an[8] American[9] film[10] actor[11] and[12] producer.[13] He[14] is[15] famous[16] for[17] his[18] performances[19] as[20] Han[21] Solo[22] in[23] the[24] original[25] Star[26] Wars[27] trilogy[28] and[29] as[30] the[31] title[32] character[33] of[34] the[35] Indiana[36] Jones[37] film[38] series.[39]

Indiana Jones Colonel[1] Henry[2] Walton[3] "Indiana"[4] Jones,[5] Jr.,[6] Ph.D.[7] is[8] a[9] fictional[10] character[11] and[12] the[13] protagonist[14] of[15] the[16] Indiana[17] Jones[18] franchise.[19] The[20] character[21] first[22] appeared[23] in[24] the[25] 1981[26] film[27] *Raiders*[28] *of*[29] *the*[30] *Lost*[31] *Ark*,[32] to[33] be[34] followed[35] by[36] *Indiana*[37] *Jones*[38] *and*[39] *the*[40] *Temple*[41] *of*[42] *Doom*[43] in[44] 1984,[45] *Indiana*[46] *Jones*[47] *and*[48] *the*[49] *Last*[50] *Crusade*[51] in[52] 1989,[53] and[54] *Indiana*[55] *Jones*[56] *and*[57] *the*[58] *Kingdom*[59] *of*[60] *the*[61] *Crystal*[62] *Skull*[63] in[64] 2008.[65] Jones[66] is[67] most[68] famously[69] played[70] by[71] Harrison[72] Ford[73] and[74] has[75] also[76] been[77] portrayed[78] by[79] River[80] Phoenix[81] (as[82] the[83] young[84] Jones[85] in[86] The[87] Last[88] Crusade[89]).

Figure 4.1: Example documents used to illustrate cover density ranking. Terms are numbered to make it easier to identify their positions within each document. The documents' text is taken from the associated Wikipedia articles.

order for results, cover density ranking orders the documents within each coordination level by a measure of term co-occurrence.

Clarke *et al.* define a document as a sequence of terms: i.e., $d = (t_1, t_2, \ldots, t_{|d|})$ where $|d|$ is the number of terms in d. A document *extent* $\mathcal{E}$ is a sequence of terms $(t_p, \ldots, t_q)$ in d and is represented by the ordered pair (p, q) where $1 \leq p \leq q \leq |d|$. An extent satisfies a set of terms T if all the terms in T appear in the extent. For example, if a document d contains all the terms in T, the extent $(1, |d|)$, which represents the entire document, satisfies T. An extent is a *cover* for T if and only if it satisfies T and does not also contain a smaller extent that also satisfies T. The set $\mathcal{C}$ is the set of all covers for T in a document d.

Consider the documents shown in Figure 4.1 and the query "Ford trilogy character." Users prefer for documents that contain all search terms to be ranked ahead of those containing a smaller subset. For this query, the biographical sketch of Harrison Ford is the preferred result because it includes all three query terms. Indiana Jones's biography is ranked before the films because the film synopses each contain one search term. The cover sets for each document given this query are shown in Table 4.1. For the biographical sketch of Indiana Jones, the extent $(11, 73)$ satisfies the terms "Ford" and "character," but this extent is not a cover because it is not minimal. As another example, Table 4.1 also gives the coordination level and cover sets for these documents given the query "Indiana Jones ark crusade."

Scoring a cover set follows two intuitions:

1. "the more covers contained within a document, the more likely the document is relevant" and
2. "the shorter the cover, the more likely the corresponding text is relevant" [CCT00].

Table 4.1: Example of cover sets using the documents shown in Figure 4.1.

Query	Document	CL	$\mathcal{C}$
Ford trilogy character	*Raiders of the Lost Ark*	1	$\{(15,15),(20,20)\}$
	Indiana Jones and the Last Crusade	1	$\{(14,14)\}$
	Harrison Ford	3	$\{(2,33)\}$
	Indiana Jones	2	$\{(21,73)\}$
Indiana Jones ark crusade	*Raiders of the Lost Ark*	2	$\{(5,19),(18,30)\}$
	Indiana Jones and the Last Crusade	2	$\{(1,6),(2,23),(6,27)\}$
	Harrison Ford	1	$\{(36,37)\}$
	Indiana Jones	3	$\{(32,51)\}$

Legend
CL coordination level
$\mathcal{C}$ cover set

Given a cover set $\mathcal{C}$ where $\mathcal{C} = \{(p_1, q_1), (p_2, q_2), \ldots, (p_n, q_n)\}$, Clarke *et al.* suggest the following formula to score $\mathcal{C}$:

$$score(\mathcal{C}) = \sum_{j=1}^{n} score(p_j, q_j) \tag{4.1}$$

where

$$score(p, q) = \begin{cases} \dfrac{\mathcal{H}}{q - p + 1} & \text{if } q - p + 1 > \mathcal{H} \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

In this formula, $\mathcal{H} \in [1, \infty)$ is a tuning parameter. Covers smaller than $\mathcal{H}$ receive a score of 1, and longer covers receive scores proportional to the inverse of their length. The suggested default value for $\mathcal{H}$ is 16, which is shown to have good performance. Table 4.2 gives the scores and the final ranking for the documents using the queries from Table 4.1. Note that documents are ranked first by coordination level and then by cover density scoring within each coordination level.

Cover density ranking does not measure the frequency of individual search terms but rather the frequency

Table 4.2: Example of scoring cover sets. Documents are ordered (top-to-bottom) by their final position in the list of results.

Query	CL	$score(\mathcal{C})$	Document
Ford trilogy character	3	0.5	Harrison Ford
	2	0.302	Indiana Jones
	1	2.0	*Raiders of the Lost Ark*
		1.0	*Indiana Jones and the Last Crusade*
Indiana Jones ark crusade	3	0.8	Indiana Jones
	2	2.455	*Indiana Jones and the Last Crusade*
		2.0	*Raiders of the Lost Ark*
	1	1.0	Harrison Ford

Legend
CL coordination level
$\mathcal{C}$ cover set

and proximity of the search terms' co-occurrences in a document. A document may contain many instances of each search term and yet have a single cover (when all the search terms appear in discrete groups). Hence, cover density ranking not only considers term matches but also forms a crude approximation for phrase matching. When searching for information, a phrase approximation can be particularly useful when the order of the information is unknown (e.g., names written with the surname before the given name). Generating a cover set for a document can be done efficiently by exploiting the property that no two covers can start or end at the same position in the document.

Clarke *et al.* show that cover density ranking compares favorably with more traditional ranking schemes. Nevertheless, they do suggest limiting the use of cover density ranking to queries with 1–3 terms or blending its score with a more traditional scoring function for longer queries. Major benefits of cover density ranking include its simplicity and independence from collection statistics such as average document length or a term's document frequency. Not incorporating these collection statistics enables the underlying documents to be partitioned among multiple machines (i.e., split into disparate collections) and scored in parallel with no need for a master index, which most ranking schemes require. Furthermore, the cover sets used when scoring a document can be reused in the result snippets that most retrieval systems present to users as part of the search results.

4.1.2 Structured documents

Scoring structured documents differs significantly from scoring unstructured documents. Wilkinson *et al.* [Wil94] showed that retrieval systems benefit from indexing the constituent parts of a document individually, particularly when a small portion of the document should be returned as a result. The challenge is properly defining a function that combines the scores of the individual fields that compose the complete document. Ad-hoc definitions (e.g., summing the individual scores) are simple to apply but may damage the original properties of the ranking function [RZT04]. Thus, the primary goal when generalizing cover density ranking to structured documents is retaining its original scoring intuitions.

Unlike many other similarity measures that merely consider the frequency of query terms within a document, cover density ranking considers the position of terms within the document. Previous work that extends scoring functions developed for unstructured text to handle structured documents assumes traditional similarity measures. For example, Robertson *et al.* [RZT04] preserved the properties of the original scoring function by concatenating the various fields that compose the structured document and scoring the concatenation. However, this approach is not applicable because a structured document should not create new covers, which would occur if its parts were concatenated. As an example, consider a structured document

Table 4.3: Example of structured documents (relational data) from IMDb. The film id (e.g., $\mathbf{d}_1$) merely identifies the structured document.

id	**Film**	year	type
$\mathbf{d}_1$	*Raiders of the Lost Ark*	1981	movie
$\mathbf{d}_2$	*Indiana Jones and the Temple of Doom*	1984	movie
$\mathbf{d}_3$	*Indiana Jones and the Last Crusade*	1989	movie
$\mathbf{d}_4$	"The Young Indiana Jones Chronicles"	1992–1993	TV series
$\mathbf{d}_5$	*Indiana Jones and the Kingdom of the Crystal Skull*	2008	movie

$\mathbf{d}$ comprising two fields $f_1 = (\ldots, t_1)$ and $f_2 = (t_2, \ldots)$ where t_1 and t_2 are query terms. Concatenating these fields produces $d' = f_1 f_2 = (\ldots, t_1, t_2, \ldots)$, but note that whereas each field previously contained a cover with a coordination level of 1, the concatenation of the fields produces a cover with a coordination level of 2 (i.e., $\mathcal{C}_d = \{(|f_1|, |f_1| + 1)\}$ where $|f_1|$ is the length of the field f_1). Hence, concatenating the individual fields can create new covers not present in the original text. Furthermore, the order in which the fields are concatenated affects the covers of the resulting document.

Analogues of the definitions used by cover density ranking can be defined for structured documents. A structured document $\mathbf{d}$ is a set of fields where each field is an unstructured document: $\mathbf{d} = \{d_1, d_2, \ldots, d_{|\mathbf{d}|}\}$. A structured extent $\boldsymbol{\mathcal{E}}$ is a subset of the document's fields. A structured extent satisfies a term set T if all the terms of T appear in the structured extent; the individual terms may be present in difference fields. A structured cover is a structured extent that satisfies T and does not contain a smaller structured extent that also satisfies T. The set of all structured covers of a structured document $\mathbf{d}$ is denoted by $\boldsymbol{\mathcal{C}}$. Like the unstructured version, structured documents are initially ranked by coordination level; structured cover density ranking orders the documents within each coordination level.

Consider the relational data shown in Table 4.3 and the query "movie of Indiana Jones." The given set of structured documents each contain three different fields (i.e., title, year, and type). The structured extent {title, year, type} from document $\mathbf{d}_2$ satisfies the query terms, but this set of fields is not a structured cover because the extent {title, type} also satisfies this set of terms.

Scoring structured covers follows the unstructured scoring definition, i.e.

$$score(\boldsymbol{\mathcal{C}}) = \sum_{\boldsymbol{\mathcal{E}} \in \boldsymbol{\mathcal{C}}} score(\boldsymbol{\mathcal{E}}) \tag{4.3}$$

where

$$score(\boldsymbol{\mathcal{E}}) = \begin{cases} \dfrac{\boldsymbol{\mathcal{H}}}{|\boldsymbol{\mathcal{E}}|} & \text{if } |\boldsymbol{\mathcal{E}}| > \boldsymbol{\mathcal{H}} \\ 1 & \text{otherwise} \end{cases} \tag{4.4}$$

Like $\mathcal{H}$ in the original ranking function, $\boldsymbol{\mathcal{H}} \in [1, \infty)$ is a tuning parameter. Decreasing $\boldsymbol{\mathcal{H}}$ rewards documents

Table 4.4: Example ranking documents by structured cover density scoring.

Query	CL	$\mathcal{C}$	$score(\mathcal{C})$	Document
movie of Indiana Jones	4	{title, type}	1.0	$\mathbf{d}_2$
		{title, type}	1.0	$\mathbf{d}_5$
	3	{title, type}	1.0	$\mathbf{d}_3$
	2	{title}	1.0	$\mathbf{d}_4$
		{title, year}	1.0	$\mathbf{d}_1$
1984 movie of Indiana Jones	5	{title, year, type}	0.667	$\mathbf{d}_2$
	4	{title, type}	1.0	$\mathbf{d}_5$
	3	{title, type}	1.0	$\mathbf{d}_3$
	2	{title}	1.0	$\mathbf{d}_4$
		{title, year}	1.0	$\mathbf{d}_1$

Legend
CL coordination level
$\mathcal{C}$ structured cover set

that contain smaller subsets of fields that completely satisfy a query's term set. When multiple documents score identically, ties are broken using the mean cover density scores (that is, the scores of the unstructured fields) to capture term proximity within each field.

Table 4.4 gives the ranking for the structured documents from Table 4.3 for two queries. For the second query ("1984 movie of Indiana Jones"), the mean cover density score is used to break the tie between $\mathbf{d}_4$ and $\mathbf{d}_1$. "The Young Indiana Jones Chronicles" ($\mathbf{d_4}$) is ranked before *Raiders of the Lost Ark* ($\mathbf{d_1}$) because the former contains two search terms in the same field (i.e., title) whereas the latter contains the terms in separate fields. Hence, the mean coordination level (of the fields that contain terms) is 2 for "The Young Indiana Jones Chronicles" while it is only 1 for *Raiders of the Lost Ark*.

4.1.3 Normalization

Fang *et al.* [FTZ04] investigated heuristics that IR scoring functions should satisfy. Because cover density ranking abandons traditional $tf \cdot idf$ weighting used to score documents, many of these constraints do not apply. Nevertheless, the general intuitions are still applicable as evidenced by the similarity between Clarke *et al.*'s scoring intuitions and more traditional heuristics. For example, cover density ranking favors documents that contain more covers just like the traditional heuristic that favors documents that contain more instances of a query term. The second scoring intuition for cover density ranking also vaguely satisfies the following length normalization constraint: "the score of a document should decrease if we add one extra occurrence of a non-relevant word" [FTZ04]. This constraint is satisfied by cover density ranking if the extra term appears in a cover (and the cover is longer than $\mathcal{H}$); it is not satisfied when the extra term is merely appended to the document. This failure can have important consequences particularly when the document collection contains a mix of long and short documents.

Assume that a long and short document both contain information relevant to a given query and that this information is identical. A retrieval system should prefer the more specific document (i.e., the shorter one) to minimize the time the user must spend to process the results. This preference is reflected by a variety of evaluation metrics that have been proposed [Dun97, dVKL04] (see also Lalmas and Tombros [LT07]). Fortunately, it is simple to satisfy the more general constraint by normalizing a document's cover density score by the document's length. This normalization does not alter the original scoring intuitions but does allow cover density ranking to satisfy the more general retrieval heuristic.

4.2 SVM rank

Even though it is possible to define many different scoring functions (such as structured cover density ranking) for keyword search in databases, a key research question is why certain scoring functions outperform others. The search techniques evaluated in Chapter 3 provide comparable search effectiveness despite their different factors and normalizations. It would be ideal if a scoring function included only the most effective factors and normalizations from the previous work rather than creating a new scoring function from scratch. SVM rank is a learning-to-rank scheme that achieves this objective. Machine learning is preferable to deriving a new scoring function by hand due to machine learning's ability to consider many potential factors and to provide confidence measures for its results. Moreover, scoring functions typically include a number of parameters that have previously been tuned by researchers using a small number of queries. It is reasonable to believe that significant improvement is possible simply by applying better tuning methodologies (e.g., via machine learning).

This section describes SVM rank. It begins with a brief overview of existing scoring functions to motivate the learning-to-rank approach. Next, an overview of machine learning (as it applies to IR) is given. This section concludes with an overview of the features used to train the machine learning model; the following section evaluates both structured cover density ranking and SVM rank against the evaluation benchmark described in Chapter 3.

4.2.1 Motivation

Despite the recent research interest, scoring functions for keyword search in relational databases have become increasingly complex while providing only modest benefits with regards to the quality of search results. Many researchers to date have relied on intuition to create new scoring functions. For example, many search techniques assume that the relevance of a search result is negatively correlated with the total edge weight of a result. That is, a result with a larger sum of edge weights is less relevant than a result with a smaller

Table 4.5: Summary of scoring functions used by proximity search techniques. Systems that do not propose an alternative scoring function are omitted (i.e., each row contains at least one unique definition •).

System	Features			Factors		Parameters
	$\lvert N \rvert$	$weight(e)$	$weight(n)$	$\sum_e weight(e)$	$\sum_n weight(n)$	
DBXplorer	•					0
BANKS	○	•	•	•	•	1
BANKS-II		•	•	•	•	1
DPBF		○		•		0
BLINKS		•	○	•	•	3
EASE		○		•		0
Golenberg *et al.*		•		○		2
CSTree		○	○	○	•	2

Legend

- • unique definition
- ○ uses existing definition

$n \in N$ node in result

e edge in result

sum of edge weights. Ignoring the issue that the literature contains little evidence for this relationship, one significant problem is that minimizing the total sum of edge weights is known to be NP-complete. Some systems propose alternative semantics (e.g., minimizing the weight of paths between the root of a result tree and its leaves [KPC+05, HWYY07]) that enable more efficient enumeration of search results. No existing work studies the impact different semantics have on search effectiveness nor have researchers validated the underlying assumption—that is, whether or not relevance is negatively correlated with the weight of result trees. These intuitions should be validated to ensure their correctness. After all, why should search techniques approximate solutions to an intractable problem that need not be solved?

Tables 4.5 and 4.6 summarize the features advocated by search techniques that propose a novel scoring scheme. Many search techniques are omitted from these tables because they address performance issues (i.e., enumeration) rather than ranking. Distinctions among the ranking schemes are indicated in the tables where a unique definition (•) differs from previous work instead of using an existing technique (○) to score results. Each row of the table contains at least one unique element. For example, DISCOVER ranks results identically to DBXplorer so DISCOVER is omitted from Table 4.5.

The aforementioned tables illustrate two general trends. First, the number of "features" (that is, atomic values from a search result, user's query, or database) has increased steadily as the various scoring functions distinguish between a search result's root, leaves, interior nodes, etc. Accompanying this increase is a rise in the number of "factors" (that is, values derived from one or more features). SPARK [LLWZ07a] provides an excellent example, for it multiples two IR-style scoring functions (pivoted normalization weighting [SCH+99] and the extended boolean model [SFW83]) by its own size normalization factor to compute a result's score.

Table 4.6: Summary of IR-style scoring functions used for relational keyword search. The factors *ntf*, *ndl*, and *idf* are components of pivoted normalization scoring; ⊕ indicates the combination of attribute scores.

System	Features			Factors						Parameters
	$\lvert N\rvert$	$\lvert N\rvert / \overline{\lvert N\rvert}$	$\lvert N'\rvert$	*nsize*	L^2 *norm*	*ntf*	*ndl*	*idf*	⊕	
DISCOVER-II	○			○		○	○	○	●	1
Liu *et al.*	○	○		●		○	●	●	●	2^a
SPARK	○		○	●	●	○	●	●	○	4
EASE						●	○	●	○	1
SAINT					●	○	○	○	1	
CSTree					●			●	2	

Legend

- ● unique definition
- ○ uses existing definition
- ⊕ attribute combination

N nodes in result
$\overline{\lvert N\rvert}$ average result size
N' non-leaf nodes

[a]The two occurrences of s are assumed to be independent although this issue is not addressed in the original paper.

Second, as the number of factors increases so does the total number of free parameters, which are used to weight the importance of each factor in the final scoring function. This is problematic because tuning these parameters is expensive, requiring exploration of a low-dimensional grid of possible parameter values or heuristics that partially explore many dimensions [TZC+06]. Unfortunately, the existing literature does not suggest that these experiments are being conducted by researchers.

4.2.2 Machine Learning

Traditionally, scoring functions developed by the IR community included only a small number of features (e.g., term frequency (*tf*), document frequency (*df*), and document length (*dl*)), which allowed the parameters of these functions to be tuned empirically [Sal71]. In different contexts, many other features have proven useful for predicting the relevance and importance of results. For example, PageRank [BP98] is a well-known algorithm for determining the prestige of web pages. Incorporating these features into existing or even new scoring functions is non-trivial because increasing the number of features increases the number of free parameters, which makes empirical tuning of scoring functions prohibitively expensive.

Machine learning offers an alternative to creating scoring functions by hand and also may be used to determine the relative weights of different features. The major benefit of applying machine learning to this problem is that machine learning scales to many more factors than hand tuning and also provides confidence measures for its results. In this case, the goal of machine learning is to determine the weight of each feature that maximizes the relevance of results because some features are more important than others.

The process of assigning weights to features to maximize the relevance of results is described as ordinal regression. Ordinal regression falls between classification—that is, predicting a categorical variable (e.g., relevant or non-relevant)—and regression—that is, predicting a real number as a function of inputs.

Given a document collection $C = \{d_1, d_2, \ldots, d_{|C|}\}$ and query Q, the objective of an IR system is to provide a ranking σ that optimally orders the documents according to their relevance to Q. Achieving this optimal ranking is difficult, and IR systems provide an approximation, σ', of the optimal. An IR system is evaluated based on how closely its ranking approximates the optimal ranking.

This work uses a ranking support vector machine (SVM) to perform ordinal regression. A SVM is a type of large-margin classifier, which attempts to maximize the decision boundary between classes. In simple terms, the larger the margin of a classifier, the more certainty is provided in the predicted decision. The goal of a ranking (or ordinal regression) SVM is to learn a function f such that for any d_i, d_j

$$f(\phi(q, d_i)) > f(\phi(q, d_j)) \Rightarrow \sigma(d_i) > \sigma(d_j)$$

where $\phi(q, d)$ is the feature vector of the query and a document d and $\sigma(d)$ is the position of d in the optimal ranking. The SVM essentially learns a function f that will reproduce the optimal ranking for the given training data. The feature vector, $\phi(q, d)$, is an n-dimensional vector of features that represents important characteristics of the query and document—for example, term frequency, document length, and (in hyperlinked environments) PageRank. Let P be the set of pairs (i, j) where $\sigma(d_i) > \sigma(d_j)$. Formally, a ranking SVM optimizes

$$\min_{\vec{w}, \xi_{ij} \geq 0} \frac{1}{2} \vec{w}^T \vec{w} + \frac{C}{P} \sum_{(i,j) \in \mathcal{P}} \xi_{ij}$$

subject to

$$\forall (i, j) \in P, \quad \vec{w}^T f(\phi(q, d_i)) \geq \vec{w}^T f(\phi(q, d_j)) + 1 - \xi_{ij}$$

where $\vec{w}$ is the decision hyperplane that separates different classes (ranks) and ξ is a slack variable that allows for training errors. Allowing for training errors is necessary because reproducing the optimal ranking σ may not be possible given the available features. When the optimal ranking cannot be reproduced, the number of errors should be minimized. In other words, the retrieval system's ranking σ' should be as close as possible to the optimal ranking σ. The function f identifies the ideal weights for each feature by minimizing the number of pairs that are swapped (with regard to the optimal ranking) when training the SVM. The output of the SVM is a vector of weights that provides the best performance for the training instances—i.e., the SVM maximizes the relevance of the results.

4.2.3 Features

The features included in any scoring function should be efficient to compute and highly correlated with the ideal ranking of the search results [GKS08]. A review of existing scoring functions identified ten different features that are used in eight different factors. Some features included in previous work cannot be computed efficiently. For example, calculating the average number of nodes per result ($\overline{|N|}$) as proposed by Liu *et al.* [LYMC06] requires enumerating all possible results for each query. Including this feature in a scoring function immediately precludes any top-k query processing scheme. Although query processing is largely orthogonal to ranking, efficient execution is critical to achieve acceptable end-to-end performance for keyword search in relational databases.

Twelve different factors are derived from the features present in existing scoring function. The additional factors revolve around simple ways to combine multi-valued features (e.g., averaging the edge weights). Table 4.7 lists all the factors. Some factors subsume the features shown in Tables 4.5 and 4.6. For example, pivoted normalization weighting [SCH+99] includes normalized term frequency (*ntf*), normalized document length (*ndl*), and inverse document frequency (*idf*) factors. Rather than considering these factors in isolation, they are combined in pivoted normalization weighting, which is a state-of-the-art scoring function [Sin01]. By no means is the selection of twelve factors exhaustive—many more features and factors could be derived—but stopping at twelve limits the time required to train the model.

The factors are generally a superset of those included in previous work. All the variations of these factors that appear in the literature are not explored because doing so would greatly expand the search space. For example, there are nearly 100 different formulations of pivoted normalization weighting alone if all the proposed modifications are considered independently. In addition, several factors have been created to improve the tractability of enumerating search results (e.g., distinct root semantics [KPC+05, HWYY07]), and these factors are typically compared to more traditional methods for scoring results (in this case, minimizing the total edge weight), which are included in the twelve factors considered in this work.

Given the variety of existing techniques to calculate the value of each factor, one is chosen instead of exploring all the alternatives. BLINKS's formula is used for deriving the weights of edges in the data graph:

$$weight(e) = \ln(1 + out(u)) \tag{4.5}$$

where e is an edge from u to v and $out(u)$ is the outdegree of u. Node weights are computed using the PageRank algorithm [BP98], which is the most common method in the literature. Pivoted normalization weighting's document-related factors are computed over an entire search result (i.e., the tree composed of database tuples (vertices in the data graph)) and collection statistics are taken from the entire database.

Table 4.7: Correlations between factors and the relevance of results ($p < 0.01$). Larger absolute values indicate stronger correlations.

Factor	r	ρ	τ
$\lvert N \rvert$	-0.007	-0.041	-0.045
$\lvert E \rvert$	-0.007	-0.041	-0.045
$\min weight(e)$	-0.096	-0.039	-0.036
$\max weight(e)$	0.149	0.243	0.215
$\sum weight(e)$	0.048	0.142	0.123
$\overline{weight(e)}$	0.113	0.172	0.154
$\min weight(n)$	0.043	0.008	0.005
$\max weight(n)$	-0.097	-0.009	-0.010
$\sum weight(n)$	-0.095	0.035	0.026
$\overline{weight(n)}$	-0.095	0.031	0.024
pivoted normalization	0.264	0.389	0.310
Euclidean (L^2) distance	-0.132	-0.219	-0.207

Unlike traditional IR, the collection statistics are not well-defined when searching semi-structured and relational data (see Mass and Mandelbrod [MM05], Vittaut *et al.* [VPG05], and Clarke [Cla05]).[3] The decision to use statistics from a complete search result and the entire database follows Robertson *et al.*'s previous work [RZT04] and techniques from XML retrieval [Cla05].

Correlating Scoring Factors with Relevance

Before presenting the creation of the learned scoring function, the correlation between each factor and the relevance of search results is examined. Three different measures of the correlation between each factor and relevance are considered: Pearson's r [Pea20], which measures the degree of linear relationship [RN88]; Spearman's ρ [Spe04], which measures the degree of dependence with any monotonic function; and Kendall's τ [Ken38], which is the number of bubble-sort swaps required to transform one list into another. Kendall's τ is perhaps the best metric for comparing ordinal correlations [Joa02], but Pearson's r and Spearman's ρ are also reported because they are more common. In general, the trends are similar regardless of the measurement used.

Table 4.7 summarizes the correlations. The correlations are weak, but all the values are significant at the 0.01 level.[4] Pivoted normalization weighting is most correlated with relevance. The Euclidean distance (which is 0 if the result contains all terms present in the query Q and 1 if the result does not contain any query terms) has one of the most negative correlations. These two results are not altogether unsurprising

[3] It is straightforward to show that calculating exact collection statistics in the context of relational keyword search requires violating desirable properties of the original IR scoring function (as done by DISCOVER-II [HGP03] and Liu *et al.* [LYMC06]), estimation (see SPARK [LLWZ07a, LWL+11]), or enumeration of all possible search results *a priori* (as proposed by EASE [LOF+08]).

[4] With the exception of Kendall's τ for the feature $\min weight(n)$, all values are significant at the 0.001 level.

because pivoted normalization is a state-of-the-art scoring function and the Euclidean distance is a very coarse measure of relevance. While most relevant results will contain all the query terms, the converse is not true: a result is not relevant just because it contains the query terms—it must address the query's underlying information need.

The correlations for different edge weight factors are extremely surprising. The table simply includes the raw values of each factor. In contrast, proximity search systems minimize the weight of results. Hence, a larger total edge weight ($\sum weight(e)$) is expected to be *negatively* correlated with relevance or—stated conversely—a smaller sum should be positively correlated with relevance. These results indicate the exact opposite: results with higher weights are more likely to be relevant than results with smaller weights! This result differs substantially from the conventional wisdom in this field, and it will be discussed further in Section 4.4.

4.2.4 Model

Because the evaluation benchmark described in Chapter 3 does not provide both training and testing data, 10-fold cross validation [Koh95] mitigates the threat of overfitting a scoring function derived via machine learning. In 10-fold cross validation, the data is randomly partitioned into ten different folds (subsets). Training uses nine of the folds (that is, 90% of the original data), and the model is tested against the final fold. This process is repeated so each fold is used exactly once to test the model.

SVM^{rank} [Joa06] is used to learn the weight for each factor in the scoring function. SVM^{rank} is designed to efficiently solve ordinal regression problems and operates on a set of training instances. The training instances are derived from potential query results, and each instance consists of a feature vector of all the factors included in the scoring function and the ideal rank of the result.

Because different enumeration algorithms have been shown to omit results [GKS08], the top-1000 results returned by 9 different search techniques [BHN+02, HP02, HGP03, KPC+05, LYMC06, DYW+07, HWYY07, LLWZ07a, CW10b] are pooled to create the set of training instances. Pooling ensures that as many different query and result semantics as possible are considered and increases the generality of the scoring function. It also decouples the challenges of enumeration and ranking although the training instances only include results that have been deemed relevant by at least one existing system.

4.3 Evaluation

This section compares structured cover density ranking and SVM rank with previous strategies for ranking search results.

4.3.1 Experimental Setup

The benchmark described in Chapter 3 is used to evaluate the two proposed scoring functions. For the experiments described in this chapter, the retrieval depth was 1000 results, and unlike the experiments presented in Chapter 3, the execution time for search techniques was not limited to one hour. One minor modification needed for SVM rank is described below.

Binary relevance judgments were originally created for the benchmark. While these judgments would be sufficient to train a classifier, they are not suitable for ordinal regression, and the limited number could potentially diminish training effectiveness. The binary relevance judgments are augmented with marginal relevance judgments of the results returned by each search technique. The existing set of relevant results forms the *ideal recall base* for each query. The ideal recall base includes all non-overlapping results [LKK+07]. In recent years, evaluation forums have moved toward marginal relevance judgments, and search techniques should return results in decreasing order of their estimated marginal relevance. Hence, the supplied relevance judgments are extended by creating the *full recall base*, which includes all results that have any relevance to the query. The full recall base is created by pooling the results returned by each system, and any result that completely overlaps an ideal result is judged marginally relevant.[5] Relevance scores are assigned according to the practice used at INEX: the marginal relevance of a result is the ratio of relevant information to the total information [LKK+07, LT07]. Issues related to overlap have been investigated in the context of XML retrieval [KJAA05] so they are not explored further here.

The search effectiveness metrics described in Chapter 3 (precision @ k, MRR, and MAP) are used for these experiments in addition to normalized discounted cumulative gain (nDCG). NDCG was designed for evaluations with non-binary (i.e., marginal) relevant assessments. Like MAP, nDCG is scored in the range $[0, 1]$ with higher scores being better. The family of cumulative gain metrics rewards systems that return results in the order of their relevant to the query. Including nDCG is warranted due to the extension of the binary relevance assessments to include marginal relevance assessments of search results.

Given a large number of factors and available training instances (see Table 4.9 on page 72), a SVM may fail to terminate within a timely fashion. The SVM is given 1% of the available training instances and the remainder are ignored. This subset always includes all the ideal (completely relevant) results and—if possible—at least as many marginally relevant and irrelevant results as the number of ideal results to ensure the training set covers the gamut of available relevance judgments.

The parameters for structured cover density ranking are set as follows: $\mathcal{H} = |Q|$ (that is, the number of terms in the query) and $\boldsymbol{\mathcal{H}} = 1$. These parameters are not tuned for the datasets and queries because such

[5] Unlike XML, which is tree-structured, relational keyword search systems create result trees from a data graph. Two results are overlapping when one is a subset of the other and not when they merely share tuples in common.

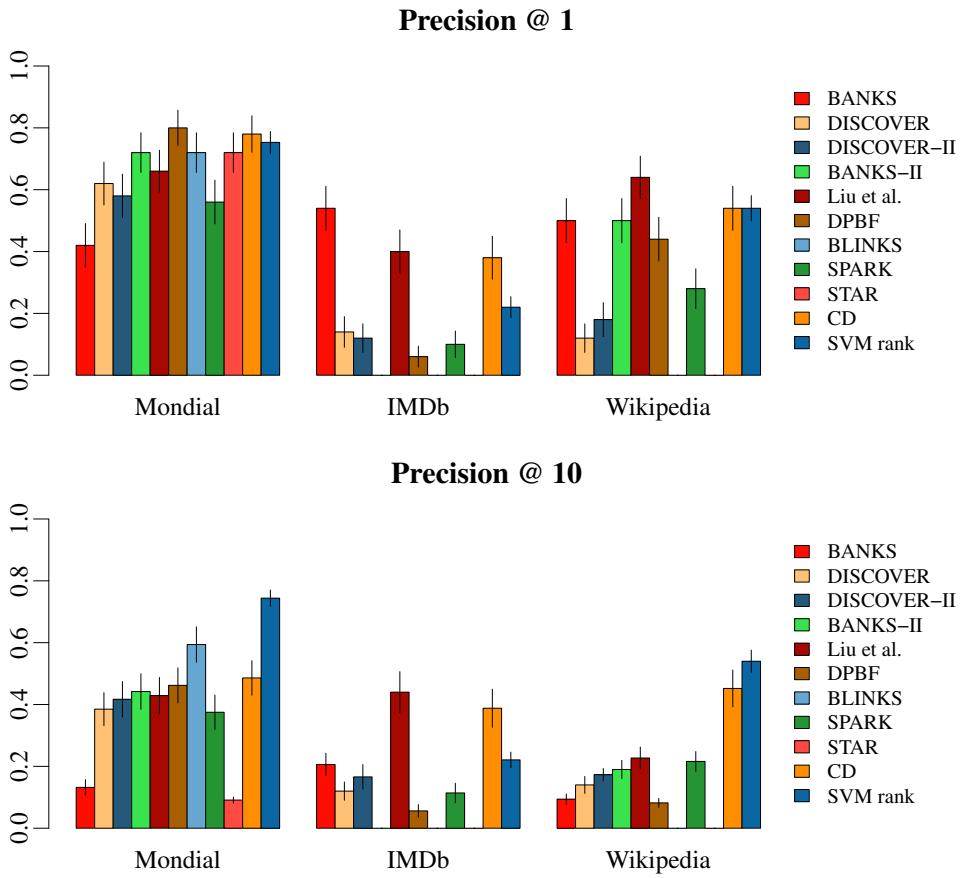


Figure 4.2: Precision @ k ($\in [0, 1]$, higher is better) for the search techniques on each dataset. Omitted bars indicate that the search technique failed to retrieve *any* results for any query. Search techniques are ordered by date of publication, left to right in the graphs and top to bottom in the legends. The error bars denote the standard error of the mean.

tuning would overstate the effectiveness of the ranking scheme and bias the results in its favor. Additional tuning would undoubtedly improve structured cover density ranking's effectiveness. Structured cover density ranking was implemented via DISCOVER-II's naive query processing strategy. It does not use the pooled results from other search techniques.

4.3.2 Experimental Results

Figure 4.2 presents results for precision @ k. These results are similar to those previously reported in Chapter 3; the minor deviations are due to the full recall base, which increases the total number of relevant results. Even though structured cover density ranking is never the most effective ranking scheme as measured by P@1 and P@10, it tends to lag the best systems only slightly. Moreover, its performance is very stable: it is never worse than third among the various scoring schemes and is usually second-best for these two metrics. On Mondial and Wikipedia, SVM rank lags slightly behind the best systems for P@1, but it scores best for

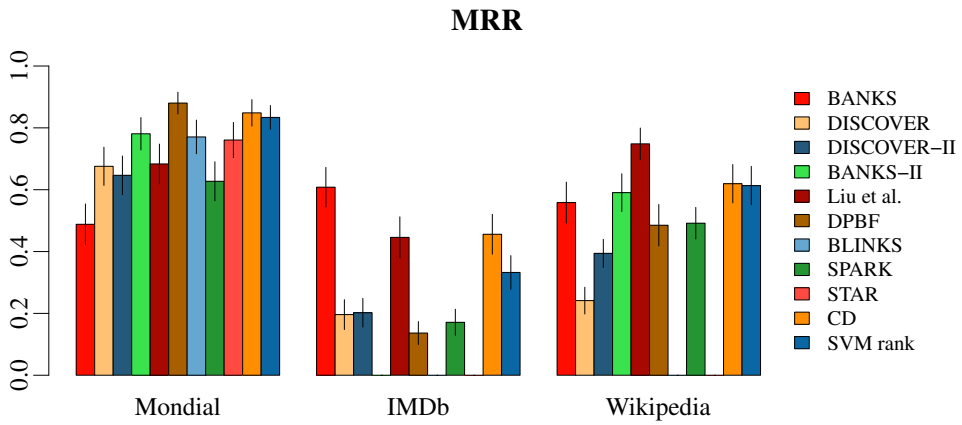


Figure 4.3: MRR ($\in [0, 1]$, higher is better) for the search techniques on each dataset. Omitted bars indicate that the search technique failed to retrieve *any* results for any query. Search techniques are ordered by date of publication, left to right in the graph and top to bottom in the legend. The error bars denote the standard error of the mean.

P@10. However, SVM rank falters on the IMDb dataset, scoring only half as well as the most effective search techniques, which is likely due to their slight variations on the scoring factors included in SVM rank. For example, BANKS uses a different edge weighting scheme than SVM rank, which likely accounts for BANKS's greater search effectiveness for this dataset. SVM rank outperforms the search techniques that share exactly the same factors.

The results for MRR (Figure 4.3) are very similar to the results of P@k. In fact, the relative performance of each system compared to the others is largely unchanged from P@1. Structured cover density ranking is second-best on all three datasets. SVM rank continues to provide excellent performance on the MONDIAL and Wikipedia datasets but is only average for IMDb.

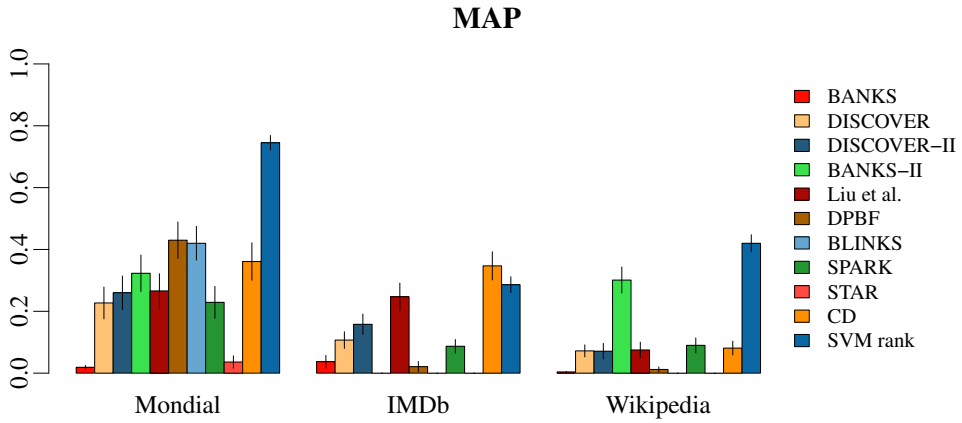


Figure 4.4: MAP ($\in [0, 1]$, higher is better) for each search technique and dataset. Search techniques are ordered by date of publication, top to bottom in the legend and left to right in the graph. Omitted bars indicate that the system failed to retrieve *any* results for any query. The error bars denote the standard error of the mean.

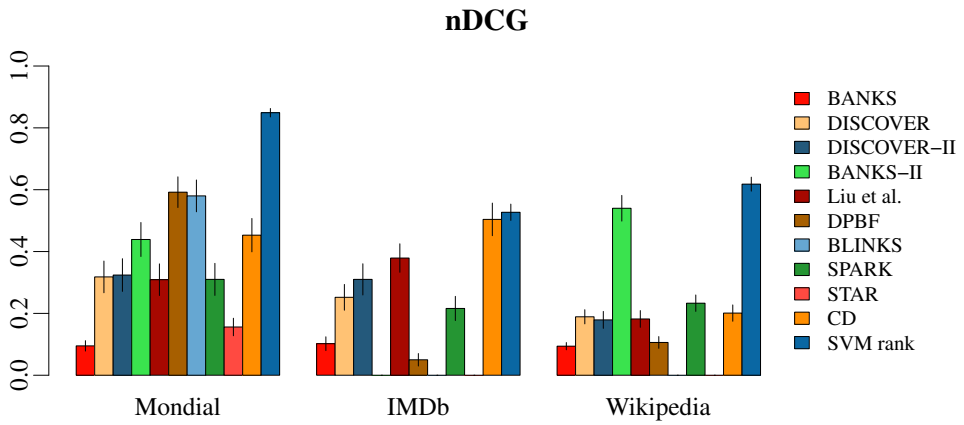


Figure 4.5: NDCG ($\in [0, 1]$, higher is better) for each search technique and dataset. Search techniques are ordered by date of publication, top to bottom in the legend and left to right in the graph. Omitted bars indicate that the search technique failed to retrieve *any* results for any query. The error bars denote the standard error of the mean.

Figure 4.4 presents MAP for each system and dataset. Structured cover density ranking continues to be very steady as it is always in the top half of the rankings. In comparison, most of the other scoring schemes (with the exception of SVM rank) that perform well on one dataset perform poorly on the other two. SVM rank nearly doubles the performance of any previous search technique for the MONDIAL database and also outscores the other search techniques for Wikipedia queries. On the IMDb dataset, SVM rank is narrowly beaten by structured cover density ranking. The results for MAP are considerably lower than those of P@k and MRR due to many search techniques failing to enumerate all the relevant results (e.g., due to different search semantics). Unlike P@k and MRR, these results differ dramatically from those previously reported in Chapter 3; this deviation is again attributable to using the full recall base in lieu of the ideal recall base supplied with the evaluation framework and extending the retrieval depth to 1000 results.

Figure 4.5 shows nDCG for each search technique and dataset. In terms of the relative rank of each search technique, these results are similar to MAP. However, nDCG rewards search techniques that rank more relevant results ahead of less relevant results. Several search techniques improve significantly under nDCG; indeed every search technique improves at least slightly compared to MAP. With the exception of the IMDb dataset, structured cover density ranking's performance relative to the other systems is unchanged. SVM rank shows a moderate boost in its measured effectiveness; this improvement is sufficient for SVM rank to edge structured cover density ranking as the best system on the IMDb dataset, which indicates that SVM rank is more likely than structured cover density ranking to rank more relevant results ahead of less relevant results for these queries.

4.4 Analysis of SVM rank

This section analyzes the linear function that SVM rank uses to score search results.[6] Many features can be eliminated without having a significant impact on search quality. Limitations of the approach are also discussed.

4.4.1 Training and Features

Using machine learning to create a scoring function potentially introduces several sources of error in the results. These sources have a negligible effect, as shown in the following paragraphs.

Sensitivity

As mentioned previously, cross validation is used to train and to test the scoring function. Because cross validation requires randomly partitioning the query workload, it is possible that the choice of partitions biases the results. To investigate this possibility, retrieval effectiveness across 10 different random partitions is compared (i.e., the cross validation is repeated 10 times). Table 4.8 shows the results of this experiment. As evidenced by the table, different cross folds do not significantly impact the results.

A second source of possible error stems from using a fraction of the available training instances. Table 4.9 provides the search effectiveness from different percentages of training instances where the set of training instances is randomized for each percentage. Because all the ideal results are retained for each query, doubling the percentage does not exactly double the number of training instances. The table shows that increasing the number of training instances increases the time required to train the model, from approximately 5 minutes (per fold) for 0.5% to almost 20 minutes for 3.0%. However, the impact that the training percentage has on

[6] No analysis of structured cover density ranking is given due to its few design decisions and tuning parameters.

Table 4.8: Comparison (nDCG) of SVM rank trained using different cross folds. The "baseline" is graphed in Figure 4.5.

	nDCG		
Dataset	baseline	μ	σ
Mondial	0.849	0.862	0.006
IMDb	0.527	0.525	0.002
Wikipedia	0.618	0.621	0.007

Legend
μ mean
σ standard deviation

Table 4.9: Mean number of training instances and average training time across all cross-folds for different percentages of training instances. MAP and nDCG improve only slightly with more training instances. Both metrics are averaged over all 3 datasets.

	Training		Effectiveness	
%	Instances	Time (s)	MAP	nDCG
0.5	2279	298	0.457	0.657
1.0	3430	321	0.489	0.663
2.0	5909	550	0.509	0.664
3.0	8453	1024	0.516	0.667

search effectiveness is negligible. MAP sees moderate improvement when moving from 0.5% to 3.0%, but the impact on nDCG is much smaller.

Feature Selection

Feature selection tries to reduce the number of factors in the scoring function without significantly impacting search effectiveness. Starting with the baseline scoring function that includes twelve different factors, the factor with the lowest weight (that is, it contributes least to the final ranking) is removed and a new model without that factor is created. This process is then repeated to progressively eliminate factors.

While greedy backward selection is non-optimal—finding the optimal requires searching all possible combinations of factors [KJ97]—it does provide a simple measure of the importance of each factor. There is overlap among the factors: a principal component analysis shows that 98% of the variability is attributed to

Table 4.10: Comparison of retrieval effectiveness when removing factors (listed in Table 4.7 on page 65) from the scoring function. The factors that have been removed are *cumulative*—i.e., the second variation of SVM rank (third row of the table) does not include $\max weight(n)$ or $\sum weight(n)$. SVM rank (minimal) includes the 2 factors that were *not* removed: $\min weight(n)$ and pivoted normalization. It is the same as the next-to-last row of the table (but is duplicated for clarity). The **best score** for each dataset is bolded.

Removal		Mondial			IMDb			Wikipedia		
			Improvement			Improvement			Improvement	
Order	Factor	nDCG	Δ	%	nDCG	Δ	%	nDCG	Δ	%
	SVM rank	0.849	—	—	**0.527**	—	—	0.618	—	—
1	$\max weight(n)$	0.854	0.005	0.6	0.525	-0.002	-0.4	0.627	0.009	1.5
2	$\sum weight(n)$	**0.866**	**0.017**	**2.0**	0.515	-0.012	-2.3	0.630	0.012	1.9
3	$\overline{weight(n)}$	0.855	0.006	0.7	0.518	-0.009	-1.7	0.616	-0.002	-0.3
4	Euclidean distance	0.673	-0.177	-20.8	0.454	-0.073	-13.6	0.560	-0.058	-7.7
5	$\sum weight(e)$	0.681	-0.169	-19.9	0.464	-0.063	-12.0	0.576	-0.042	-6.8
6	$\lvert N \rvert$	0.679	-0.170	-20.0	0.446	-0.081	-15.4	0.580	-0.038	-6.1
7	$\max weight(e)$	0.676	-0.173	-20.4	0.456	-0.071	-13.5	0.573	-0.045	-7.3
8	$\overline{weight(e)}$	0.691	-0.158	-18.6	0.469	-0.058	-9.4	**0.634**	**0.016**	**2.8**
9	$\lvert E \rvert$	0.711	-0.139	-16.4	0.494	-0.033	-6.3	0.619	0.001	0.2
10	$\min weight(e)$	0.772	-0.077	-9.1	0.495	-0.032	-6.1	0.623	0.005	0.8
	SVM rank (minimal)	0.772	-0.077	-9.1	0.495	-0.032	-6.1	0.623	0.005	0.8

six different components. Hence, it should be possible to eliminate a number of factors without significantly impacting search quality.

Table 4.10 shows nDCG as factors are removed from the scoring function. Note that the number of factors removed is cumulative—i.e., the first variation is SVM rank without $\max weight(n)$ and the second variation is SVM rank without $\max weight(n)$ and without $\sum weight(n)$. The final variation, SVM rank (minimal), includes the factors *not* appearing in the table: $\min weight(n)$ and pivoted normalization weighting. As evidenced by the table, retrieval effectiveness remains largely unchanged as several factors are removed. However, performance drops precipitously when the fourth factor (the Euclidean distance between a result and the query) is removed. The impact is most noticeable for the MONDIAL dataset. This factor is undoubted noisy, having almost no impact for many queries but is very important for a few.

All but two factors can be removed from the scoring function without significantly impacting search effectiveness. Retrieval effectiveness actually improves on the Wikipedia dataset as additional features are eliminated. This trend is not completely surprising given that pivoted normalization weighting was developed by the IR community to score lengthy text documents and the Wikipedia articles are most similar to this use case.

4.4.2 Comparison with Alternative Functions

If the order in which features are removed (see Table 4.10) is indicative of importance, node weights (with the exception of the minimum node weight, which is included in SVM rank (minimal)) are relatively unimportant when ranking search results. Similarly, the total sum of edge weights is the first edge-based factor to be dropped. This result clashes sharply with conventional wisdom in this field. In particular, all proximity search techniques attempt to rank search results in order of their total edge weight. Strangely enough, other factors (e.g., the minimum edge weight) are more indicative of relevance as evidenced by its stronger correlation with relevance (see Table 4.7 on page 65) and by its longer retention in feature selection. This result opens the door for future debate regarding the design of new scoring functions.

Figure 4.6 considers the two versions of SVM rank and two alternatives that score search results more conventionally. SVM rank (minimal), which includes only two factors ($\min weight(n)$ and pivoted normalization), is compared to a scoring function using pivoted normalization and $\sum weight(e)$ and to one using $\sum weight(n)$ and $\sum weight(e)$. Both of these scoring functions are more similar to those proposed in previous work. The weights for the various factors are determined using SVM^{rank}; hence, the alternatives may be viewed as an upper bound of the search effectiveness of existing techniques. As evidenced in the figure, SVM rank and SVM rank (minimal) always outperform the more conventional scoring functions.

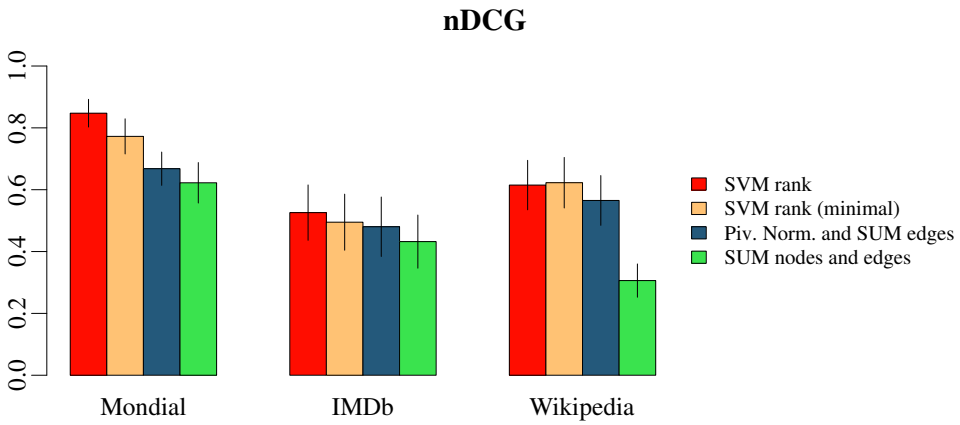


Figure 4.6: Comparison of retrieval effectiveness among SVM rank and traditional scoring functions proposed in the literature. Higher scores are better. The error bars denote the standard error of the mean.

Contrary to the established norm in this area, ranking results by their total edge weight is not an ideal ranking scheme. This conclusion overturns general intuition for these scoring functions and questions the importance of algorithms that attempt to enumerate results in the order of their total edge weight.

Justification for Minimizing Total Edge Weight

Goldman *et al.* [GSVGM98] first proposed proximity search in databases. Intuitively, users want results that are more closely related to be ranked ahead of results that are less closely related. For example, if two nodes are connected by a single edge, this relationship is stronger than one that requires multiple edges and intermediary nodes. Goldman *et al.* [GSVGM98] do not consider search effectiveness in their evaluation, but BANKS [BHN+02] provides anecdotal evidence that total edge weight is correlated with relevance. In NAGA [KSI+08] and STAR [KRS+09], Kasneci *et al.* claim that a Steiner-tree-based scoring function is correlated with relevance.

Reconciling these claims with the results from the correlation of the various scoring factors (Table 4.7 on page 65) and SVM rank's analysis is non-obvious, but three factors might account for this discrepancy. First, Kasneci *et al.* search the YAGO knowledge base [SKW07], which includes millions of facts connected by predefined relationships. Its structure is considerably different from the graphs created from a relational database, and NAGA's evaluation [KSI+08] did not include graphs created from relational data. Second, NAGA [KSI+08] does indicate that its weighting of graph edges (based on the strength of its beliefs regarding each relationship) outperforms BANKS's scoring function [BHN+02], which cannot capture the informativeness of different relationships. Hence, alternative edge weighting schemes might produce different results. BANKS significantly outperforms SVM rank on the IMDb dataset for the highest ranked results (see the P@1 and MRR results in Figure 4.2 on page 68 and Figure 4.3 on page 69), and one possible explanation is its different

method for assigning edge weights. A formal examination of these different edge weighting schemes would undoubtedly be useful for future work in this area. Third, the general intuition—results should be composed of closely related tuples—seems valid, particularly for identifying the *most relevant* result, but it may not be as good at identifying marginally relevant results, which would cause it to perform worse at higher recall levels (e.g., on MAP and nDCG). Again, a formal study of this phenomenon seems prudent for future work in this area.

4.4.3 Threats to Validity

There are two major issues that may have impacted the evaluation of SVM rank. First, the enumeration of results is not addressed by SVM rank. Many enumeration algorithms have already been proposed; it would be straightforward to combine SVM rank with any one of these algorithms similar to the way structured cover density ranking uses DISCOVER-II's naive query processing strategy. Second, all possible feature and factor combinations that have been previously proposed were not considered separately. In particular, using a different edge weighting scheme might have produced different results, particularly with regards to the importance of minimizing the total edge weight of results. Unfortunately, there is no existing work that indicates which edge weighting scheme is best.

This work uses a linear SVM to fit a linear function for scoring results. This decision follows naturally from most existing relational keyword search scoring functions, which are linear. In contrast, most scoring functions used by the IR community are non-linear and include—for example—logarithmic damping of term frequency and inverse document frequency. Although the edge weighting scheme is non-linear (it applies a logarithmic normalization of each vertex's outdegree), all the factors are combined linearly, and it is certainly possible for a non-linear combination function to provide even better results.

Although the methodology for assessing relevance follows from the methodology of the XML retrieval community, it is possible that this process partially explains the results. A result is relevant if it completely overlaps with a result in the ideal recall base, and the relevance of a result is the ratio of its relevant information to total information (e.g., when a single tuple is relevant to the query). Hence, larger results are more likely to be judged relevant than smaller results because larger results are more likely to contain relevant information. This technique may also partly explain why the sum of edge weights is not indicative of relevance—results with more nodes and edges are actually more likely to be marginally relevant. Relevance judgments in previous evaluations of keyword search techniques for databases (e.g., see SPARK [LLWZ07a, LWL+11]) differ markedly—the relevant result(s) must be minimal, a practice without precedent in traditional IR.

Applying SVM rank to the set of results enumerated by other search techniques may also account for its superior performance. Different search techniques have different semantics for considering a result to be valid—for example, some approaches allow results to contain only a subset of the search terms while others require all search terms to be present in a result. If some results judged relevant were not enumerated by a particular search technique, then enumeration and ranking are no longer orthogonal tasks and should be addressed separately in the evaluation of relational keyword search techniques. That is, a system's failure to enumerate a relevant result precludes it from ranking that result and—depending on the metric—diminishes its measured effectiveness. Using each search technique's scoring function to rank the set of results enumerated by any system would completely disassociate enumeration and ranking in an evaluation.

Chapter 5

User Queries in Relational Databases

The selection of a representative query workload is an important aspect of the evaluation of retrieval systems. Allowing researchers to construct their own queries introduces a strong potential for bias because it is easy to create queries—even unconsciously—that favor proposed algorithms and ranking schemes. This potential for bias is the reason that the evaluation benchmark presented in Chapter 3 was developed prior to the ranking schemes presented in Chapter 4. Nevertheless, it is possible that the benchmark's queries do not reflect real user queries. For example, users might express their information needs differently than the queries in the benchmark. Hence, it is necessary to understand what users search for and how users express their information needs to ensure that evaluation query workloads are as realistic as possible. Answering these questions is an important step for validating the benchmark that forms a cornerstone of this dissertation.

This chapter investigates the types of queries that users would submit to a relational keyword search system if one was available for real-world retrieval tasks. A framework for analyzing queries in the context of relational keyword search is described and compared to classification schemes characterizing web searches. A taxonomy classifies the user's intent behind a query and how the user expresses this intent. Analysis of an Internet search engine log indicates that queries submitted to a relational keyword search system will likely be focused on retrieving a single database entity, which is considerably different than the informational searches submitted to existing search engines. Relatively few queries reference more than one database entity unless the additional entity is used to disambiguate the query, but the number of entities referenced varies considerably across different datasets. Examining previous evaluations of relational keyword search systems reveals that most evaluations contain more complex search tasks than present in existing search logs. This analysis allows the creation of representative query workloads for evaluations; these synthetic workloads are much more similar to real user queries than those used previously in the literature. This work closes an

important gap in the existing evaluation methodology and promotes continued improvement to relational keyword search systems.

5.1 Analysis Framework

This section describes the framework created to analyze queries. It starts by describing the users of these systems. User goals when searching a relational database are then discussed as is how to measure the complexity of the queries that users submit.

No relational keyword search system appears to have been adopted as the primary means to search a well-known data repository. Hence, determining the types of queries that users would submit to such a system is a challenge. Fortunately, there are a large number of well-known websites that all use data that could be stored relationally.[1] As a surrogate for real user queries submitted to a relational keyword search system, this analysis uses an Internet search engine query log. This necessity actually has two benefits. First, Internet search engines receive far more traffic than most web sites; this traffic provides more flexibility to select datasets of interest. Second, it allows analyzing queries that result in click-throughs to large well-known sites (e.g., Wikipedia, Amazon, and IMDb).

5.1.1 Users

Two classes of users exist for a relational keyword search system—naive users and experts. While the first group is traditionally the target for keyword search systems, experts are also likely to use these systems once they rival the performance of structured query languages. Naive users use keyword search because they have little alternative apart from learning a database query language (viz. SQL). Alternative query interfaces such as QBE [Zlo75] and search forms [CBC+09] require some understanding of the database schema by the user, making these interfaces less accessible than keyword search. Moreover, these approaches can lead to frustration on the part of users when unexpected results are returned in response to a query [JCE+07].

In contrast, expert users know SQL. Why would someone who knows SQL choose to use a keyword search interface? First, the user may not be familiar with the relational schema. Enterprise databases contain hundreds or even thousands of interrelated tables [YPS09, Sri10], which is a significant obstacle when first encountering the database. Second, the complexity of the schema coupled with the verbosity of SQL leads to instances where an imprecise keyword query is actually more efficient for the user than a more precise SQL statement that requires considerable time and effort to write.

[1] Alternatives to relational databases for storing data are becoming popular (e.g., NoSQL databases) as many applications trade ACID compliance for performance and scalability [Lea10, Sto10]. For this high-level analysis, the underlying architecture is not critical.

A small study [SW05] of doctoral students and faculty in a database group found many cases where keyword search is more effective than SQL even for a group of knowledgeable SQL users. On average, a keyword search interface leads to users submitting fewer queries than when using SQL,[2] and the desired information was usually retrieved much faster via keyword search. This study suggests that expert users will use a keyword search interface if they believe that it will be more efficient than writing an SQL statement. Essentially, a keyword search interface is better for simple data exploration and retrieval tasks whether or not the user already knows SQL.

5.1.2 User intent

A preliminary examination of a search engine query log suggests that queries can be described by the user's objective behind making the search. Three general goals were identified:

- locate a particular database entity,
- obtain information from the database, and
- retrieve a resource stored in the database.

These goals appear to be comprehensive (i.e., all queries could be classified by one of these goals) and are similar to those identified for web search [Bro02, RL04, JBS08] where they are labeled navigational, informational, and resource.[3] Due to high-level similarities between web search and keyword search in databases, the existing verbiage is adopted to categorize queries.

Navigational The intent of a navigational query is to locate a particular database entity that the user already has in mind. These queries have exactly one correct (relevant) result. If the database has a browsable interface (e.g., it is published online), navigational searches arrive at the desired entity faster than browsing via the existing interface. For expert users, a navigational keyword query is often faster than writing the necessary SQL statement by hand [SW05].

Informational Informational queries are closest to classic IR tasks although they may be very broad. They seek to obtain information about a particular topic in response to the user's information need. These queries often take the form of question answering or "undirected" requests to learn more about a topic or desire a list of results. Exploratory search—that is, open-ended queries posed by users trying to learn more about a general topic—also falls into this category.

[2] Even though only a single SQL statement was necessary, users typically reformulated their queries to obtain the desired result(s).

[3] Broder [Bro02] and Jansen *et al.* [JBS08] term resource queries "transactional" to encompass more general web-mediated activities such as purchasing a product. Because transactions refer to a different concept in the context of databases, Rose and Levinson's terminology [RL04] is used instead.

Table 5.1: Categories of user intent and example queries (pertaining to IMDb) for each category

Category	Characteristic	Example
Navigational	people, product, company names	Indiana Jones, Sean Connery
	single result (database entity)	*The Parent Trap* 1998
Informational	answer to specific question	when is Julie Andrews's birthday
	general information	*The Lord of the Rings* Oscars
	list	Tom Hanks 2004 [movies]
Resource	images, music, or video	battle Gettysburg Devil's Den images

Resource A resource query aims to retrieve a resource, which is not information, that is stored in the database. It frequently relates to media (e.g., images, music, or video) or entertainment. In some cases, the desired resource can take the form of text (e.g., music lyrics), but in such cases, the objective is the resource and not any information it may contain.

Table 5.1 lists the major characteristics used to classify queries. Beside each characteristic is an example query. The interpretation of queries is often subjective and open to interpretation. Moreover, in many instances the query classification will differ from that of the query in the context of web search. These differences are discussed in the following paragraphs, and as shown in Section 5.2, the percentage of queries in each category differs significantly from web searches.

Navigational queries in relational keyword search particularly differ from navigational queries in the context of web search although they do share the same high-level objective—that is, to locate a specific, known object (entity vs. web site). The navigational category for relational keyword search is essentially extracted from the informational category of web searches. Many web queries (e.g., names of celebrities) lack a single, authoritative web site, but in the context of a relational database, these queries have a single, well-defined result because the database is assumed not to contain redundant information.

5.1.3 Query Expression

The way users express their intent—that is, the content of their query—is also an important component of this analysis. Some queries (e.g., navigational queries) are typically much easier to handle than others (e.g., informational queries expressed as a natural language question). If users only submit "simple" queries, then existing search techniques are likely to be good enough for all practical purposes. Measuring the complexity of a query is a step toward understanding the usefulness of various search techniques.

To be a candidate for retrieval via keyword search, the desired data must have a textual name, label, or succinct description that will be matched by the query. When all search terms are present in the same tuple, even naive search techniques (e.g., concatenating all attribute values and searching the resulting text via traditional IR techniques) will return meaningful results. The complexity of relational keyword search

Table 5.2: Classifications of query expression and example queries

Classification	Example
Entity	Tom Hanks, *Gettysburg*
Entity-weak entity	*The Hunt for Red October* quotes
Entity-relationship	*Star Wars* cast
Multi-entity	Sean Connery James Bond

increases with the number of different entities that must be considered. For example, it is generally more difficult to identify the relationships among three database tuples than it is to identify the relationships between two. Hence, the number of entities referenced in a query grossly approximates its complexity.

As a refinement for this naive approach to estimate the complexity of a query, the following scheme is used to classify queries by the number and type of entities that they reference.

Entity An entity query references a single database entity.

Entity-weak entity These queries reference an entity and a weak (dependent) entity in the database.

Entity-relationship An entity-relationship query refers to a particular entity's relationships with other entities.

Multi-entity A multi-entity query references multiple database entities.

Complexity increases moving down the list. For example, an entity and a weak (dependent) entity are very closely related so these queries should be easier to answer than finding the relationship between two arbitrary entities in the database. The most precise classification that is appropriate is chosen when labeling a query. Table 5.2 provides example queries for the classification scheme.

The classification uses an entity-relationship model instead of the logical database design (i.e., relations) due to the variations of the latter. For example, the same entity-relationship model can produce different schemas due to different methods for data normalization. Normalization separates related information to eliminate redundancy, but from the conceptual view of the user, the information is still a cohesive whole. From the user's perspective, a relational keyword search system should return an appropriate conceptual view of the information regardless of the database's schema. A general characterization also allows comparing queries across different databases (e.g., a lyrics database and IMDb).

Classification of the user's intent when searching is orthogonal to classification of the query's expression. The latter investigates how the former is expressed by the user. There is some relationship between the two classifications, for a query that exactly matches a single entity is classified as navigational. Nevertheless, the converse does not necessarily hold—not all navigational queries are also single entity queries. For example, the query "Disney's *The Little Mermaid*" is a multi-entity query because it refers to both a company and a

film, but it is clear that the user has a single result in mind. In this case, referencing more than one entity improves the precision of results.

5.2 Query Analysis

This section describes the analysis of a search engine query log. Workloads from several evaluations of relational keyword search techniques are also examined to illustrate the differences between those workloads and the queries that users currently submit to search engines. Finally, how to generate synthetic query workloads for evaluating relational keyword search systems is also described.

5.2.1 Search Engine Logs

Because relational keyword search systems have yet to be deployed for real-world retrieval tasks, the analysis uses an AOL query log. The AOL query log contains approximately 20 million queries submitted by 650 thousand users over three months during 2006. Each entry in the log contains an anonymized user id, query (case-shifted with most punctuation removed), the time when the query was submitted, and if the user clicks on a result, the rank of the result and the domain portion of the URL for the link. The analysis for this work focuses on the query and click URL. Pass *et al.* [PCT06] provide more information about the dataset and its characteristics.

The analysis of the query log focused on two overarching research questions. First, what is the potential applicability of relational keyword search? Although relational keyword search has been the focus of much academic research, no effort appears to have been made to determine the market penetration that these systems might have. For example, if a system was deployed tomorrow and was as good as existing search techniques, how many sites would use it? Estimating how many queries might require relational keyword search techniques would also be valuable. Second, how do queries executed against a single database compare to web queries? For example, are they shorter or longer? Does the classification of user intent differ depending on context, and how is the intent behind each query expressed?

What is the applicability of relational keyword search? To answer this question, the URLs in the query log were ranked by their number of click-throughs. Less than 1% of domains account for half of all user click-throughs in search results [PCT06]. Exactly 80 websites all had in excess of 10,000 click-throughs. These websites were examined to identify those that potentially have relational database backends.[4] Each site's actual system architecture was not determined but rather if the site's data and typical use cases suggested

[4]This arbitrary cutoff was chosen to avoid manual analysis of the 1.6 million sites that receive fewer than 10,000 click-throughs in the query log.

Table 5.3: Query statistics of the original search log and datasets

	$\lvert Q\rvert$	$[\![q]\!]$	$\overline{[\![q]\!]}$	σ
AOL	20061708	1–245	2.51	1.81
Wikipedia	120112	1–95	2.91	1.94
Amazon	95435	1–50	3.55	1.95
IMDb	99234	1–96	2.75	1.55
lyrics	40499	1–111	3.81	2.52
eBay	4221	1–6	1.45	0.62

Legend
$\lvert Q\rvert$ total number of queries
$[\![q]\!]$ range in number of query terms
$\overline{[\![q]\!]}$ mean number of terms per query
σ standard deviation of number of terms per query

a relational database would be suitable. For example, DBLP stores information in flat text files, but this website would be classified as amenable to using a relational database. Of the original 80 websites, roughly half could use a relational database backend. The most frequent query genres identified in these websites were shopping, entertainment, and travel. The remaining websites were typically web portals (e.g., Yahoo!), "communities" [Kle99] of related content (e.g., Myspace and GeoCities), or web applications (e.g., MapQuest and Hotmail).

Five datasets were created from the URLs that received the greatest number of click-throughs. When classifying queries using the taxonomy presented in the previous section, queries that are empty (i.e., "") or are navigating to a particular website (e.g., "ebay") are not considered. However, when classifying queries in the context of web search (to compare against relational keyword search queries), navigation queries that target a particular web site are included. Table 5.3 lists the datasets.

Comparison of Web and Relational Queries

A simple metric that is commonly used in the literature when analyzing search engine query logs is query length. Table 5.3 contains summary statistics for query length. As evidenced by the table, the query statistics for the five datasets are similar to those for the entire query log. The results (both for the datasets and the query log) agree with previous studies [JSBS98, WSJS01, SJWS02] that found web queries to be very short, containing 2–3 terms on average.

With the exception of eBay, the mean number of terms per query is slightly higher for the five datasets than for the query log. One possible explanation for this trend is that queries resulting in click-throughs to these sites are more complex—that is, they reference more entities or attributes than typical web queries [LYMC06].

Table 5.4: User intent (% of total) in the context of web search (left) and relational keyword search (right).

	Web			RDBMS		
Dataset	Navigational	Informational	Resource	Navigational	Informational	Resource
Wikipedia	4.8	95.2	2.4	59.2	38.8	2.0
Amazon	13.2	74.0	12.8	23.2	63.6	9.2
IMDb	6.4	85.6	8.0	77.2	12.4	7.2
lyrics	14.8	12.8	72.4	11.6	7.2	81.2

While the difference is minor (and probably does not support this explanation) for Wikipedia and IMDb, it is plausible for Amazon and lyrics queries, which tend to contain an additional term per query.

The eBay dataset is an outlier. Although eBay received a large number of click-throughs ($> 77,000$) in the original query log, more than 90% of these queries are simply trying to reach the site itself. Discarding all the queries navigating to eBay's homepage reduces the number of eBay queries to just a few thousand so it is not considered further in the analysis.

What is the user intent behind the queries? For each dataset, two sets of 250 queries were manually classified according to user intent. These queries were sampled without replacement from the complete set of queries in each dataset. One set was classified in the context of web search (using the Jansen *et al.*'s characteristics [JBS08]); the other was classified in the context of relational keyword search using the taxonomy presented in Section 5.1.2. The same set of queries was not classified because the set classified in the context of web search include queries navigating to the website (e.g., "ebay") so results are comparable across the two different taxonomies.

Table 5.4 contains the results of the analysis and shows that query interpretation varies widely depending on context. With the exception of the lyrics dataset, the percentage of navigational queries increases significantly for relational keyword search. This change is most visible for Wikipedia and IMDb where the majority of queries transition from informational to navigational. Unlike navigational and informational queries, the percentage of resource queries is relatively unaffected by context.

Across the different datasets, considerable variation exists in user intent. Wikipedia and IMDb queries are the most similar. In the context of web search, the vast majority of queries resulting in click-throughs to these sites are informational. These informational queries (e.g., a person's name) become navigational when only considering a single data source because the database has one authoritative result for these queries. Amazon queries are not dissimilar, but the percentage of informational queries remains high due to the vague product

Table 5.5: Contingency table (% of total) of user intent and query expression for IMDb. The unclassified 3.2% of queries did not pertain to IMDb content.

	Navigational	Informational	Resource	Total
Entity	75.2	6.0	4.8	86.0
Entity-weak entity	0.4	3.2	0.8	4.4
Entity-relationship				
Multi-entity	1.6	3.2	1.6	6.4
Total	77.2	12.4	7.2	96.8

descriptions that comprise many queries. In fact, these descriptions are often just shopping categories—for example, "home organization."

The lyrics dataset provides a counter-example to the others, for the majority of its queries are always classified as resource. The classification consistency is explainable by a single keyword that appears in more than 80% of these queries—"lyrics." Including this keyword identifies the user's objective to retrieve a resource.

How are queries expressed in the context of a single data source? Although the user's intent can be determined fairly easily, investigating a query's expression in the context of relational keyword search requires a data model. A relatively simple model is evident for IMDb data (person, film, character, and production company entities) and lyrics data (artist, song, album, and company entities) so these two datasets are the focus of this part of the analysis. Because the results thus far indicate that IMDb and lyrics queries are at two extremes, the results for Amazon and Wikipedia queries would likely fall somewhere in between the results for these two datasets.

For the IMDb and lyrics datasets, 250 queries were manually classified by query expression. These queries are the same ones used when classifying user intent so the relationship between user intent and query expression can also be investigated.

Tables 5.5 and 5.6 are contingency tables for the IMDb and lyrics datasets. These tables show how queries are typically expressed for each user objective. Most IMDb queries (Table 5.5) are navigational, and the queries reference a single database entity. For example, the user's query might be "Harrison Ford" to locate the desired entity in the database. Queries referencing weak (dependent) entities, relationships, or more than one entity are much more rare. These results differ significantly from lyrics queries (Table 5.6). A slight majority of these queries reference more than one database entity. The difference is likely due to the content of each dataset. Because many song titles are not unique, providing the artist name in addition to the song

Table 5.6: Contingency table (% of total) of user intent and query expression for lyrics queries.

	Navigational	Informational	Resource	Total
Entity	11.2	3.2	30.0	44.4
Entity-weak entity				
Entity-relationship				
Multi-entity	0.4	4.0	51.2	55.6
Total	11.6	7.2	81.2	100.0

title improves the precision of results. Hence, many times users are not searching for just song lyrics; instead, they are searching for the lyrics of a particular artist's rendition of a song. This situation also arises when searching for films whose titles are not unique (e.g., remakes). Nevertheless, the most common means to distinguish these films is by their year of release instead of specifying a person who stars in the desired film. For example, users specify "*The Parent Trap* 1998" (which is a single-entity query) instead of "*The Parent Trap* Dennis Quaid" (which is a multi-entity query).

Even though the manually classified queries did not include any entity relational queries, this category is still useful. For example, as many as 2% of the IMDb queries include the term "cast," which would be interpreted as an entity-relationship query. Hence, dropping this category would make the taxonomy incomplete.

Summary

The original research questions are reviewed to summarize the query log analysis. First, relational keyword search techniques are applicable as evidenced by the proportion of data-driven websites that could use them to search their underlying data repositories. Second, web search queries and relational keyword search queries differ. In particular, relational keyword search queries tend to be more targeted (due to the collection being searched) than web search queries as indicated by the increase in the number of navigational queries. In relational keyword search, many queries can be expressed by referencing a single database entity (e.g., IMDb queries) although including additional entities may be necessary to resolve ambiguity (e.g., lyrics queries).

5.2.2 Relational Keyword Search

As stated previously, the trend when evaluating relational keyword search systems is for researchers to construct their own query workloads (see Chapter 2 and Webber [Web10]). With the exception of Liu *et al.*'s evaluation [LYMC06] that used queries sampled from a search engine query log, there is little evidence

Table 5.7: Query lengths in previous evaluations contrasted with statistics from the queries analyzed for this study (*italicized datasets*). The last (non-italicized) IMDb dataset is from the benchmark presented in Chapter 3.

Dataset	$\lvert Q\rvert$	$[\![q]\!]$	$\overline{[\![q]\!]}$
lyrics [LYMC06]	50	2–20	6.7
lyrics	250	1–18	4.3
IMDb [HWYY07]	40	2–8	5.0
IMDb [LLWZ07a]	22	2–3	2.4
IMDb [DKS08]	4	2–3	2.5
IMDb [QYC09]	20	3–5	3.3
IMDb [CW10a]	50	1–26	3.9
IMDb	242	1–10	2.8

Legend
$\lvert Q\rvert$ total number of queries
$[\![q]\!]$ range in number of query terms
$\overline{[\![q]\!]}$ mean number of terms per query

that these query workloads are representative of the types of queries users would submit to these datasets. Hence, the previous analysis is repeated using query workloads that appear in the literature to investigate their representativeness of real user queries.

Table 5.7 lists the statistics for query workloads that have appeared in previous evaluations. One interesting trend is that the mean number of terms per query is almost universally greater in these evaluations than the queries that users actually submit to search engines (see Table 5.3 on page 83). This trend could have two important repercussions on the validity of results derived from these workloads. First, additional terms suggest that queries are better specified, which should make it easier for these systems to identify the relevant results. Under-specified queries, which are extremely common [NJ09], lead to ambiguity. Such ambiguity complicates the task of ranking results in order of their relevance to the user's information need. Second, more terms might have an impact on the performance of these systems. When more terms are specified, there are more ways to relate the terms. If the additional terms allow a system to hone in on the desired result(s) more easily, performance will improve; alternatively, the additional terms might increase the search space and cause the system's performance to degrade. Existing work has not adequately investigated these possibilities.

Even though the IMDb queries from the benchmark presented in Chapter 3 have a much larger mean than the queries from the search log, the difference is due to five queries that include a quote from a movie. When these five queries are omitted, the maximum number of terms fall to 7 and the mean number of terms drops to 2.91, both of which are consistent with the queries from the search log.

Table 5.8 contains the results for the analysis of user intent. Liu *et al.*'s queries [LYMC06] are most similar to actual user queries, which is expected because their queries are also sampled from a search engine

Table 5.8: User intent (% of total) in previous evaluations contrasted with this analysis (*italicized datasets*, copied from Tables 5.5 and 5.6). The last (non-italicized) IMDb dataset is from the benchmark presented in Chapter 3.

Dataset	Navigational	Informational	Resource	Total
lyrics [LYMC06]	2	2	90	94.0
lyrics	12.4	7.2	80.4	100.0
IMDb [LLWZ07a]	23	77		100.0
IMDb [DKS08]	25	75		100.0
IMDb [CW10a]	40	60		100.0
IMDb	86.0	4.4	6.4	96.8

log. For the IMDb workloads appearing in the literature, there is a heavy skew toward informational queries. This skew represents one of the dangers of having researchers design their own evaluation workloads. In this case, existing evaluations showcase the abilities of relational keyword search, but these types of queries are not the kind that users are most likely to submit to a search engine. Users often search for a single film or individual within IMDb and will expect any new search techniques to be at least as good as existing systems. By focusing on more complex queries, previous evaluations have not ensured that relational keyword search is making progress toward replacing existing systems.

The expression of a user's intent in queries from previous evaluations (Table 5.9) shows a similar bias. The majority of queries reference multiple entities. Far fewer focus on a single entity even though these queries are the most likely to be submitted by users. The IMDb queries from the benchmark presented in Chapter 3 actually strike a good balance between real user queries and the queries that are typically used to

Table 5.9: Query expression (% of total) in previous evaluations contrasted with this analysis (*italicized datasets*, copied from Tables 5.5 and 5.6). The last (non-italicized) IMDb dataset is from the benchmark presented in Chapter 3.

Dataset	Entity	Entity-weak entity	Entity-relationship	Multi-entity	Total
lyrics [LYMC06]	34		60		94
lyrics	44.4		55.6		100
IMDb [LLWZ07a]	14	9	77		100
IMDb [DKS08]	25		75		100
IMDb [CW10a]	40	10	50		100
IMDb	86.4	4.4	6.4		97

evaluate relational keyword search techniques. Unlike other query workloads, the benchmark's queries are not heavily skewed toward any extreme (navigational vs. informational or single-entity vs. multi-entity).

5.2.3 Synthetic Generation

Sampling queries from search engine logs is an excellent way to ensure a query workload is representative. However, there are two cases that defy this approach. First, some popular datasets do not receive many click-throughs in search engine logs. DBLP is one example, for its specialized content makes it unlikely to receive much traffic from those outside the computer science community. Other datasets which are compilations of other material (e.g., MONDIAL [May99]) will also lack entries in Internet search engine logs. For these datasets, an analysis of user intent and query expression from similar datasets can drive the construction of a query workload. For example, DBLP's content is generally similar to IMDb's (both contain information regarding professional associations among people) so users would be likely to submit similar types of queries to each database. Second, many existing evaluations of relational keyword search systems use subsets of a large database because even state-of-the-art techniques have trouble with databases containing more than 100,000 tuples as evidenced by the empirical evaluation in Chapter 3. When an original database is truncated, some queries that appear in the search engine log may no longer have relevant results in the smaller database. Synthetically generating a query workload avoids this problem by using queries that are similar to those executed against the entire dataset.

Query templates [NJ09] provide a straightforward method for creating representative query workloads whose characteristics are similar to those observed in a search engine log. Query templates are abstractions of the queries that users actually submit to a search engine. For example, a query template might be "⟨Person.name⟩;" the actual name is specified when the user performs a search. Another way to view a query template is as a search form that the user fills in when querying the database.

Analysis using query templates proceeds backward from the search engine query log. Queries in the log are converted to query templates by replacing substrings in the query with their description from an underlying data model. As examples, consider the queries "Harrison Ford" and "Tom Hanks movies." The first will be converted to the query template "⟨Person.name⟩" where ⟨Person.name⟩ refers to the Person entity and its name attribute, and the second will become the query template "⟨Person.name⟩ movies." Query templates simplify the analysis of the original search engine query log.

Table 5.10 lists the five most common query templates for the IMDb and lyrics queries that were manually classified. To create a query workload from the query templates, concrete terms are substituted in place of

Table 5.10: Most frequent query templates appearing in search log

Dataset	Template	%
IMDb	⟨Person.name⟩	40.8
	⟨Film.title⟩	30.0
	⟨Film.title⟩ ⟨Film.type⟩	2.8
	⟨Film.title⟩ ⟨Film.year⟩	1.6
	⟨Character.name⟩	1.2
lyrics	⟨Artist.name⟩ lyrics	23.6
	⟨Song.title⟩ lyrics	19.2
	⟨Artist.name⟩ ⟨Song.title⟩ lyrics	7.2
	⟨Song.title⟩	5.6
	⟨Artist.name⟩	5.6

the types. For example, a tuple is selected from the Person table, and the value of its name attribute is substituted for ⟨Person.name⟩.

Real user queries are unlikely to be uniformly distributed over all database entities. For example, popular actors and actresses will be the target of more searches than their stunt doubles. Query templates allow this bias to be reflected in the workload. For example, one can approximate the popularity of individuals by the number of films in which they appear, and when selecting a person's name for substitution in the query template, the likelihood of choosing people is weighted by their popularity. This approach allows generated query workloads to be more representative of actual user queries and—unlike sampling queries from the original search log—allows workloads to be easily adapted to reflect users' search trends.

5.2.4 Threats to Validity

Performing this analysis using the query log from an Internet search engine will likely reflect some of the limitations of existing search engines—namely, web search engines are not designed to connect disparate pieces of information. This limitation may partially account for the relative lack of multi-entity queries from the classifications, but many websites contain pages that can answer queries with closely-related entities. For example, IMDb cast pages can be retrieved when two characters from the same film appear in the user's query. Hence, the limitation of existing search techniques does not imply that users will cease submitting navigational queries when more sophisticated search techniques are available. Some queries that pertain to the datasets used in this analysis may have been omitted due to the use of click-throughs to define the datasets. For example, if a user searches for "Harrison Ford," the user must click on an IMDb link to be included it in the IMDb dataset. This issue is not critical if one assumes that users click on links that are relevant; missing a few queries that did not result in click-throughs is not significant given the large number of captured queries.

Lack of data redundancy and under-specified keyword queries result in a bias away from informational queries and toward navigational queries when classifying queries from a search engine log. Consider the question "When is Sean Connery's birthday?" A user might reason that search results about people are likely to include their birthdays and use the query "Sean Connery" to express this information need. However, this informational query is identical to a precisely specified navigational query requesting the person entity whose name is "Sean Connery." Hence, there is a bias to label under-specified queries as navigational. Unfortunately, the percentage of under-specified queries cannot be determined precisely without an overt description of the user's information need, which is not available from the search engine log (and rarely obtained except in controlled studies).

Having a single annotator for all the queries may have influenced these results. As part of future work, it would be ideal if multiple annotators classified each query; high inner-annotator agreement would indicate the applicability of this classification scheme. Having a single annotator does increase the consistency of the classifications—if there is a systemic bias, it will be consistent across all the results.

One site that is conspicuously absent from this analysis is Facebook, which was the most visited website in 2010 [Law10b]. Social networking sites are ideal candidates for relational keyword search systems. Facebook is not included in the datasets because the AOL query log is from 2006, prior to Facebook's explosion in popularity. Hence, relatively few queries ($< 8{,}000$) resulted in click-throughs to this popular service, and more than 80% of these queries were navigating to the site itself.

5.3 Experiments

This section experimentally validates synthetic query generation and shows that the characteristics of synthetic workloads closely resemble those of the original search engine query log. These experiments demonstrate the usefulness of this work for the construction of query workloads.

Synthetic Workloads

A synthetically-generated query workload should have characteristics similar to the queries appearing in a search engine log. Using query templates to create the workload ensures that the user intent and query expression are consistent with the results of the analysis described in Section 5.2. In addition, the query statistics and the relevant collection statistics of the synthetically-generated workloads are much more consistent with real user queries than the workloads used previously in the literature.

Four workloads with different numbers of queries were synthetically generated. These queries are based on eight different query templates from IMDb. From the previous analysis, these query templates account

Table 5.11: Query statistics for IMDb workloads generated synthetically

Source	$\|Q\|$	$[\![q]\!]$	$\overline{[\![q]\!]}$	σ
Synthetic	80	1–11	2.73	1.76
	160	1–9	2.76	1.41
	400	1–11	2.73	1.42
	800	1–10	2.62	1.30
Log	99234	1–96	2.75	1.55

Legend

$|Q|$ total number of queries
$[\![q]\!]$ range in number of query terms
$\overline{[\![q]\!]}$ mean number of terms per query
σ standard deviation of the number of terms per query

for just under 80% of IMDb queries in the search log. Table 5.11 provides the query statistics for the four workloads and from the search engine log (the last row is copied from Table 5.3 on page 83). As evidenced by the table, the basic query statistics for the synthetic workloads are very similar to those of the complete search engine log. More importantly, they are much more consistent than those used in previous evaluations (compare to Table 5.7 on page 87).

Another approximation for the complexity of a query is its initial search space (i.e., the number of database tuples that contain at least one search term) [HWYY07]. Performance is generally expected to decrease as more tuples contain search terms because more relationships must be considered to relate the information. Figure 5.1 depicts the initial search space for queries in the original search engine log, previous evaluations, and in the synthetic workloads. The initial search space is estimated by summing the number of tuples that contain each search term. Although some tuples will contain multiple search terms, this number is relatively small. Using the queries from the search engine log whose terms appear in fewer than 2 million tuples, the approximation error is just 2.63% using the sum of the frequencies, which is considerably smaller than the error when using the maximum collection frequency of a search term (14.5%) or the mean collection frequency of a search term (51.94%).

As shown in Figure 5.1, the initial search space for queries using the synthetic workloads is much more similar to the initial search space of queries from the search engine log than that achieved by previous evaluations. Previous evaluations fail to capture the range in the collection frequencies for each query whereas the synthetic workloads created using query templates are similar to the search log. In addition, the median number of tuples containing search terms is rarely comparable to the median for real user queries. The median for BLINKS's evaluation is approximately two orders of magnitude smaller, and the median for the queries used in the benchmark presented in Chapter 3 is more than an order of magnitude larger. For the benchmark's queries, the greater collection frequency is attributable to some queries containing "schema"

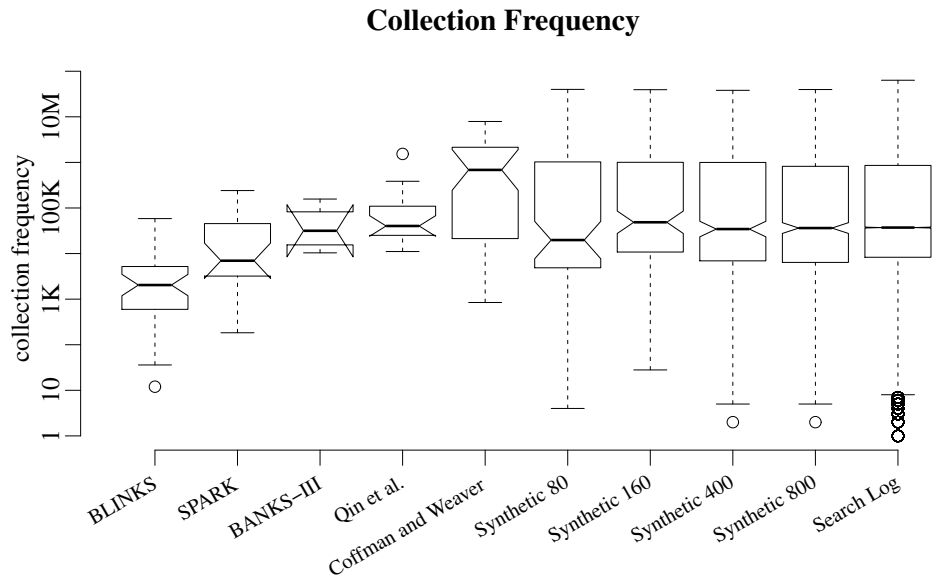


Figure 5.1: Box plots of the total number of tuples containing search terms for each query. Note that the y-axis has a log scale. Coffman and Weaver refers to the IMDb queries from the benchmark presented in Chapter 3.

terms that match every tuple in a relation (e.g., the term "character" matches every character in the database). The higher median collection frequency might indicate that the benchmark's queries are more difficult than real user queries and thus account for the poor performance of the various search techniques in the evaluation in Chapter 3. However, it is also important to remember that the evaluation used a subset of IMDb whereas Figure 5.1 considers the entire database. Within the database subset, the median collection frequency of query terms is much smaller so the impact on performance should be minimal. Using query templates to generate queries is much more robust than having researchers create representative query workloads by hand.

Chapter 6

Related Work

This chapter reviews other research related to the work in this dissertation. A complete description of relational keyword search techniques is not provided here but is available in Appendix B. Each section in this chapter corresponds to a chapter in this dissertation.

6.1 DB&IR

Whereas Chapter 1 focuses on how different applications for databases and retrieval systems led to their differences, Weikum [Wei07] goes further in stating that the differences between the database and IR communities stem from different notions of users. For database systems, the user is an application programmer whose queries are precise, known in advance, and should be answered exactly and as quickly as possible. In comparison, IR systems see users as non-technical, with fixed cognitive facilities (i.e., too much information is as unhelpful as too little), and whose queries are approximate representations of an underlying information need, often requiring refinement (and even guidance) to obtain the desired results.

Attempts have been made to reconcile the theory underlying databases and IR. Fuhr and Rölleke [FR97] propose a probabilistic relational algebra that models both imprecision and probabilistic models from IR. Chaudhuri *et al.* [CRW05] outline the core functionality required of a DB&IR system, discuss different architectures to create such a system, and propose their own extension to the existing relational algebra used in databases. These approaches to retool the theory underlying relational databases have not met with success, perhaps due to their timing or the difficulty in constructing a new database management system (DBMS) that can compete with those that have seen decades of active development. An immediate drawback to this work that stifles its adoption is that it cannot be applied to *existing* data repositories. In contrast, keyword search in relational databases can be applied to the databases currently in use, and the techniques

described imbue relational databases with the ability to retrieve and rank results in an IR fashion. XML IR also faces many of the same challenges as keyword search in relational databases but is not explored in this work.

6.2 Relational Keyword Search

Chen *et al.* [CWLL09] and Chaudhuri and Das [CD09] both presented tutorials on keyword search in databases. Yu *et al.* [YQC10] provides an excellent survey of relational keyword search techniques. Chu *et al.* [CBC+09] propose matching keyword queries against a set of search forms created for a database. Forms relevant to the query are returned to the user who must populate the correct search form with the data and submit it to retrieve the desired information. Baid *et al.* [BRL+10] suggest terminating the search after a predetermined period of time and allowing the user to guide further exploration of the search space via forms. Neither of these proposed search techniques are included in the evaluations in this dissertation because both require an intermediate step to obtain search results. Furthermore, it is not clear how to evaluate a system that returns a set of forms and requires the user to select the one(s) relevant to the query.[1]

Webber [Web10] reviews effectiveness evaluations of keyword search in relational databases. Most of the issues raised are similar to those highlighted by the survey in Chapter 2. Webber states that improvement in relational keyword search will likely be the result of improvements in evaluation methodology and suggests that "individual research groups [...] create more thorough, credibly independent, and re-usable test collections" [Web10]. The benchmark described in Chapter 3 satisfies this call.

6.3 Evaluation of Relational Keyword Search Techniques

Science reports that 56% of researchers rarely use datasets from the published literature in original research papers [Sci11]. For the evaluation of keyword search in relational databases, this situation was an unfortunate reality because no evaluation workloads were publicly available. The evaluation benchmark presented in Chapter 3 is the first of its kind for evaluating keyword search in relational databases. In comparison, previous evaluations were ad hoc and lack any standardization. Objective empirical evaluation is at the heart of IR [Sin01], and the evaluation in Chapter 3 is patterned after the success of established evaluation forums such as TREC and INEX. The empirical evaluation supersedes many that have appeared to date in the literature by comparing both runtime performance and search effectiveness across many more search techniques than previous evaluations (see also Chapter 2). This work is also the first to confirm claims

[1] No standard IR effectiveness metric immediately applies to this scenario.

regarding the unpredictable performance of existing search techniques [BRL+10]. Making the datasets, queries, and relevance judgments available to other researchers gives the community a clear bar with which to measure future progress.

6.4 Ranking Schemes

Relatively little work focuses on just ranking relational keyword search results. Most research papers propose schemes to improve both runtime performance and search effectiveness rather than focusing on either enumeration or ranking. The evaluation presented in Chapter 4 indicates that structured cover density ranking has much more reliable performance than existing ranking schemes, and SVM rank outperforms all previous schemes that have been described in the literature. One unique aspect of structured cover density ranking is its independence from collection statistics. Indeed, the algorithm itself is embarrassingly parallel due to this feature. Although parallel algorithms have not been explored to date for keyword search in relational databases, they are likely to become more critical as researchers attempt to overcome existing performance problems.

Using machine learning to weight different factors when scoring search results has not been previously investigated in the context of relational keyword search. Machine learning in IR is a well-studied topic. Sebastiani [Seb02] provides an overview of automated text categorization. Joachims [Joa02] applies ordinal regression to clickstream data and shows that the learned function outperforms existing search engines. Burges *et al.* [BSR+05] and Richardson *et al.* [RPB06] also apply learning to rank to web search. Cao *et al.* [CXL+06] address how to adapt ranking SVMs to document retrieval. Geng *et al.* [GLQL07] consider the problem of feature selection and propose an approach based on the importance and similarity of the various features. Joachims [Joa02] and Richardson *et al.*'s [RPB06] work is most related to SVM rank, but the context and the features both differ. Incorporating Cao *et al.*'s work [CXL+06] on ranking SVMs would have undoubtedly improved the search effectiveness of SVM rank for precision @ k and MRR, the instances where SVM rank was only comparable to existing systems.

6.5 Query Analysis

Broder's taxonomy for web search [Bro02] is one of the earliest investigations of user intent in web searches. User intent refers to an "affective, cognitive, or situational goal as expressed in an interaction with a web search engine" [JBS08]. Broder classifies queries as navigational, informational, and transactional. Rose and Levinson [RL04] later reexamined Broder's work and replaced Broder's notion of "transactional" queries with

Table 6.1: Nandi and Jagadish's classification of query expression. 40% of the queries remain unclassified.

Category	%	Example
Entity	36	name of actor, movie title
Entity-attribute	20	Terminator cast
Multi-entity	2	Angelina Jolie Tomb Raider
Aggregate	2	highest box office revenue

"resource" queries where the user's goal is merely to obtain something. For the informational and resource goals, Rose and Levinson also identify sub-goals for greater classification granularity. In a later study of the user intent of web searches, Jansen *et al.* [JBS08] developed an automatic classification scheme based on query characteristics (e.g., query length, specific terms, and general content).

The major factor distinguishing the work in Chapter 5 from this previous research is the application to keyword search within relational databases. There is not a bijection between the classification of queries in the context of web search and in the context of relational keyword search. Furthermore, analysis shows that the frequency of each class of queries differs considerably by context (see Table 5.4 on page 84). The classification of web queries (also Table 5.4) agrees with the previous results, but the percentage of queries in each category differs substantially depending on the dataset.

Nandi and Jagadish [NJ09] investigated query templates for queries resulting in click-throughs to IMDb. They created four categories for classifying queries: single entity queries, entity-attribute queries, multi-entity queries, and queries that involved aggregate functions. Approximately 100,000 queries are classified via an automated technique, and the results are shown in Table 6.1. Unfortunately, Nandi and Jagadish's work only considered a single dataset (IMDb) and only 60% of the set of queries was classified. The work presented in Chapter 5 extends Nandi and Jagadish's investigation by considering additional datasets. Results for IMDb queries are consistent with Nandi and Jagadish's findings—namely, the majority of queries reference only a single entity and only a small percentage of queries reference multiple entities.

More generally, Paşca *et al.* investigated extracting entities and attributes from web documents [PLB+06] and query logs [Paş07]. Although the latter work in particular could be applied for the analysis of query expression, the key difference is that the underlying relational database (in relational keyword search) already defines the entities and attributes of interest—i.e., they need not be extracted. Yin and Shah [YS10] propose an approach to build a taxonomy of user intents from "intent phrases;" generic intents for a particular class of entities are similar to query templates. Applying their approach would determine which generic intents (query templates) have similar meanings but is orthogonal to the major goal of the analysis—namely, how to synthetically generate a realistic query workload.

Chapter 7

Future Work

Baid *et al.* succinctly summarize the research challenges facing keyword search in relational databases: "building such systems requires addressing many issues, including robustness, accuracy, reliability, and privacy" [BRL+10]. They go on to lament the performance issues that plague existing search techniques. It is difficult to move on to the aforementioned research challenges without first addressing existing problems.

This chapter explores three avenues for future work. The first builds incrementally on the research presented in this dissertation. These extensions address more recent research that re-examines assumptions that underly the evaluation of relational keyword search techniques. The second area for future work is unquestionably the most critical. Existing keyword search techniques for databases are hobbled by their poor and oftentimes unpredictable performance. Previous work in the computer science community has addressed some of the fundamental issues that must be overcome, but this previous work has not been integrated by other researchers. The third avenue is orthogonal to the work presented in this dissertation. It addresses how to visualize and browse through relational keyword search results; visualization will likely be a central component of these systems when they are deployed in real-world settings.

7.1 Evaluation

Despite the extensive work that has already been completed regarding the evaluation of keyword search in databases, additional work remains. Creating the benchmark described in Chapter 3 required making some simplifying assumptions where—at that time—there were open research questions. These research questions are summarized here.

- What are the characteristics of a representative query workload? Can representative workloads be generated synthetically in instances where a query log is not available or when only a portion of a large dataset is available?
- Given the scalability problems that plague existing systems, how should a database subset be constructed? That is, do database subsets accurately reflect the runtime performance and search effectiveness of the original database?

The investigation of query workloads in Chapter 5 indicates that user queries are typically much simpler than the queries used to evaluate proposed search techniques. Nevertheless, the work in Chapter 5 was only able to investigate a small number of databases and a small number of queries for each database. The existing manual classification should be augmented by automatic classifications of all the queries for the datasets. An automated classification scheme would validate the existing results and allow additional datasets to be considered. In addition, the evaluation benchmark should be updated to take advantage of the work on how to generate synthetic query workloads from query templates. A much larger set of queries could be distributed with the benchmark. It would also be advantageous to create "tracks" that evaluate different aspects of search techniques. For example, the benchmark could distinguish single-entity queries from multi-entity queries, and search techniques could receive separate scores for each type of query.

7.2 Runtime Performance

Addressing the runtime performance issues of existing search techniques is the most critical aspect of future work in this field. This section considers two issues: memory utilization and execution time.

7.2.1 Memory

The memory required by many implementations of existing search techniques is simply unacceptable. Table 3.8 on page 45 shows that even the most memory-efficient implementations of graph-based approaches require nearly 1 GB to store the data graph for the benchmark's IMDb subset. This subset is conservatively 25 times smaller than the original database (see Table 3.1 on page 29), which means that these search techniques would need no less than 24 GB of memory for their representation of the original database. The IMDb database contains tens of millions of vertices and more than one hundred million edges, but its size is small compared to the potential application of these search techniques in social networks, electronic medical records, and government databases. The graph implementation used by the search techniques in this evaluation is JGraphT, which was the most memory-efficient Java graph library found during implementation. Nevertheless,

there are several techniques that may be able to improve memory efficiency. Researchers should carefully investigate these possibilities to determine their feasibility and trade-offs.

Kacholia *et al.* [KPC+05] state that their implementation of BANKS-II requires only $16 \cdot |V| + 8 \cdot |E|$ bytes of memory, which means that graphs with tens of millions of vertices and edges can fit within memory.[1] While this array-based implementation is certainly more efficient, it does have its own drawbacks. First, it requires additional data structures (these need not reside in main memory) that map database tuples to unique ids. More importantly, these ids must be dense—that is, memory consumption actually depends upon the largest vertex id. Second, requiring the set of vertex ids to be dense makes updating this representation difficult. In particular, inserting new vertices (due to the insertion of tuples into the database) requires allocating additional space and then copying the original graph. In an update-intensive database, keeping the graph consistent with the underlying data is likely to be prohibitively expensive, but no existing work has investigated this cost. A graph implementation that does away with Java's dynamically-sized collections is unlikely the answer to existing memory problems: the advantages of lower memory consumption will be offset by decreased performance. Nevertheless, more research in necessary to determine whether or not the advantages of keeping the graph within main memory outweigh these disadvantages.

Memory consumption can also be dramatically reduced via compression. The Web Graph Framework[2] is designed to manipulate large web graphs. This framework can compress web graphs with millions of vertices and a billion edges in as little as 26.57 bits / vertex and 3.08 bits / edge [BV04]. While the compression ratios are compelling, the current implementation does have downsides. First, it is optimized specifically for web graphs and does not support all typical graph operations (e.g., efficiently finding all neighbors of a given vertex) that many existing graph-based search techniques require. Second, the compression does not allow modification to the graph—merely efficient access. Hence, the existing framework is not immediately suitable to keyword search in databases, but it would be interesting to investigate different search techniques that can take advantage of this framework.

Dalvi *et al.* [DKS08] propose a multi-granular graph representation and suggest that search heuristics should be sensitive to which portions of the graph are cached within memory. Despite raising the level of complexity when implementing search techniques, this idea is probably the most practical. Multi-granular graphs allow the search to concentrate in closely-related clusters, and given the simplicity of most user queries, it is unlikely that the vast majority of searches will even expand outside of a few clusters. Moreover, these clusters could be partitioned among multiple machines to both reduce the memory footprint and

[1] To provide a concrete example of the difference between an array-based graph implementation and one using Java's dynamically-sized collections, JGraphT consumes at least $32 \cdot |V| + 56 \cdot |E|$ bytes of memory. The difference in memory consumption for the entire IMDb data graph is 2.5 GB verses no less than 13.5 GB.

[2] http://webgraph.dsi.unimi.it/

enable parallel processing within different clusters. Nevertheless, Dalvi *et al.* only implement the backward search heuristic for their multi-granular graph representation, which suggests that exploiting this graph representation is non-trivial. Ideally the graph implementation itself could be made more memory efficient by loading only those portions of the graph necessary and caching little used portions on disk. This approach would achieve the major advantages of the multi-granular graph representation without forcing developers to consciously control for it. Making existing graph implementations memory-conscious would go a long way to making existing memory issues disappear.

7.2.2 Execution Time

The empirical results presented in Chapter 3 emphasize the necessity of improving the runtime efficiency of search techniques. In general, schema-based approaches have the best performance. They falter only as the maximum size of results increases because the number of candidate networks grows exponentially with the size of results. Researchers have investigated and presented solutions to the chief issue plaguing these approaches—namely, the step of generating candidate networks. DISCOVER's candidate network generation algorithm [HP02] is inefficient due to the number of partial results generated and due to the cost of isomorphism detection. Markowetz *et al.* [MYP07] addresses both of these issues by 1) adding joins to only certain nodes when expanding a candidate network and 2) avoiding isomorphism detection by controlling the order that joins are added. Yu *et al.* [YQC10] further improve the efficiency of this algorithm via a template-based approach.

The poor performance of the graph-based systems on the benchmark presented in Chapter 3 indicates that these techniques are not as scalable as hoped. An interesting research question is whether or not these search techniques can provide acceptable execution times. Answering this question is challenging because there is only one user input to keyword search systems for databases—the query. Nevertheless, the complexity of the search process depends upon the following two factors:

- the initial search space (that is, the number of tuples (vertices) that contain search terms) and
- the number of tuples (vertices) related to those containing search terms.

Controlling these two factors to ascertain the best- and worst-case runtime of proposed search techniques poses an interesting challenge.

It is tempting to control the first factor (that is, the initial search space) by seeding the database with new terms. The distribution of these terms can be fixed for precise control of the initial search space, which would give researchers precise control over scalability experiments. Another option is to create queries with terms that appear with a given frequency in the database. However, both of these approaches are poor because

neither ensures that queries will resemble real user queries. Query segmentation provides a good alternative to these approaches when evaluating how a smaller initial search space affects overall performance. Most web search engines use query segmentation to disambiguate queries, which usually improves the relevance of search results. For example, the query "New York" should be considered a single entity rather two separate terms. For many of the queries in the evaluation benchmark (Chapter 3), a good query segmentation technique could dramatically prune the search space, improving search effectiveness and potentially runtime performance. Given the set of correct results, it is easy to replicate the advantage that an optimal query segmentation algorithm would provide. The optimal query segmentation algorithm not only identifies the concepts in a query but also identifies exactly those database tuples (vertices in the data graph) referred to by these concepts.[3]

Table 7.1 contains the results from running various search techniques while using an optimal query segmentation algorithm. The runtimes reflect the minimum initial search space required to answer the queries in the benchmark. As evidenced by the table, the improvement in response time is considerable for many search techniques, but some search techniques actually require more time to identify the top-10 results.[4] Nevertheless, the important result of this experiment is that the response times—particularly for IMDb and Wikipedia—are still more than a few seconds and longer than most users are willing to wait.

If reducing the size of the initial search space cannot correct existing runtime performance issues, the only other possible area for improvement is to improve the process of identifying relationships among search terms. A guided search heuristic that directs each search technique toward the set of correct results was created to investigate this possibility. Tuples (vertices) that do not appear in a relevant result are immediately discarded by the guided search heuristic so the number of relationships that must be considered is minimized. Table 7.1 also contains the results for each search technique while using the guided search heuristic. Unlike optimal query segmentation, the guided search heuristic does significantly improve performance, particularly for the larger datasets (IMDb and Wikipedia).[5] However, these performance improvements are insufficient insofar as runtime still remains unacceptably high for a keyword search system. Even when combining both optimal query segmentation with the guided search heuristic, the response time for some search techniques remains unacceptable, particularly on the IMDb dataset.

[3]Query segmentation ordinarily does not have this power, but given the interest in determining search techniques' best-case performance, it is reasonable to make it "optimal."

[4]Although this result may be unintuitive, a smaller initial search space results in fewer possible ways to relate the search terms. Consequently, there are fewer total results because many irrelevant results (i.e., those with tuples containing the search terms but not actually relevant) cannot be enumerated.

[5]DISCOVER and DISCOVER-II do not enjoy the same improvement as the graph-based search techniques because each potential candidate network must still be executed by the database. Even though irrelevant candidate networks fail to return any results and execute extremely quickly, the number of such candidate networks minimize the performance advantage proffered by the guided search heuristic.

Table 7.1: Best-case performance using optimal query segmentation, guided search, and an oracle approach (query segmentation and guided search).

(a) Mondial

		query seg.		guided		oracle	
System	resp.	resp.	%	resp.	%	resp.	%
BANKS	1778.9	898.7	49.5	154.7	91.3	145.6	91.8
DISCOVER	8.0	3.7	53.8	6.1	23.8	3.5	56.3
DISCOVER-II	6.5	3.8	41.5	6.1	6.1	3.7	43.1
BANKS-II	143.4	189.2	−31.9	48.1	66.5	38.4	73.2
DPBF	5.2	3.6	30.8	1.6	69.2	1.6	69.2
BLINKS	14.1	54.8	−288.7	7.1	49.7	6.9	51.1

(b) IMDb

		query seg.		guided		oracle	
System	resp.	resp.	%	resp.	%	resp.	%
BANKS	2686.0	3451.3	−28.5	236.6	91.2	51.3	98.1
DISCOVER	227.9	31.9	86.0	96.7	57.6	11.3	95.0
DISCOVER-II	201.8	30.9	84.7	98.6	51.1	11.6	94.3
BANKS-II	3604.3	3275.5	9.1	457.2	87.3	548.3	84.8
DPBF	2042.1	1640.5	19.7	40.4	98.0	34.1	98.3
BLINKS	—	—	—	—	—	—	—

(c) Wikipedia

		query seg.		guided		oracle	
System	resp.	resp.	%	resp.	%	resp.	%
BANKS	2229.9	3535.8	−58.6	27.8	98.8	1.2	99.9
DISCOVER	43.1	10.1	76.6	28.7	33.4	5.3	87.7
DISCOVER-II	39.8	10.1	74.6	38.2	4.0	5.3	86.7
BANKS-II	2755.8	3032.2	−10.0	31.5	98.9	15.5	99.4
DPBF	251.3	70.7	71.9	25.0	90.1	1.0	99.6
BLINKS	—	—	—	—	—	—	—

Legend

query seg.	query segmentation	resp.	mean response time (in seconds) to retrieve 10 results, lower is better
guided	guided search		
oracle	query segmentation and guided search	%	percentage improvement

These experiments indicate that existing search techniques are unlikely to provide the response time that users have come to expect from keyword search systems. Is there any hope for keyword search in databases to provide acceptable response times? Yes, but first researchers must abandon some intuitive notions about enumeration and ranking. Chapter 4 indicates that ranking results by edge weight is simply not an ideal ranking scheme. In fact, edge weight can be completely ignored without sacrificing search effectiveness. This result opens the door for new enumeration algorithms that do not approximate the group Steiner problem, which is intractable. For example, STAR [KRS+09] uses a breadth-first search to identify a result tree and then iteratively improves the weight of this tree. Simply using the first phase of this algorithm

(which is equivalent to enumerating results by height rather than weight [GKS08]) could significantly improve enumeration performance.

This preliminary examination of best-case runtime has not been performed previously by researchers but should accompany proposed search techniques. The results of this preliminary analysis indicate that a break must be made with current approaches if relational keyword search is to scale to large datasets. Other components of a usable system are also scattered throughout the literature. Precomputing all possible results up to a given size is extraordinarily expensive both in initial effort and in the size of the index. However, these approaches do have an advantage in that all related information is contained in a single location. Denormalizing relational data (e.g., combining closely-related information) would reduce the number of relationships (edges in the data graph), which is the most computationally expensive part of schema- and graph-based approaches. Applying this approach would allow simple queries (the majority of user queries as evidenced in Chapter 5) to be answered efficiently; only more complex queries would invoke the more computationally expensive search heuristics. Researchers should also design algorithms that are embarrassingly parallel to take advantage of many low-cost commodity machines. Parallel algorithms have received practically no attention to date in the literature; it is high-time for researchers at least to consider this factor when designing new search techniques.[6]

7.3 Visualization

The visualization of search results is another research issue that deserves mention. The simplistic ranked list of results is not the quintessential interface. Kaki and Aula [KA05] show that even a simple modification—filtering results by categories—increases the speed and accuracy of retrieval tasks. The research community has not yet determined the best method to display search results, but in the context of searching databases, a ranked list cannot capture the relationships among the results, which is perhaps the most important factor for relational keyword search. This inability limits knowledge building by association and domain exploration, both of which are critical components of exploratory search [sch09]. For this reason, incorporating a browsing framework is important: it allows users to navigate among the various search results to discover their relationships.

Displaying tree-structured results is another challenge. One insidious problem is how to highlight commonalities among search results. Consider the two results shown in Figure 7.1 and even their graphical representation in Figure 7.2. Is it easy to identify all the database tuples shared by even just two results?

[6] A parallel implementation (which may be difficult) is distinct from a parallel algorithm. The design of new algorithms should consider using multiple machines (particularly to increase the available working memory) even if their research prototype is not parallel. In particular, algorithms that share considerable state should be avoided in favor of those that lend themselves to efficient parallel implementations.

```
person:  name=Ford, Harrison, ...
+ cast:  ...
| + movie:  name=Indiana Jones and the Last Crusade, ...
|   + cast:  ...
|     + person:  name=Connery, Sean, ...
+ cast:  ...
  + movie:  name=Raiders of the Lost Ark, ...
    + cast:  ...
      + person:  name=Allen, Karen, ...
```

```
character:  name=Indiana Jones, ...
+ cast:  ...
| + movie:  name=Raiders of the Lost Ark, ...
|   + cast:  ...
|     + person:  name=Allen, Karen, ...
+ cast:  ...
  + movie:  name=Indiana Jones and the Last Crusade, ...
    + cast:  ...
      + person:  name=Connery, Sean, ...
```

Figure 7.1: Tree-structured search results. Nodes common to different results are difficult to identify.

Tuples common to multiple results are probably more relevant to the query, yet commonalities among the search results are easy to overlook. The top-k results may also repeat the same nodes, showing different combinations of the relationships among them. Both issues suggest that a better solution exists.

Humans process visual cues much more acutely than scanning text [SCL+99]. 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. For example, combining the two results shown in Figure 7.2 into a graph clearly shown their commonalities and differences with only a modicum of user effort (compare Figure 7.2 with Figure 7.3). When examining search results, users should be able to drill down on graph nodes and edges to acquire additional information. Given that graph visualizations are already a common means for visualizing networks, the semantic web, and other systems that contain significant structure, they should be incorporated into a novel user interface for relational keyword search.

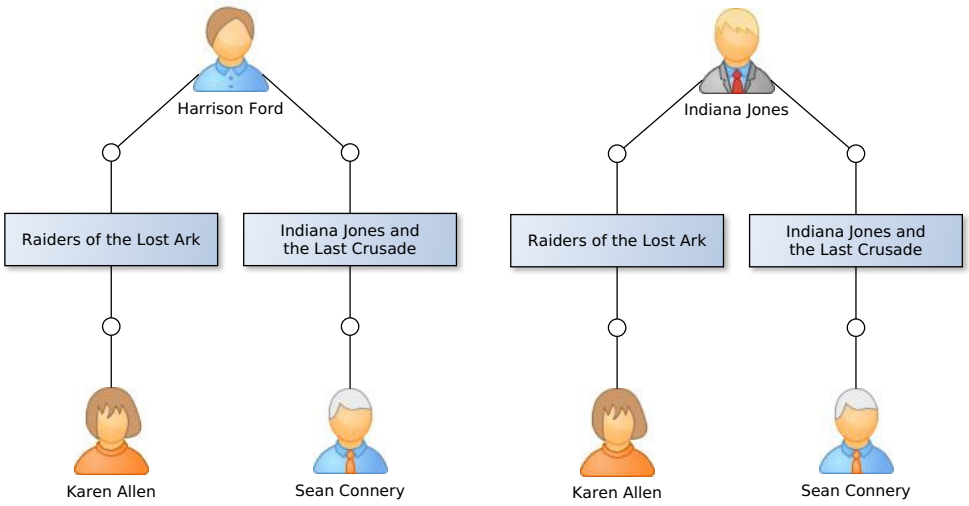


Figure 7.2: Visualization of search results shown in Figure 7.1. Commonalities among results are still difficult to identify.

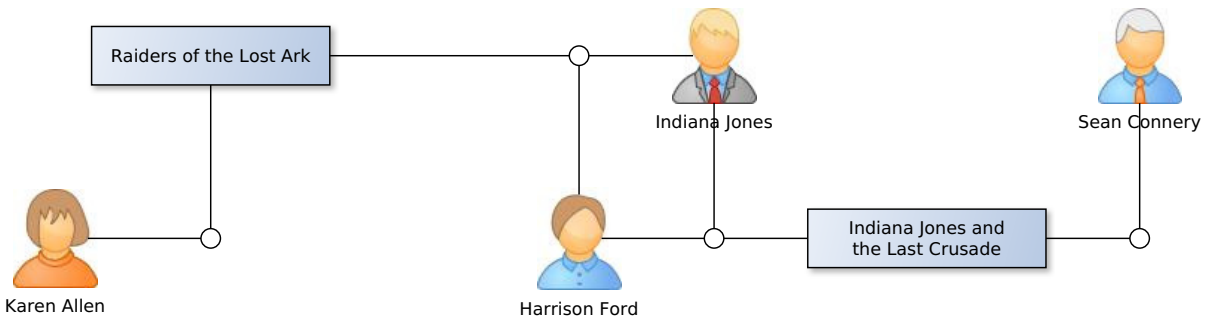


Figure 7.3: Example of visualizing multiple search results simultaneously. More information is presented in this graph, but it is trivial to identify the relationship between Karen Allen (far left) and Sean Connery (far right).

Chapter 8

Conclusion

As information becomes available in digital form, people expect to use this information effectively. Unfortunately, the data repositories typically used to store this information (i.e., relational databases) are not amenable to the search techniques developed by the IR community to find relevant information. Relational keyword search techniques hold promise for making the information stored in relational databases accessible to non-technical users, but despite significant research interest during the past decade, little real progress appears to have been made. This dissertation considers the challenges that the existing search techniques must overcome before they can be used for real-world retrieval tasks in existing data repositories. The objective of this research is to push research prototypes toward practical retrieval systems.

The contributions of this research center on the evaluation of relational keyword search techniques and novel schemes to improve search effectiveness. These contributions are summarized in the following paragraphs.

Objective empirical evaluation has been a cornerstone of the IR community [Sin01]. Regrettably, the practices that enabled search effectiveness to double within the first six years after the inception of a large-scale, standardized evaluation forum (TREC) have not been applied to the evaluation of relational keyword search techniques. This failure has led to unrealistic evaluations and makes it difficult to compare different search techniques because previous approaches must be reimplemented from scratch to effect a comparison [Web10]. Researchers [CWLL09, Web10] have called for the creation of frameworks to evaluate the runtime performance and search effectiveness of relational keyword search strategies. The most significant contribution of this dissertation is a benchmark that satisfies these calls for an independent test collection with publicly available queries and relevance judgments. That benchmark was presented at CIKM 2010 [CW10a] and has since been used by other researchers [BTN11, MS12] to improve their own experimental evaluations. In addition, a major

empirical evaluation of existing search techniques accompanies the benchmark's description in Chapter 3. The baselines for previous work free researchers from reimplementing others' research and allow them to focus on novel improvements to relational keyword search.

Two such novel ranking schemes that improve search effectiveness are structured cover density ranking and SVM rank. Structured cover density ranking (described in Chapter 4 and presented at KEYS 2010 [CW10b]) is a generalization of an unstructured ranking scheme that adheres to users' expectations regarding the order of search results. Unlike prior work, which focuses on a relational context, structured cover density ranking applies to any form of structured information including XML. In addition, it does not depend upon collection-wide statistics to score results; this independence makes structured cover density ranking simple to parallelize. SVM rank, which is also described in Chapter 4, was presented at CIKM 2011 [CW11b] and uses machine learning to create a scoring function for relational keyword search. Machine learning is significantly more robust than tuning scoring functions by hand, which was the trend for previous scoring functions proposed in the literature. Both structured cover density ranking and SVM rank improve upon the retrieval effectiveness of previous search techniques. A major research finding accompanying SVM rank is that certain factors (viz. edge weight) are not correlated with the relevance of search results. This opens the door for novel enumeration algorithms that do not approximate the group Steiner problem.

This dissertation also explores research questions regarding the evaluation of relational keyword search techniques. That research was presented at WSCD 2012 [CW12] and investigates what users search for in relational databases. Key components of this work include a model for user interaction and a taxonomy to classify both the intent behind queries and the expression of this intent. Comparing real user queries to those being used to evaluate relational keyword search techniques reveals a number of shortcomings. First, user queries are less complex than those devised by researchers to evaluate proposed search techniques. Second, existing query workloads fail to capture the wide variation in the search space explored by relational keyword search techniques.

These shortcomings and potential for inadvertent bias during previous evaluations may account for the experimental discrepancies that litter the literature. The extensive empirical evaluation that accompanies the benchmark indicates that many previous results are not reproducible: runtime performance is worse and search effectiveness is lower than claimed by previous evaluations. In addition, the scalability of some search techniques is abysmal, not capable of handling databases that contain more than tens of thousands of tuples. Even though memory concerns are somewhat trivial to address, it is disappointing that these concerns persist for small datasets even after a decade of research. Runtime performance is more critical since investigation reveals that even in the best case search techniques fail to provide acceptable response times for large datasets. More work is necessary to replace existing enumeration techniques with algorithms

that have acceptable performance. With regard to search effectiveness, most search techniques score lower than reported by previous evaluations, and it appears that the size of the dataset plays an important role when measuring the quality of search results. These findings indicate that this field needs an established benchmark with which to measure incremental progress toward the objective of developing practical relational keyword search techniques. Now that such a benchmark is available, it is beginning to be discovered by other researchers (e.g., Bicer *et al.* [BTN11] and Mass and Sagiv [MS12]). Wide-scale adoption of this benchmark will move the field forward by enabling fair and meaningful comparison of proposed search techniques. This benchmark, combined with contributions to novel ranking schemes (structured cover density ranking; the design, implementation, and analysis of SVM rank; and observations on the irrelevance of some factors commonly used as measures of search effectiveness) and an understanding of the true nature of user queries, will empower future research to develop efficient and effective keyword search techniques for relational databases.

Appendix A

Overview of Databases and Information Retrieval

Databases and information retrieval (IR) systems evolved separately despite their similar goal to manage information. While their divergence into separate subfields of computer science may be accidental, the effects of this divergence have profound impacts on today's attempts to construct complex integrated systems. With policy makers, researchers, and even the general public now demanding unprecedented access to information, researchers are attempting to reconcile the different models and techniques used by both fields to manage information and to provide users with effective means to access that information. This appendix provides a high-level overview of the key concepts in both fields and explores the historical reasons for their differences.

A.1 Databases

Most popular database systems today trace their roots to System R [ABC+76, BAC+81] developed at IBM during the 1970s [SMA+07]. At that time, there was a single database market—business data processing, commonly referred to as online transaction processing (OLTP). OLTP systems manage transaction-oriented applications[1] such as ATMs, reservation systems, and product sales. For these applications, the integrity of the data is paramount. Early databases were architected to provide a uniform method to store and retrieve data while offering a high degree of concurrency. Simultaneous access by many users was critical, given that the database was the central data repository for the enterprise. Various data control issues such as

[1] "Transaction" in this context refers to business transactions.

security, privacy, and backup and recovery gave rise to the database management system (DBMS) to manage databases efficiently.

As time progressed, it became apparent that the information stored within a database could provide companies with a significant competitive advantage in attracting and retaining customers. This realization, coupled with the increasing amount of information being stored electronically, created a second application for these systems—online analytical processing (OLAP). OLAP is a special case of decision-support systems, which aim to provide users with the high-level information necessary to make complex decisions. For example, decision-support systems can be used to determine what products to stock when a hurricane is forecasted to make landfall[2] or what applicants to admit to a university. OLAP and more general decision-support systems typically build upon OLTP systems since the latter provide a wealth of historical data that can be used to make good predictions.

The critical observation to make about these database applications is that both depend on the accuracy of the underlying data. Inaccurate or inconsistent data cannot be tolerated. Hence, one of the primary goals when designing a database is eliminating redundant information. Duplication of data should be avoided for a variety of reasons that are mentioned here due to their impact on the relational model, which underlies most databases in use today [SKS06]. First, superfluous copies of data require additional storage space. Even though the capacity of magnetic disks has increased rapidly during the past few decades, an enterprise incurs real costs to store duplicated data. Second, redundant copies of information invite inconsistency in the data. Consider the result if a bank records a customer's address with each of the customer's accounts and all the addresses are not changed when the customer moves. Now the customer's true address in not known, and the bank will send statements to the wrong address. Third, even if the aforementioned problem is handled (that is, if any duplicated data is modified, all copies are updated or none are changed), updating information in the database is more expensive because all copies must be changed to avoid inconsistent data. Although these issues may be obvious, they have a profound impact on how data is organized in a database and why retrieving information can be a challenging task. The following paragraphs present the relational model in more detail to illustrate how this popular paradigm organizes information.

The Relational Data Model

The utility of early information processing systems of the 1950s was seriously constrained due to the medium for persistent data storage—magnetic tape. Sequential data access largely limited the tasks that could be undertaken by these systems to processing records in a fixed order. Magnetic disks—first introduced

[2] The answer is surprising. Walmart sends (among other products) strawberry Pop-Tarts to the areas where a hurricane will make landfall [Ayr07].

Harrison Ford
His father was Irish, his mother Russian-Jewish. He was a lackluster student at Maine Township High School East in Park Ridge Illinois (no athletic star, never above a C average). After dropping out of Ripon College in Wisconsin, where he did some acting and later summer stock, he signed a Hollywood contract with Columbia and later Universal. . . .

Filmography

Air Force One	1997
President James Marshall	
Patriot Games	1992
Jack Ryan	
Indiana Jones and the Last Crusade	1989
Indiana Jones	
Indiana Jones and the Temple of Doom	1984
Indiana Jones	
Star Wars: Episode VI - Return of the Jedi	1983
Han Solo	
Raiders of the Lost Ark	1981
Indiana Jones	
Star Wars: Episode V - The Empire Strikes Back	1980
Han Solo	
Star Wars: Episode IV - A New Hope	1977
Han Solo	

. . .

Raiders of the Lost Ark **(1981)**
Archeologist and adventurer Indiana Jones is hired by the US government to find the Ark of the Covenant before the Nazis.

Director	Steven Spielberg
Writers	Lawrence Kasdan (screenplay)
	George Lucas, Philip Kaufman (story)

Cast

Harrison Ford	Indiana Jones
Karen Allen	Marion Ravenwood
Paul Freeman	Dr. René Belloq
John Rhys-Davies	Sallah

. . .

Indiana Jones
Dr. Henry "Indiana" Jones, Jr. (born July 1, 1899), also known as Indy, is a fictional professor, archaeologist, and adventurer. . .

Filmography

Indiana Jones and the Last Crusade	1989
Harrison Ford / River Phoenix (as Young Indy)	
Indiana Jones and the Temple of Doom	1984
Harrison Ford	
Raiders of the Lost Ark	1981
Harrison Ford	

. . .

Figure A.1: Entertainment information used to illustrate the relational data model. The text is taken from IMDb.

in the late 1950s—provided random access to any storage location and greatly expanded the possibilities for information systems. Data structures like linked lists and trees took advantage of the ability to access data efficiently regardless of its location on disk. Network databases [BW64, Dod69, BBB+71, MHS+71] and hierarchical databases [McG77, TL76] were constructed around these data structures and required programmers to manipulate the data structures directly to access data. This dependency between data and the database implementation made these systems complex to use and difficult to modify either the data (e.g., including more information about entities modeled in the database) or the implementation. Edgar Codd proposed the relational model [Cod70] to provide data independence—that is, the physical data storage should be independent of its access mechanism. The relational model also allowed for non-procedural querying of data, which freed users to focus on *what* to retrieve rather than *how* to retrieve it. However, it was not until the early 1980s that the relational model came into prominence following several landmark research projects [ABC+76, SHWK76, BAC+81] that proved its performance to be comparable to other database (i.e., network and hierarchical) models. Relational databases have been largely unchallenged in the past two decades as the preeminent paradigm for information systems.

The major advantage of the relational model as compared to previous data models is its simplicity. All information is conceptually stored in relations, which are commonly referred to as tables. Relations model both entities, the real-world objects represented in the database, and relationships among these entities. Using the example entertainment information shown in Figure A.1, relations would be created for people,

Person	id
Harrison Ford	10
Sean Connery	11
Karen Allen	12
Steven Spielberg	13

Role	id
actor	14
actress	15
director	16
composer	17

Character	id
Indiana Jones	7
Marion Ravenwood	8
Professor Henry Jones	9

Film	year	id
Raiders of the Lost Ark	1981	18
Indiana Jones and the Last Crusade	1989	19

Cast				
filmId	personId	roleId	characterId	id
18	10	14	7	1
18	13	16		2
19	10	14	7	3
19	11	14	9	4
19	13	16		5
18	12	15	8	6

Figure A.2: Example IMDb relational tables.

characters, films, and the roles individuals have making a film (e.g., acts, directs, or produces). An additional "cast" relation is used to represent the relationship among all these entities. Relations are highly structured, comprising a fixed set of attributes (table columns) whose values have well-defined domains. Attributes model details about a particular entity. Continuing the example, a film has a title, release date, length, rating, budget, and production company. A tuple (table row) contains data pertaining to a particular entity such as the character Indiana Jones or the film *Raiders of the Lost Ark*.

All tuples within a relation must be distinguishable. A primary key uniquely identifies the tuples within a relation. Foreign keys express relationships among entities. A foreign key contains the value of a primary key but appears in a different relation. In the running example, the cast relation will contain several foreign keys: one to a particular person, another to a film the person is associated with, yet another to the role the person had making the film, etc. Other real-world examples of foreign keys relating information abound. For example, SSNs are used to track individuals enrolled in the social security program and have been co-opted to identify individuals when filing taxes. A hospital might use this (or a similar unique id) number to associate a patient's X-rays and lab reports with the patient's name, address, birthday, medication list, etc. In these cases, the unique identifier relates different entities—namely, an individual with their tax returns or the services provided as part of medical care. A critical facet of a database (which gives it an advantage over other approaches to data storage) is that databases enforce referential integrity—that is, the tuple referenced by a foreign key must exist. Enforcing this constraint ensures that the database is always consistent with regards to related information.

As mentioned previously, eliminating redundant information is critical to prevent the possibility of inconsistency in a database. Normalization ensures that redundant information does not appear in different relations. While the details are beyond the scope of this introduction, the underlying intuition is straightforward. All

information pertaining to a single entity appears in one relation, and foreign keys denote associations between different entities. While the inclusion of foreign keys does protect the integrity of the database, the end result is that related information is physically separated in the database. For example, no single tuple in the movie database states that the film *Raiders of the Lost Ark* starred Harrison Ford and Karen Allen. Instead, this information can only be obtained by following the foreign keys in the cast relation to the person and film referenced by those foreign keys. Connecting related pieces of information in a relational database is referred to as a join operation. The relational data model allows joins to be performed efficiently, giving the appearance that the data is well-integrated (i.e., appears to be stored together) while avoiding redundancy. Figure A.2 illustrates a well-designed relational database for the running movie example. Notice that the cast relation references the unique id numbers present in the other relations and recovering the conceptual view of the information (e.g., an individual's filmography) requires joining the various relations.

A.2 Information Retrieval

Whereas databases developed primarily to address business data processing, the origins of IR are considerably different. The advent of the printing press in the 15th century and inexpensive paper in the 19th century allowed writing and disseminating large amounts of text. Library systems required effective means to categorize and search this text, which led to the creation of partial indexes for documents. For example, an index could be manually maintained for titles of books, articles written by an author, or the content's subject matter. However, indexing the actual text of articles or books was simply not feasible. In 1945, Vannevar Bush [Bus45] envisioned the "memex," a machine capable of storing an individual's personal library and providing instant access to its information. Users created trails through the content based on their associations rather than by the aforementioned indexing schemes. By the 1950s, it was apparent that text archives could be searched automatically. Luhn [Luh57] laid the groundwork for modern information retrieval by proposing to use words to index documents and to use statistical methods to compare queries with documents.

Manning *et al.* define information retrieval as "finding material [. . .] of an unstructured nature (usually text) that satisfies an information need from within large collections [. . .]" [MRS08]. Users and information needs are at the heart of IR. An information need is a topic about which the user desires more information. A good retrieval system should find documents that satisfy the user's information need. Two challenges make this task complex. First, a user's information need is expressed as a query, which is the user's attempt to describe the desired information. The relevance of documents is not judged relative to the query (which may be vague or ambiguous) but to the user's original information need. Second, many documents in a given collection may pertain to the user's query. These documents should be ranked in order of their estimated

Table A.1: Comparison of fundamental data types in DBs and IR systems.

	DB	IR
data structures	relation, B+ tree	document, inverted index
queries	formal, structured (e.g., XQuery and SQL); join multiple relations	natural language (i.e., keywords, free text)
results	relation, likely the result of joining multiple relations; exact; objective measure of correctness	documents; inexact, ranked by estimated relevance; subjective measure of correctness

relevance so that the user examines only as many as necessary to satisfy the information need. Given that the only input to the system (the query) is merely a proxy for what is desired, retrieval systems understand that users may iteratively refine their queries and view results until the desired information is found. Users may also abandon their search if they feel the desired information does not exist or will not be retrieved by the system (e.g., because the user cannot specify the information need precisely).

The central data structure for an IR system is the inverted index, which maps terms to the set of documents that contain each term. Answering boolean queries is as simple as intersecting (or unioning) the sets of documents that contain each term. However, an unordered set of results is not particularly useful when many documents satisfy the query, and ranked retrieval systems have been shown to be more effective than systems that use pure boolean queries [Moo61, Tur94]. Ranking query results requires a model to score documents (and subsequently to sort them in order of their estimated relevance). Two popular strategies for ranked retrieval are the vector space model and probabilistic models. The vector space model views documents and queries as sets of terms. The score of a document for a particular query is merely the similarity of their sets of terms. Probabilistic models attempt to rank documents in decreasing probability of relevance to the user's query [Rob77]. Probability theory provides a formal foundation to reason about uncertainty inherent to the user's information need and whether or not a document contains content that will satisfy the information need.

Both the vector space model and probabilistic models rely heavily on determining the importance of individuals terms within a document. Three factors are commonly used to measure this value. Term frequency expresses the intuition that terms that appear more frequently in a document are considered more important. Terms that appear in many documents are considered common, or stated differently, terms that appear in fewer documents are more important than terms that appear in many documents. Scoring functions typically include an inverse document frequency factor to express this belief. Finally, longer documents contain more terms than shorter documents, which would enable them to score higher using just the two factors mentioned previously. Hence, document length is generally normalized to compensate for the advantage of longer documents containing more terms.

Appendix B

Description of Existing Search Techniques

A wide variety of techniques to support keyword search in relational databases have been proposed in the literature. Unfortunately, there is neither a standard problem definition nor a standard definition for valid query results. Most proposed techniques adopt definitions that enable different query processing (i.e., enumeration) strategies in an attempt to improve performance. Some search techniques' enumeration strategies omit results, which is justifiable because retrieval systems are not evaluated by their completeness (that is, whether or not they identify all possible results) but rather by user satisfaction (that is, how well do the query results address the user's information need). In most cases, the differences among search techniques are relatively insignificant, and the minutia among different problem definitions do not warrant detailed discussion in this dissertation.

This chapter presents three different approaches to supporting keyword search in relational databases. A general overview of each approach is presented prior to descriptions of specific search techniques based on that approach. Schema-based approaches are specific to relational databases. These search techniques reuse the underlying database to the greatest possible extent including for query processing. Graph-based approaches are much more general and apply to any data that can be modeled as a graph. These search techniques are applicable to other sources of structured data such as XML. The third approach moves much of the work required for efficient query processing to an offline phase. Possible search results are indexed prior to any searches; this preliminary step allows traditional IR techniques to be used for query processing and ranking.

B.1 Schema-based Approaches

Schema-based search techniques support keyword search over relational databases via direct execution of SQL commands. These techniques model the relational schema as a graph where vertices are relational tables and edges denote foreign keys between tables. Figure B.1 illustrates the general architecture for schema-based systems. Query processing follows three phases. First, database tuples that contain search terms are identified. Second, candidate networks (SQL expressions) that could relate these tuples are enumerated systematically. Third, the execution engine executes these SQL expressions to identify results, which are then returned to the user. Because there are many possible ways to relate the search terms, the execution engine is generally responsible for selecting only the most promising SQL expressions. The remainder need not be executed once the top-k highest-ranking results are known. The three separate phases of query processing are described in more detail in the following paragraphs.

IR Engine The IR engine is responsible for identifying tuples that may be relevant to the user's query. An inverted index typically maps query terms to database tuples. The inverted index can be external to the database (e.g., using Apache Lucene[1] to index text) or part of the database itself since major DBMSs (e.g., Oracle, DB2, SQL Server, MySQL, and PostgreSQL) all support full text indexes over attributes. In addition, other database indexes (e.g., B+ trees) can be used to identify tuples that match other types of data in the query (e.g., numeric values or dates). Assuming database indexes exist on attributes that should be matched, reusing these indexes eliminates the potential for inconsistency between the database and an external index. The output from the IR engine is sets of tuples (grouped by database relation) that contain information perceived to be relevant to the query (i.e., tuples that contain search terms).

Candidate Network Generator A tuple set is the set of tuples from each relation that contains search terms. Figure B.2 lists the tuple sets generated by two different schema-based systems for the query "Ford

[1] http://lucene.apache.org/

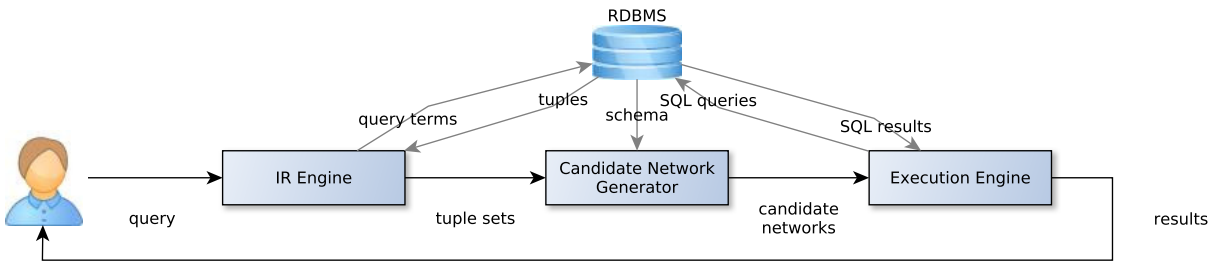


Figure B.1: Conceptual architecture of schema-based search techniques.

(a) DISCOVER

Table	Terms	Tuples
Character	$\emptyset$	{8}
	{ford}	$\emptyset$
	{jones}	{7, 9}
	{ford, jones}	$\emptyset$
Person	$\emptyset$	{11, 12, 13}
	{ford}	{10}
	{jones}	$\emptyset$
	{ford, jones}	$\emptyset$
Film	$\emptyset$	{18}
	{ford}	$\emptyset$
	{jones}	{19}
	{ford, jones}	$\emptyset$

(b) DISCOVER-II

Table	Terms	Tuples
Cast	$\emptyset$	{1, 2, 3, 4, 5, 6}
	Q	$\emptyset$
Character	$\emptyset$	{8}
	Q	{7, 9}
Person	$\emptyset$	{11, 12, 13}
	Q	{10}
Role	$\emptyset$	{14, 15, 16, 17}
	Q	$\emptyset$
Film	$\emptyset$	{18}
	Q	{19}

Figure B.2: Tuple sets generated by DISCOVER and DISCOVER-II for the query "Jones Ford" and the database instance shown in Figure A.2 on page 113. For brevity, the Cast and Role tables are omitted from DISCOVER because all the tuples in these tables belong to the free tuple set. DISCOVER-II creates a single non-free tuple set for each relation whereas DISCOVER creates a tuple set for each combination of search terms.

Jones." The free tuple set of each relation is the set of tuples that do not contain any search terms. The free tuple sets are used to relate the tuples that contain search terms. The expanded schema graph [YQC10] comprises all tuples sets and free tuple sets connected by their relationships in the schema graph. The candidate network generator takes the expanded schema graph and outputs a set of candidate networks, which are join expressions that may relate the tuples that contain search terms. Candidate networks must be minimal: removing even a single tuple set must either exclude search terms or break the result tree into two separate subtrees. Candidate networks are pruned according to a number of rules.

1. Isomorphic candidate networks are discarded because isomorphic candidate networks produce identical results.
2. The candidate network cannot contain a construct of the form $R \leftarrow S \rightarrow R$ when there is a single foreign key from the relation S to the relation R. Any tuple $s \in S$ must refer to the same tuple $r \in R$ and duplicating r would mean that the candidate network is not minimal.
3. The number of non-free tuple sets cannot exceed the number of search terms. Results will not be minimal when there are more non-free tuple sets than search terms. That is, there exists a smaller candidate network that contains all the search terms.

Figure B.3 illustrates the candidate networks generated for DISCOVER-II's tuple sets shown in Figure B.2.

Execution Engine Converting a candidate network to an SQL expression is straightforward: relationships are replaced by their foreign key references, and tuples from non-free tuple sets are restricted to those that

$\|T\|$	id	parent	T	valid	pruning condition
1	1		Character^{Q}	✓	
	2		Person^{Q}	✓	
	3		Film^{Q}	✓	
2	4	1	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset}$		
	5	2	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset}$		
	6	3	$\text{Film}^{Q} \leftarrow \text{Cast}^{\emptyset}$		
3	7	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Character}^{\emptyset}$		2
	8	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Character}^{Q}$		2
	9	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Person}^{\emptyset}$		
	10	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Person}^{Q}$	✓	
	11	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Role}^{\emptyset}$		
	12	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Film}^{\emptyset}$		
	13	4	$\text{Character}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Film}^{Q}$	✓	
	14	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Character}^{\emptyset}$		1
	15	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Character}^{Q}$		1
	16	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Person}^{\emptyset}$		2
	17	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Person}^{Q}$		2
	18	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Role}^{\emptyset}$		
	19	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Film}^{\emptyset}$		
	20	5	$\text{Person}^{Q} \leftarrow \text{Cast}^{\emptyset} \rightarrow \text{Film}^{Q}$	✓	
	...				

Figure B.3: Candidate networks generated by DISCOVER-II. The candidate networks of size 1 are the tuple sets of tables that contain search terms (see Figure B.2). Valid candidate networks may produce results to the keyword query when executed against the database. Candidate networks that are not pruned are expanded until a maximum size is reached.

contain search terms. Figure B.4 shows two candidate networks and their corresponding SQL queries. The SQL queries constructed from candidate networks must be evaluated to produce results to the user's original query. Determining an efficient execution strategy is critical because the number of candidate networks grows exponentially with their maximum size. Most query processing techniques reuse common subexpressions across candidate networks or avoid executing candidate networks by showing that their best results will not be ranked within the top-k results for the query. Ranking and query execution schemes are where most schema-based approaches differ.

DBXplorer Agrawal *et al.* [ACD02] proposed one of the first systems for keyword search in databases. Because the database might not provide full text indexes, a symbol table maps terms to tuples. The symbol table conceptually resembles a traditional inverted index, and various design alternatives are explored. Query processing follows the general strategy previously outlined. DBXplorer ranks results by the total number of joins in the candidate network.

id	SQL
1	**SELECT** * **FROM** Character **WHERE** Character.id **IN** (/* *characters matching search terms* */);
10	**SELECT** * **FROM** Character **JOIN** Cast **ON** Character.id = Cast.characterId **JOIN** Person **ON** Cast.personId = Person.id **WHERE** Character.id **IN** (/* *characters matching search terms* */) **AND** Person.id **IN** (/* *people matching search terms* */);

Figure B.4: SQL queries created for candidate networks. The id number refers to Figure B.3.

DISCOVER DISCOVER [HP02] creates a set of tuples for each subset of search terms that appear in the database relations. Determining the optimal execution strategy that has the lowest total evaluation cost for a set of candidate networks is an NP-complete problem so a greedy algorithm is proposed to enumerate results. The greedy algorithm prioritizes candidate networks that generate the fewest intermediate results and also candidate networks that share common subexpressions with other candidate networks. This greedy algorithm executes in time $O((|S| \cdot T_{\max})^2)$ where $|S|$ is the number of candidate networks and $T_{\max}$ is the maximum size of candidate networks. DISCOVER, like DBXplorer, ranks results by the total number of joins in the candidate network that produces the result.

DISCOVER-II Hristidis *et al.* [HGP03] refined DISCOVER by adopting an IR-style scoring function to rank results. Pivoted normalization weighting [SCH+99] is a state-of-the-art scoring function [Sin01]:

$$\sum_{t \in Q} \frac{1 + \ln(1 + \ln tf)}{1 - s + s \cdot \left(\frac{dl}{avgdl}\right)} \cdot qtf \cdot \ln\left(\frac{N+1}{df}\right) \tag{B.1}$$

where t is a query term, $(q)tf$ is the frequency of the (query) term, s is a constant (usually 0.2), dl is the document length, $avgdl$ is the mean document length, N is the number of documents in the collection, and df is the number of documents that contain t. The values of each database attribute form a distinct document collection. The total score for a result T is the sum of all the individual attribute scores normalized by the size of the result:

$$score(T, Q) = \frac{1}{|T|} \sum_{a} score(a, Q)$$

where a is an attribute, $score(a, Q)$ is pivoted normalization weighting's score for the attribute, and $|T|$ is the size of the candidate network. Including an IR-style scoring function enables two opportunities for optimization. First, only the top-k highest scoring results are returned to the user instead of the set of all results containing the search terms. Second, OR semantics become the default for query results; the IR-style scoring function is responsible for preferring results that contain more search terms. Scalability is further

improved by creating only a single tuple set for each database relation instead of the exponential number required by DISCOVER. Hristidis *et al.* propose a variety of strategies to execute candidate networks and to terminate execution once the top-k results are known.

Liu *et al.* Liu *et al.* [LYMC06] build on DISCOVER-II's framework but ignore performance considerations and focus exclusively on search effectiveness. They propose four new normalizations to adapt pivoted normalization weighting to a relational context. The resulting function bears some resemblance to pivoted normalization weighting (Equation B.1):

$$score(a, Q) = \frac{1}{(1-s) + s \cdot \frac{|T|}{\overline{|T|}}} \sum_{t \in Q} \frac{1 + \ln(1 + \ln tf)}{\left(1 - s + s \cdot \left(\frac{dl}{avgdl}\right)\right) \cdot (1 + \ln avgdl)} \cdot qtf \cdot \ln\left(\frac{N^g + 1}{df^g}\right)$$

where $\overline{|T|}$ is the average size of results, N^g is the number of tuples in the database, and df^g is the number of tuples that contain the term. Scores from individual attributes are combined according to the following function:

$$score(T, Q) = \max_a score(a, Q) \cdot \left(1 + \ln\left(1 + \ln\left(\frac{\sum_a score(a, Q)}{\max_a score(a, Q)}\right)\right)\right)$$

where a is an attribute and $score(a, Q)$ is the attribute's score as determined by the previous function. Query terms that match the names of database relations or attributes are handled explicitly and phrase weighting boosts the scores of results that match phrases in the user's query.

SPARK Previous schema-based approaches that feature IR-style scoring required monotonic score aggregation functions for efficient execution. These monotonic score aggregation functions combined the scores of individual attributes linearly to determine a result's overall score. In contrast, most IR scoring functions are not monotonic because a linear combination of component scores destroys desirable properties of most IR weighting schemes (see Robertson *et al.* [RZT04]). Unlike previous work, SPARK [LLWZ07a, LWL+11] does not calculate the scores of individual attributes. Instead, the result is scored as a single entity:

$$score(T, Q) = \left(\sum_{t \in Q} \frac{1 + \ln(1 + \ln tf)}{1 - s + s \cdot \left(\frac{dl}{avgdl}\right)} \cdot qtf \cdot \ln\left(\frac{N + 1}{df}\right)\right) \cdot completeness(T, Q) \cdot nsize(T)$$

where $completeness(T, Q)$ is derived from the extended boolean model [SFW83] and rewards results that contain more search terms

$$completeness(T, Q) = \left(1 - \left(\sum_{1 \le i \le |Q|} \left(\frac{tf_i}{\max_{1 \le j \le |Q|} tf_j} \cdot \frac{idf_i}{\max_{1 \le j \le |Q|} idf_j}\right)^p\right)^{\frac{1}{p}}\right)$$

and p is a tuning parameter ($p \in [1, \infty)$) that enforces AND semantics. The size normalization factor is computed according to the following function:

$$nsize(T) = (1 + s_1 - s_1 \cdot |T|) \cdot (1 + s_2 - s_2 \cdot |T^{nf}|)$$

where s_1 and s_2 are tuning parameters and $|T^{nf}|$ is the number of non-free tuple sets in the candidate network producing the result. One downside to SPARK's approach is that collection statistics (e.g., average document length (*avgdl*) and the inverse document frequency factor) must be estimated. Including a non-monotonic scoring function also precludes the query processing schemes previously developed by Hristidis *et al.* [HGP03]. Luo *et al.* propose a skyline sweep algorithm to enumerate results; this algorithm also minimizes the total number of database probes made during query execution.

Qin *et al.* Qin *et al.* [QYC09] explore different query processing strategies to support keyword search in databases. Different semantics for query results (connected trees (the problem statement used in this dissertation), distinct root [KPC+05, HWYY07], and communities [QYCT09]) are all considered, and experiments indicate that all three can be supported efficiently on relational databases via SQL queries. The performance improvements are achieved by aggressive pruning of tuples that will not contribute to results. This pruning reduces the size of temporary relations created by database joins, thereby reducing the overall query processing cost.

B.2 Graph-based Approaches

Graph-based approaches assume the database is modeled as a weighted graph where the weights of edges indicate the importance of relationships. Proximity search strategies attempt to minimize the weight of result trees. This task is a formulation of the group Steiner tree problem [DW71], which is known to be NP-complete [RW90]. In addition to ranking results by their total edge weight, many search techniques also include a prestige (i.e., node weight) factor to prefer results that contain more highly-referenced database tuples. Graph-based search techniques are more general than schema-based approaches, for relational databases, XML, and the Internet can all be modeled as graphs. As with the schema-based approaches, a wide variety of ranking schemes have been proposed in the literature.

None of the graph-based search techniques described in the literature operate on the database itself. Instead, the database is explicitly converted to a graph. Tuples become vertices in the graph, and edges denote foreign keys. Most search techniques also distinguish edges induced by foreign keys in the database

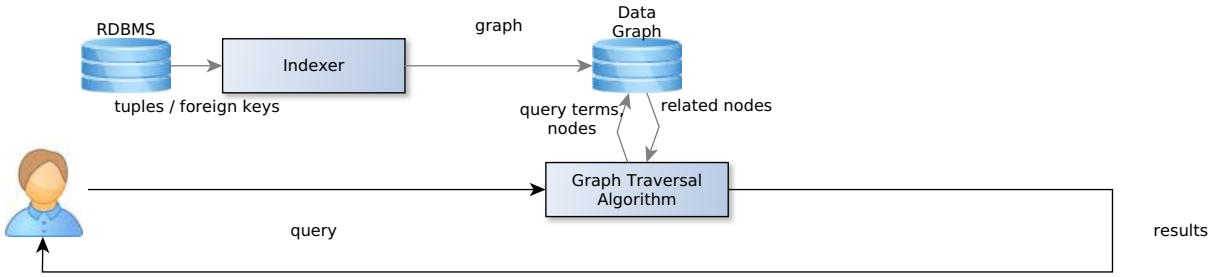


Figure B.5: Conceptual architecture of graph-based search techniques.

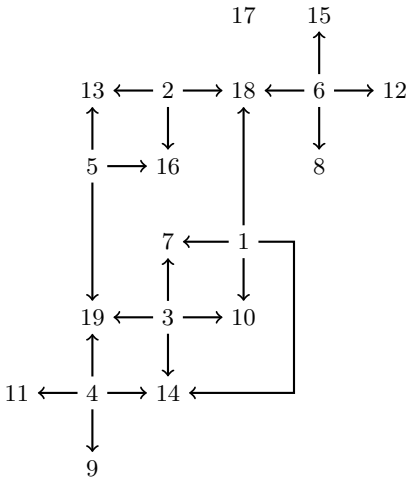
from "backward" edges, which connect the same vertices but reverse the edge's direction. The addition of the backward edges allow directionality to be considered; weighting the backward edges higher than the "forward" edges discourages the inclusion of common relationships that users would find uninformative. For example, the relationship between Sean Connery, Harrison Ford, and the film *Indiana Jones and the Last Crusade* is less common than the relationship between these two individuals and the actor role (which is shared by many people across many different films). Edge weighting and directionality allow these search techniques to suppress these common relationships. Figure B.5 depicts the conceptual architecture of graph-based approaches.

BANKS BANKS [BHN+02] enumerates results by searching the graph backwards from vertices that contain query keywords. The backward search heuristic concurrently executes copies of Dijkstra's shortest path algorithm [Dij59], one from each vertex that contains a search term. When a vertex has been labeled with its distance to each search term, that vertex is the root of a directed tree that is a result to the query. Figure B.6 illustrates the backward expanding search heuristic on example data from IMDb. Even though the backward search heuristic is non-optimal, it does illustrate the core idea behind proximity search strategies. BANKS ranks results according to the following scoring function:

$$score(T) = (1 - \lambda)\left(\frac{1}{1 + \sum_e weight(e)}\right) + \lambda\left(\frac{\sum_n weight(n)}{|N|}\right)$$

where e is an edge and n a node in the result tree, N is the set of all nodes in the result tree, and λ is a tuning parameter. Because results are not guaranteed to be enumerated in ranked order, a small fixed-size heap buffers the results. Once the heap is full, the highest scoring result is returned to the user.

BANKS-II BANKS-II [KPC+05] augments the backward search heuristic [BHN+02] by also searching the graph forward from potential root nodes. This strategy has an advantage when the query contains a



edge	weight
$7 \to 1$	5.883
$7 \to 3$	5.883
$8 \to 6$	1.585
$9 \to 4$	3.322
$10 \to 1$	9.412
$10 \to 3$	9.412
$11 \to 4$	9.255
$12 \to 6$	7.098
$13 \to 2$	10.071
$13 \to 5$	10.071
$14 \to 1$	22.897

edge	weight
$14 \to 3$	22.897
$14 \to 4$	22.897
$15 \to 6$	22.102
$16 \to 2$	19.909
$16 \to 5$	19.909
$18 \to 1$	10.772
$18 \to 2$	10.772
$18 \to 6$	10.772
$19 \to 3$	9.974
$19 \to 4$	9.974
$19 \to 5$	9.974

		Predecessors		
Distance	Vertex	connery	ford	Notes
0.000	10	—	$\emptyset$	contains “connery”
	11	$\emptyset$	—	contains “ford”
1.000	1	$\emptyset$	{10}	vertex 10 is predecessor on shortest path to “ford”
	3	$\emptyset$	{10}	vertex 10 is predecessor on shortest path to “ford”
	4	{11}	$\emptyset$	vertex 11 is predecessor on shortest path to “connery”
4.322	9	{4}	$\emptyset$	...
6.883	7	$\emptyset$	{1}	
	7	$\emptyset$	{1, 3}	2 different paths to “ford”
10.974	19	$\emptyset$	{3}	
	19	{4}	{3}	path to all search terms ⇒ root of result tree
11.772	18	$\emptyset$	{1}	
11.974	5	{19}	{19}	path to all search terms ⇒ root of result tree
22.045	13	{5}	{5}	path to all search terms ⇒ root of result tree
23.897	14	$\emptyset$	{1}	
	14	$\emptyset$	{1, 3}	2 different paths to “ford”
	14	{4}	{1, 3}	path to all search terms ⇒ root of 2 result trees
...				

Figure B.6: BANKS's backward search heuristic. The reduce clutter in the depiction of the data graph, no edge weights are given and no backward edges are shown. The weights of all forward edges (i.e., those shown in the data graph) are 1.0; the weights of the backward edges are shown in the tables on the right. The data graph is the same one presented in Figure 1.4 on page 12 and corresponds to the relational data in Figure A.2 on page 113.

common term or when a copy of Dijkstra's shortest path algorithm reaches a vertex with a large number of incoming edges. Spreading activation prioritizes the search but may cause the bidirectional search heuristic to identify shorter paths after creating partial results. When a shorter path is found, the existing results must be updated recursively, which potentially increases the total execution time. BANKS-II also adopts slightly different search semantics: only the first (and presumably highest-scoring) result rooted at a given vertex is returned to the user. Other answers with the same root are simply discarded. In keeping with these semantics, BANKS-II also modifies its predecessor's scoring function to

$$score(T) = \left(\sum_{p} weight(p)\right)\left(\sum_{n} weight(n)\right)^{\lambda}$$

where p is the path from the root of the result tree to a leaf, n is taken from the root and leaves of the result tree, and λ is a tuning parameter.

DPBF Although finding the optimal group Steiner tree is an NP-complete problem, there are efficient algorithms to find the optimal tree for a fixed number of terminals (i.e., search terms). DPBF [DYW+07] is a dynamic programming algorithm for the optimal solution but remains exponential in the number of search terms. After finding the optimal solution, additional results are enumerated in approximate order. DPBF scores results solely by their total edge weight—i.e.,

$$score(T) = \left(\sum_{e} weight(e)\right)^{-1}$$

where e is an edge of the result tree.

BLINKS He *et al.* [HWYY07] propose a bi-level index to improve the performance of bidirectional search [KPC+05]. BLINKS partitions the graph into blocks and constructs a block index and intra-block index. These two indexes provide a lower bound on the shortest distance to keywords, which dramatically prunes the search space. BLINKS also proposes a significantly more involved scoring function:

$$score(T) = \left(\alpha\, weight(r) + \beta \sum_{t} weight(l) + \gamma \sum_{t} weight(p)\right)^{-1}$$

where r is the root of the result tree, l is the leaf that contains the search term t, p is the path from r to the leaf containing t, and α, β, and γ are all tuning parameters.

Golenberg *et al.* Golenberg *et al.* [GKS08] identify the following properties that keyword search systems for databases should satisfy.

- They must generate all possible answers.[2]
- Results should be enumerated in an order that is highly correlated with the final ranking.
- The enumeration algorithm must be efficient, which means a polynomial delay between successive answers [JYP88].

Golenberg *et al.* propose a novel enumeration algorithm that satisfies all three of these constraints. Golenberg *et al.* enumerate result trees in approximate order by height rather than weight and permit the results to be ranked in a different order before being returned to the user. The proposed ranking function is dependent on the set of previously returned results for the query:

$$score(T) = \left(\left(\sum_{e} weight(e) \right)^{-1} + \epsilon \cdot redundancy(T, \mathbb{T}) \right)$$

where $redundancy(T, \mathbb{T})$ penalizes the current result T based on its similarity to the existing set of results $\mathbb{T}$ and ϵ is the penalty factor.

BANKS-III Dalvi *et al.* [DKS08] address the problem of searching graphs that are considerably larger than main memory. A multi-granular graph representation both permits a "global" view of the search and takes advantage of portions of the graph that are already cached within memory. Existing search techniques can take advantage of the multi-granular representation to save I/O costs and direct the search toward portions of the data graph most likely to yield results. Both improve the overall efficiency of the algorithm.

STAR STAR [KRS+09] is a pseudopolynomial-time algorithm for the Steiner tree problem. It computes an initial solution quickly and then improves this result iteratively. Although STAR approximates the optimal solution, its approximation ratio is significantly better than previous heuristics [BHN+02, KPC+05, HWYY07].

B.3 Precomputing Results

A downside of the schema- and graph-based approaches is that they perform the majority of work at query time. Although the database tuples (vertices in a data graph) are indexed *a priori*, the relationships among them are discovered as part of the search process. Identifying these relationships is expensive; in the case

[2]Even though retrieval systems are not evaluated by their completeness (i.e., whether or not they enumerate all possible results), achieving this goal is clearly desirable. A system cannot rank an answer that has not been enumerated.

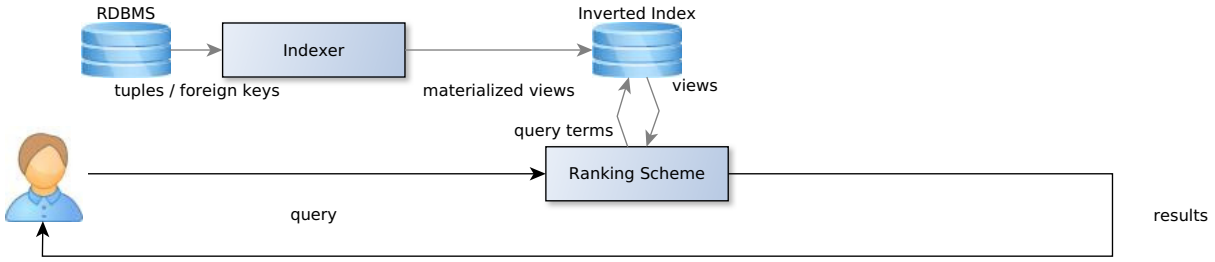


Figure B.7: Conceptual architecture of search techniques that precompute results.

of proximity search, finding even the best result is intractable. Approaches based on precomputing results moves the discovery of relationships to an offline phase. Essentially, all possible results are materialized prior to any searches. These materialized views are indexed via a traditional inverted index, which is scanned at query time to identify results that satisfy a query.

These search techniques are closest to traditional IR. Most variation among these systems stems from different semantics for query results and how to support these semantics efficiently via indexing and query processing. Figure 1.2 on page 10 illustrates several different logical views over relational data; these logical views are similar to the materialized views created by these search techniques.

EKSO Su and Widom [SW05] propose the EKSO system, which reuses the full text search features included in many relational databases. A virtual document comprises values (e.g., text) from related tuples. Virtual documents are rooted by a single tuple from a "root relation," which is a relation whose primary key is not composed entirely of foreign keys. Starting from the root tuple, a breadth-first traversal of the database identifies related content, but each relation in the database is visited only once. Because virtual documents are materialized within the database, triggers (which are fired in response to updates, inserts, and deletes) keep each virtual document consistent with its underlying data. Results are ranked by the underlying database's full text search engine.

EASE Li *et al.* [LOF+08] model unstructured web documents, semi-structured documents, and relational data as graphs to create a unified approach for keyword search in heterogeneous data collections. EASE defines a query result to be an r-radius Steiner graph, which is a subgraph whose radius[3] does not exceed r. Extracting an r-radius Steiner graph from an r-radius subgraph is straightforward; hence, the core problem is how to identify these subgraphs.[4] An inverted index is used to identify the subgraphs that satisfy a query.

[3]The *eccentricity* of a graph vertex is the maximum distance between it and any other vertex in the graph. The *radius* is the minimum eccentricity across all the vertices in the graph.

[4]Although there are an exponential number of r-radius subgraphs for a given graph [YQC10], Li *et al.* only consider those rooted at a different graph vertex. Of this subset, only the maximal r-radius subgraphs (i.e., those not contained by another r-radius subgraph) must be indexed.

EASE proposes the following function to score results:

$$score(T, Q) = \sum_{1 \le i < j \le |Q|} similarity((t_i, t_j), T) \cdot (score(t_i, T) + score(t_j, T))$$

where t is a query term ($t \in Q$), and T is an r-radius Steiner graph. The similarity function measures the similarity between search terms:

$$similarity((t_i, t_j), T) = \frac{1}{N_i \cup N_j} \sum_{n_i \in N_i, n_j \in N_j} \sum_{n_i \rightsquigarrow n_j} \frac{1}{(|n_i \rightsquigarrow n_j| + 1)^2}$$

where N_i comprises vertices of T that contain term t_i (and similarly for N_j), $n_i \rightsquigarrow n_j$ is a path between n_i and n_j, and $|n_i \rightsquigarrow n_j|$ is the length of the path. Finally, scoring individual search terms—the $score(t, T)$ component of the scoring function—uses a slight variation of pivoted normalization weighting [SCH+99]. Critically, all the component scores can be computed offline, which allows the top-k results to be identified efficiently.

SAINT Li *et al.* [LFZ08] propose tuple units as the atomic units for retrieval. Similar to EKSO's construction of virtual documents [SW05], a tuple unit comprises tuples related by foreign key references. Later work [FLW11] examined combining multiple tuple units to answer queries and proposed structure-aware indexes to identify results efficiently. Ranking tuple units is similar to EASE's scoring function [LOF+08]:

$$score(T, Q) = \sum_{i=1}^{|Q|} \left(\sum_{j=1}^{|Q|} similarity((t_i, t_j), T) \cdot \left(\frac{\ln(1 + tf)}{(1 - s) + s \cdot \left(\frac{dl}{avgdl} \right)} \cdot \ln \left(\frac{N + 1}{df + 1} \right) \right) \right)$$

where the collection statistics are taken over the tuple units derived from the database and the similarity factor is scored identically to EASE's similarity factor.

CSTree Li *et al.* [LFZW11] propose the compact Steiner tree to efficiently approximate the top-k solutions to the Steiner problem. A compact Steiner tree is a group Steiner tree whose terminals have the shortest path to the root. A given graph has at most $|V|$ compact Steiner trees; these trees can be recovered efficiently when the distance and path(s) from each term to vertices containing or related to that term are precomputed. Query results are obtained by ranking the roots of compact Steiner trees by their distance from the search terms and then reconstructing the path between the root and terminals that contain the search terms. The size of the index is minimized by not storing distances (or paths) longer than a predetermined threshold.

Query results are scored according to the following function:

$$score(T, Q) = \max_r \left(\sum_{t \in Q} (\alpha \cdot weight(r) + (1 - \alpha) \cdot relevance(r, t) + \beta weight(r \rightsquigarrow t)) \right)$$

where r is the root of the compact Steiner tree, t is a term in the query Q, $weight(r)$ is the prestige of the root (e.g., as determined by PageRank [BP98]), $weight(r \rightsquigarrow t)$ is the minimum distance from r to a vertex v that contains the term t,

$$relevance(r, t) = \begin{cases} \ln(1 + tf) \cdot \ln idf & \text{if } r \text{ contains } t \\ \max_v \left(probability(r \rightsquigarrow v) \cdot relevance(v, t) \right) & \text{otherwise} \end{cases}$$

and $probability(r \rightsquigarrow v)$ is the probability of randomly walking from r to v in the graph.[5]

B.4 Summary

The following paragraphs summarize the differences among the various approaches to support keyword search in relational databases. Performance is addressed first before comparing the various ranking strategies. Unfortunately, few papers published in the literature confine their work to improving only runtime performance or search effectiveness. The lure of improving both appears just too tantalizing for researchers to avoid. However, such changes make overall progress difficult to measure: consistent baselines are non-existent and even the simplest research questions have not been laid to rest. Most evaluations reported in the literature vary so many parameters at once that it is difficult to determine the reason(s) for improvement.

One issue that has been largely ignored in previous work is different semantics for identifying results. Proximity search traditionally implies strict AND semantics—a result is not valid unless it contains all the search terms. In contrast, IR-style ranking schemes use OR semantics, and their scoring functions reward results that contain more search terms. However, the question of semantics goes still deeper—some search techniques [LOF+08, QYCT09] define results to be subgraphs instead of trees. No user studies have been reported in the literature to determine how users perceive this difference and which approach they prefer although previous work in multimedia retrieval [dVKL04] indicates that the amount of irrelevant information should be minimized. Hence, query semantics remain an area where additional research is required.

[5] Li *et al.* do not specify how to calculate their inverse document frequency factor.

Table B.1: Algorithmic comparison of graph-based search techniques. A technique's performance ratio is its approximation bound on the ideal answer (i.e., computing the optimal group Steiner tree).

System	Ratio	Time	Memory
BANKS	$O(l)$	$O(n^2 \log n + n \cdot e)$	$O(l \cdot n^2)$
BANKS-II	$O(l)$	$O(n^2 \log n + n \cdot e)$	$O(l \cdot n)$
DPBF	1	$O(3^l n + 2^l((l + \log n) \cdot n + e))$	$O(2^l n)$
BLINKS	$O(l)$	⟨dependent on partitioning⟩	$O(\sum_b N_b^2 + BP)$
Golenberg *et al.*	2^a	$O((e + n \log n) \cdot l \cdot \lvert T_i \rvert)$	$O(l \cdot n)$
STAR	$O(\log l)$	$O(\frac{1}{\epsilon} \cdot \frac{\max_e weight(e)}{\min_e weight(e)} \cdot e \cdot t \cdot (n \log n + e))$	$O(l \cdot n)$

[a] The algorithm enumerates results by height rather than by weight.

Legend

n	number of nodes in data graph	T_i	number of nodes in ith result
e	number of edges in data graph	N_b	size of block b in index
t	number of unique terms in database	B	number of blocks in index
l	number of terms in query	P	number of node separators in graph partition

B.4.1 Performance

Table B.1 compares the graph-based approaches by worst-case execution time and memory requirements. As evidenced by the table, the theoretical execution time and memory consumption vary widely, and these upper bounds are unlikely to be realized in practice due to the heuristics that each technique uses to approximate the intractable group Steiner problem. Thus, while algorithmic analysis has been used in the literature to argue for the superiority of particular search techniques, the lack of tight (lower) bounds dictates that these approaches be evaluated empirically. Defining similar execution time bounds for the schema-based approaches is difficult due to their brute-force approach of creating all possible join expressions that might relate search terms. In particular, although the total number of such expressions grows exponentially with their size, it is difficult to predict how many must be executed for a particular query. The approaches that precompute potential results do relatively little work at query time, which obviates the need for traditional worst-case analysis of execution time.

B.4.2 Ranking

The wide variation in existing scoring functions and their performance in the empirical evaluation presented in Chapter 3 indicates that different schemes work better for different datasets or particular types of queries. Of particular interest is the distinction that more recent work gives to the root, leaves, and path between each search term and the root of the result tree. The simplicity of BANKS's egalitarian treatment of nodes and edges has disappeared as systems use alternative semantics to improve performance. With the distinction comes additional parameters as typified by BLINKS's scoring function, which contains three free parameters. Unfortunately, the existing literature does not identify the tradeoffs with regards to search effectiveness

among the various scoring functions. Other differences such as how to assign edge weights in the data graph have been almost completely ignored. Although different schemes exist, no existing evaluation has considered what effect this decision might have on its results.

Similar to the ranking schemes advocated by the various proximity search heuristics, the deviations between IR-style scoring functions have not been well-studied. As an example, consider how SPARK scores a result as a single unit instead of combining the scores of its constituent attributes. Robertson *et al.* [RZT04] had previously established the theoretical superiority of this approach, but in the context of relational keyword search, efficient enumeration dictates the estimation of several factors including average document length, inverse document frequency, and the number of documents generated by each candidate network. A small study of the approximation error [LLWZ07b] suggested acceptable accuracy (usually within 30% relative error) but also found instances where the error was substantial (e.g., 700% relative error). Whether or not the approximation error negates the theoretical advantage proffered by the approach was not addressed, which is a significant shortcoming. Other differences such as calculating collection statistics from a single table or the entire database [LYMC06] are not investigated individually in evaluations. Hence, it is difficult to ascertain which changes to the scoring function account for reported improvements in search effectiveness. Again, previous work supplies a variety of options but provides little insight into why particular ranking schemes outperform others in the evaluation of these systems.

Bibliography

[ABC+76] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: Relational Approach to Database Management. *ACM Transactions on Database Systems*, 1:97–137, June 1976.

[ACD02] Sanjay Agrawal, Sarajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 5–16, February 2002.

[AH07] Jesse Alpert and Nissan Hajaj. We knew the web was big.... http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html, July 2007.

[AHE+04] Brain S. Alper, Jason A. Hand, Susan G. Elliott, Scott Kinkade, Michael J. Hauan, Daniel K. Onion, and Bernard M. Sklar. How much effort is needed to keep up with the literature relevant for primary care? *Journal of the Medical Library Association*, 92(4):429–437, October 2004.

[AYCR+05] Sihem Amer-Yahia, Pat Case, Thomas Rölleke, Jayavel Shanmugasundaram, and Gerhard Weikum. Report on the DB/IR panel at SIGMOD 2005. *SIGMOD Record*, 34(4):71–74, December 2005.

[Ayr07] Ian Ayres. *Super Crunchers: Why Thinking-by-Numbers Is the New Way to Be Smart*. Bantam Dell, New York, NY, 2007.

[BAC+81] M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, and R. A. Yost. System R: An architectural overview. *IBM Systems Journal*, 20(1):41–62, 1981.

[BBB+71] Charles W. Bachman, Richard E. Batchelor, I. Marvin Beriss, Charles R. Blose, Turgut I. Burakreis, Vincent Delia Valle, George G. Dodd, William Helgeson, John Lyon, A. (Tax) Metaxides, Gerald E. McKinzie, Paul Siegel, Warren G. Simmons, Larry L. Sturgess, Harrison Tellier, Sharon B. Weinberg, and George T. Werner. *COnference of DAta SYstems Languages (CODASYL) Data Base Task Group Report*. ACM, New York, NY, October 1971.

[BCF+07] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. Technical report, W3C, January 2007. http://www.w3.org/TR/xquery/.

[Ber01] Michael K. Bergman. The Deep Web: Surfacing Hidden Value. *The Journal of Electronic Publishing*, 7:1–17, August 2001.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 431–440, February 2002.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 564–575, August 2004.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998.

[BRL+10] Ankanksha Baid, Ian Rae, Jiexing Li, AnHai Doan, and Jeffrey Naughton. Toward Scalable Keyword Search over Relational Data. *Proceedings of the VLDB Endowment*, 3(1):140–149, 2010.

[Bro02] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[BS09] Roger E. Bohn and James E. Short. How Much Information? 2009 Report on American Consumers. Technical report, Global Information Industry Center, University of California, San Diego, December 2009.

[BSR+05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to Rank using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 89–96, August 2005.

[BTN11] Veli Bicer, Thanh Tran, and Radoslav Nedkov. Ranking Support for Keyword Search on Structured Data using Relevance Models. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1669–1678, October 2011.

[Bus45] Vannevar Bush. As We May Think. *Atlantic Monthly*, 176:101–108, July 1945.

[BV04] Paolo Boldi and Sebastiano Vigna. The WebGraph Framework I: Compression Techniques. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 595–602, May 2004.

[BW64] C. W. Bachman and S. B. Williams. A General Purpose Programming System for Random Access Memories. In *Proceedings of AFIPS 1964 Fall Joint Computer Conference*, volume 26 of *AFIPS '64 (Fall, part I)*, pages 411–422, October 1964.

[CAE+76] D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development*, 20(6):560–575, November 1976.

[CBC+09] Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, and Jeffrey Naughton. Combining Keyword Search and Forms for Ad Hoc Querying of Databases. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 349–360, June 2009.

[CCT00] Charles L. A. Clarke, Gordon V. Cormack, and Elizabeth A. Tudhope. Relevance ranking for one to three term queries. *Information Processing and Management*, 36(2):291–311, 2000.

[CD09] Surajit Chaudhuri and Gautam Das. Keyword Querying and Ranking in Databases. *Proceedings of the VLDB Endowment*, 2:1658–1659, August 2009.

[Cis11] Cisco Visual Networking Index: Forecast and Methodology, 2010–2015. Technical report, Cisco, 2011.

[CKC+08] Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and Diversity in Information Retrieval Evaluation. In *SIGIR '08: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 659–666, July 2008.

[Cla05] Charles L. A. Clarke. Controlling Overlap in Content-Oriented XML Retrieval. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 314–321, August 2005.

[Cle97] Cyril Cleverdon. The Cranfield Tests on Endex Language Devices. In *Readings in Information Retrieval*, pages 47–59. 1997.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13:377–387, June 1970.

[Coh98] William W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 201–212, June 1998.

[Col08] Graham P. Collins. The Discovery Machine. *Scientific American*, 298(2):39, 2008.

[com10] comScore. Global Search Market Grows 46 Percent in 2009. http://www.comscore.com/Press_Events/Press_Releases/2010/1/Global_Search_Market_Grows_46_Percent_in_2009, January 2010.

[Cor10] Anne-Marie Corley. Real-Time Search Stumbles Out of the Gate. *IEEE Spectrum*, February 2010. Online: http://spectrum.ieee.org/telecom/internet/realtime-search-stumbles-out-of-the-gate.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proceedings of the 2005 Conference on Innovative Data Systems Research*, CIDR '05, pages 1–12, January 2005.

[CW10a] Joel Coffman and Alfred C. Weaver. A Framework for Evaluating Database Keyword Search Strategies. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 729–738, October 2010.

[CW10b] Joel Coffman and Alfred C. Weaver. Structured Data Retrieval using Cover Density Ranking. In *KEYS '10: Proceedings of the 2nd International Workshop on Keyword Search on Structured Data*, pages 1–6, June 2010.

[CW11a] Joel Coffman and Alfred C. Weaver. An Empirical Performance Evaluation of Relational Keyword Search Systems. Technical Report CS-2011-07, University of Virginia, 2011.

[CW11b] Joel Coffman and Alfred C. Weaver. Learning to Rank Results in Relational Keyword Search. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, October 2011.

[CW12] Joel Coffman and Alfred C. Weaver. What Are We Searching For? Analyzing User Objectives When Searching Relational Data. In *Proceedings of the 2nd Workshop on Web Search Click Data*, WSCD '12, February 2012.

[CWLL09] Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. Keyword Search on Structured and Semi-Structured Data. In *Proceedings of the 35th SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 1005–1010, June 2009.

[CXL+06] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting Ranking SVM to Document Retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 186–193, August 2006.

[DEGP98] Shaul Dar, Gadi Entin, Shai Geva, and Eran Palmon. DTL's DataSpot: Database Exploration Using Plain Language. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 645–649, August 1998.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, January 2008.

[Dij59] Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DKS08] Bhavana Bharat Dalvi, Meghana Kshirsagar, and S. Sudarshan. Keyword Search on External Memory Data Graphs. *Proceedings of the VLDB Endowment*, 1(1):1189–1204, 2008.

[DLK06] Hoa Trang Dang, Jimmy Lin, and Diane Kelly. Overview of the TREC 2006 Question Answering Track. In *TREC 2006: Proceedings of the 16th Text REtrieval Conference*, November 2006.

[Dod69] George G. Dodd. Elements of Data Management Systems. *ACM Computing Surveys*, 1:117–133, June 1969.

[Dun97] M. D. Dunlop. Time, Relevance and Interaction Modelling for Information Retrieval. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '97, pages 206–213, July 1997.

[dVKL04] Arjen P. de Vries, Gabriella Kazai, and Mounia Lalmas. Tolerance to Irrelevance: A user-effort oriented evaluation of retrieval systems without predefined retrieval unit. In *Recherche d'Informations Assisteee par Ordinateur*, RAIO '04, April 2004.

[DW71] S. E. Dreyfus and R. A. Wagner. The Steiner Problem in Graphs. *Networks*, 1(3):195–207, 1971.

[DYW+07] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding Top-k Min-Cost Connected Trees in Databases. In *ICDE '07: Proceedings of the 23rd International Conference on Data Engineering*, pages 836–845, April 2007.

[Eco10] Special report: Managing data. *The Economist*, 394(8671):3–5, 2010.

[Fal08] Deborah Fallows. Search Engine Use. Technical report, Pew Internet and American Life Project, August 2008. http://www.pewinternet.org/Reports/2008/Search-Engine-Use.aspx.

[FGKL02] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas. INEX: Initiative for the Evaluation of XML Retrieval. In *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, August 2002.

[FLW11] Jianhua Feng, Guoliang Li, and Jianyong Wang. Finding Top-k Answers in Keyword Search over Relational Databases Using Tuple Units. *IEEE Transactions on Knowledge and Data Engineering*, 23(12):1781–1794, December 2011.

[Fox02] Susannah Fox. Search engines. Technical report, Pew Internet and American Life Project, July 2002. http://www.pewinternet.org/Reports/2002/Search-Engines.aspx.

[FR97] Norbert Fuhr and Thomas Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems*, 15:32–66, January 1997.

[FTZ04] Hui Fang, Tao Tao, and ChengXiang Zhai. A Formal Study of Information Retrieval Heuristics. In *SIGIR '04: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 49–56, July 2004.

[GCM+08] John F. Gantz, Christopher Chute, Alex Manfrediz, Stephen Minton, David Reinsel, Wolfgang Schlichting, and Anna Toncheva. The Diverse and Exploding Digital Universe. Technical report, International Data Corporation (IDC), March 2008.

[GCS+07] John F. Gantz, Christopher Chute, Wolfgang Schlichting, John McArther, Stephen Minton, Irida Xheneti, Anna Toncheva, and Alex Manfrediz. The Expanding Digital Universe. White paper, International Data Corporation (IDC), March 2007.

[GI95] Jerrold W. Grossman and Patrick D. F. Ion. On a Portion of the Well-Known Collaboration Graph. *Congressus Numerantium*, 108:129–131, 1995.

[Gil05] Mark L. Gillenson. *Fundamentals of Database Management Systems*. John Wiley & Sons, Inc., 2005.

[GKS08] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword Proximity Search in Complex Data Graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 927–940, June 2008.

[GLQL07] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature Selection for Ranking. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 407–414, July 2007.

[GR11] John Gantz and David Reinsel. Extracting Value from Chaos. Technical report, International Data Corporation (IDC), June 2011.

[GSVGM98] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity Search in Databases. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 26–37, August 1998.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style Keyword Search over Relational Databases. In *Proceedings of the 29th International Conference on Very Large Data Bases*, VLDB '03, pages 850–861, September 2003.

[HHP06] Heasoo Hwang, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: A System for Authority-based Search on Databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 796–798, June 2006.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of the 29th International Conference on Very Large Data Bases*, VLDB '02, pages 670–681. VLDB Endowment, August 2002.

[HWYY07] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 305–316, June 2007.

[Jav11] Java Platform, Standard Edition 7 API Specification. http://docs.oracle.com/javase/7/docs/api/, 2011. Retrieved 2 February 2012.

[JBS08] Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. Determining the informational, navigational, and transactional intent of Web queries. *Information Processing & Management*, 44(3):1251–1266, May 2008.

[JCE+07] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making Database Systems Usable. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 13–24, June 2007.

[Joa02] Thorsten Joachims. Optimizing Search Engines using Clickthrough Data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, July 2002.

[Joa06] Thorsten Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, August 2006.

[JS06] Bernard J. Jansen and Amanda Spink. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Information Processing and Management*, 42(1):248–263, 2006.

[JSBS98] Bernard J. Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real Life Information Retrieval: A Study Of User Queries On The Web. *SIGIR Forum*, 32(1):5–17, April 1998.

[JYP88] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On Generating All Maximal Independent Sets. *Information Processing Letters*, 27(3):119 – 123, 1988.

[KA05] Mika Käki and Anne Aula. Findex: improving search result use through automatic filtering categories. *Interacting with Computers*, 17(2):187–206, 2005.

[Ken38] M.G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, June 1938.

[KJ97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–324, 1997.

[KJAA05] Jaana Kekäläinen, Marko Junkkari, Paavo Arvola, and Timo Aalto. TRIX 2004 – Struggling with the Overlap. In Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltán Szlávik, editors, *Advances in XML Information Retrieval*, volume 3493 of *Lecture Notes in Computer Science*, pages 145–162. Springer Berlin / Heidelberg, 2005.

[Kle99] Jon M. Kleinberg. Hubs, Authorities, and Communities. *ACM Computing Surveys*, 31, December 1999.

[Koh95] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1137–1143, San Francisco, CA, 1995. Morgan Kaufmann Publishers Inc.

[KPC+05] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 505–516, August 2005.

[KRS+09] Gjergji Kasneci, Maya Ramanath, Mauro Sozio, Fabian M. Suchanek, and Gerhard Weikum. STAR: Steiner-Tree Approximation in Relationship Graphs. In *Proceedings of the 25th International Conference on Data Engineering*, ICDE '09, pages 868–879, March 2009.

[KSI+08] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 953–962, April 2008.

[Law10a] Ryan Lawler. Facebook: 2B Videos Viewed Per Month. http://gigaom.com/video/facebook-2b-videos-viewed-per-month/, June 2010.

[Law10b] Ryan Lawler. Facebook was the top search term in 2010 for second straight year, December 2010.

[LCVA02] Wen-Syan Li, K. Selcuk Candan, Quoc Vu, and Divyakant Agrawal. Query Relaxation by Structure and Semantics for Retrieval of Logical Web Documents. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):768–791, July 2002.

[Lea10] Neal Leavitt. Will NoSQL Databases Live Up to Their Promise? *Computer*, 43:12–14, February 2010.

[LFZ08] Guoliang Li, Jianhua Feng, and Lizhu Zhou. RETUNE: Retrieving and materializing tuple units for effective keyword search over relational databases. In Qing Li, Stefano Spaccapietra, Eric Yu, and Antoni Olivé, editors, *Conceptual Modeling - ER 2008*, volume 5231 of *Lecture Notes in Computer Science*, pages 469–483. Springer Berlin / Heidelberg, 2008.

[LFZW11] Guoliang Li, Jianhua Feng, Xiaofang Zhou, and Jianyong Wang. Providing built-in keyword search capabilities in RDBMS. *The VLDB Journal*, 20:1–19, February 2011.

[LKK+07] Mounia Lalmas, Gabriella Kazai, Jaap Kamps, Jovan Pehcevski, Benjamin Piwowarski, and Stephen Robertson. INEX 2006 Evaluation Measures. In Norbert Fuhr, Mounia Lalmas, and Andrew Trotman, editors, *Comparative Evaluation of XML Information Retrieval Systems*, volume 4518 of *Lecture Notes in Computer Science*, pages 20–34. Springer Berlin / Heidelberg, 2007.

[LLWZ07a] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. SPARK: Top-k Keyword Query in Relational Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 115–126, June 2007.

[LLWZ07b] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. SPARK: Top-k Keyword Query in Relational Databases. Technical Report UNSW-CSE-TR-0708, The University of New South Wales, March 2007.

[LOF+08] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 903–914, June 2008.

[LT07] Mounia Lalmas and Anastasios Tombros. INEX 2002 - 2006: Understanding XML Retrieval Evaluation. In Costantino Thanos, Francesca Borri, and Leonardo Candela, editors, *Digital Libraries: Research and Development*, volume 4877 of *Lecture Notes in Computer Science*, pages 187–196. Springer Berlin / Heidelberg, 2007.

[Luh57] H.P. Luhn. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM Journal of Research and Development*, 1(4):309–317, October 1957.

[LV03] Peter Lyman and Hal R. Varian. How Much Information. Technical report, University of California at Berkeley, 2003. Available from http://www.sims.berkeley.edu/how-much-info-2003.

[LWL+11] Yi Luo, Wei Wang, Xuemin Lin, Xiaofang Zhou, Jianmin Wang, and Keqiu Li. SPARK2: Top-k Keyword Query in Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 23:1763–1780, December 2011.

[LYMC06] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective Keyword Search in Relational Databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 563–574, June 2006.

[May99] Wolfgang May. Information Extraction and Integration with Florid: The Mondial Case Study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from http://dbis.informatik.uni-goettingen.de/Mondial.

[MCB+11] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. Big data: The next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute, May 2011.

[McG77] W. C. McGee. The information management system IMS/VS, Part I: General structure and operation. *IBM Systems Journal*, 16(2):84–95, 1977.

[MHS+71] A. Metaxides, W. B. Helgeson, R. E. Seth, G. C. Bryson, M. A. Coane, G. G. Dodd, C. P. Earnest, R. W. Engles, L. N. Harper, P. A. Hartley, D. J. Hopkin, J. D. Joyce, S. C. Knapp, J. R. Lucking, J. M. Muro, M. P. Persily, M. A. Ramm, J. F. Russell, R. F. Schubert, J. R. Sidlo, M. M. Smith, and G. T. Werner. *COnference of DAta SYstems Languages (CODASYL) Data Base Task Group Report*. ACM, New York, NY, April 1971.

[Mil67] Stanley Milgram. The Small-World Problem. *Psychology Today*, 1:61–67, May 1967.

[Mit98] Antonija Mitrovic. Learning SQL with a Computerized Tutor. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, pages 307–311, February 1998.

[MM05] Yosi Mass and Matan Mandelbrod. Component Ranking and Automatic Query Refinement for XML Retrieval. In Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltn Szlvik, editors, *Advances in XML Information Retrieval*, volume 3493 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin / Heidelberg, 2005.

[Moo61] Calvin Mooers. From a point of view of mathematical etc. techniques. In R. A. Fairthorne, editor, *Towards information retrieval*, pages xvii–xxiii. Butterworths, London, 1961.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, 2008.

[MS12] Yosi Mass and Yehoshua Sagiv. Language Models for Keyword Search over Data Graphs. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 363–372, February 2012.

[MYP07] Alexander Markowetz, Yin Yang, and Dimitris Papadias. Keyword Search on Relational Data Streams. In *Proceedings of the 2007 ACM International Conference on Management of Data*, SIGMOD '07, pages 605–616, New York, NY, June 2007.

[MZ11] Mary Madden and Kathryn Zickuhr. 65% of online adults use social networking sites. Technical report, Pew Research Center, August 2011. Online: http://pewinternet.org/Reports/2011/Social-Networking-Sites.aspx.

[NBY97] Gonzalo Navarro and Ricardo Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems*, 15:400–435, October 1997.

[Net12] February 2012 Web Server Survey. http://news.netcraft.com/archives/2012/02/07/february-2012-web-server-survey.html, February 2012.

[New01] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

[NJ09] Arnab Nandi and H. V. Jagadish. Qunits: queried units for database search. In *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*, CIDR '09, January 2009.

[Paş07] Marius Paşca. Organizing and Searching the World Wide Web of Facts - Step Two: Harnessing the Wisdom of the Crowds. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 101–110, New York, NY, USA, May 2007. ACM.

[PCT06] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A Picture of Search. In *InfoScale '06: Proceedings of the 1st International Conference on Scalable Information Systems*, May 2006.

[Pea20] Karl Pearson. Notes on the History of Correlation. *Biometrika*, 13(1):25–45, October 1920.

[PLB+06] Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and Searching the World Wide Web of Facts - Step One: the One-Million Fact Extraction Challenge. In *Proceedings of the 21st National Conference on Artificial Intelligence*, AAAI '06, pages 1400–1405. AAAI Press, July 2006.

[QYC09] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Keyword Search in Databases: The Power of RDBMS. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 681–694, June 2009.

[QYCT09] Lu Qin, J.X. Yu, Lijun Chang, and Yufei Tao. Querying Communities in Relational Databases. In *Proceedings of the 25th International Conference on Data Engineering*, ICDE '09, pages 724–735, March 2009.

[RC96] Daniel E. Rose and D. R. Cutting. Ranking for Usability: Enhanced Retrieval for Short Queries. Technical Report 163, Apple, 1996.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the Hidden Web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 129–138, San Francisco, CA, September 2001. Morgan Kaufmann Publishers Inc.

[RL04] Daniel E. Rose and Danny Levinson. Understanding User Goals in Web Search. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 13–19, May 2004.

[RN88] Joseph Lee Rodgers and W. Alan Nicewander. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician*, 42(1):59–66, February 1988.

[Rob77] S. E. Robertson. The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4):294–304, 1977.

[RPB06] Matthew Richardson, Amit Prakash, and Eric Brill. Beyond PageRank: Machine Learning for Static Ranking. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 707–715, May 2006.

[RW90] Gabriele Reich and Peter Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 1990.

[RZT04] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *CIKM '04: Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 42–49, November 2004.

[Sal71] Gerard Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., 1971.

[SCH+99] Amit Singhal, John Choi, Donald Hindle, David Lewis, and Fernando Pereira. AT&T at TREC-7. In *Proceedings of the Seventh Text REtrieval Conference (TREC-7)*, pages 239–252, November 1999.

[sch09] m.c. schraefel. Building Knowledge: What's beyond Keyword Search? *Computer*, 42(3):52–59, March 2009.

[Sci11] Challenges and Opportunities. *Science*, 331(6018):692–693, February 2011.

[SCL+99] Marc M. Sebrechts, John V. Cugini, Sharon J. Laskowski, Joanna Vasilakis, and Michael S. Miller. Visualization of Search Results: A Comparative Evaluation of Text, 2D, and 3D Interfaces. In *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–10, August 1999.

[Seb02] Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34:1–47, March 2002.

[SFW83] Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[She10] Zack Sheppard. 5,000,000,000. http://blog.flickr.net/en/2010/09/19/5000000000/, September 2010.

[SHWK76] Michael Stonebraker, Gerald Held, Eugene Wong, and Peter Kreps. The Design and Implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222, September 1976.

[Sin01] Amit Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, December 2001.

[SJWS02] Amanda Spink, Bernard J. Jansen, Dietmar Wolfram, and Tefko Saracevic. From E-Sex to E-Commerce: Web Search Changes. *Computer*, 35(3):107–109, March 2002.

[SKS06] Avi Silberschatz, Hank Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 5 edition, 2006.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, May 2007.

[SMA+07] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The End of an Architectural Era: (It's Time for a Complete Rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1150–1160. VLDB Endowment, September 2007.

[SOSL04] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin. SQLator: An Online SQL Learning Workbench. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '04, pages 223–227, June 2004.

[Spe04] C. Spearman. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology*, 15(1):72–101, January 1904.

[Sri10] Divesh Srivastava. Schema Extraction. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 3–4, October 2010.

[Sto10] Michael Stonebraker. SQL databases v. NoSQL databases. *Communications of the ACM*, 53:10–11, April 2010.

[SW05] Qi Su and Jennifer Widom. Indexing Relational Database Content Offline for Efficient Keyword-Based Search. In *Proceedings of the 9th International Database Engineering & Application Symposium*, IDEAS '05, pages 297–306, July 2005.

[TL76] D. C. Tsichritzis and F. H. Lochovsky. Hierarchical Data-Base Management: A Survey. *ACM Computing Surveys*, 8:105–123, March 1976.

[TM69] Jeffrey Travers and Stanley Milgram. An Experimental Study of the Small World Problem. *Sociometry*, 32(4):425–443, December 1969.

[TRE] TREC Overview. http://trec.nist.gov/overview.html.

[Tur94] Howard Turtle. Natural Language vs. Boolean Query Evaluation: A Comparison of Retrieval Performance. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 212–220, July 1994.

[Twi11] 200 million Tweets per day. http://blog.twitter.com/2011/06/200-million-tweets-per-day.html, June 2011.

[TZC+06] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation Methods for Ranking Functions with Multiple Parameters. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 585–593, November 2006.

[Voo02] Ellen M. Voorhees. The Philosophy of Information Retrieval Evaluation. In *CLEF '01: Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems*, pages 355–370. Springer-Verlag, 2002.

[VPG05] Jean-Nol Vittaut, Benjamin Piwowarski, and Patrick Gallinari. An Algebra for Structured Queries inBayesianNetworks. In Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltn Szlvik, editors, *Advances in XML Information Retrieval*, volume 3493 of *Lecture Notes in Computer Science*, pages 91–106. Springer Berlin / Heidelberg, 2005.

[Wal10] Hunter Walker. Great Scott! Over 35 Hours of Video Uploaded Every Minute to YouTube. http://youtube-global.blogspot.com/2010/11/great-scott-over-35-hours-of-video.html, November 2010.

[Web10] William Webber. Evaluating the Effectiveness of Keyword Search. *IEEE Data Engineering Bulletin*, 33(1):54–59, 2010.

[Wei07] Gerhard Weikum. DB&IR: Both Sides Now. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 25–30, June 2007.

[Wil94] Ross Wilkinson. Effective Retrieval of Structured Documents. In *SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 311–317, August 1994.

[WMZ08] William Webber, Alistair Moffat, and Justin Zobel. Statistical Power in Retrieval Experimentation. In *Proceeding of the 17th ACM International Conference on Information and Knowledge Management*, CIKM '08, pages 571–580, October 2008.

[WSJS01] Dietmar Wolfram, Amanda Spink, Bernard Jansen, and Tefko Saracevic. Vox Populi: The Public Searching of the Web. *Journal of the American Society for Information Science and Technology*, 52(12):1073–1074, 2001.

[WZSD95] Ross Wilkinson, Justin Zobel, and Ron Sacks-Davis. Similarity Measures for Short Queries. In *Fourth Text REtrieval Conference (TREC-4)*, pages 277–285, November 1995.

[XIG09] Yanwei Xu, Yoshiharu Ishikawa, and Jihong Guan. Effective Top-k Keyword Search in Relational Databases Considering Query Semantics. In Lei Chen, Chengfei Liu, Xiao Zhang, Shan Wang, Darijus Strasunskas, Stein L. Tomassen, Jinghai Rao, Wen-Syan Li, K. Seluk Candan, Dickson K.W. Chiu, Yi Zhuang, Clarence A. Ellis, and Kwang-Hoon Kim, editors, *Advances in Web and Network Technologies, and Information Management*, volume 5731 of *Lecture Notes in Computer Science*, pages 172–184. Springer, 2009.

[YPS09] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Summarizing Relational Databases. *Proceedings of the VLDB Endowment*, 2:634–645, August 2009.

[YQC10] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. *Keyword Search in Databases*. Morgan and Claypool Publishers, 1st edition, 2010.

[YS10] Xiaoxin Yin and Sarthak Shah. Building Taxonomy of Web Search Intents for Name Entity Queries. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1001–1010, New York, NY, USA, April 2010. ACM.

[Zlo75] Moshé M. Zloof. Query by Example. In *Proceedings of the National Computer Conference and Exposition*, AFIPS '75, pages 431–438, May 1975.

Acronyms

ACID atomicity, consistency, isolation, and durability.

AOL America OnLine.

API application programming interface.

DB database.

DBLP Digital Bibliography & Library Project.

DBMS database management system.

DCG discounted cumulative gain.

EMR electronic medical record.

IMDb Internet Movie Database.

INEX Initiative for the Evaluation of XML retrieval.

IR information retrieval.

JVM Java VM.

MAP mean average precision.

MB megabyte.

MRR mean reciprocal rank.

nDCG normalized discounted cumulative gain.

OLAP online analytical processing.

OLTP online transaction processing.

QBE query by example.

RDBMS relational database management system.

SQL Structured Query Language.

SSN social security number.

SVM support vector machine.

TREC the Text REtreival Conference.

URL uniform resource locator.

VM virtual machine.

XML eXtensible Markup Language.

Glossary

11-point interpolated average precision The mean of interpolated precision at the recall levels of $0.0, 0.1, 0.2, \ldots, 1.0$ for each information need in the collection.

attribute Labeled domain of a relation; conceptually, a column of a table.

average precision The mean of the precision values calculated after each relevant result is retrieved. A value of 0 is assigned to any relevant result that is not retrieved. See also precision.

candidate network A connected tree of database relations where the database schema contains a foreign key for adjacent relations.

coordination level The number of terms common to both a document and query.

cross validation Statistical technique used to assess how the results of a statistical analysis will generalize to an independent data set. Example data is partitioned into complementary subsets with the analysis being performed on one subset (the training data) and validated against the other subset (the testing data).

data independence Decoupling of data storage and access from application software.

data integration The ability to combine related pieces of data within an information system.

data integrity Accuracy of the data contained within an information system.

database A highly-organized collection of data related to an enterprise.

database management system A set of programs that controls access to a database.

DB&IR Research field focused on integrating database and IR technologies.

deep web Online content not indexed by search engines. This content is typically accessible via searchable databases. Also known as the *invisible web* or *hidden web*.

discounted cumulative gain A search effectiveness metric where the relevance of a result is penalized proportionally to its position in the list of results (because the user may abandon the search before seeing the result).

document In the context of IR, whatever unit(s) over which a retrieval system operates.

entity Real-world object or event from an environment to be modeled.

execution time The elapsed wall-clock time of an algorithm for a given problem instance.

expanded schema graph All tuple sets and free tuple sets connected by the relationship(s) among the relations in the schema graph [YQC10]. See also schema graph.

foreign key An attribute of a relation that references the value of a primary key of another relation.

hidden web See deep web.

information need The topic about which the user desires more information.

information nugget A fact or atomic piece of information pertaining to an information need. See Dang *et al.* [DLK06].

interpolated precision Given a recall level r ($\in [0, 1]$), the highest precision for any recall level $r' \geq r$.

machine learning Algorithms that infer characteristics of an unknown distribution from example data; generalizing from the given examples allows useful output even for unseen data. See also supervised learning.

mean average precision The mean of average precision values calculated for each query in an evaluation workload.

mean reciprocal rank The mean of reciprocal rank values calculated for each query in an evaluation workload.

normalization Process of organizing relations to minimize redundancy.

normalized discounted cumulative gain Discounted cumulative gain (DCG) normalized by dividing by the ideal DCG at that position.

ordinal regression A type of regression used to predict an ordinal variable (i.e., a variable where only the relative ordering between different values is significant).

pooling Process by which relevance assessments are collected for a document collection and query. Typically, the top-k results returned by a number of retrieval systems are assessed; other results are assumed to be irrelevant.

precision The proportion of retrieved results that are relevant. See also recall.

precision @ k The proportion of retrieved results that are relevant within the first k results returned by a retrieval system.

primary key One or more attributes of a database relation that uniquely identify a tuple of that relation.

query The expression of an information need that is used as input to a retrieval system.

recall The proportion of relevant results retrieved by a system. See also precision.

reciprocal rank The reciprocal of the position of the highest-ranked relevant result returned by a system.

referential integrity Ensures that a value that appears in one relation for a given set of attributes also appears for a set of attributes in another relation [SKS06].

regression Statistical techniques that predict the relationship between a dependent variable and one or more independent variables.

relation A subset of the Cartesian product of a list of domains. A relation comprises a heading and an unordered set of tuples that share the same types (i.e., the values are drawn from the same list of domains); conceptually, a table with no duplicate rows.

relational data See relational model.

relational database A database based on the relational model.

relational database management system A software package for storing and retrieving data that gives the end user the impression that the data is well-integrated even though the data can be stored with no redundancy [Gil05]. See also database management system.

relational keyword search Intuitively, search techniques that connect disparate data (i.e., tuples) via relationships present in a relational database.

relational model An abstract model where all data and the relationships among that data are represented via relations. Each relation comprises fixed-format records (i.e., tuples).

relevant Containing information of value with respect to an information need.

response time The (typically wall-clock) time elapsed from initially requesting a service until its associated response. While this value may be the same as execution time, response time can also measure the time required for an algorithm that computes results incrementally to complete some portion of the total amount of work.

schema The formal description of a database's structure; its logical design.

schema graph A graph representation of the database schema where vertices model relations and edges represent foreign keys between relations.

semi-structured data Data that does not conform to the formal structure of a database model but contains markup to identify semantic structure. See also XML.

structured data See relational data.

structured document See structured data.

supervised learning Machine learning that infers a function from labeled training data.

tuple A single entry in a relation; conceptually, a table row.

tuple set Literally a set of tuples from a relation. For schema-based search techniques, a tuple set is the set of tuples from each relation that contain a search term.

free tuple set The set of tuples that do not contain any search terms.

non-free tuple set A tuple set that is not a free tuple set.

unstructured data Data that does not contain semantic markup to denote its structure. See also semi-structured data and relational data.

unstructured document See unstructured data.