PRINCIPLES OF USABLE QUERY INTERFACES

BY

ARASH TERMEHCHY

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Marianne Winslett, Chair
Professor Jiawei Han
Professor ChengXiang Zhai
Professor H.V. Jagadish, University of Michigan, Ann Arbor

Abstract

People generate and share a huge variety of information that can be used to derive useful insights and create value for society. Keyword queries are by far the most popular method for users to search, explore, and understand such information. Because search is more effective when information is well organized, people often choose to add structure to their information, producing semi-structured or even structured data. However, the new structure is not always added in the right places, i.e., where it will maximize the improvement in search results. Further, seemingly unimportant changes in the structure of the information tend to affect search results in undesirable ways. In addition, current keyword search and exploration systems are not very effective for certain types of information and queries. This thesis addresses these three issues by showing how to determine where to add structure to information for maximal benefit, and how to use the resulting structure to provide effective and robust answers to keyword queries.

Our first contribution is to determine what structure should be imposed on a given collection of information. Since the resources for converting information to a more structured format are always limited, our goal is to impose structure in a manner that will maximize the improvement for subsequent queries. In more technical terms, this means discovering and maintaining exactly the parts of a database schema that will most improve users' search and exploration experience. We show that adding structure corresponding to the most "popular" parts of a schema does not generally maximize search effectiveness. We formalize the problem of deciding which parts of a schema to extract and materialize, prove that the problem is NP-hard, propose polynomial-time solution algorithms, and prove that their effectiveness is within a constant factor of optimal.

Given a collection of semi-structured information, our next contribution is a principled way to take this structure into account when answering keyword queries. Because the same information can be represented in many ways, we propose the principle of *design independence*, which postulates that search results should not depend on the details of the design chosen for the information's structure. Otherwise, search may be effective over one schema design but provide poor rankings over another equivalent design. We formalize this principle and show that current search approaches for semi-structured information are not design independent. We propose a new approach to keyword search over semi-structured information, prove that it is design independent, and show that it gives better search results in practice than all previous approaches.

A schema describes the concepts that underlie a collection of information, but users do not usually explicitly specify these concepts in keyword queries. Thus the query term *Java* could refer to an island, a programming language, or a drink. When query terms can refer to multiple concepts, it is hard to devise effective heuristics to determine which answers should rank highest. In this thesis, we use the probability ranking principle and other widely accepted principles to provide a solid formal model for ranking the answers to such queries over semi-structured data. In an extensive empirical study, we show that our method generally provides more effective rankings than previous approaches.

Schemas provide information about concepts and the relationships between them. Current keyword search approaches generally assume that all types of relationships are equally important to users, which can lead to poor ranking of answers. We provide a way to quantify the importance of relationships between concepts, provide an efficient way to use this metric to rank answers, and show that users prefer the resulting rankings over those of previous methods.

If a query is very broad and underspecified (e.g., *vacation*), then even well-ranked answers are unlikely to please users. In such situations, we believe that the best approach is to detect that this is a difficult query and then use other techniques, such as query completion suggestions, to improve the user's experience. We provide a probabilistic model that accurately predicts how underspecified a query is, together with algorithms that quickly compute this measure when a user queries semi-structured data.

To My Wife and My Parents.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Marianne Winslett. From giving me the freedom to pursue a research topic of my own choice and guiding me through the difficult research problems, to improving my writing and presentation skills or giving professional and personal advice, she has been of immense help all throughout these years of graduate school. This thesis would not have been possible if not for her help, guidance, support and encouragement. She has taught me not only how to be a good student, but rather how to be a good professional. I feel privileged to have her as my advisor.

I would like to thank my committee member Professor Jiawei Han for his support and insightful comments on my research and job hunting. I greatly appreciate his efforts in creating a productive environment for data and information management research in the University of Illinois at Urbana-Champaign. I am also extremely thankful to another of my committee members, Professor ChengXiang Zhai, who introduced me to the subject of probabilistic models for retrieval. Finally, I would like to express my gratitude to my other committee member, Professor H.V. Jagadish, for the insightful discussions on usability issues in data management. I am thankful to Professor Vagelis Hristidis for his helpful comments and discussions on the problems of keyword search over database systems. I also thank Professor Mehdi Harandi, the associate head of the department of computer science, who helped me through some difficult periods in my graduate studies.

I thank my collaborators Austin Gibbons, Yodsawalai Chodpathumwan, Ali Vakilian, and Shiwen Cheng for their hard work that helped with many parts of this thesis. I also thank my groupmates Sruthi Bandhakavi, Joana Trindade, Ragib Hasan, Adam J. Lee, Kazuhiro Minami, Rishi Sinha, Soumyadeb Mitra, Huong Vu Thanh Luu, and Naoki Tanaka for providing useful feedback on my work. I am thankful to our summer interns, Adarsh Prasad, Soravit Changpinyo, Prachi Jain, Naga Varun Dasari, Alan Xia, and Di Xu, who helped me to perform extensive user studies and experiments for my thesis. I also thank the graduate students in the data and information research laboratory, Yizhou Sun, Zhenhui Li, Peixiang Zhao, Manish Gupta, Tim Weninger, Xiao Yu, Ming Ji, Bolin Ding, and Yanen Li, for sitting through my talks and providing useful comments. Donna Coleman provided invaluable assistance in every aspect of my graduate life, from arranging for conference travel to deciphering the complex departmental regulations.

I appreciate the support of my friends Ali Namazifard, Erfan Ghazinezam, Bidjan Ghahreman, Fargol Boya, Amir Nayyeri, Behrooz Touri, Reza Zamani Nasab, Omid Fatemieh, Parya Moinzadeh, Ali Farhadi, Hananeh Hajshirzi, Hamed Asghari, Majid Kazemian, Hadi Tavassol, Banafsheh Behzad, Babak Behzad, Abdolreza Osouli, Shadi Ansari, Hossein Ahmadi, Behnam Khatami, Ghazaleh Moghadam, Hamid Chitsaz, Amir Behgooy, Salem Darisavi, Professor Abbas Amimansour, Mino Roghani, Professor Ali Tousi, Ali Heydari, Hani Ramezani, Milad Moridzadeh, Saman Zonos, Reza Mahani, Sima Chehrenegar, Amir Marmarchi, Elahe Javadi, Yasahr Heidari, Mina Kamouie, Amir Houmansadr, Saloumeh Karimi, Nima Honarmand, Razie Irani, and Amir Barati who helped me have a wonderful time in Urbana-Champaign.

Finally, I am deeply thankful to my wife Maryam for her love and support during my graduate studies. I also thank my parents, Abolghasem and Azam, for their helpful encouragement. I would also like to thank my brothers Toraj, Ehsan, and Amir, and my sister Atefeh for their continued support.

I thank Yahoo! Labs for recognizing and assisting my research through a Yahoo! Key Scientific Challenges Award. My research in this thesis was also supported by NSF grants number 0938071, 0716532, and 0938064.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Limitations of Search and Exploration of Information

People and organizations generate and share a huge variety of unstructured information that can potentially be used to derive useful insights and bring value to society. However, unstructured information has limited capacity to satisfy users' information needs [32]. For instance, suppose that a user wants to find information about the athlete Michael Jordan, and submits keyword query Q_1: *Michael Jordan* to the Wikipedia collection (*www.wikipedia.org*) whose excerpts are shown in Figure 1.1. Many articles in the collection contain Q_1's two terms, and the search system does not have enough evidence to determine which article best satisfies the information need behind Q_1. Thus, the search system may give a high rank to articles that do not satisfy the information need behind Q_1, such as an article whose *url* is *wikiMichael_I._Jordan* rather than *wikiMichael_Jordan*.

To help overcome this challenge, we can convert the unstructured information to a semi-structured or structured format that makes some or all of the information's concepts and relationships explicit [23, 14, 55, 13, 7, 39, 36, 135, 32]. By *structure*, we mean the kind of semantic information about entities and relationships that one finds in a database schema.[1] A search system can use the resulting structure (*schema*) to improve users' experience as they search and explore the data. For example, Figure 1.2 depicts a more structured representation of the information shown in Figure 1.1. Users can learn the schema of the data set and use query languages such as XQuery or SQL to express their exact information needs and receive precise answers. This paradigm, however, has two important issues.

First, organizing huge volumes of unstructured information into a semi-structured[2] format is error-prone and takes a lot of computational and human resources [21, 1, 70]. To organize unstructured information, developers must identify and implement hundreds of extraction rules, or design and train complex machine learning based modules, to discover instances of concepts and add them to the resulting semi-structured data set. Developers must also maintain the modules and update them over time [105], which is especially awkward for specialized domains that require collaboration between developers and domain experts, such as medicine and law.

[1] In this dissertation, our approaches to answering keyword queries do not consider syntactic structure, such as the division of a document into sections, paragraphs, and sentences. For example, for our purposes an ordinary HTML file is unstructured text.

[2] In this thesis, we view structured information as a special case of semi-structured information, and will not consider it separately.

Since an organization will have limited resources available for adding structure, we should add structure only where it will significantly improve the user experience. For instance, if queries about athletes are exceedingly rare or are already answered well using unstructured information, then we should not expend a lot of resources to find and label all instances of concept *athlete*. Further, if the error rate of a module that discovers instances of a concept is high, the original unstructured data may serve the information needs behind the queries about this concept more effectively than its more structured version. Hence, the challenge is how to create a systematic framework that accurately predicts *the degree of improvement in satisfying users' information needs* using *erroneous programs and limited resources*, and determines *the most cost effective organization* for the data.

A second issue is that most users are not familiar with the notion of a schema or with query languages like XQuery or SQL, and have no wish to learn [69, 16, 9, 57, 26, 87, 149, 90, 6]. Even sophisticated users face challenges, as semi-structured data sets typically have complex schemas with hundreds of tables, columns, or XML elements. With little documentation available for the database design, and even less motivation to study it, even computer-savvy users find it challenging to explore the database and formulate appropriate queries [154]. Further, as the schema evolves over time, even an expert user may not be able to pose the correct query for her needs. Visual interfaces are an option that is excellent for many situations [146], but they must be customized individually for each domain, which is very expensive.

Keyword query interfaces are a popular alternative [155, 19, 9, 26, 57, 61, 60, 79, 85, 119, 124, 134, 149, 150]. With keyword query interfaces, users do not need to know schema details or any query language. For example, suppose a user wants to find papers about *Integration* published in the *VLDB* conference, in the database fragment in Figure 1.3. The user submits keyword query *Integration VLDB*, whose desired answer is the paper at node 2. Since the query is not framed in terms of the data's actual structure, the search system should determine the information need behind the query and rank highest the potential answers most closely related to the user's information need. For example, the paper at node 9 should not be the top-ranked answer, as it does not satisfy the user's information need. Thus, the challenge is how to create a system that finds users' desired answers for *most if not all keyword queries* over *most if not all semi-structured data sets*.

This thesis proposes systematic and principled solutions to these challenges. We set forth solid theoretical foundations for effective, robust, and cost-effective keyword search systems based on widely accepted principles in statistics, information retrieval, and database systems. We provide efficient algorithms for large scale search and exploration systems, and present the results from extensive user studies that validate these theoretical frameworks over real world data. We elaborate on the main contributions of this thesis in the following sections.

```
<article>
<url>wiki/Michael_Jordan</url>
<body>
  Michael Jeffrey Jordan
  (born February 17, 1963) is a
  former American professional
  basketball player, ... .
  Michael Jordan had spent
 his entire career with the Chicago Bulls ... .
</body>
</article>

<article>
<url>wiki/Michael_Jordan_statue</url>
<body>
   The Michael Jordan statue, officially
   known as The Spirit, ... .
</body>
</article>

<article>
<url>wiki/Michael_I._Jordan</url>
<body>
    Michael I. Jordan is a leading
    researcher .... . He is currently a full
    professor at the University of California,
    Berkeley... .
</body>
</article>
...
</collection>
```

Figure 1.1: Wikipedia article excerpts

1.2 Cost Effective Data Organization

To determine where to expend resources to add structure to information, we have created a method to quantify the expected improvement in satisfying users' information needs that will result if we convert the instances of a particular concept in unstructured data to a semi-structured format. We use the widely accepted effectiveness metric *precision at k*, so our approach provides understandable and verifiable predictions about the improvement in user satisfaction that can be expected from adding structure to a given collection of information.

Given a budget for the resources available for discovering instances of concepts, plus particular error rates for the discovery processes, we model the problem of choosing the most cost-effective organization for the data as an optimization problem. We use this model to explore the effectiveness of some commonly used heuristics for organizing the information in sets of documents. For instance, it is generally assumed that *the more relevant and popular documents we organize, the more effective search experience we can provide*. Interestingly, we show that sometimes *the fewer*

```
<collection>
<article>
<url>wiki/Michael_Jordan</url>
<body>
  <athlete> Michael Jeffrey Jordan </athlete>
   (born
   <birthdate> February 17, 1963 </birthdate>
   ) is a former
   <nationality> American </nationality>
   professional basketball player, ... .
   <athlete> Michael Jordan </<athlete>
   had spent his entire career with the
   <club> Chicago Bulls </club>
    ... .
</body>
</article>

<article>
<url>wiki/Michael_Jordan_statue</url>
<body>
  <art-form> The Michael Jordan statue </art-form>
  , officially known as
  <art-form> The Spirit </art-form>
  ... .
 </body>
</article>

<article>
<url>wiki/Michael_I._Jordan</url>
<body>
  <scientist> Michael I. Jordan </scientist>
   is a leading researcher .... . He is currently a
   <position> full professor </position>
   at the
   <school> University of California, Berkeley </school>
   ... .
</body>
</article>
...
</collection>
```

Figure 1.2: A more structured version of the Wikipedia article excerpts

relevant and popular documents we organize, the more effective search experience we can provide. We prove that the optimization problem is NP-hard in the general case, and propose approximation algorithms with provably bounded approximation ratios. We use standard benchmarks to validate the formal model and optimization algorithms, along with extensive experimental studies.

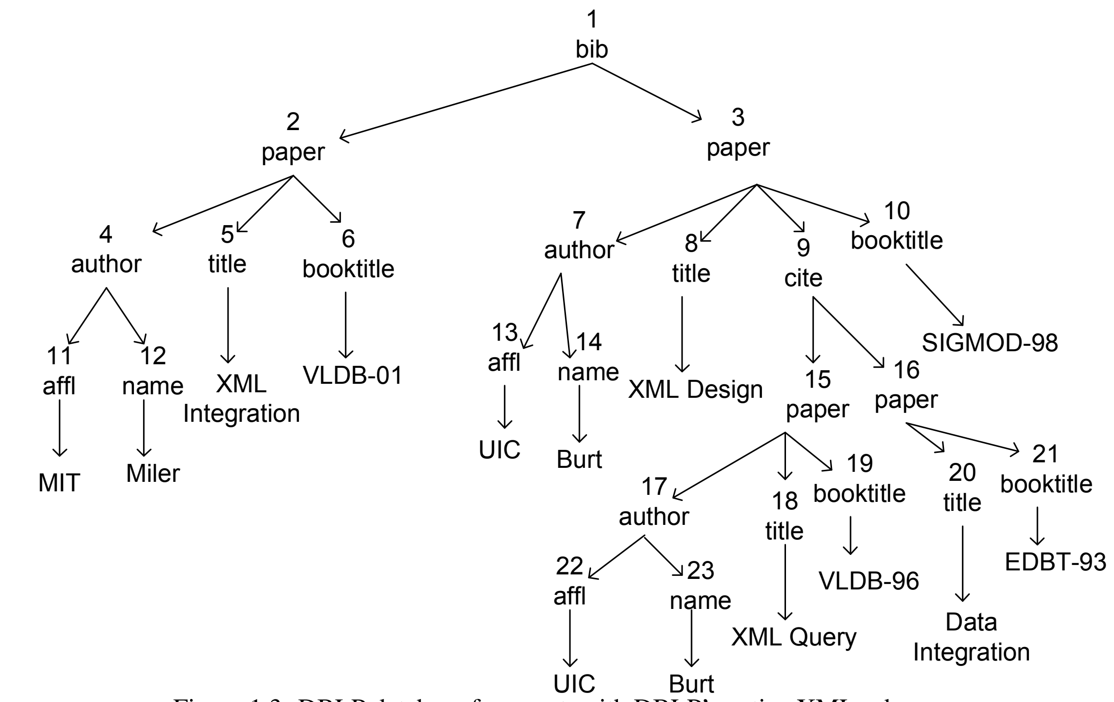


Figure 1.3: DBLP database fragment, with DBLP's native XML schema

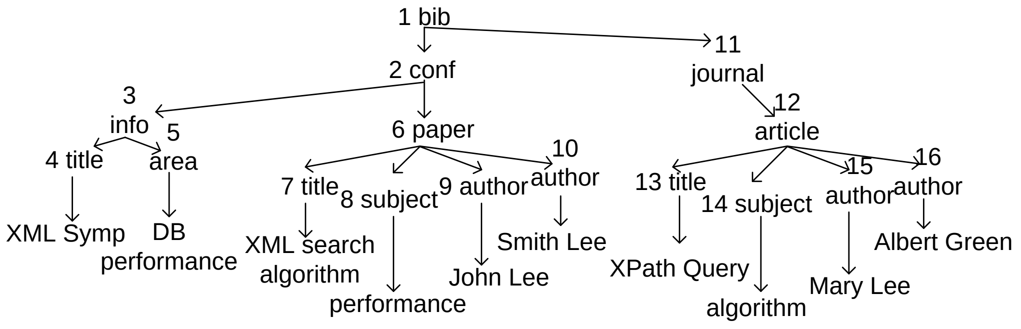


Figure 1.4: A bibliographic database

1.3 Design Independent Query Interfaces

Different data sources may be designed differently. Moreover, the design of the same data set may evolve over time. Database administrators (DBAs) may revise the database design over time to address issues such as redundancy, space overhead, performance, and usability [30, 137]. For instance, in the database excerpted on the right in Figure 5.3, path *paper/booktitle* is repeated under every subtree of element *proceedings*. The DBA may normalize the schema as shown in the fragment on the left in Figure 5.3 [4]. Or, as each paper has more than one author, the DBA may add a new node *authors* to the database excerpted in Figure 1.4 and create the new database excerpted in Figure 1.5 to make the database more usable. A DBA may also remove nodes such as *authors* to save space.

Current database management systems provide physical independence for users. Users do not have to modify their XQuery queries if the physical representation of the data changes. One goal of having a keyword query interface is to

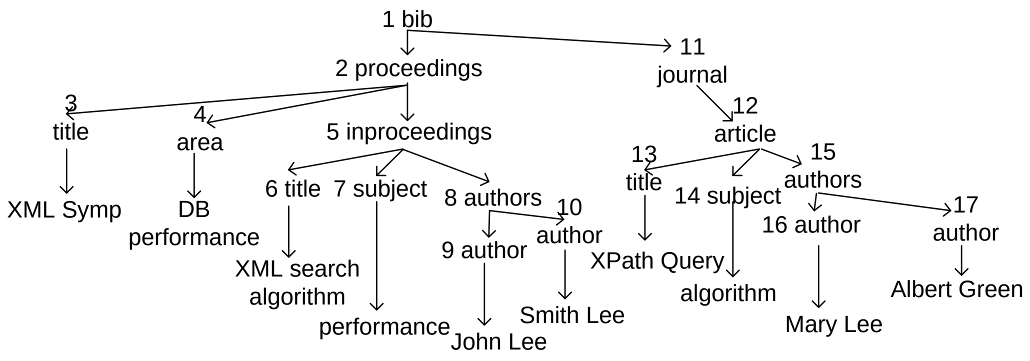


Figure 1.5: A bibliographic database

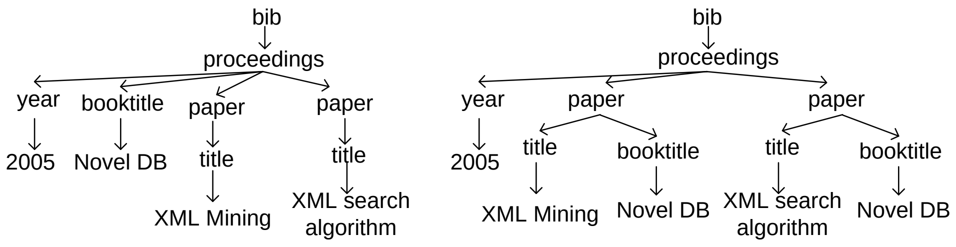


Figure 1.6: Normalization of a bibliographic database

allow users to find their desired information without knowing anything about the schema of the database [57, 149, 90, 6, 129]. Hence, a keyword query interface should have the property that its answers to a query stay the same when the database schema changes, if possible. In other words, users should not have to change their queries to get the same answers over the new schema. For instance, a keyword query interface should return the *paper* at node 6 in Figure 1.4 and the *inproceedings* at node 5 in Figure 1.5 for query *John Lee XML*, as they both represent the entity in question. Otherwise, users would have to learn aspects of the new schema to formulate the new query, which contradicts the goal of having a user-friendly system. If the results of a keyword query depend on a specific style of database design, the search system will perform poorly for data sets that follow other styles. Hence, no matter how the data is organized, a keyword query system should return the same answers as long as the data set contains the same information.

We introduce and formally define this property of *design independence* for keyword query systems [130, 131]. Design independence provides a way to measure the degree of *logical data independence* sought by the architects of modern data models [25]. To the best of our knowledge, design independence has not previously been defined and explored for any kind of query interface, including keyword query interfaces.

For a given set of content, the ranking produced by a design independent query interface is invariant under

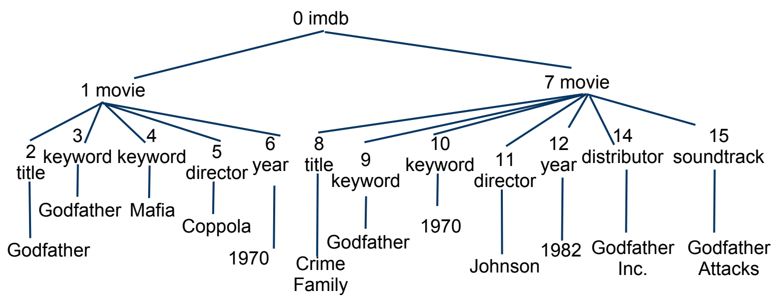


Figure 1.7: IMDB fragment

equivalence-preserving reorganizations of the structure of the data. We analyze this property for current keyword query interfaces and prove that they do not generally provide design independence. We have developed a novel keyword query answering method that is design independent. In addition to providing theoretical results for the worst case, we conduct an extensive empirical study to measure the design independence of a variety of keyword query answering methods, using real-world data sets. We show that our new method has higher average design independence than other methods. We also perform an extensive user study and show that the ranking provided by the new query answering approach is as good as or better than other methods.

1.4 Principled Ranking of Keyword Query Answers

As a basis for creating effective, extensible, and robust keyword query systems, it is crucial to develop principled formal models that correctly characterize the problem of which answers to rank highest. A keyword query system that is built atop clearly defined principles should have far fewer parameters to tune than more ad hoc approaches. Moreover, it should be possible to interpret and estimate its parameters analytically. More importantly, such a model may be able to guarantee that given the information needed to compute its parameters, *it delivers the best possible ranking* for every keyword query and data set. In other words, it will provide *optimal ranking of answers* for every keyword query and data set, given enough information.

Researchers have addressed similar challenges in document retrieval by creating formal models, most notably the probabilistic model and language model [29, 157]. However, it is ineffective to apply document ranking models and techniques directly to keyword search over databases. In fact, document retrieval models assume that a term refers to the same concept throughout the entire document [29, 104]. This assumption is usually false in databases. For example, Figure 3.6 shows part of the movie database IMDB, *www.imdb.com*. Different occurrences of term *Godfather* under the same movie at node 7 refer to different concepts: a company, a soundtrack, and a keyword about the movie. To rank answers well, the query answering system must take these differences into account.

Researchers have extended the formal models of document retrieval for keyword search over databases [102, 116, 78, 34, 33]. However, these approaches use relatively ad-hoc heuristics to define their models [116, 78, 34, 33]. In other words, it is not clear why their ranking methods should provide the highest utility to users, given the information available to use in ranking. Hence, they are not guaranteed to perform reasonably well for arbitrary databases. To address this problem, we propose a new formal model for keyword queries over structured data, called the *S*tructured Language Model (SLM). We show that SLM provides an effective ranking of answers for every query and data set, given the information made available to it. Our justification is based on the *probability ranking principle*, which is a widely accepted principle for ranking answers in an information retrieval system [109]. To the best of our knowledge, no previous model for keyword queries over structured data guarantees such a property. For the case where no training data is available, we show how to use widely accepted principles and heuristics in statistics and information retrieval to estimate the parameters of SLM. We provide the results of extensive experiments using two real databases, a real query log, and queries from a standard benchmark. Our results show that SLM's ranking of answers is better than that of previous keyword query systems.

1.5 Using Relationships in the Schema for Effective Ranking

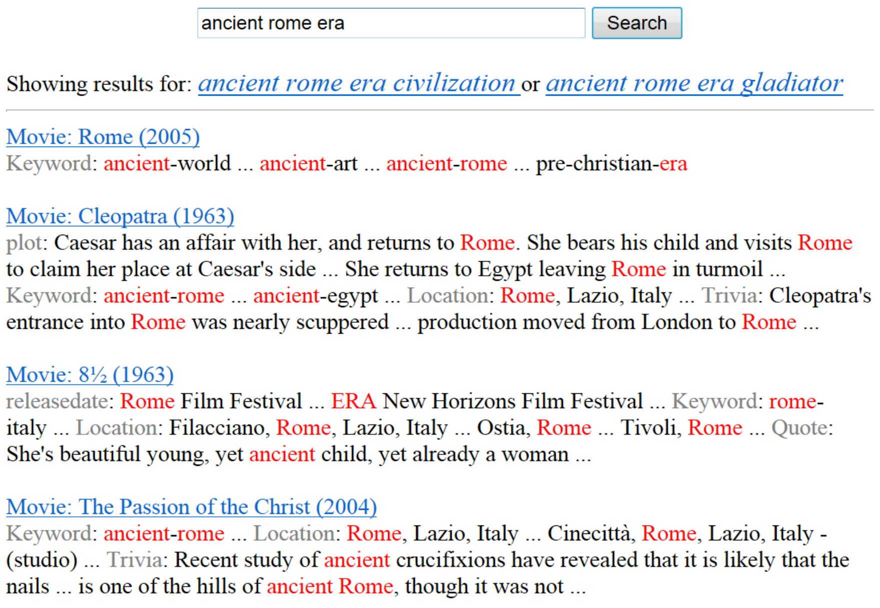
Different types of relationships in a database have different degrees of importance for users. For instance, users who submit the query *Han data mining* to a bibliographic data set are generally more interested in books about data mining that are *authored* by Han, rather than the books on this subject that are *edited* by Han. A query answering system should quantify the importance of these relationships for users, and give a higher rank to answers that represent more important relationships between data items.

Previous methods model the data set as a large graph, and use heuristics such as the length of the shortest path between data items to compute the importance of the relationship between them [26, 149, 9, 151, 54]. We show that these methods are often misleading and return ineffective search results. We propose a novel approach that uses metrics from information theory to quantify the importance of relationships between concepts in a schema. We propose novel approaches to use this information to rank the answers of a keyword query, and show that this approach avoids the pitfalls of the previous methods. Since the naive algorithms to compute these metrics are prohibitively inefficient, we present efficient algorithms to compute them. Our extensive user study with two real-world data sets shows that our ranking method is efficient at query time, and provides better ranking than previous approaches.

1.6 No Query Left Behind

INEX	SemSearch
ancient Rome era	Austin Texas
Movies Klaus Kinski actor good rating	Carolina
true story drugs addiction	Earl May
	Lynchburg Virginia
	San Antonio

Table 1.1: Example difficult queries from INEX and SemSearch benchmarks



Showing results for: *ancient rome era civilization* or *ancient rome era gladiator*

Movie: Rome (2005)
Keyword: ancient-world ... ancient-art ... ancient-rome ... pre-christian-era

Movie: Cleopatra (1963)
plot: Caesar has an affair with her, and returns to Rome. She bears his child and visits Rome to claim her place at Caesar's side ... She returns to Egypt leaving Rome in turmoil ... Keyword: ancient-rome ... ancient-egypt ... Location: Rome, Lazio, Italy ... Trivia: Cleopatra's entrance into Rome was nearly scuppered ... production moved from London to Rome ...

Movie: 8½ (1963)
releasedate: Rome Film Festival ... ERA New Horizons Film Festival ... Keyword: rome-italy ... Location: Filacciano, Rome, Lazio, Italy ... Ostia, Rome ... Tivoli, Rome ... Quote: She's beautiful young, yet ancient child, yet already a woman ...

Movie: The Passion of the Christ (2004)
Keyword: ancient-rome ... Location: Rome, Lazio, Italy ... Cinecittà, Rome, Lazio, Italy - (studio) ... Trivia: Recent study of ancient crucifixions have revealed that it is likely that the nails ... is one of the hills of ancient Rome, though it was not ...

Figure 1.8: Our results for difficult query *ancient Rome era*, with query suggestions returned

Intuitively, if the user enters a vague and underspecified query, the top-ranking answers are not likely to meet her needs. For example, Table 1.1 lists several queries from two recent benchmarking competitions [142, 133]. These are hard queries, in the sense that no current query-answering method or potential ranking approach performs very well for them. It is important for a keyword query system to recognize such queries and take appropriate action, such as warning the user, reformulating the query, suggesting query completions, or diversifying the top-ranked answers. For example, Figure 1.8 shows the ranked results for query *ancient Rome era*, using one of our implemented ranking algorithms. Our algorithms determine that this is an ambiguous query, which guides the system to generate query reformulation suggestions. On the other hand, if a keyword query interface employed these techniques for queries with high-quality results, it could reduce user satisfaction and waste computational resources. Hence, it is helpful for a keyword query interface to recognize underspecified queries and respond accordingly.

To the best of our knowledge, there has not been previous work on predicting or analyzing the difficulty of keyword queries over structured data. Researchers have proposed methods to detect difficult queries over unstructured text document collections [132, 161, 122]. These techniques are not directly applicable to semi-structured data sets, because they do not consider structure. For example, given the query *Godfather*, a keyword query system should consider which concepts in the schema are most likely to correspond to the user's information need: a movie title, production company name, descriptive keyword, plot summary, etc. We show empirically that direct adaptations of query difficulty prediction techniques for unstructured data are ineffective for structured data.

We analyze the characteristics of difficult queries over structured data and propose a novel method to detect such queries, by taking advantage of the structure of the data to gain insight about the degree of the difficulty of a query given the database. We used several popular and representative algorithms for keyword search over semi-structured data to show that our approach predicts the degree of difficulty of a query efficiently and effectively.

1.7 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 discusses related work. Chapters 3 and 4 introduce a series of novel keyword search and ranking algorithms that exploit the relationship between concepts in semi-structured data to deliver effective results for keyword queries. Those chapters include our experimental results on the effectiveness of these query answering techniques over real-world data sets.

Chapter 5 presents the concept of design independence and formally analyzes this property for current keyword query interfaces over semi-structured data. The chapter introduces a novel keyword query answering approach for semi-structured data that is provably design independent and explains our empirical results regarding its effectiveness and design independence over real-world large scale data sets.

Chapter 6 introduces the *Structured Language Model* (SLM), which is a novel, principled, and formal query answering model for structured data. This chapter provides methods to estimate the parameter of SLM, and presents empirical results on the effectiveness of SLM over real-world databases and query workloads.

Chapter 7 studies the problem of detecting hard queries over databases and proposes novel models and efficient and algorithms to effectively distinguish difficult queries over databases.

Chapter 8 formalizes the novel problem of optimizing the annotation and organization of data given limited resources. It introduces efficient approximation algorithms to solve the problem.

Chapter 9 concludes the dissertation and points out our future research directions.

Chapter 2

Related Work

2.1 Measuring the Effect of Adding Structure

Researchers have recognized that adding organization and structure to data sets reduces or eliminates current barriers in effectively addressing users' information needs [16, 32, 37, 42, 21, 115, 14]. Many systems use programs such as named entity and relationship recognizers to annotate or extract concepts and relationships in unstructured data [32, 37, 42, 21, 115, 23, 14, 55, 13, 39, 36, 135]. Our focus in this thesis is on determining what additional structure will most improve the user experience, through better ranking of answers to queries.

Since information extraction over a large collection of text documents is time consuming, researchers have proposed methods to detect and extract information from only a "promising" subset of documents or concepts. The intent is that the targeted documents or concepts are more likely to contain relevant information and/or be interesting to users [68, 71, 1, 70, 63, 74]. While the ultimate goal of information extraction is to improve the quality of query results, that line of work has not directly measured the improvements, instead relying on intermediate metrics such as the number of relevant documents organized or the "extractability" of a concept [63] as a proxy for query result improvement, and assuming error-free extraction. This thesis builds on that line of work by directly exploring the relationship between the cost of extraction, the choice of concepts to extract, the error rate of the information extraction modules, and the quality of ranking of answers to subsequent keyword queries, as measured by precision at k. Our goal is to help guide organizations as they decide where to expend their limited budgets for information extraction and annotation.

The previous work discussed above focuses on the effectiveness of queries over the *extracted* data only. This amounts to an implicit assumption that extracting and/or annotating more text documents always leads to more effective search. This thesis shows that this is not true if query processing also considers unstructured data, which may already satisfy the users' information needs in its original form. We also show that the extraction of one concept or entity may impact the effectiveness of search over other concepts, which violates an implicit assumption in previous work.

To measure the cost of extraction, previous work focused on the number of documents processed. This thesis considers a wider range of possible costs, such as the time and money to develop and maintain information extraction and annotation programs [117, 56]. Previous empirical studies suggest that the average life of some types of real-world Web information extraction programs is about two months, due to the dynamic nature of the Web [56]. These observations motivate our consideration of additional cost measures.

Researchers have considered the effect of the quality of online and ad hoc information extraction on the effectiveness of a single SQL query [70]. This setting is simpler than for keyword queries, because a user's information need is clearly stated in a SQL query and the query answers do not have to be ranked, which simplifies the relationship between query result effectiveness and information extraction quality. We build on this work by considering the more complex setting of a large workload of keyword queries. In this setting, previous work has empirically examined the effect of the quality of information extraction on the quality of query answers [24]. We add to this understanding by providing a formal framework to capture and optimize this relationship.

Concept annotation and extraction can be aimed at recognizing named entities and proper nouns such as *Michael Jordan*, or at disambiguating the meaning of terms found in dictionaries, such as *Java*, which can be a drink, an island, or slang for coffee. Researchers have used word sense disambiguation techniques to improve the effectiveness of keyword queries over unstructured text documents [83, 121, 113, 114, 123, 112]. We build on this work by providing a formal framework that can be used to explore the relationship between word sense disambiguation and the effectiveness of keyword queries, while considering the cost of annotation and extraction.

2.2 Design Independence

The goal of our work on design independence is to ensure that a keyword query will receive the same answers in the same order when the structure of the underlying data changes, as long as the two versions of the data are in some sense equivalent. To the best of our knowledge, the issue of design independence has not been examined previously for keyword queries over semi-structured data sets. In pursuing this line of inquiry, however, we build on previous work from the database community that seeks to characterize what it means for two databases to have equivalent content.

In the context of data exchange and data integration, researchers have studied the information preservation and query preservation properties of XML and relational schema mappings, i.e., translations from one schema version to another [66, 45]. The goal there is to identify schema mappings where every SQL or XQuery query over the source schema can be manually translated to a new query over the target schema. Our goal is different: we want to identify mappings where the keyword query system can return the same answers *without changing the query*, and we propose a new keyword query system that does this.

Researchers have experimented with providing logical data independence by representing a database as an instance of one single "universal" relation [136], so that each database query maps to a query over the universal relation.

Universal relation users do not need to know how to join relations to build the universal relation, but they must know a query language, the attribute names, and which information is stored in which attribute. Thus a universal-relation-based system is harder to use than a keyword query interface. Our work does not assume a common representation for all databases, such as a universal relation. Instead, we provide keyword query answering approaches that can return the same answers over different designs of the same database content. This provides more logical data independence than a universal relation approach. For instance, if the database designer changes attributes' names, our query system will still return the same query answers over the old and new databases. Universal relation users would have to rewrite their queries for the new database.

This thesis focuses on data-centric semi-structured data sets, rather than text-centric data such as HTML files with section and paragraph tags, or XML files containing just long text fields. Researchers have analyzed the effect of modifications of text-centric XML files, where the content of the output file is a subsequence of the content of the input file [3]. These modifications may not preserve the content or structure of the XML file, which puts them beyond the scope of the schema modifications we consider in this thesis.

2.3 Identifying Difficult Queries

Researchers have proposed methods to predict the difficulty of queries over unstructured text documents [132, 161, 59, 27, 138, 122]. Some methods use the statistical properties of the terms in the query to predict its difficulty, such as the average inverse document frequency of the terms in the query and the number of documents that contain at least one query term [59]. The common idea behind these methods is that the more discriminative the query terms are, the easier the query will be. Empirical evaluations indicate that these methods have limited prediction accuracy [132].

A better and popular approach called the *clarity score* argues that an easy query is sufficiently distinctive to separate the user's desired documents from other documents in the collection [132, 27]. Hence, its top ranked answers belong to very few topics, which are very likely to be the desired topics. On the other hand, the top ranked documents of a difficult query describe a variety of topics, many of which are irrelevant to the user's information need.

For example, consider a set of documents containing news stories. The top ranked documents for query *European financial crisis* will be mainly about financial news, but the top ranked answers for query *European crisis* may cover several additional topics, such as a political scandal, parliamentary showdown, or mysterious mass poisoning linked to vegetable consumption. Thus *European crisis* is more difficult than *European financial crisis*.

The clarity score provides a better estimate of the difficulty of a query for text documents than clues such as the number of terms in the query or the inverse document frequencies of its terms [132]. To measure the number of topics in the top ranked document of a given query, some systems compare the probability distribution of terms in the returned documents with the probability distribution of terms in the whole collection. If these probability distributions are relatively similar, the top ranked documents contain information about almost as many topics as the whole collection, indicating that the query is difficult [132, 122].

Consider a database as a collection of information about entities and their attributes. From this point of view, a "topic" in a database consists of entities that pertain to a similar subject. It is hard to define a formula that can partition an arbitrary set of entities into topics, as it requires finding an effective similarity function for entities. Such a similarity function would require domain knowledge and an understanding of users' preferences [58]. For instance, in practice, different attributes may have different degrees of impact on the similarity between entities. As a hypothetical example, suppose movies *The Bourne Identity* and *Mission Impossible 4* from the Internet Movie Database (IMDB) share k terms in their *genre* attributes. Suppose *Mission Impossible 4* and *High School Musical 3* also share k terms in their *distributor* attribute, which gives the distribution company for the movie. If the other attributes of these three movies do not contain any common term, then without additional domain knowledge, *Mission Impossible 4* and *High School Musical 3* would be seen as just as likely to be about the same subject and satisfy the same information need as *Mission Impossible 4* and *The Bourne Identity*. This thesis provides empirical results that confirm this argument, and shows that a straightforward extension of clarity scores is a poor predictor of the difficulty of a query over structured data.

To compute clarity scores, some systems use a precomputed set of topics and assign each document to at least one topic in the set [27]. To estimate the difficulty of a query, those systems compare the probability distribution of topics in the top ranked documents with the probability distribution of topics in the whole collection, and consider a query to be difficult if the two distributions are similar. The drawback of this approach is that we need domain knowledge about the data and its users to create an appropriate set of topics. It would be better to find an effective domain independent approach to predict the difficulty of queries.

Some methods use machine learning techniques to learn the properties of difficult queries and predict whether a new query will be difficult [152]. As these methods are intended for unstructured information, they have the limitations described above when applied to structured data. Moreover, they depend on the existence of sufficient quantities of good-quality training data, which is not always available in practice.

2.4 Effective Keyword Query Answering

Much research has focused on how to provide user-friendly query systems for semi-structured and structured data, especially for keyword search [136, 69, 16, 2, 5, 9, 26, 57, 61, 60, 79, 82, 85, 87, 89, 119, 149]. Two recent surveys offer an overview of the issues associated with supporting keyword queries over structured and semi-structured data

[155, 19]. Effectiveness, i.e., delivering the desired answers in the desired order, is always a key issue [144, 19], and is a central focus of this thesis. In the chapters that follow, we examine the previously proposed methods in detail.

In the case of unstructured information (text documents), formal probabilistic models justified by the probability ranking principle have been found effective in deciding how to rank answers to keyword queries [29, 157]. We use a similar approach to provide effective probabilistic models for ranking answers to keyword queries over structured data.

Chapter 3

Using Structural Information in Keyword Search Over Semi-Structured Data

3.1 Background

In this chapter, we propose a ranking approach for keyword queries that exploits the presence of structure in semi-structured data while avoiding overreliance on shallow structural details. This new ranking approach has higher precision and recall and better ranking quality than previous approaches. More specifically, we make the following contributions:

- We develop a theoretical framework that defines the degree of relatedness of query terms to XML subtrees, based on *coherency ranking* (CR), an extended version of the concepts of data dependencies and mutual information.
- We show how CR avoids the pitfalls that lower the precision, recall, and ranking quality of previous approaches, which rely on intuitively appealing but insufficiently principled heuristics that we analyze within the framework of CR. In particular, we show that for a given set of content, the ranking produced by CR is invariant under equivalence-preserving reorganizations of the schema, thus avoiding reliance on shallow structural details.
- We show how to deploy CR in XML query processing, using a two-phase approach. The first phase is a precomputation step that extracts the meaningful substructures from an XML database, before the query interface is deployed. During normal query processing, we use the results of the precomputation phase to rank the substructures in the database that contain the query keywords. Precomputation needs to be repeated after structural changes in the database that introduce new node types, so that subtrees containing those types of nodes can be correctly ranked. However, the results of the precomputation phase are not affected by non-structural updates to the content of a populated database, so the precomputation phase rarely or never needs to be repeated as the database content evolves.
- Since naive methods are prohibitively inefficient for the precomputation step, we present and evaluate optimization and approximation techniques to reduce precomputation costs. Our experiments show that these optimiza-

Portions of this work has appeared in [126] and [129].

tions improve precomputation performance by orders of magnitude while introducing negligible approximation errors.

- Our extensive user study with two real-world XML data sets shows that CR is efficient at query time, and has higher precision and recall and provides better ranking than previous approaches.
- For domains where information-retrieval-style statistics such as term frequency (TF) and inverse document frequency (IDF) [110] or PageRank [11] can be helpful in ranking query answers, we show how to combine CR with such measures to improve the precision of query answers. We provide an empirical study of how to combine information-retrieval-style methods and CR.

Although our focus in this chapter is on XML keyword queries, CR could also be adapted to work with for relational and graph databases (e.g., RDF, OWL, XML with ID/IDREF), and natural language queries. Our precomputation algorithm could also be adapted to for purposes other than keyword search, such as finding highly correlated elements and patterns [76], or discovering approximate functional dependencies [67] in XML databases.

This chapter is organized as follows. Section 3.2 motivates the work by examining previous approaches. Section 3.3 introduces the basic principles of CR and Section 3.4 provides the mathematical tools to quantify CR scores. Section 3.5 presents optimization and approximation techniques for database precomputation. Section 3.6 describes our implementation and Section 3.7 presents empirical results.

3.2 Motivation

In this section, we analyze current approaches to XML keyword search, in terms of their precision, recall, and ranking capabilities.

3.2.1 Basics

We model an XML database as a tree $T = (r, V, E, L, C, D)$, where V is the set of nodes in the tree, $r \in V$ is the root, E is the set of parent-child edges between members of V, $C \subset V$ is a subset of the leaf nodes of the tree called *content nodes*, L assigns a label to each member of $V - C$, and D assigns a data value (e.g., a string) to each content node. We assume no node has both leaf and non-leaf children, and each node has at most one leaf child; other settings can easily be transformed to this one. Each node can be identified by its *path* from the root; e.g., node 5 in Figure 3.1 has path */bib/paper/title*. Each *subtree* $S = (r_s, V_s, E_s, L_s, C_s, D_s)$ of T is a tree such that $r_s \in V_s$, $V_s \subseteq V$, $E_s \subseteq E$, $L_s \subseteq L$, $C_s \subseteq C$, and $D_s \subseteq D$.

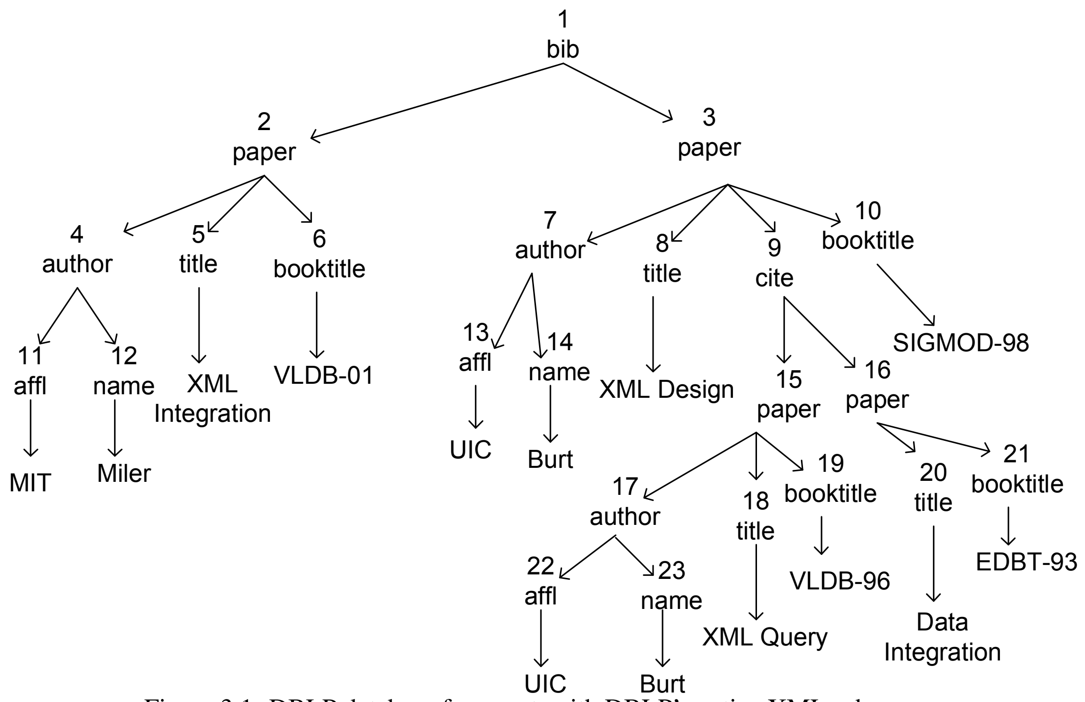


Figure 3.1: DBLP database fragment, with DBLP's native XML schema

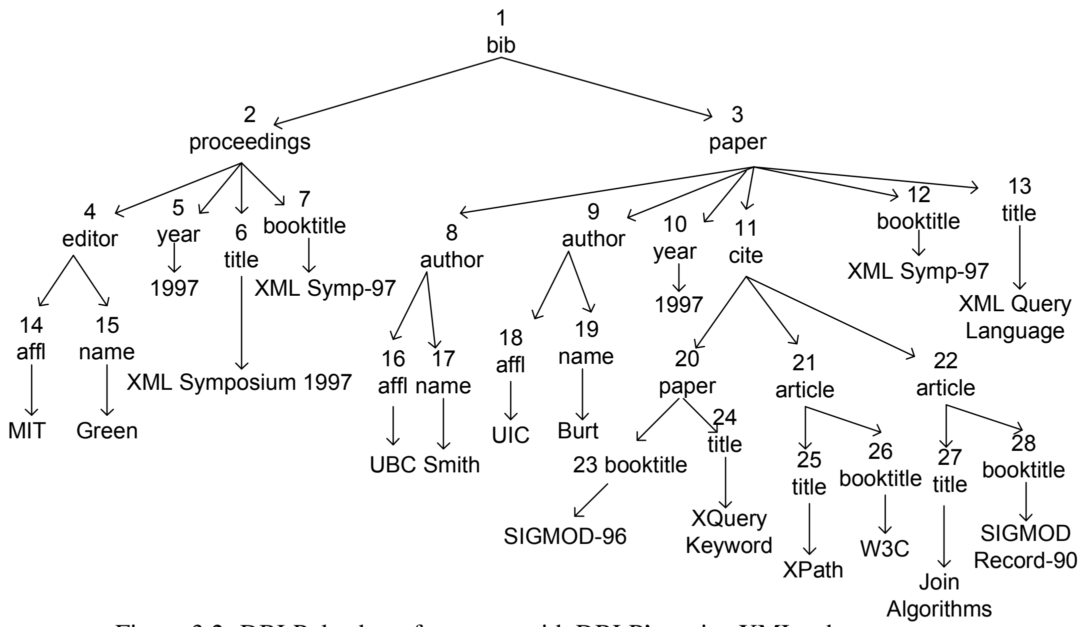


Figure 3.2: DBLP database fragment, with DBLP's native XML schema

A *keyword query* is a sequence $Q = t_1 \cdots t_q$ of terms. A subtree S is a *candidate answer* to Q iff its content nodes contain at least one instance of each term in Q. (The containment test can rely on stemming, stop words, synonym tests, and other refinements, although our experiments do not use these.) The root of a candidate answer is the *lowest common ancestor* (LCA) of its content nodes. When no confusion is possible, we identify a candidate answer by its root's node number.

The IR community has been developing retrieval techniques for *text-centric* XML [96], where structures are simple

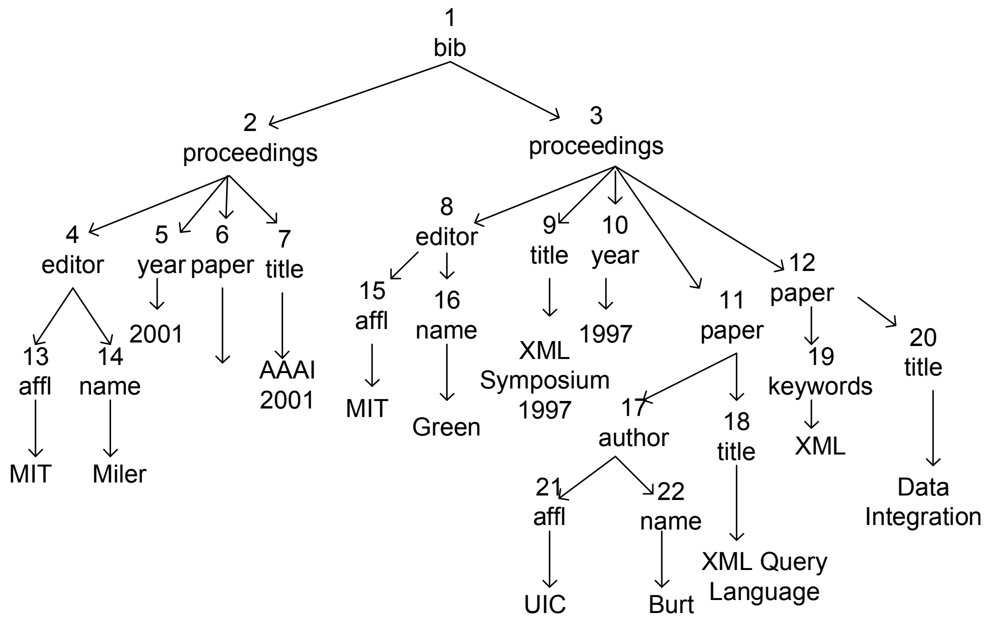


Figure 3.3: Reorganized DBLP database

and structural information plays almost no role in retrieval. The metadata of such content (e.g., title, author, conference) is the primary target of keyword search in data-centric XML, and the techniques we propose in this chapter are not intended for use with very long text fields, which we consider in the following chapter.

3.2.2 Current Approaches

The consensus in keyword search for relational and XML databases is that the best answers are the most specific entities or data items containing the query terms [2, 5, 9, 26, 57, 61, 60, 79, 82, 85, 87, 89, 119, 149]. Thus in XML databases, answers should include the most specific subtrees containing the query terms. The specificity of a subtree depends on the strength of the relationship between its nodes. For instance, if two nodes merely belong to the same bibliography, such as titles of two different papers, then the user will not gain any insight from seeing them together in an answer. If the nodes belong to the same paper, such as the paper's title and author, the user will surely benefit from seeing them together. If the nodes represent titles of two different papers cited by one paper, the answer might be slightly helpful.

The *baseline method* for XML keyword search returns *every* candidate answer ([119], with modest subsequent refinements [57, 61, 150]). For instance, consider the DBLP fragment from *www.informatik. uni-trier.de/* shown in Figure 3.1. The answer to query *Integration Miller* is (rooted at) node 2. This approach has high recall but very low precision. For example, for query Q_1 = *Integration EDBT*, the baseline approach returns the desired answer of node 16, but also the unhelpful root node. In Q_2 = *Integration VLDB* for Figure 3.1, candidate answers node 9 and

1 are unhelpful. The node 9 tree contains two otherwise-unrelated papers cited by the same paper, and the node 1 tree contains otherwise-unrelated papers in the same bibliography. A good approach should either not return these answers, or rank them below the helpful answers.

Researchers have worked to boost the precision of the baseline method by filtering out unhelpful candidate answers. One approach eliminates every candidate answer whose root is an ancestor of the root of another candidate answer [124, 149]; the LCAs of the remaining candidate answers are called the *smallest* LCAs (SLCAs). The SLCA approach relies on the intuitively appealing heuristic that far-apart nodes are not as tightly related as nodes that are closer together. For Q_1 in Figure 3.1, the SLCA approach does not return node 1. However, it does still return node 9 for Q_2; as it does not rank its answers, the user will get a mix of unhelpful and desirable answers. SLCA's recall is less than the baseline's: for query Q_3 = *XML Burt*, nodes 3 and 15 are desirable; but since node 3 is an ancestor of node 15, node 3 will not be returned. MaxMatch [90] improves the precision of SLCA, by eliminating an SLCA answer if it has the same root as another SLCA answer A, and its matched keywords are a subset of A's matched keywords. But it still has SLCA's effectiveness problems: it still returns node 9 for Q_2, and does not return nodes 3 for query Q_3. It does not rank its answers either.

XSearch [26] removes every candidate answer having two non-leaf nodes with the same label. The idea is that non-leaf nodes are instances of the same entity type if they have duplicate labels (DLs), and there is no interesting relationship between entities of the same type. We refer to this heuristic as *DL*. For instance, the subtree rooted at node 9 does not represent a meaningful relationship between nodes 19 and 20, because they have the same label and type. Therefore, node 9 should not be an answer to Q_2. XSearch ranks the remaining answers by the number of nodes they contain and their TF/IDF.

DL is not an ideal way to detect nodes of similar type. For example, nodes *article* and *paper* in Figure 3.2 have different names but represent similar objects. As a result, for the query Q_4 = *SIGMOD XPath*, DL returns node 11, which is undesirable. DL cannot detect uninteresting relationships between nodes of different types, either; it does not filter out node 1 for query Q_5 = *UBC Green* in Figure 3.2. Further, sometimes there *are* meaningful relationships between similar nodes, even in a database with few entity types. For example, DL does not return any answer for Q_6 = *Smith Burt* in Figure 3.2, as it filters out node 3. XSearch's distance-based ranking scheme does not help to avoid DL's pitfalls.

We also review the MLCA approach [87] used to retrieve meaningful relationships from XML documents with a natural language query interface [88]. Similar to DL, MLCA assumes that only the closest nodes of different types are meaningfully related. For instance, node 16 in Figure 3.2 will not be associated with node 15 because node 16 *has a different node with the same label (name) closer to it*. Therefore, the MLCA approach successfully filters out the undesirable answers for queries like Q_5.

However, MLCA has many of the DL and SLCA problems. MLCA does not identify uninteresting relationships between nodes of the same type; e.g., it returns undesirable node 1 for query Q_7 = *Smith Green*. Second, it can return undesirable answers when a child is null due to being optional. For example, consider Figure 3.1, but with node 10 null. For query Q_8 = *XML Design VLDB*, MLCA returns the undesirable tree rooted at node 1 and including nodes 8 and 6 in Figure 3.1. Third, MLCA is not appropriate to use with databases that contain instances of more than one entity type. Consider Figure 3.2 with *proceedings*'s *title* renamed to *venue*. For the query *editor XML Query*, MLCA returns the subtree rooted at the root of the database containing nodes 4 and 13. Since there is no meaningful relationship between nodes 13 and 4, MLCA should not return this answer. These precision problems are important, as MLCA does not rank its answers. Fourth, MLCA misses desirable answers when similar nodes have a meaningful relationship. In Figure 3.3, MLCA finds no interesting relationship between nodes 9 and 17, as there is another node with the label *title* closer to node 17. To improve precision, CVLCA combined MLCA and DL into one approach [85]. But the combination still has the precision problems described above, and lower recall than either approach alone. [85] also introduced another approach, called VLCA. VLCA uses the DL heuristic; therefore, VLCA has the same problems as the XSearch method. CVLCA and VLCA do not rank their returned answers.

XRank [57] finds the LCAs using a modest modification of the baseline approach and uses a PageRank-based approach to rank subtrees. It has the same precision problems as the baseline method. Also, PageRank is effective only in certain domains and relationships and is not intended for ranking subtrees. For instance, all the *Proceedings* tags in Figure 3.1 have the same PageRank.

XReal [6] filters out entity types that do not contain many of the query terms. Then it ranks subtrees higher if they and their entities' types have more of the query terms. For instance, DBLP has few books about *Data Mining*, so XReal filters out all *book* entities when answering the query *Data Mining Han* – even Han's textbook. XReal does not consider the relationship *between* nodes of a subtree when it ranks its answers, so irrelevant subtrees can be ranked very high. For query Q_{10} = SIGMOD 1997, XReal ranks the subtree rooted at node 3 in Figure 3.2 higher than the 1997 papers with *booktitle* SIGMOD. This is because *SIGMOD* occurs very frequently in subtrees rooted at *cite*. Even if we ignore the importance of the *cite* entity type in XReal ranking, XReal still ranks papers published in 1997 below papers that cite multiple articles and papers published in 1997. In general, IR ranking techniques do not take advantage of XML's structural properties effectively.

Another method is to determine the interesting entity types manually and return only the instances of these entities that contain the keyword query. For instance, assume that DBLP contains only entity types *paper* and *proceedings*. Then, this method returns only the subtrees rooted at *paper* and *proceedings* nodes that contain the input keywords. However, we still have to rank the candidate answers, as usually more than one instance of each entity type match the keyword query. For example, both papers rooted at node 2 and 3 are candidate answers for the query *XML VLDB*

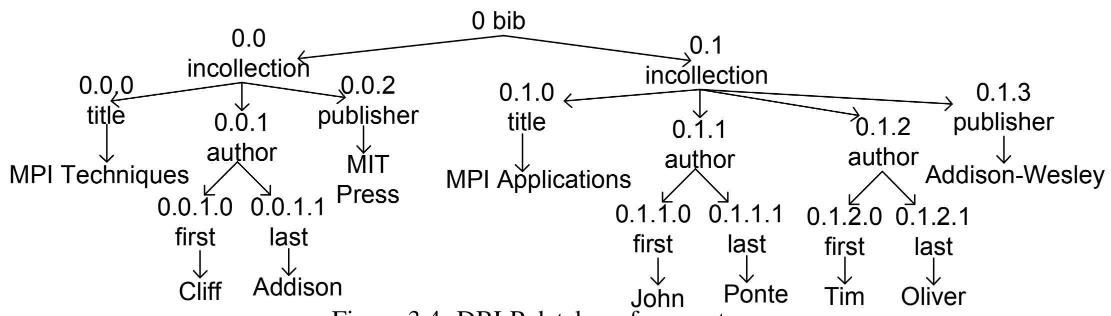


Figure 3.4: DBLP database fragment

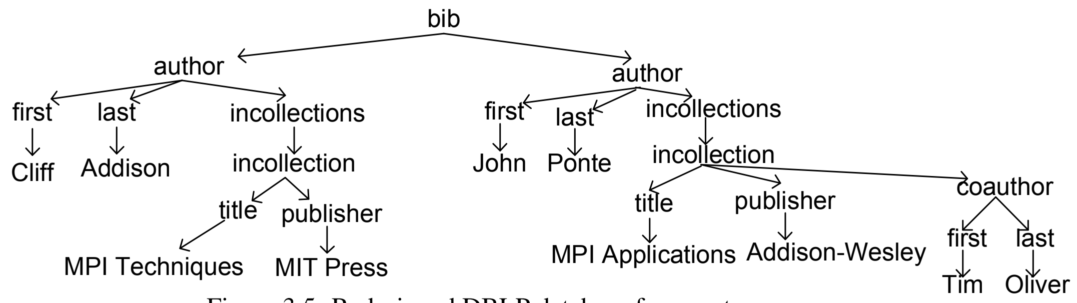


Figure 3.5: Redesigned DBLP database fragment

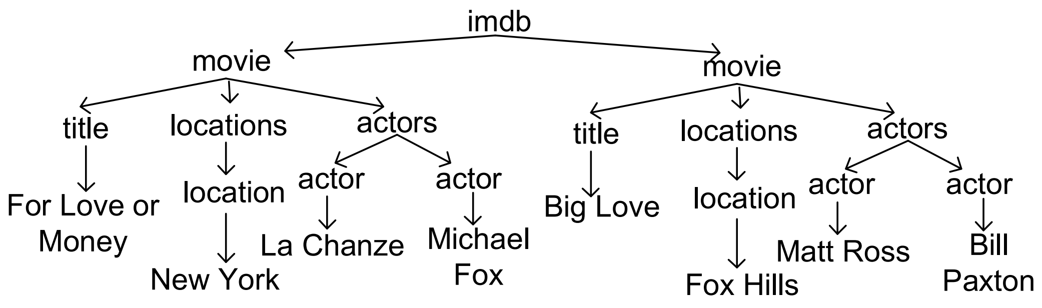


Figure 3.6: IMDB database fragment

in Figure 3.1. Nevertheless, the desired answer is the paper rooted at node 2. Furthermore, this method is domain dependent and appropriate only for the databases that consist of relatively small number of entity types.

The first key shortcoming of all these methods is that they *filter out answers instead of ranking them* and they *rely on very shallow structural properties to rank answers*. As relevance is a matter of degree, good ranking schemes are generally more effective than filtering [109]. Since these methods rely on ad hoc heuristics, they are ineffective for many queries, as our experimental results illustrate. Second, these methods are dependent on the current schema of the database. If the schema is reorganized in an equivalence-preserving manner, their query answers may change.

3.3 Coherency-based Ranking

In this section, we propose an approach that uses deep structural information to rank the candidate answers instead of filtering them. First, we analyze previously proposed structural properties that could be used for this purpose in XML, relational, and graph databases and show they deliver poor ranking. We argue that the meaningfulness and interestingness of a candidate answer depends on the types of its nodes and their relationships. We show that the interestingness of the relationship between two entity types can be measured by the information content and correlation between its nodes. We also show that our proposed approach does not have the problems of the previously proposed approaches and delivers better ranking. Furthermore, we argue that as this approach takes advantage of deeper structural information, it provides almost the same answers for different designs of the same database.

Current XML keyword search systems implicitly assume that all relationships between nodes in a candidate answer are equally relevant to the input queries [26, 6, 57]. The *distance* between nodes n and m is the number of nodes in the path between n and m. Thus, they rank higher the subtrees where the nodes containing query terms are less distant from each other. This heuristic is too ad hoc to work well in general: consider the subtrees rooted at *cite* and *bib* in Figure 3.1, which contain paper titles. Te query terms in these two subtrees are the same distance apart,, but in a large bibliographic database with papers on many topics, such as the original DBLP data set, the relationship between the *title*s under *cite* is more meaningful than between the *title*s under *bib*. Even in a small bibliographic database where all the *paper*s under *bib* are on the same narrow topic, this rule will still hold for citations of papers that are themselves *title*s under *bib*, though it may be violated for citations of external papers. Our work is targeted toward large diverse data sets, where ranking keyword query answers is especially important and challenging. Also, our experiments show that many candidate answers have the same maximum distance. This is particularly important because flat XML databases (like the original version of DBLP) are proliferating, due to the use of semantic tagging of text corpora using information extraction codes [35]. Most candidate subtrees have the same distance in flat XML databases, so we need to be able to rank subtrees with the same distance between query terms.

One might think that the subtrees that occur deeper in the database (i.e., where the root of the subtree is further from the DB root) present more meaningful relationships. However, this method has the same problems as the distance-based ranking. For instance, node 2 is at a higher level than node 9 in Figure 3.1, but node 2's subtree represents a more meaningful relationship (information about one paper) than node 9's (information about different papers cited by the same paper).

Researchers have also considered the problem of how to rank answers to keyword queries in relational databases [9, 2, 5, 89, 82]. These approaches create a join tree whose tuples contain the query terms, and rank the tree based on its maximum distance. The ranking can also consider DBA-specified weights on the edges in the join tree [5, 82]; however, we seek an automated approach.

Researchers have found that different types of relationships have different degrees of importance to the users in relational and graph database keyword search systems [9, 151, 54]. They suggest that the fewer instances of type B associated with each instance of type A, the stronger their relationship is. One approach ranks a tuple higher if many tuples link to it, but it links to just a few [9]; the same idea can be used for graph databases [54]. Another idea employs the notion of join selectivity to measure the degree of closure among a set of tables [151]. We call this group of methods *association based approaches*. We can also extend these approaches to work with the semi-structured data such as XML. For instance, this approach can recognize that the relationship between two *title*s under *bib* in a large bibliographic database covering multiple topics (such as the original DBLP database) is more meaningful than the relationships between *title*s under *cite*. This is because each *title* under *bib* is associated with many more other *title*s than the number of *title*s associated with a *title* under *cite*. This heuristic improves the ranking quality of distance based approaches; however, as explained in the following paragraphs, these heuristics are misleading.

First consider Figure 3.6 which illustrates some information about movies and TV shows from *www.imdb.com*. Node type *location* shows the filming locations of a movie. The number of actors in each movie greatly exceeds the number of its filming locations on average, considering all instances of the original IMDB data set. The fragment shown in Figure 3.6 has twice as many number of actors as filming locations per movie, which is typical of the IMDB data set. Hence, according to association based heuristics, the relationship between *title* and *location* is more interesting to most users than the relationship between *title* and *actor*. Therefore, for query *Love Fox* these methods rank the *movie* subtree on the right higher than the *movie* subtree on the left in Figure 3.6. However, we argue that the relationship between the title of a movie and its actor(s) is more interesting to most users than the relationship between its title and its filming location(s). The main search query form on the home page of the IMDB web site (*www.imdb.com*) enables users to search on several attributes of a movie, such as titles and actors. However, it does not query the database based on the filming locations of a movie. There is a query form in the bottom of another page of the web site (*www.imdb.com/search*) that enables users to find movies based on their filming locations. Even in that form, the filming location has been placed after other attributes. It is reasonable to assume that the developers of the IMDB web site put the more important and more frequently queried attributes on the main query form of the database. Thus, the relationship between the title of a movie and its actor(s) is more important to most users of IMDB than the relationship between the title of a movie and its filming location(s). Of course, in some cases users may be interested in the filming locations of a movie, but they are in the minority and our goal is to deliver the best possible ranking for the majority of users. Interestingly, distance-based approaches will give the same rank to both candidate subtrees for query *Love Fox*.

As another example, consider the fragments of DBLP shown in Figure 3.4. If we compare the relationship of *title* with *author* and *publisher* in Figure 3.4, we see that there are more authors associated with a title than there are publishers for the title on average. The same argument holds for the original DBLP data set, as there is more than one author for each paper but only one publisher per *incollection* (i.e., an article in a book) on average. Hence, association based approaches assume that the relationship between *title* and *publisher* is more important than the relationship between *title* and *author*. Thus, they rank the right *incollection* subtree higher than the left one for query Q_{11} = *MPI*

Addison. However, we argue that the relationship between *title* and *author* of an *incollection* is more interesting than the relationship between its *title* and its *publisher*. Developers of graphical user interfaces place the most queried database fields in the query forms. They make these decisions based on user feedback, query logs, and their own experiences. The DBLP data set has some query forms that perform search over specific fields, where users choose their field(s) of interest before submitting their queries at *dblp.uni-trier.de*, *dblp.mpi-inf.mpg.de/dblp*, and *dblp.l3s.de*. All these forms support searches based on author names, but none of them support searches via the publisher fields. They also show *author* as a facet when returning the candidate answer, so that users are able to filter the results based on specific author(s). However, they do not show the *publisher* field as a facet. This indicates that the *publisher* field is less interesting to most DBLP users than *author*. In other words, the relationship of a book or article's *title* to its *author* is generally more relevant for query answering than the relationship between the book or article's *title* and its *publisher*. Of course, there are exceptions to this rule (e.g., an author looking for a potential publisher for her book), but it holds in general for large bibliographic databases.

Note also that distance-based approaches will give the same rank to the candidate subtrees for query Q_{11}, as both candidate answers have the same maximum distance. Consider the redesign of Figure 3.4 shown in Figure 3.5. The new design categorizes articles and books based on their authors, which makes the title closer to the publisher than it is to the article's author – even though the new design is still normalized [4, 153]. Thus, distance-based ranking methods rank the candidate subtree on the right higher than the candidate subtree on the left for query Q_{11} in Figure 3.5.

A second problem with association based methods is that these measurements are not normalized by the number of distinct values for each type. For example, suppose that type B has just three values. Having a distinct value of type A associated with just two values of type B does not mean that the relationship is very meaningful. Yet having the same association when B has over 100 distinct values could mean that the relationship between the two is very close.

A third problem is that the join-selectivity method does not address the case where the join path does not cover all the tuples in the relations. For instance, consider a universitydatabase where each professor advises a few students and a few professors offer on-line courses for hundreds of students. Since the on-line course join path does not cover all tuples in the faculty relation, its join selectivity can be as small as or even smaller than the advisor relationship.

Finally, these approaches do not rank different candidate tuples that come from the same table. These problems, plus the lack of certain other forms of normalization discussed later, mean that these heuristics will not rank answers well in general.

The right intuition is that type A is more related to type B if each instance of type A is associated with relatively few instances of type B **and** each instance of type B is associated with relatively few instances of type A. In other words, from the value of one type we can predict the value of the other. The closer this association is between the types in a subtree, the more the subtree represents a meaningful and coherent object. This intuition can be generalized to more than two types, and can be normalized based on the number of values of each type. We can use this idea to measure the strength of the relationships in a candidate answer and rank the candidate answers according to these strengths. We will show how to quantify these relationships in Section 3.4. We will measure the correlation between node types, considering all instances of the node types in the complete underlying database. Using this information, we rank the most highly correlated candidate answers first. We call this approach **coherency ranking** (CR).

CR correctly ranks the subtree on the left higher than the subtree on the right in Figure 3.4 for Q_{11} = *MPI Addison*, as the number of articles and books a publisher publishes is far more than the number of the publications of an author (considering all the instances in the original DBLP database). Thus, knowing the name of an author predicts the title of its publications more precisely than knowing the publisher of an article or a book. CR also gives more importance to the relationship between *title* and *actor* than the relationship between *title* and *location* in Figure 3.6. CR handles the cases where association based methods deliver acceptable ranking. For instance, according to CR the relationship between *title*s under *cite* is more meaningful than the relationship between the *title*s under *bib* in the original DBLP database whose fragments shown in Figure 3.2.

CR gives the intuitively desirable ranking for all the queries considered in Section 3.2. The *title* of a *paper* is associated with only one *booktitle* and there are on average fewer than 50 papers in each proceedings in the original DBLP database. However, each citation is co-cited with around 40 other citations on average in the original DBLP database. Thus, *title* and *booktitle* are more correlated than two citations in the same paper. Thus, CR ranks the candidate answer rooted at node 2 first and the candidate answer rooted at node 9 second for Q_2 = *Integration VLDB* in Figure 3.1. It also ranks the candidate answer rooted at node 1 last. As we will discuss in Section 3.4, users can set the minimum acceptable amount of correlation for the returned candidate answers. If this value is set to be greater than zero, CR will not return any spurious candidate answers such as the candidate answers rooted at the root of the database. CR delivers a similar ranking for Q_4 = *SIGMOD XPath* and ranks a paper about “XPath” in “SIGMOD” higher than the subtree rooted at node 11 in Figure 3.2. CR also ranks a paper published by an *author* named “Green” and affiliated with “UBC” higher than the candidate answer rooted at the root of the database in Figure 3.2 for Q_5 = *UBC Green*. This is because each institute has a limited number of authors and each author is affiliated with a few institutes, but all authors and institutes in the original database are associated with each other.

The correlation between each pair of author and editor in the original DBLP database is very low, as almost every author in the database can be associated with each editor and vice versa. Thus, CR recognizes that the candidate answer rooted at node 1 shown in Figure 3.2 is not a meaningful answer to Q_7 = *Smith Green* and will not return it or will return it as the last candidate answer in the ranked list of answers. Even if node 10 in Figure 3.1 is null, CR still omits or ranks last the subtree rooted at node 1 for Q_8 = *XML Design VLDB*. This is because considering all information in the original DBLP database, the correlation of *title* of one *paper* and *booktitle* of another *paper* whose only relationship is that they are placed in the same database is very low. CR will deliver a similar result for Q_9 =*editor XML Query* in Figure 3.2. Each *booktitle* is associated with only one *year* value, as each conference in the DBLP database is held once per year. Each *year* value is associated with many *booktitle* values in DBLP. However, each *year* value is associated with many more papers and even more papers that are cited by these papers. On the other hand, each paper can be cited by other papers published in different years. Thus, the *booktitle* and *year* of a paper are more correlated than its *year* and its cited papers. Hence, CR ranks a paper that is published in "1997" in "SIGMOD" higher than the candidate answer rooted at node 3 in Figure 3.2 for query Q_{10} = SIGMOD 1997.

As for recall, unlike previous methods, CR correctly returns both nodes 3 and 15 for Q_3 = *XML Burt* in Figure 3.1, and returns node 3 as an answer to query Q_6 = *Smith Burt* in Figure 3.2. CR's recall will be equal to that of the baseline approach and higher than that of all other approaches discussed earlier, as long as the minimum correlation threshold is close to zero. Users of approximate retrieval systems sometimes like to see as many relevant answers as possible (i.e., high recall) and sometimes they are interested in only the most relevant answers (i.e., high precision) [96]. CR enables users to control the tradeoff between the quantity (recall) and quality (precision) of the returned answers.

The idea of CR is close to the concept of functional dependencies (FDs) and approximate FDs [65, 31], which have been used to distinguish fine-grained entities in a database [95]. Researchers have extended the concept of a FD to apply to XML data [4, 153]. However, two obstacles prevent us from using FDs here. First, FDs are unidirectional, and we need a bilateral relationship. Second, FDs are defined on two specific sets of attributes. For instance, to identify the most meaningful subtrees, should we use the FD *title* $\rightarrow$ *booktitle author*, *author* $\rightarrow$ *booktitle title*, or another FD? We remedy this problem using an extended version of mutual information.

Mutual information and other statistical measurements are used in the data mining community to find correlated data items [10, 94, 76, 100]. However, these approaches typically consider only binary variables, and our variables (XML types, or paths) are usually not binary. Second, these approaches look for sets of elements whose correlation exceeds a specific threshold; the exact value of the correlation is not of interest. However, to correctly rank query answers, we may need to compute exact correlations. Third, their measurements are upward closed, meaning that if a set of variables is correlated, at least one subset of them is correlated [10, 94, 76, 100]. This makes the correlation easy to compute. But in keyword search, we may prefer one answer over another that contains it, even if the larger tree has a higher correlation. Thus our measurement should not be upward closed. Third, these approaches redefine the correlation measurement to make it easier and faster to compute. For example, one approach only considers mutual information among pairs of variables in a set, rather than among all possible subsets of the set [94]. But we must consider the correlation among all nodes in a subtree, not just pairs. For example, consider a bibliographic database with conference names, years, and paper titles. The correlation between conference names and years will be low, as each conference is held in multiple years and each year has many different conferences. But the correlation between the three types conference *name*, *year*, and *title* is quite high, as the conference name and year together are associated with only a small set of paper titles. Overall, we conclude that we cannot simply apply previous work from the data mining community to rank candidate answers.

Mutual information has been used to improve retrieval effectiveness for unstructured and semi structured documents [73, 107, 72, 118]. Jones et al. and Petkova et al. use mutual information between terms to extract phrases in unstructured and semi-structured documents. Petkova et al. also use information gain to discover the correct ancestral order among tag labels in ambiguous XML structural queries and refine the queries. Schenkel and Theobald use mutual information to expand the structural queries over XML documents using relevance-feedback techniques. Jang et al. use mutual information to eliminate query ambiguities in cross-language information retrieval. These approaches consider mutual information between terms, rather than between schema elements. Our work is orthogonal to these methods and can use these methods to improve its effectiveness.

As shown in our experiments, in application domains where PageRank or IR ranking techniques are applicable, they can be combined with CR by weighting and adding the ranks suggested by each approach. As mentioned earlier, our techniques in this chapter are for data-centric XML, not text-centric XML. Nodes with a lot of text, such as the body of an article, are likely to be very highly correlated with certain other fields. The metadata of such content (e.g., title, author, conference) is the primary target of keyword search in data-centric XML, and the techniques we propose in this chapter are not intended for use with very long text fields.

3.4 Measuring Coherency

In this section, we develop a ranking framework for XML query answers, based on the concept of CR. For brevity, we will use the term *DB* to refer to XML DBs. Also, in this section we ignore all non-content leaf nodes, as they do not affect rankings.

3.4.1 Preliminaries

Consider the depth-first traversal of a tree, where we always visit the children of a node in alphabetic order of their labels. Each time we visit a node, we output its number (or content); each time we move up one level in the tree, we output *-1*. The result is the unique *prefix string* for that tree [156]. For instance, the prefix string for the subtree rooted at node 4 in Figure 3.1 is *4 11 MIT -1 -1 12 Miller -1 -1*.

Trees T_1 and T_2 are **label isomorphic** if the nodes of T_1 can be mapped to the nodes of T_2 in such a way that node labels are preserved and the edges of T_1 are mapped to the edges of T_2. A **pattern** concisely represents a maximal set of isomorphic trees (its **instances**). The pattern can be obtained from any member of the set, by replacing each node number in its prefix string by the corresponding label. For instance, pattern *bib paper title -1 -1* corresponds to trees *1 2 5 -1 -1* and *1 3 8 -1 -1* in Figure 3.1. P_1 is a subpattern of P_2 if P_1's trees are all subtrees of P_2's trees.

Definition 3.4.1. *A tree $S = (r, V_s, E_s, L_s, \emptyset, \emptyset)$ is a* ***root-subtree*** *of tree $T = (r, V, E, L, C, D)$ if S is a subtree of T and each leaf node of S is the parent of a leaf node of T.*

For instance, *1 2 5 -1 6 -1 -1* is a root-subtree in Figure 3.1. If the root-subtree is a path, we call it a **root-path**. Each path from root to leaf in a root-subtree is a root-path, so each root-subtree contains at least one root-path. Intuitively, the *value* of a root-subtree is the content that was pruned from its leaves. For example, the value of *1 2 5 -1 6 -1 -1* is ("XML Integration", "VLDB").

Definition 3.4.2. *Let S' be a subtree of T, and let S be a subtree of S', such that S is a root-subtree of T and every leaf of S' is also a leaf of T. Let $c_1, \ldots, c_n$ be the list of content nodes encountered in a depth-first traversal of S'. Then $(c_1, \ldots, c_n)$ is the* **value** *of S.*

Every maximal set of isomorphic root-subtrees in a tree T corresponds to a pattern. Each root-subtree pattern includes one or more root-path patterns, corresponding to the root-paths of its root-subtrees. The **size** of a root-subtree pattern is the number of root-path patterns it contains. The **values** of a pattern are all the values of its instances. The only patterns we consider hereafter are those of root-subtrees.

Information theory allows us to measure the association between the values of random variables in tables [28]. We now translate information theory metrics to apply to XML. Each root-path pattern p represents a discrete random variable that takes value a with probability $P(a) = \frac{1}{n} count(a)$, where $count(a)$ is the number of instances of p with value a and n is the total number of instances of p in the DB. In Figure 3.1, if p = *bib paper title -1 -1*, $P(p =$"XML Design"$) = \frac{1}{2}$. Since a root-subtree pattern defines the association between the patterns of its root-paths, the root-subtree pattern represents the joint distribution of random variables. For instance, the root-subtree pattern t_1 = *bib paper title -1 booktitle -1 -1* represents an association between root-path patterns p_1 = *bib paper title -1 -1* and p_2 = *bib paper booktitle -1 -1*. The probability of each value of a root-subtree pattern can be defined in the same manner as the probability of each value of a root-path pattern. For example, the probability of $P(p_1 =$"XML Design", $p_2 =$"SIGMOD") = $\frac{1}{2}$ in Figure 3.1. Generally, one value of a root-path can be associated with one or more values of other root-path(s). Before showing how to apply information-theoretic concepts to patterns, we quickly review the traditional definitions of entropy and joint entropy.

Definition 3.4.3. *Given a pattern p that takes values $a_1, \ldots, a_n$ with probabilities $P(a_1), \ldots, P(a_n)$ respectively, the* ***entropy*** *of p is $H(p) = \sum_{1 \le j \le n} P(a_j) \lg (1/P(a_j))$.*

Intuitively, the entropy of a random variable indicates how predictable the variable is. The entropy is minimal (zero) when all instances of p have the same value, and maximal ($\lg n$) when no two instances of p have the same value. When the pattern is not a root-path pattern, we refer to its entropy as *joint* entropy as well, because it explains the behavior of a joint random variable. We refer to the *joint* entropy of pattern t both by $H(t)$ and $H(p_1, \ldots, p_n)$ where $p_1, \ldots, p_n$ are the root-path patterns of t. The properties of XML entropy are the same as for tabular data, except for the following property:

Proposition 3.4.4. *If root-tree pattern t has exactly one root-path pattern p, then $H(p) \leq H(t)$.*

This property does not hold for traditional joint entropy, where $H(x, x) = H(x)$.

In the context of the relational model, researchers [31] have defined FDs based on conditional entropy [28]. We extend conditional probability and conditional entropy to apply to XML variables:

Definition 3.4.5. *Given pattern t containing two root-path patterns p_1 and p_2 that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$. respectively, the* ***conditional probability*** *of $p_1|p_2$ is: $P(p_1 = a_i|p_2 = b_j) = \frac{P(p_1=a_i,p_2=b_j)}{P(p_2=b_j)}$.*

Definition 3.4.6. *Given pattern t including two root-path patterns p_1 and p_2 that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$ respectively, the* ***conditional entropy*** *of $p_1|p_2$ is:*

$$H(p_1|p_2) = \sum_i \sum_j P(p_1 = a_i|p_2 = b_j) \lg (1/P(p_1 = a_i|p_2 = b_j)).$$

The conditional entropy of two root-path patterns is the entropy of the first one given information about the value of the second. The next property follows from the properties of conditional entropy in the original formulation of entropy:

Proposition 3.4.7. *Given pattern t with two root-path patterns p_1 and p_2, $H(p_1|p_2) = H(p_1, p_2) - H(p_2)$. Furthermore,*

$$H(p_1, \ldots, p_n) = \sum_{1 \leq i \leq n} H(p_i|p_1, \ldots, p_{i-1}).$$

3.4.2 Coherency Ranking & NTC

Mutual information [28] measures the correlation between two random variables:

Definition 3.4.8. *The* ***mutual information*** *of random variables A and B that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$ respectively, is:*

$$M(A, B) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} p(a_i, b_j) \lg (p(a_i, b_j)/p(a_i)p(b_j)). \tag{3.1}$$

where variable A, B takes value (a_i, b_j) with probability $p(a_i, b_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

We have:

$$M(A, B) = H(A) + H(B) - H(A, B). \tag{3.2}$$

We extend the traditional definition of mutual information to apply to XML.

Definition 3.4.9. *The **mutual information** of a pattern t with size two is*

$$M(t) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} P(a_i, b_j) \lg \left(P(a_i, b_j)/P(a_i)P(b_j)\right) \tag{3.3}$$

where t takes value (a_i, b_j) with probability $P(a_i, b_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

According to the definition of the values of a pattern, the a_is and b_js are the values of the root-paths of pattern t. For instance, in Figure 3.6 pattern *imdb movie title -1 locations location -1 -1 -1* takes values ((“For Love of Money”,“New York”),(“Big Love”,“Fox Hills”)). Therefore, we have P(“For Love of Money”,“New York”) = $1/2$ and P(“Big Love”,“Fox Hills”) = $1/2$. Also, we have P(“For Love of Money”) = $1/2$, P(“Big Love”)= $1/2$, P(“New York”) = $1/2$, P(“Fox Hills”)= $1/2$. Thus, the value of mutual information for this pattern is 1.

Mutual information measures the reduction of uncertainty in the values of one root-path pattern that comes from knowing the value of the second one. Mutual information is minimal when each value of one root-path pattern is associated with every value of the other root-path pattern, and is maximal when each value of one root-path pattern is associated with just one value of the other and vice versa. For instance, the mutual information of pattern *bib paper title -1 author -1 -1* whose paths are *bib paper title -1 -1* and *bib paper author -1 -1* is higher than the mutual information of *bib paper booktitle -1 author -1 -1* whose paths are *bib paper booktitle -1 -1* and *bib paper author -1 -1* in Figure 3.1. Therefore, the relationship between the first two variables is stronger.

The next proposition follows immediately from the properties of mutual information, and is useful in computing mutual information.

Proposition 3.4.10. *Given pattern t containing root-path patterns p_1 and p_2, we have $M(t) = H(p_1) + H(p_2) - H(t)$.*

To compute the mutual information of t, we consider only the instances of the root-path p_i that are subtrees of the instances of the root-pattern t. Notice that with non-XML mutual information we have $M(A, A) = H(A)$, but as redefined for XML data, we have $M(t) \leq H(p)$, where t includes only one root-path pattern p.

Total correlation [143] is closely related to mutual information; it measures the correlation between more than two random variables.

Definition 3.4.11. *The **total correlation** of the random variables $A_1, \ldots, A_n$ is:*

$$I(t) = \sum_{1 \leq i \leq n} H(A_1) - H(A_1, \ldots, A_n). \tag{3.4}$$

We extend this definition to apply to XML DBs:

Definition 3.4.12. *Let $p_1, \ldots, p_n$ be the root-path patterns of pattern t. The **total correlation** of t is:*

$$I(t) = \sum_{1 \leq i \leq n} H(p_i) - H(p_1, \ldots, p_n). \tag{3.5}$$

Total correlation does not consider the possibility that a pattern may have higher entropy just because it has more values. Similarly, total correlation goes up when one root-path pattern has more diverse values, even if it is not correlated with the other root-path patterns. For example, in Figure 3.3, consider patterns $t =$ *bib proceedings editor name -1 -1 title -1 -1 -1* and $t' =$ *bib proceedings paper author name -1 -1 title -1 -1 -1*. Intuitively, the relationship between a paper and its author is as close as between a proceedings and its editor. But the root-paths ending with *paper title* and *paper author* have more values than those ending with *proceedings title* and *proceedings editor*, so $I(t) > I(t')$. NTC addresses this, in a manner similar to that used to normalize mutual information [147]. We define the **normalized total correlation (NTC)** of t as:

$$\hat{I}(t) = f(n) \times (\frac{I(t)}{H(p_1, \ldots, p_n)}). \tag{3.6}$$

It can be tricky to compare patterns of very different sizes. As discussed earlier, adding new nodes to a pattern that already contains all the query terms should not improve its rank. NTC solves this problem by dividing the total correlation by the sum of the entropies of all root-path patterns, thereby penalizing sets with more root-path patterns. Nonetheless, the range of total correlation values for $n > 1$ root-path patterns is $[0, \frac{n-1}{n}]$, as the maximum total correlation for n root-path patterns is reached when they all have the same entropy as their root-subtree pattern. Thus as n grows, the total correlation may also increase, which may penalize small candidate answers during ranking. NTC removes this bias by including a factor $f(n)$ that is a decreasing function of the answer size (number of root-path patterns) n, for $n > 1$. Based on the observations above, we can stipulate that as n grows, $f(n)$ must decrease at least as fast as $\frac{n}{n-1}$. Through empirical observation we found that $f(n) = n^2/(n-1)^2$ performs well in practice. A systematic exploration of the options for $f(n)$ is an interesting area for future work.

CR uses NTC to rank subtrees and filter answers for XML keyword queries, as follows. First we extend each candidate answer to be a root-subtree, by adding the path from the root of that answer to the root of the DB. Then we rank the candidate answers in the order given by the NTC of their root-subtrees' patterns. (We rank patterns with only one root-path highest, in descending order of entropy, as our $f(n)$ function is undefined for them.) If desired, one can set a low threshold $NTC_{min} \geq 0$ appropriate for the domain, and include only candidate answers whose patterns have an NTC greater than NTC_{min}. With a good ranking function such as NTC, one does not really need this cutoff, as the worst answers appear last; but its use can improve the user experience by removing particularly unhelpful candidate answers.

For example, consider the query *XML SIGMOD* in Figure 3.1. When computed over all of DBLP, the NTC of the *title* of a paper and its *booktitle* is 1.47, so node 3 will be returned as an answer. The subtree that connects nodes 18 and 10 will be returned as the second answer (NTC = 0.42), and the root node will be ranked last (NTC = 0). For Q_2 = *Integration VLDB* in Figure 3.1, pattern *bib paper cite paper title -1 -1 -1 -1 paper booktitle -1 -1* has NTC = 0; its candidate answers will be ranked last. Similarly, *bib paper title -1 booktitle -1 -1* (NTC = 1.47) outranks *bib paper cite paper title -1 -1 paper booktitle -1 -1 -1 -1* (NTC = 0.84). CR also correctly differentiates between the relationship of nodes 15 and 16 and nodes 2 and 3 in Figure 3.1, and ranks the former higher because the NTC of two papers cited in the same publication is greater than the NTC of two papers listed in the same bibliography. In this chapter, we set $NTC_{min} = 0$ unless otherwise noted, which is an extremely conservative setting.

NTC sheds light on the behavior of previously-proposed heuristics. In SLCA, the intuition is that closer nodes are more strongly correlated. DL assumes that repeated root-path patterns will have lower correlation than non-repeated root-path patterns in a subtree. MLCA assumes that two root-path patterns that are closer together will be more correlated than two that are far apart.

3.5 Precomputing Coherency Ranks

NTCs can be computed offline in advance with a populated version of the DB, and then used at query time. In this section we introduce methods to make this precomputation efficient.

A naive approach to finding NTCs for all DB patterns is to generate all patterns, find all their instances, then compute the instances' NTCs. We can generate all patterns using existing data mining algorithms to generate candidate subtrees in a forest [156, 140, 103]. These works find trees in the forest that have at least one subtree isomorphic to the candidate; but to compute its NTC, we must find *all* instances of a pattern. To do this, we can express a newly found pattern as an XML twig query and use existing algorithms to find its instances [12, 92].

Unfortunately, the naive method is so inefficient as to be impractical. First, this method generates and finds all types of subtrees, but we only want root-path subtrees, and there are relatively few root-paths in XML DBs [22]. Twig query algorithms examine each node in the DB, while we only need to consider root-paths to find all the instances of a root-tree pattern. Second, to compute the entropy of a pattern, we will have to find and store all its values in a table-like data structure and then scan the stored values. This table can be much larger than the DB itself. If the XML file is huge and has a deep nested structure, the size of the table could be prohibitively large. For example, consider the size of such a table for the pattern *bib paper cite paper title -1 -1 -1 cite paper title -1 -1 -1 -1* in Figure 3.1.

Third, each NTC must be computed separately; there is no upward closure property [10] to help us, no general way to detect that NTC is below a cutoff threshold [100], no way to use sampling [67] (because the number of distinct values is quite close to the number of instances for some patterns), and no way to use XML query selectivity estimation techniques (because they assume that the conditional entropy between root-paths is zero). Fourth, as there is no limit on the size of a candidate pattern, the process of finding all patterns in a large database can take an extremely long time. To address this problem, the tree mining methods use the number of instances of a pattern as a way to prune unlikely candidate answers. Unfortunately, we do not expect our target patterns to be relatively infrequent. There could be very few infrequent patterns, due to noisy data. Finally, these methods scan the whole database to find the instances of each pattern, which is very costly.

We use four approaches to speed up precomputation. First, we exploit the typical characteristics of users' keyword queries. Second, we use properties of NTC and entropy to reduce computation. Third, we use compressed indexes to save memory space and speed up pattern generation. Fourth, we introduce methods to estimate entropy and approximate NTC efficiently. We discuss each of these below.

Keyword Search Parameters. Previous studies of internet document retrieval have shown that the average query has roughly 2.5 keywords [145]. As users want specific answers, we expect the number of nodes in the user's ideal answer to be quite low. That is, if the query has many keywords (e.g., a long paper title), probably many of those words reside in just a few nodes of the top-ranked answer. Hence we introduce a domain-dependent upper bound MAS (maximum answer size) on the value of n (pattern size) considered when precomputing NTCs. Since the average length of keyword queries for different bibliographic databases is between 1.81 to 3.66 including stop words [148], setting MAS to 4 or 5 seems reasonable for bibliographic DBs.

Zero NTCs. We refer to the root of a candidate answer as the LCA of its pattern. We show that if a pattern's LCA has only one instance in the DB (e.g., the DB root), its NTC is very close to zero.

A **prefix-path** is a path whose root is the root of the DB and its last node is not a leaf node or a parent of a leaf node. For instance, */bib/paper* is a prefix-path in Figure 3.1. If the last node of prefix-path s is the LCA of pattern p, s is a prefix-path of p. For instance, */bib/paper* is a prefix-path of *bib paper title -1 booktitle -1 -1* in Figure 3.1. Prefix-paths such as u that are created by adding nodes to prefix-path s are the **super-paths** of s. For instance, */bib/paper* is a super-path of */bib* in Figure 3.1. Every prefix-path has at least one **instance** in the DB. For example, */1/2* is an instance of */bib/paper* in Figure 3.1. Each instance of prefix-path s is a **prefix-path instance** of at least one instance of s's patterns. Given i that is an instance of prefix-path s, the distinct values of root-path p whose prefix-path instance is i is a **projection** of p under i, which is written as V_p^i. $|V_p^i|$ shows the number of such values.

Consider pattern t that contains paths $p_1, \ldots, p_n$ and its prefix-path s has only one instance in the DB. If $p_1, \ldots, p_n$ do not share any other prefix-path that is a super-path of s, each instance of p_i is associated with every instance of p_j, for $i \neq j$ and $1 \leq i, j \leq n$. Therefore, for the joint probability of $(p_1, \ldots, p_n)$ we have: $P(p_1, \ldots, p_n) = P(p_1) \cdots P(p_n)$. Hence, $H(t) = \sum_{1 \leq i \leq n} H(p_i)$. Considering formula (4.3), the value of NTC of t will be zero. Assume that $p_1, \ldots, p_n$ share at least one prefix-path l other than s. l is a super-path of s. Therefore, each instance of p_i will be associated with at least $|V_{p_j}^s| - |V_{p_j}^l|$ instances of p_j, for $i \neq j$ and $1 \leq i, j \leq n$ where $|V_{p_j}^l| = \max_{m \in l} |V_{p_j}^l|$ and m is an instance of l. If the value of $|V_{p_j}^l|$ is far less than the value of $|V_{p_j}^s|$, similar to the previous case, we can assume that: $P(p_1, \ldots, p_n) = P(p_1) \cdots P(p_n)$ and therefore, the value of NTC of t will be zero. This usually happens for the patterns whose LCAs is the root of DB. For instance, the number of distinct values for $/bib/paper/title$ or $/bib/paper/author$ under the root of DB are by far larger than the number of distinct values for the same paths under $/bib/paper/$. In other words, knowing the value of root-path p_i does not help to narrow down the value of another root-path p_j when their only relationship is that they belong to the same DB.

During the parsing of the input XML data set, we identify the prefix-paths such as s that have only one instance in the DB and store the numbers of the distinct values of their paths under all possible super-paths. If the numbers of the distinct values of all paths under s are by far larger than the numbers of their distinct values under all super-paths of s, we ignore all patterns whose LCA is s during NTC computation.

Entropy Computation. Since we need only to compare different values of a pattern to compute its entropy, we can use the hashed values of the pattern instead of its actual values. This reduces the memory consumption of the entropy computation process and speeds up the comparison operation, as the hashed values generally occupy less space than the original values. The precision of this method depends on how conflict-free the hash function is. Our experiments use the FNV-64 hash function [47], whose output is relatively short and is almost conflict-free.

Approximation. To reduce the number of patterns to find, we adapt an approximation approach designed for mutual information [80]. We use another correlation measurement in information theory called **interaction information** [97]:

Definition 3.5.1. *The **interaction information** of random variables $A, \ldots, A_n$ is:*

$$Intr(t) = -\sum_{T \subseteq \{A_1, \ldots, A_n\}} (-1)^{n-|T|} H(T). \tag{3.7}$$

We extend the definition of interaction information [97] to work with XML:

Definition 3.5.2. *Let $p_1, \ldots, p_n$ be the root-path patterns of pattern t. Then the **interaction information** of t is:*

$$Intr(t) = -\sum_{T \subseteq \{p_1,\ldots,p_n\}} (-1)^{n-|T|} H(T). \tag{3.8}$$

In contrast to total correlation, this metric subtracts the correlation inside the patterns defined by T from the correlation of T itself. For instance, consider (bib/book/title, bib/book/author, bib/book/price), where all share the same *book* parent node, in a bibliographic DB. Their interaction information is close to zero because the correlation of (title, author) with the price of the book is almost equal to the sum of the correlations of (title, price) and (author, price). On the other hand, the interaction information between root-path patterns (bib/book/ISBN, bib/book/title, bib/book/author), where they share the same *book* parent, is negative. This is because the combination of ISBN and title does not provide more information about the author than ISBN alone.

We can compute the entropy of a pattern t with root-paths $p_1, \ldots, p_n$, by using the entropies and interaction information of its subtrees T as follows [80]:

$$H(t) = \sum_{1\leq i\leq n} H(p_i) + \sum_{T \subseteq \{p_1,\ldots,p_n\}} (-1)^{|T|} Intr(T). \tag{3.9}$$

We can approximate the entropy of pattern t by computing the information interaction of subtrees T of size m, up to a value $m \leq EV$, where $0 < EV < n$. The formula works well if the sum of the correlations of the subtrees T is close to the total correlation of the pattern, as for title, author, and price in the last example. However, its error is quite high for cases like ISBN, title, and author, where there are multiple keys or quasi-keys (e.g., ISBN and book title) in the pattern t. While keyword queries do not usually provide multiple key or quasi-key values, experiments with real-world queries showed us that the errors were high enough in practice for us to prefer a different formula, based on the fact that the maximum value of the entropy of a pattern t is the summation of the entropies of its root-paths $p_1, \ldots, p_n$:

$$H(t) = \sum_{1\leq i\leq n} H(p_i) + \min\,(0, \sum_{T \subseteq \{p_1,\ldots,p_n\}} (-1)^{|T|} Intr(T)) \tag{3.10}$$

Using the above approach, we compute the exact NTC of patterns up to a certain size $EV \leq MAS$. Then, we approximate the NTC of larger patterns using formula (3.10).

How should we choose the value of EV? Even if the information interaction starts to approach zero for subset T of size $m < n$, it may increase for the subsets of size $m+1$. Thus, one should set EV to be as large as possible, based on the computational time available for precomputation (e.g., half a day). Our experiments in Section 3.7 on multiple data sets show that even low values of EV are very effective.

3.6 Search & Ranking Algorithms

```
Input: XML data file $data$
Input: Maximum size $EV$ of patterns to compute exact NTC for
Input: Maximum size $MAS$ of patterns to compute exact or estimated NTC for
Input: Minimum frequency thresholds $MIN\_FREQ$
Output: Table $CT$ of NTCs for $data$

   /* Create path indices for all frequent root-path patterns */
1  $indxs$ = Depth_First_Scan($data$, $MIN\_FREQ$);
   // Compute the entropy for root-path patterns
2  forall $p \in indxs$ do
3      $p$.entropy();
   /* Initialize the set of prefix classes */
4  $pfxSet \leftarrow \emptyset$;
   /* Add all root-path patterns as one prefix class */
5  $pfxSet$.add($indxs$);
6  for $k = 2$ to $MAS$ do
7      $nextPfxSet \leftarrow \emptyset$;
8      last = ();
9      forall $pfx \in pfxSet$ do
10         forall $p \in pfx$ do
               /* Compute all prefix classes whose prefixes are $p$ */
11             $nextPfx \leftarrow \emptyset$;
12             forall $q \in pfx$ do
13                 $Jnt \leftarrow$ join_Pattern($p, q$);
14                 forall $r \in Jnt$ do
15                     if subTrees($r$) $\not\subset pfxSet$ then
16                         continue;
17                     if $k \leq EV$ then
                           /* Join indices and get frequency as well as NTC */
18                         $w \leftarrow$ join_Indices($r, indxs$);
19                         if $w.freq < MIN\_FREQ$ then
20                             continue;
21                         CT[$r$] $\leftarrow w.\hat{I}$;
22                     else
23                         CT[$r$] $\leftarrow$ approximate$\hat{I}$($r$);
24                     if $k \neq MAS$ then
25                         $nextPfx$.add($r$);
26             if $k \neq MAS$ then
27                 $nextPfxSet$.add($nextPfx$);
28     $pfxSet \leftarrow nextPfxSet$
29 return $CT$;
```

Figure 3.7: Algorithm to compute NTCs

3.6.1 Ranking Precomputation Algorithms

The algorithm in Figure 4.5 shows the overall process of computing NTCs. Our first challenge is to generate the patterns efficiently (lines 1-20), by generating only root-subtree patterns and generating every pattern only once. For efficiency, we should use information gained in previous stages to find instances of future patterns. As discussed earlier, modern tree mining algorithms produce new candidates by joining current candidate trees together, and adding new nodes to current candidate trees. When extending a candidate, these algorithms cannot use information about the positions of the instances of the current candidate in the DB. Therefore, they have to scan all the information in the DB to enumerate the number of instances for the new candidate. Furthermore, they generate many candidates that are not patterns. Therefore, we extend current tree mining approaches to generate patterns more efficiently.

To help ensure that we generate patterns only once, we impose a somewhat counterintuitive transitive $<$ relationship on patterns, where -1 is greater than all other labels:

Definition 3.6.1. *For patterns t and t', we have $t < t'$ iff the prefix string of t' is a prefix of the prefix string of t, or the prefix string of t is lexicographically greater than the prefix string of t'. We have $t = t'$ if both patterns have the same prefix string.*

The first part is counterintuitive, but it will be justified when we prove the correctness of our algorithm.

Two subtrees of t are *siblings* if their roots are siblings in t.

Definition 3.6.2. *A pattern t is* ***sorted*** *when for every pair of sibling proper subtrees u and u', $u \leq u'$ iff the root of u appears before the root of u' in the prefix string for t.*

For instance, pattern *bib paper author -1 title -1 -1 paper author -1 -1* is sorted, while pattern *bib paper author -1 -1 paper author -1 title -1 -1* is not.

Lemma 3.6.3. *Every prefix of a sorted pattern is sorted.*

The proof of the lemma uses the concept of the **last path** of tree t, which is the path that starts at the last node with only one child in the depth first traversal of t, and ends at the last node in the depth first traversal of t. For instance, *10 SIGMOD* is the last path in Figure 3.1. The *last rightmost node* of a subtree is the last node in its last path.

Proof. Suppose we have $s \geq r$ for subtrees s and r. Let s' be obtained from s by removing its last rightmost node. Then $s' \geq r$. Removing the last rightmost node of a sorted tree t will remove the rightmost node of the proper subtrees of t whose roots are in the rightmost root-path of t. Therefore, these subtrees will still be greater than or equal to their left siblings. $\square$

The **level** of a last path is the level of the parent of its root in t. Everything that is not on the last path of t is t's **prefix-tree**. All patterns with the same prefix-tree pattern form a **prefix class**. For example, all root-path patterns belong to the same prefix class, as their prefix-tree is empty. We can describe a pattern t as $t = (px, pa, l)$, where px is its prefix-tree, pa is its last path, and l is the level of its last path. We use the following operation in line 13 of the NTC computation algorithm in Figure 4.5 to generate sorted patterns of size n from sorted patterns of size $n - 1$.

Definition 3.6.4. *Given patterns $t = (px, pa_t, l_t)$ and $r = (px, pa_r, l_r)$, we define their **join** $\oplus$ as follows:*

- *If $l_r < l_t$, then $t \oplus r = (t, pa_r, l_r)$.*
- *If $l_r = l_t$, then $t \oplus r$ is a set of patterns of the form $s = (t, pa_r, l_r)$. For each common prefix path c of pa_t and pa_r such that $c \neq pa_t$ and $c \neq pa_r$, we include pattern $s = (t, pa_s, l_s)$, where pa_s is created by removing the nodes of c from n_r. The level of pa_s is $l_r + |c|$.*
- *If $l_r > l_t$, then $t \oplus r$ is null.*

For example, the patterns *bib paper author -1 -1* and *bib paper title -1 -1* have the same prefix-tree (null) and the same levels (0). Therefore their join contains *bib paper author -1 -1 paper title -1 -1* and *bib paper author -1 title -1 -1*. The two generated patterns belong to the same prefix class. The join of the first of them with the second is null, as the level of the first (0) is less than the level of the second (1). The result of joining the second pattern with the first one is *bib paper author -1 title -1 -1 paper title -1 -1*. The NTC algorithm in Figure 4.5 only needs to join patterns from the same prefix class (lines 12 and 13). The join of two sorted patterns is not generally sorted, so the call to join_Pattern in line 12 must check the result patterns to ensure that they are sorted. We call a pattern a **node-subtree** of pattern t at node v if it is induced by removing all nodes from t except for v and its descendants (if any). To see if the pattern is sorted, we must compare every node-subtree on the rightmost path of the pattern with its left sibling (if any). This function takes $O(m(d-1))$ time, where m is the number of nodes in the pattern and d is the length of its rightmost path. We ignore patterns at level 0 in our implementation, because their LCA is the tree root, so their NTC is zero. The join operation itself takes $O(m)$ time. Therefore, the exact time complexity of each join and check operation is $O(m(d-1))$.

Theorem 3.6.5. *Given the set of all root-path patterns in the DB, the join operation followed by the check operation will generate every sorted pattern exactly once.*

Proof. We prove that the algorithm generates all possible sorted patterns by induction on the size of the pattern. Assume that all possible sorted patterns of size $n - 1$ can be generated by the algorithm. According to Lemma 3.6.3, the prefix of a sorted pattern is sorted. Thus, the prefixes of patterns of size n are in the set of generated patterns of size $n - 1$. All possible last paths are also in the generated set. Therefore, the algorithm generates all the sorted patterns. Since the sorted patterns are unique, they will not be generated more than once. The exception is the case when sibling node subtrees are equal. However, these patterns are generated from two equal patterns of size $n - 1$. As we join each pattern with itself just once, the resulting patterns are generated only one time. □

Since the process of generating patterns proceeds level by level, it can take advantage of the Apriori technique. That is, it can prune patterns of size m that have subtrees that do not have any instance in the DB or have very few instances in the DB (lines 15 and 19), before finding their instances in the DB. Patterns of very low frequency often indicate noise in the DB. Line 13 of the NTC algorithm in Figure 4.5 calls the join_Pattern algorithm to generate the sorted patterns. The time complexity of generating new patterns of a particular size is $O(n^2)$, where n is the number of patterns of size $2 \leq k \leq MAS$. All the patterns found so far are stored in a hash table in main memory. The function *subTrees* in line 15 finds all subtrees of size $k-1$ for a pattern of size k. Therefore, checking whether all subtrees of a pattern are frequent takes $O(s)$ time, where s is the size of the pattern. For patterns of size larger than EV, the NTC algorithm in Figure 4.5 uses the approximation technique discussed in the previous section: it finds all the subtrees of the pattern, then uses formula (3.10) to approximate the total correlation. This step takes $O(s^3)$ time. The algorithm continues until it reaches the MAS answer size limit.

The second challenge in computing ranks is to efficiently find the instances of the generated patterns in the DB. Since the most important parts of patterns are root-path patterns, line 1 of the algorithm in Figure 4.5 scans the DB in a depth first manner and stores the instances of each root-path pattern in a separate path index. We use Dewey encoding [125] to store the root-path patterns in their path indices. Dewey encoding assigns a vector of numbers to each node. It is a combination of the Dewey number of the parent of the node and the relative position of the node among its siblings. Figure 3.4 shows the Dewey numbers for nodes of the DB tree. Each entry in the path index contains the Dewey number [125] of the parent of the leaf child of an instance, as well as the hash of the leaf child data value. For example, the path index for path */bib/incollection/title* in Figure 3.4 is (0.0.0, hash(MPI Techniques)), (0.1.0, hash(MPI Applications)). To reduce the index size, we use bitmaps to represent Dewey codes, and store the index sorted on Dewey number. In line 19, the algorithm prunes all the patterns with fewer instances than a minimum frequency threshold. The patterns with very few instances tend to have very high entropy and therefore higher NTC than more frequent patterns.

Path indices can join at different levels. The entries whose Dewey codes have the same prefix of size $l+1$ can join at level l. The join of the path index given above with itself at level 1 is given by (0.1, $\{$hash(Moore), hash(Tob)$\}$). Since the path indices are sorted on their Dewey codes, we can compute the instances of each pattern p by performing a zigzag join [50] on the path indices of p's root-path patterns (line 18). Thus, we do not need to scan the DB to find the instances of a pattern. Also, since we join the path indices in a certain order, we generate each result at most once. When joining the root-path pattern indices in line 18, the *JoinIndices* routine inserts the values of the new pattern into a temporary hash table. Then, the algorithm computes the exact NTC for the new pattern at line 21, based on the values stored in the temporary hash table. If the NTC will be approximated for larger patterns, then line 21 must also compute and store the entropy of this pattern. If a pattern has EV or more root-path patterns, the algorithm

	DBLP	IMDB	XMark
Size (in MB)	207.2	1038.8	233.7 - 2113.5
Max Depth	5	7	11

Table 3.1: Data set statistics

EV	DBLP	IMDB
2	0.290	0.250
3	0.126	0.106
4	0.061	0.052

Table 3.2: Approximation error for DBLP and IMDB

approximates the value of NTC at line 23, instead of computing the exact NTC.

3.6.2 Query Processing

The NTC value for each pattern resides in a hash table in main memory during query processing. The query processing system finds each candidate answer, generates its pattern, looks up the NTC for that pattern, and then ranks the answer accordingly. The problem of finding all candidate answers given a keyword query has been addressed in previous work [61, 124]. The algorithm proposed in [124] returns only the root of the subtree, while we need the whole structure of the subtree in order to find its pattern. The SA algorithm [61] returns the LCA, leaf nodes, and leaf node parents of each candidate answer. We extended SA to also produce the pattern of a candidate answer.

The performance of SA has been studied before [61, 124, 85], and our extensions do not affect the asymptotic time complexity of SA. Since the NTC of the candidate answers whose LCA is the root of the tree is zero, we changed SA to omit the DB root as a candidate LCA. This optimization helped the extended SA to perform better than the original SA for our queries. As shown in [61], in some circumstances the number of relevant answers is asymptotically larger than the number of candidate answers that the SA algorithm returns. However, we decrease the number of candidate answers considerably by never considering the root of the XML tree as a candidate answer.

We used Berkeley DB 4.6 [8] to store the XML data sets. Each node occupies a separate entry in the database table. The key of the node entry is the node's position in the depth-first traversal of the DB, and the value of the entry is its parent Dewey code and its content (if the node is a content node). We built an index on the parental information of the nodes to facilitate finding the parents of a node, as the SA algorithm requires. Also, we store an inverted index for the content stored in the database table.

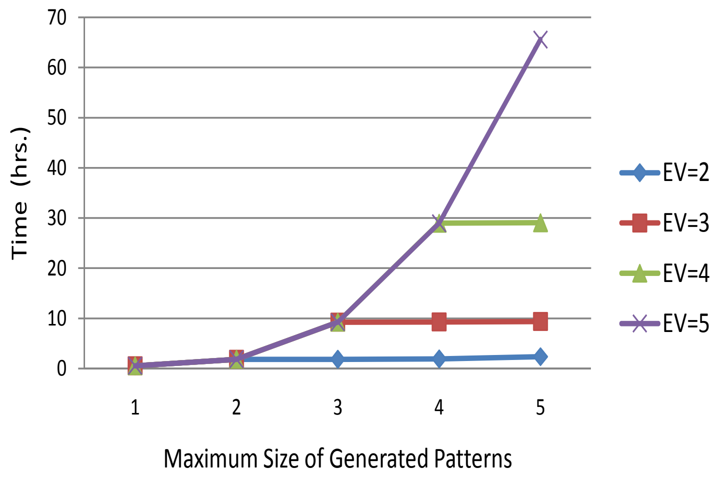


Figure 3.8: Precomputation performance for DBLP without hashing

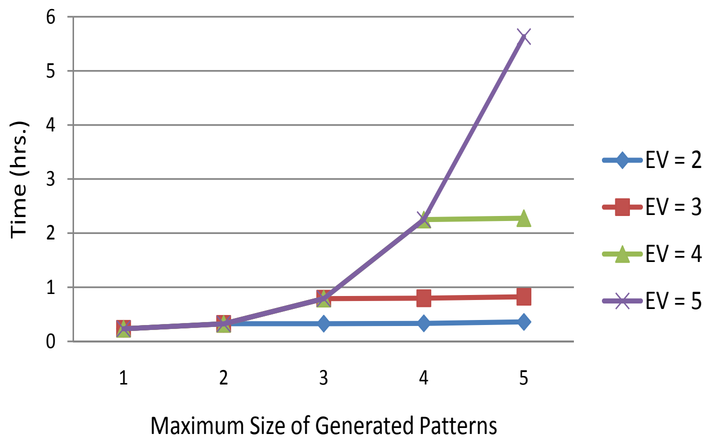


Figure 3.9: Precomputation performance for DBLP with hashing

3.7 Evaluation

We implemented and evaluated the precomputation algorithm and query processing system on a Linux machine with 4 GB memory and an Intel Pentium D 2.80 GHz. Our experiments use two real-world data sets and one synthetic data set: DBLP, IMDB, and XMark [120]. Table 3.1 presents the properties of the data sets. We used 14 different Xmark data sets whose sizes are in the range of 233.7MB to 2113.5MB to evaluate the scalability of the precomputation algorithm. We did not consider the font tags such as *emp* as elements in XMark. No previous approach has been tried on data sets as large as our version of IMDB, or as large and deep as our XMark. Because CR is not intended for use with long text fields, we did not include the long IMDB fields such as *plot* and *bloopers* in our experiments.

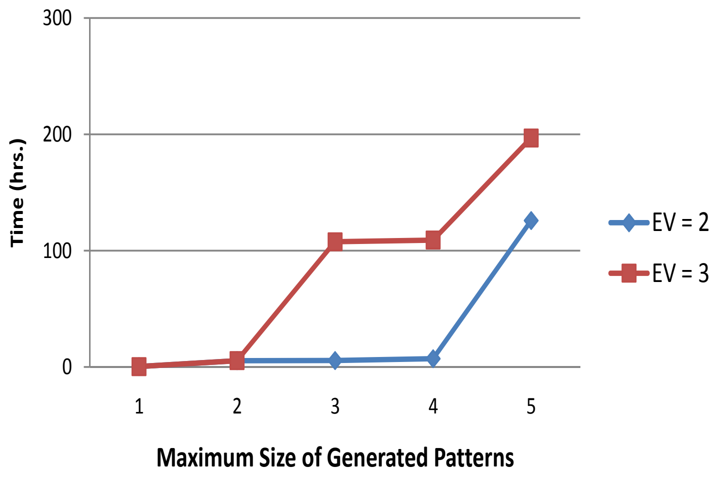


Figure 3.10: Precomputation performance for IMDB without hashing

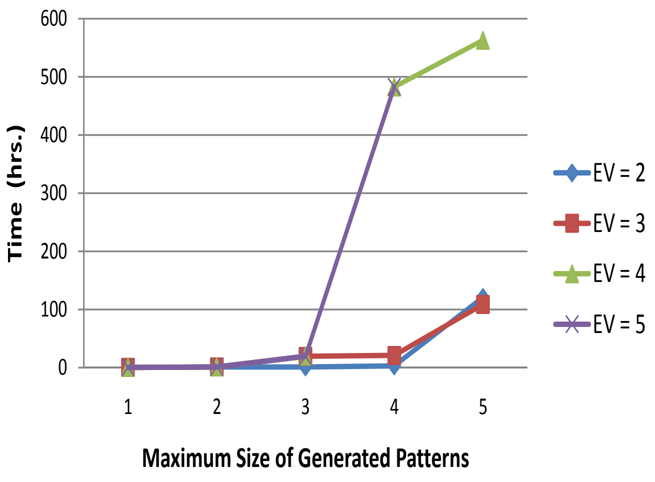


Figure 3.11: Precomputation performance for IMDB with hashing

3.7.1 NTC Precomputation Performance

We implemented the NTC precomputation algorithm in C++ using Apache's Xerces SAX parser. We set the minimum frequency threshold to 50 for all data sets; any value between 2 and 50 would have produced the same results, because IMDB and XMark have no patterns with frequencies between 1 and 49, and DBLP has only one such pattern. We set the maximum answer size MAS to 5 for DBLP and IMDB, which is a reasonable threshold for both domains as argued earlier.

Figure 3.8 and Figure 3.10 show the time spent on the exact and approximate computations for DBLP and IMDB, broken down by the value for EV, when we used the original values of the content nodes to compute NTC. This includes the time to parse the XML data and create path indices. As expected, the time for exact computation increases

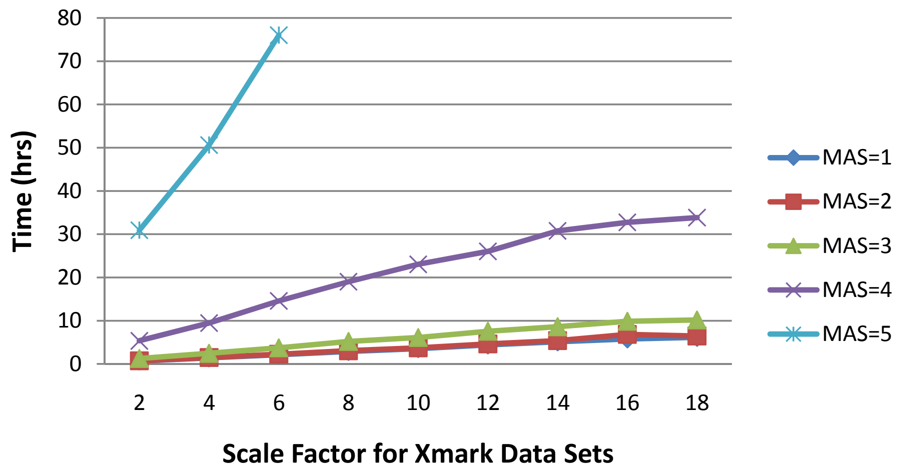


Figure 3.12: Performance of exact NTC computation for Xmark data sets

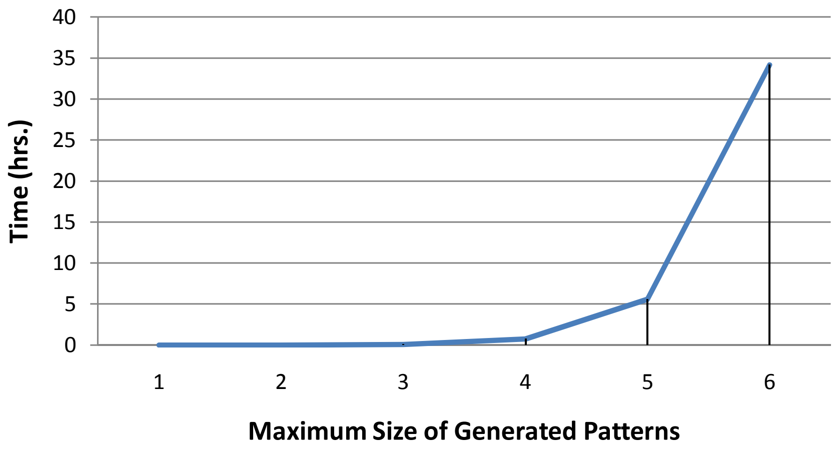


Figure 3.13: Performance of approximate NTC computation for Xmark data sets for $EV = 3$

exponentially as EV increases. The exact computation takes much longer for IMDB than for DBLP, because IMDB is larger and its structure is more nested than DBLP's, so IMDB has many more patterns and pattern instances. The exact computation for $MAS = 3$ in XMark of size 1117MB took about a week. For DBLP, the exact computation for $MAS = 5$ took about three days. The exact computation for $EV > 3$ in IMDB and XMark larger than 800MB took too long to run to completion. Overall, although approximation helps to reduce the precomputation time, precomputation still takes a long time. Thus, we repeated the experiments using the hashing technique described in Section 3.5. Figure 3.9 and Figure 3.11 depict the time spent on the exact and approximate NTC computations for the same data sets. Hashing reduces the exact NTC computation time for DBLP by more than a factor of 10 and the exact computation time for IMDB by about a factor of 5. The computation time for $EV = 3$ and $MAS = 5$ is less than for

$EV = 2$ and $MAS = 5$ in IMDB data set. The algorithm does not check wether the generated patterns whose sizes are more than EV have any instances in the database. Thus, it produces more spurious patterns for smaller EV. The exact NTC computation for $MAS = 3$ in XMark took about 87 hours. The approximate computation for $EV = 3$ and $MAS = 5$ took less than two weeks. The exact computation for $MAS = 5$ in IMDB took more than three weeks. Since the number of root-paths in an XML database is relatively small [22], the algorithm can cache all or most of the path indexes in memory. This makes choosing larger values for EV and even exact computation practical, depending on the size of the data set and the available time and resources.

Figure 3.12 shows the scalability of the exact computation for different values of MAS for Xmark data sets of size 233.7MB to 2113.5MB. The exact precomputation for $MAS = 5$ for Xmark data sets larger than 800MB took too long to run to completion. The figure shows that the precomputation algorithm scales linearly with respect to the size of the data set. Figure 3.13 shows the time spent on approximate computation of a typical Xmark data set for different values of MAS with $EV = 3$. Since the approximation time depends only on the number of pattern in the database, it takes the same time for XMark data sets of different sizes. As we see, the time scales exponentially with respect to the maximum size of generated patterns. The approximation algorithm took too long to run to completion for patterns larger than 6. Fortunately, keyword queries whose desired answers include more than 6 separate leaf nodes are very rare in practice [145, 148]. Thus, the NTC computation for such queries for very large and deep data sets (such as Xmark) can be done at query time. The query time algorithm keeps the precomputed patterns up to a given size, for instance 4, in main memory. If the size of pattern p of a candidate answer was more than the maximum size of the precomputed patterns, the query time algorithm finds all precomputed subtrees of p and approximates its NTC. Since the precomputed patterns reside in a hash table in main memory in the query-time, this step can be done very fast.

Table 3.2 shows the effectiveness of the NTC approximation technique. The table excludes XMark, because XMark content is generated randomly and is not meaningful, so the concept of an approximation error does not make sense for XMark. For the other two data sets, we sorted the list of all patterns of size up to MAS based on their exact or approximate NTC values for different values of EV. Then we computed the approximation error rate using Kendall's tau distance between two permutations of the same list of NTCs [77]. This measure penalizes the sorted approximate-computation list whenever a pattern occurs in a different position in the exact-computation list. Thus, if two lists have almost the same ordering, they have a low Kendall's tau distance. In the table, Kendall's tau is normalized by dividing by its maximum value, $n(n-1)/2$, where n is the total number of patterns in the list of NTCs.

One problem in using Kendall's tau is that the approximate and exact lists have different lengths. This is because the approximation procedure cannot distinguish and eliminate certain patterns whose frequency is less than the frequency threshold. For instance, it is not possible for an approximation run with $EV = 1$ to eliminate all patterns containing a paper with two titles. Our solution to this problem was to add these patterns to the exact list, with a zero

NTC. As shown in Table 3.2, the error rate decreases as EV increases. $EV = 3$ gives an error rate close to 10%. The approximation is a bit better for IMDB than for DBLP, because there are fewer key/semi-key elements in IMDB than in DBLP. Of course, the real test is wether approximation affects the ranking of actual user queries. In the next section, we show that is does not.

In many domains, NTC precomputation will only be done once; in domains with more fluid structure, NTC precomputation will still be a rare event, e.g., once a year. The NTC precomputation times seem quite reasonable for this rare task. For $MAS = 5$, the table of NTCs produced by precomputation occupied less than 2 MB for IMDB and about 1 MB for DBLP. Thus CR imposes only a modest memory overhead at query processing time. If desired, the table size could be further reduced by removing patterns whose correlations are below a threshold value.

3.7.2 Query Processing

At the time this research was performed, there was no standard benchmark to evaluate the effectiveness of keyword search methods for data-oriented XML. To address this problem, we used a TREC-like evaluation method [96]. We asked 15 users who were not conducting this research to provide up to 5 keyword queries for IMDB and DBLP. This resulted in 40 queries for IMDB and 25 for DBLP, shown in Table 3.3. There are more queries for IMDB than for DBLP because some of the users are not computer science researchers, and could not provide meaningful DBLP queries. We developed a keyword query processing prototype based on the baseline algorithm that returns every LCA for each query, without any ranking [119]. Our users submitted their queries to this system and judged the relevancy of each answer as relevant or irrelevant by selecting a checkbox in front of each returned answer. The prototype's user interface merged all the equal answers (subtrees with the same labels and contents), and did not present the user with any subtrees rooted at the root of the DB. We eliminated stop words from the queries before submitting them to the baseline algorithm.

Many submitted queries are relatively short. For instance, users submitted the main subject or parts of the title of a movie, with at most one related predicate, when searching for a movie in the IMDB database. This complies with the previous user studies on the length of keyword search queries [145]. There are query assistance tools for keyword search over relational databases, such as [101] that provide a ranked list of schema elements that match the keywords as user types keywords. Users can use the suggested schema elements and translate their keyword queries to the form $(e_1 : k1, \cdots, e_n : k_n)$ where $e_i, 1 \leq n$ is a schema element and $k_i, 1 \leq n$ is a keyword from input query. The keyword search interface can determine the related answers for the translated query more precisely. In order to build such a system, the DBA must make up user-friendly names for the schema elements. Also, since there may be hundreds and thousands of schema elements in the database, the DBA must group semantically similar schema elements. For instance, even a proper name such as *Clooney* matches *actor*, *director*, *producer*, *writer*, *special-*

Num	IMDB	DBLP
1	Beautiful Mind	Cis Regulatory Module 2008
2	Edward Norton	William Yurcik
3	Artificial Intelligence	Radu Sion
4	True Dreams	Barbara Liskov
5	Jackie Chan 2007	Hoarding
6	Basic Instinct	Authenticated Dictionary
7	Comedy Woody Allen Scarlett Johansson	Language Based Security
8	Peter Sellers Blake Edwards	CCS 2008
9	Belgian Detective	Closed Frequent Pattern
10	Slumdog Millionaire 2008	Social Network Evaluation
11	Crime the Godfather	Personalized Web Search
12	Forrest Gump	SASI Protocol Attack
13	Kate Winslet 1997	Trust Management Web Services
14	Pearl Harbor	Authorization Web Services
15	Susan Sarandon	VLDB Korea
16	The World of Apu	Web Conversation
17	Satyajit Ray	Maude
18	Doctor Zhivago	TinyOs Motes
19	Thumbs Up	Data Aggregation Sensys 2004
20	Grand Torino	Madden TinyDB
21	Brad Pitt Movie	TEEVE
22	Painted Skin	Stereo Vision
23	Ang Pamana	Social Visualization
24	The Owl and the Sparrow	The Dynamics of Mass Interaction
25	Hayden Christensen	Han Data Mining
26	Ben Barnes	
27	Christian Bale	
28	Fight Club	
29	The Watchmen	
30	Momento	
31	Return of the Mummy	
32	High School Musical	
33	Kate Winslet Award	
34	Britney Spears Disney	
35	Lara Croft Cambodia	
36	The Wizard of Speed and Time	
37	Mike Jittlov	
38	The Unforgiven Clint Eastwood	
39	Dakota Fanning Animation	
40	Disney Channel Movie Surfing	

Table 3.3: IMDB and DBLP queries

thanks, *miscellaneous-crew*, *cinematographer*, *soundtrack-info*, *character*, *keyword*, *art-department*, and *literature* in IMDB. The DBA can group these schema elements into 6 groups such as *people*, *soundtrack*, *companies*, *characters*, *keywords*, and *literature*. Nevertheless, CR is a domain independent keyword query interface, so we prefer to avoid reliance on domain-dependent knowledge whenever possible. Moreover. because humans are not good at considering large numbers of options [98], the number of suggested groups should not exceed seven. Given this limitation, creating

Num	IMDB	DBLP
1	Lasseter Hanks Docter	Jiawei Han Xifeng Yan
2	Brad Pitt Catherine George Clooney	Aiken Foster Kodumal
3	Boyle Trainspotting Vidler	Aiken Kodumal PLDI
4	Animation Adventure Comedy	Banerjee Amtoft Bandhakavi POPL
5	Terminator Arnold Claire	Amtoft Banerjee
6	Psycho Thriller Japanese	Hasan Winslett Radu
7	Kate Winslet Drama 2003	Hasan Winslett Policy
8	Clooney Pitt Roberts	Boyer Lee
9	Mckellen Bloom Jackson	Mittal Borisov Security
10	Drama Lynch Watts	Minami Lee
11	Clooney Pitt Mac	Han SIGMOD Frequent Patterns
12	Gould Affleck Caan	Keyword Search Databases
13	Jemison Cheadle Qin	Cabello Chambers
14	Reiner Damon Garcia	Planar Shortest Path
15	Robert Clooney Pitt	Klein Planar Flow
16	Tsunami Earth Ship	Surface Graph Computer Vision
17	Ghost Hunt Drama	TCC Public Key Cryptography
18	Family Life Nature	Bioinformatics Markov Chain
19	Romance Comedy Death	Xifeng Yan Graph Jiawei Han
20	Military World Comedy	SIGMOD Database Storage
21	Clooney Crime Warner	Binner Chen Networks
22	Julia Roberts Romance Fox	Ullman Molina SIGMOD
23	Brad Pitt Crime Warner	IPv6 Mobile Security
24	Hugh Grant Columbia	Cryptography Key Authentication
25	Clooney Action Fox	Ullman Molina Widom
26	Hijacking English	Lazebnik Schmid Ponce
27	World War Japanese	Agarwal Awan Roth
28	Mathematics Professor USA	Cootes Edwards Taylor
29		Nordstrom Nelson Phillips
30		Patterson Hennessy Architecture
31		Agarwal Roth Object Detection
32		Elmasri Navathe Database

Table 3.4: IMDB and DBLP complex queries

	Coherency Ranking	**XSearch**	**SLCA**	**MaxMatch**	**CVLCA**	**XRank**	**XReal**
Avg. Precision	0.687	0.189	0.679	0.679	0.164	0.166	0.360
Avg. Recall	1.0	1.0	0.915	0.915	1.0	1.0	0.250

Table 3.5: Average precision and recall of original methods for simple queries over DBLP

appropriate groups of schema elements for a database with a relatively complex schema will be a challenging task . Moreover, since there is more than one schema element in a group, the keyword search system still will not know the exact desired schema element(s) for each keyword. For instance, there are 7 schema elements in the *people* group, and some of them, such as *special-thanks*, *miscellaneous-crew*, and *cinematographer*, are not likely to be desirable. The same limitation applies to similar works such as [88].

However, we also wanted to measure the effectiveness of CR for structurally rich keyword queries, We define a

	Coherency Ranking	**XSearch**	**SLCA**	**MaxMatch**	**CVLCA**	**XRank**	**XReal**
Avg. Precision	0.592	0.054	0.558	0.558	0.057	0.058	0.211
Avg. Recall	1.0	0.975	0.798	0.798	0.975	0.976	0.202

Table 3.6: Average precision and recall of original methods for simple queries over IMDB

	Coherency Ranking	**XSearch-M**	**SLCA-M**	**MaxMatch-M**	**CVLCA-M**	**XRank-M**	**XReal-M**
Avg. Precision	0.687	0.687	0.679	0.679	0.687	0.687	0.360
Avg. Recall	1.0	1.0	0.915	0.915	1.0	1.0	0.250

Table 3.7: Average precision and recall of modified methods for simple queries over DBLP

	Coherency Ranking	**XSearch-M**	**SLCA-M**	**MaxMatch-M**	**CVLCA-M**	**XRank-M**	**XReal-M**
Avg. Precision	0.592	0.586	0.558	0.558	0.586	0.592	0.211
Avg. Recall	1.0	0.975	0.798	0.798	0.975	1.0	0.202

Table 3.8: Average precision and recall of modified methods for simple queries over IMDB

	Coherency Ranking	**XSearch-M**	**SLCA-M**	**MaxMatch-M**	**CVLCA-M**	**XRank-M**	**XReal-M**	**Baseline-M**
Avg. Precision	0.892	0.893	0.892	0.892	0.893	0.892	0.121	0.892
Avg. Recall	1.0	1.0	1.0	1.0	1.0	1.0	0.106	1.0

Table 3.9: Average precision and recall of modified methods for complex queries over DBLP

	Coherency Ranking	**XSearch-M**	**SLCA-M**	**MaxMatch-M**	**CVLCA-M**	**XRank-M**	**XReal-M**
Avg. Precision	0.545	0.548	0.545	0.545	0.548	0.545	0.404
Avg. Recall	1.0	1.0	1.0	1.0	1.0	1.0	0.446

Table 3.10: Average precision and recall of modified methods for complex queries over IMDB

	Coherency Ranking	**XSearch**	**PN**	**PN-M**	**XRank**	**XRank-M**	**XReal**
IMDB	0.715	0.642	0.511	0.627	0.421	0.609	0.420
DBLP	0.834	0.794	0.621	0.739	0.591	0.723	0.380

Table 3.11: Mean Average Precision (MAP) for simple queries over DBLP and IMDB

structurally rich keyword query as a keyword query that deals with at least two attributes of different types in the desired entities. For instance, query *Clooney Crime Warner* over the IMDB database refers to three three different attributes of a movie. *Clooney* matches 12 attributes, *Crime* matches 11 attributes, and *Warner* matches 12 attributes. The movies acted in or directed by *Clooney*, having the term *Crime* in their title, and distributed or produced by *Warner Brothers* company are among the desired answers. Some combinations of matching attributes are not desirable. For example, movies produced by *Clooney*, having the term *Crime* in their location, and having the term *Warner* in their camera crew are not ideal answers. These complex combinations of desired attributes cannot be captured by query

	Coherency Ranking	**XSearch**	**PN**	**PN-M**	**XRank**	**XRank-M**	**XReal**
IMDB	0.706	0.601	0.507	0.612	0.465	0.598	0.348
DBLP	0.956	0.939	0.785	0.956	0.629	0.920	0.106

Table 3.12: Mean Average Precision (MAP) for complex queries over DBLP and IMDB

assistance tools such as [101]. In order to collect such queries, we performed another round of user study. We provided our users with the definition of complex queries and some examples. Table 3.4 shows the collected complex queries for DBLP and IMDB.

We defined two different evaluation tasks for each query set to examine the effectiveness of CR. The first task compares CR with the methods discussed in Section 3.2.2: SLCA [149], XRank [57], CVLCA [85], MaxMatch [90], XReal [6], and XSearch [26] for our data sets to evaluate unranked retrieval.

XRank, XSearch, and XReal have tunable parameters. We explain these parameters and how we have set them in our experiments. XRank uses parameter $decay$ which ranges between 0 and 1 and limits the depth of the LCAs of candidate answers. XRank also uses the parameters d_1 and d_2 as damping factors in its customization of the PageRank algorithm:

$$e(v) = \frac{1 - d_1 - d_2}{N(v)} + d_1 \sum_{(u,v) \in CE} \frac{e(u)}{N_c(u)} + d_2 \sum_{(u,v) \in CE^{-1}} e(u) \tag{3.11}$$

Here, CE is the set of edges from parents to their children and CE^{-1} is the set of edges from children to parents in the DB tree. If there is only one type of edge in the database, then $d_2 = 0$ and the formula becomes the original PageRank formula. Geo et al. did not report effectiveness results for XRank and there has not been any study of the effect of tunable parameters on the effectiveness of XRank. We experimented with values for $decay$ between 0 and 1 our experiments. Since the damping factor in PageRank is 0.85 [11], we used non-zero values for d_1 and d_2, where $d_1 + d_2 = 0.85$, similar to [57]. According to our experiments, the ranking quality of XRank depends on the value of its tunable parameters. Our experiments also show that these parameters must be selected based on the properties of the data set and query set. In order to get realistic results and be fair to other methods, we trained XRank tuning parameters on a smaller training set and used those values for the actual data set and query set. Thus, we randomly selected $1/4$ of each data set and query set to tune the parameters of XRank. Then, we used these parameters for the real data set and query set.

XSearch uses two tunable parameters, α and β, that control the effect of the IR-style ranking and the size of the candidate answer, respectively. There has not been any deep study on the effect of these parameters on the effectiveness of the XSearch method. We performed an effectiveness experiments using values for α and β between 1 and 10, and compared the results to CR. Our experiments showed that changing the values of these parameters has a very low impact on the effectiveness of XSearch i.e., less than 2%. The reason is that the size of most candidate answers were the same for both data sets. XSearch shows its best ranking quality for both data sets when the values of α and β are equal. We report only the best effectiveness results for XSearch in this section.

Similar to XRank, XReal uses a tunable parameter r that limits the depth of the LCAs of the candidate answers. The authors in [6] propose 0.8 for r and uses this value in their experiments. We have used different values in the range (0,1] for r for XReal and compared the results to CR. Generally, different values of r deliver the same results in our experiments. The only exception is $r = 1$, which reduces precision, recall, and ranking quality of XReal by about 2%. We report only the maximum values for precision, recall, and ranking quality of XReal in this section.

The effectiveness measurements used in the first task are recall, precision, and fall-out [96]. *Recall* gives the fraction of the relevant candidate answers that are included in the actual answer returned to the user. *Precision* gives the fraction of the returned answers that are relevant. We consider the precision of a method to be zero whenever its recall is zero. *Fall-out* shows the proportion of non-relevant answers in the returned answers. Fall-out is closely related to precision. When there is no relevant answer, a method that returns no answers has the same precision as one that returns many irrelevant answers – even though the first method is more useful than the second. Fall-out captures this difference.

We implemented the CR query processing system in Java 1.6. The query system uses the NTCs computed using $MAS = 5$, $EV = 3$, and minimum frequency 50. The use of larger values for EV did not affect the results, which is consistent with our error rate analysis for EV. None of the queries matched more than 4 leaf nodes, which shows that the choice of maximum answer size $MAS = 5$ was reasonable. We set $NTC_{min} = 0$ for the evaluations in this paper. Larger values for NTC_{min} increase the precision but decrease the recall. If desired, one can control the recall-precision tradeoff by tuning the value of NTC_{min} in a manner similar to that used to tune the parameters of IR retrieval methods [96]. Finding the best value for NTC_{min} is a subject for future study.

Table 3.5 and Table 3.6 show the average precision and average recall of all seven approaches on IMDB and DBLP for the simple query collection, respectively. The precision of CR (shown as "Coherency" in the tables) is as high or higher than every other method, for almost every query in both data sets. CVLCA, XSearch, and XRank show the lowest precision overall. They return many irrelevant subtrees whose LCA is the root of the DB. For instance, for *William Yurcik*, these methods return subtrees describing a paper written by an author whose name contained "William" and an article authored by a different person whose name contained "Yurcik". SLCA and MaxMatch show better precision than CVLCA, XSearch, and XRank. Our queries did not reveal any differences between SLCA and MaxMatch in terms of their precision and recall. As mentioned in Section 3.2.2, XReal prunes too many candidate answers. Therefore, it delivers lower precision than other methods. The root of the database has the highest sum of the query term frequencies for most queries having more than two terms, such as *Kate Winslet 1997*, *Peter Sellers Blake Edwards*, and *Comedy Woody Allen Scarlett Johansson* in IMDB and *Closed Frequent Pattern*, *Social Network*

Evaluation, and *Cis Regulatory Module 2008* in DBLP. This also happens for the relatively short queries whose terms are very frequent in the data set, such as *VLDB Korea* in DBLP. Thus, XReal returns the root of the database as the only answer for these queries, which is not the desired result.

CR shows perfect recall for all queries. For instance, in some movies Edward Norton acted and also appeared on the sound track. SLCA and MaxMatch return only the sound track nodes. For some queries, the users marked both large and small subtrees as relevant. For instance, all methods except for CR return a director subtree or composer subtree for *Satyajit Ray*. However, our users also marked as relevant the subtrees that contain *both* director and writer information, as they were interested in movies that Satyajit Ray directed and composed. Since DBLP is flatter than IMDB, the recall of SLCA and MaxMatch is better for DBLP than for IMDB. XReal filters out the entity types that have relatively low query term frequencies. For instance, it does not return the movies produced by *Edward Norton*, as *Edward Norton* appears more frequently in the *actor* element than the *producer* element. For the same reason, XReal returns only the *inproceedings* elements containing the keyword *Hoarding* and ignores the *article* and *incollections* elements about *Hoarding*. Thus, XReal shows the lowest recall.

Our users initially proposed many queries with no relevant answers. All methods but CR presented a long list of irrelevant answers. When the users protested at having to sift through so many unwanted answers, we removed all but one of those queries from the workload: *SASI Protocol Attack* for DBLP. While quite a few papers have been written about attacks on SASI protocols, this overspecified keyword query will not find any of them in DBLP. CR returned no answer for this query, and was similarly helpful for many other queries with no relevant answers. XSearch and CVLCA return 121 answers for this query, all irrelevant; SLCA, MaxMatch, and XRank return 460 answers; and XReal returns 82 answers. The DL filtering technique of XSearch and CVLCA and the filtering approach of XReal improve their performance, but they still give many unhelpful answers. Thus, CR provides better fall-out than the other methods.

Since most methods return many irrelevant answers that are rooted at the root of the DB, we reran our experiments after modifying all the algorithms to filter out answers rooted at the DB root. This modification improves the precision and ranking quality of these methods without changing their recall. We use "-M" to distinguish these methods from their original method. For instance, we refer the modified version of SLCA as SLCA-M.

Table 3.7 and Table 3.8 show the average precision and average recall of all modified approaches on IMDB and DBLP for the simple query collection, respectively. CR shows the same precision and recall as before, as it automatically filters out the unrelated answers rooted at the root of the DB. XRank-M has the same precision and recall as CR, as it did not get a chance to remove a relevant answer. XSearch-M and CVLCA-M have the same recall as CR for DBLP but slightly lower recall for IMDB. Since IMDB is deeper than DBLP, XSearch-M and CVLCA-M filtered out some relevant answers using DL heuristics. SLCA-M and MaxMatch-M still have lower recall than CR.

The XReal-M has the same precision and ranking quality as XReal.

XReal returns elements from the lower levels of the database for some queries. For example, XReal returns only the element *actor* for the query *Christian Bale* in IMDB. Our users deem this answer not to be informative enough and want to see its ancestor element *movie*. XReal provides better precision than CR for five IMDB queries and five DBLP queries. However, XReal omits many relevant answers for these ten queries, as explained below. For the query *Fight Club*, only movies having *Fight Club* in the title were marked relevant by the users, but there are many candidate answers having these words in their keywords or production company names. Subtrees containing a title, keyword, or production company all have the same size, as they all have one non-leaf node. Thus the approaches that rely on subtree size are unable to filter out the unwanted answers. The same thing happens for queries *Beautiful Mind, Jackie Chan 2007, Pearl Harbor, Crime the Godfather, Momento, The Watchmen, Thumbs Up, Return of the Mummy, High School Musical*, and *Basic Instinct*. Our users were looking for information about movies with these titles, but there are many movies that have these words in their keywords. None of the methods had good precision for these queries. In the second evaluation task, we will see how CR addresses this problem through returning the most relevant answer first.

For query *Edward Norton*, all methods return many movies that have one actor whose first or last name is *Edward* and another actor whose first or last name is *Norton*. The same happens for *Kate Winslet 1997*. Almost the same thing happens with queries like *Han Data Mining*, where the user wants information on the book written by *Han*, but gets papers about *Data Mining* written by *Han* as well. Later we will show how CR solves this problem through ranking. For very specific queries, like *Comedy Woody Allen Scarlett Johansson*, the candidate answers are the desired ones, and SLCA, MaxMatch, and CR are equally precise. However, as explained below, SLCA and MaxMatch omit many relevant answers. This lowers not only their recall but also their precision. XReal-M has the lowest recall among the modified methods.

Table 3.9 and Table 3.10 show the average precision and average recall on the complex queries, for all modified approaches on IMDB and DBLP, respectively. Overall, we found that asking users for more complex IMDB and DBLP queries simply led to more *constrained* queries, greatly reducing the number of candidate answers, lessening the potential margin for filtering and ranking errors, and diminishing the distinctions between methods. This can most clearly be seen in the results for DBLP, where users crafted complex DBLP queries by giving more and more details about authors, titles, and venues, shrinking the set of candidate answers with each additional term. The final set of candidate answers had cardinality 1 in some cases; the largest set, which was for *Xifeng Yan Jiawei Han*, had perhaps 20 papers authored by the gentlemen in question. The users judged over 90% of the candidate answers to the DBLP queries to be relevant. If almost all candidate answers are relevant, then almost all methods will have excellent precision. If all candidate answers are good, then ranking does not matter very much either. In such a situation, even the baseline method that returns *all* candidate answers has good precision (and by definition good recall). Finally, when there are very few candidate answers and almost all of them fit the same pattern (e.g., papers coauthored by Xifeng Yan and Jiawei Han), then there is very little room for error on the part of the heuristics that filter out patterns. In such situations, almost all the methods will have good recall. Table 3.9 shows that the modified baseline algorithm that returns all candidate answers except for the ones rooted at the root of the DB (shown in the table as *Baseline-M*) has almost the same precision and recall as other methods. Thus in our results, every method except XReal (discussed below) had almost identical precision and recall. When the baseline method works very well, then no other method's benefits are very visible; tightly constrained queries do not allow any method to show its power.

The differences between methods are slightly more visible in the IMDB data set, as it has a more nested structure with more patterns, and therefore a greater margin for error in filtering and ranking. With complex queries, XReal has the lowest recall on both data sets. XSearch-M and CVLCA-M do show slightly better precision than other methods, as they successfully filter some unrelated candidate answers using DL heuristics. Since the precision and recall of almost all methods are still almost equal, the methods differ mainly in their ranking quality, discussed below.

In the second task, we evaluated the ranking effectiveness of CR, XSearch, XRank, XRank-M and XReal using *mean average precision* (MAP). Intuitively, MAP measures how many of the relevant answers appear near the beginning of the returned list of answers [96]. To compute MAP, we first consider each query Q separately. We compute the precision of all returned answers for Q, up to and including the ith relevant answer, for each value of i. The average of these precisions is called the *average precision* for Q. The MAP is the mean of the average precisions for all queries in the workload. Since XSearch uses the size of the candidate answers to rank them, it always ranks the candidate answers rooted at the root of the DB last. Thus, the MAP values of the original and modified versions of XSearch are equal. Also, since XReal returns the instances of the same entity type for an input query, the modification does not have any impact in its ranking quality.

By combining CR with IR-style ranking methods, we can take advantage of both structural and content information in the database. We combined CR with the pivoted normalization method [96] to determine the rank $r(t)$ of answer t:

$$r(t) = \alpha\hat{I}(t) + (1-\alpha)ir(t), \tag{3.12}$$

where $ir(t)$ is:

$$ir(t) = \sum_{w \in Q, E_l} \frac{1 + \ln(1 + \ln(tf(w)))}{(1-s) + s(el_l/avel_l)} \times qtf(w) \times \ln(\frac{N_l + 1}{ef_l}). \tag{3.13}$$

Here, E_l is an element whose label is l and is the parent of the leaf node m that matches term w from the input query Q. $tf(w)$ and $qtf(w)$ are the number of occurrences of w in m and Q, respectively. el_l is the length of m, and $avel_l$ is the average length of the contents of all leaf nodes m whose parent has label l. N_l is the count of nodes whose parent has label l, and ef_l is the number of nodes whose parent has label l and contain w. s is a constant; the IR community has found that 0.2 is the best value for s [96]. α is a constant that controls the relative weight of structural and contextual information in ranking. If α is set to 1, the formula uses only structural information. We used the parameter settings given in the previous section to compute $\hat{I}$.

For our queries, CR with pivoted normalization has better MAP than plain CR. The MAP for both data sets changes only slightly (at most .01%) for α between 0.1 and 0.9. It drops by more than 25% if we set α to 0. The reason is that DBLP and IMDB are data-oriented; thus, structural information is more important than contextual information.

Table 3.11 illustrates the MAP values of modified methods for the simple query collection for DBLP and IMDB. The table shows that the MAP of CR is higher than that of other methods for simple queries, for the Wilcoxon signed rank test at a confidence level of at least 95%. This confirms that NTC is better than subtree size as a basis for ranking decisions. Also, XSearch's filtering strategy lowers its ranking quality. Pivoted normalization without CR ($\alpha = 0$), shown as PN, has lower MAP than all but one other method, because PN uses only contextual information. The modified PN method (shown in the tables as *PN-M*) has better ranking quality than PN, but it has lower MAP than CR. XRank has lower MAP than PN, as its filtering strategy lowers its precision. XRank-M shows better ranking than XRank. The ranking quality of XRank and XRank-M changes considerably (about 15%) for different values of d_1, d_2, and $decay$. In order to be fair to all methods, we randomly selected $1/10$ of DBLP and IMDB and $1/4$ of the queries of the simple query set to tune the parameters of XRank-M. Then, we used these parameters for the real data set and query set. If we use the same parameters for DBLP and IMDB, the modified version of XRank will have a lower MAP value than CR for DBLP. Nevertheless, CR returns almost the same MAP values for different values of its parameter. XReal shows the lowest MAP. For most queries, the elements with higher query term frequencies are not the best answers. For instance, for the query *Crime the Godfather*, XReal ranks the movies whose *keyword* element contains *Godfather* higher than the movies whose title is *Godfather*, but the latter are the desired answers. XReal also ranks the movies produced at *Pearl Harbor* higher than the movies whose titles or keywords contain *Pearl Harbor*.

Table 3.12 illustrates the MAP values of modified methods for the complex query collection for DBLP and IMDB. CR has the highest MAP value for this query collection for DBLP and IMDB. Similar to the ranking task for simple queries, we trained XRank and XRank-M parameters over a randomly selected $1/10$ of DBLP and IMDB and $1/4$ of the complex queries. Generally, the MAP values of different methods are closer to each other for complex queries than for simple queries. This is because the complex queries have few matching subtrees (candidate answers). For complex queries, CR does specially well compared to the other methods in picking the best entity types. For instance, for query *Elmasri Navathe Database*, the user is looking mainly for the book and articles written by *Elmasri* and *Navathe* about database systems. CR successfully ranks the instance of these entity types at the top of its returned list. XSearch returns the first related answer at the fourth position of its list. XRank returns some articles written by others that cite the publications of *Elmasri* and *Navathe* at the top of its ranked list. The reason is that those articles have lots of citations and have larger PageRank value. As mentioned in Section 3.1, CR is orthogonal to PageRank style algorithms such as XRank; combining the two ia an interesting topic for future work.

In the remainder of this section, we explain how CR ranks the query answers that had low precision in the first evaluation task. We also discuss every query where CR did not rank the desired answer(s) first.

Since a book typically has fewer authors than a paper in a conference or journal, the NTC of the author and title of a book is higher than the NTC of the author and title of a paper. Therefore, for the query *Han Data Mining*, CR returns the textbook at the first result, as the user desired. XReal filters the book entity and XSearch ranks the desired answer very low.

Subtrees containing two keyword nodes, and subtrees containing a keyword node and a literature reference, have lower NTCs than subtrees containing a title and a keyword node. So for query *Godfather Crime*, CR ranks crime movies whose title contains *Godfather* above movies with *Godfather* and *Crime* as keywords and/or literature references. XSearch ranks the subtree with two keywords first, as it is smaller. XReal ranks the subtree containing keyword and literature reference nodes higher, as the frequencies of the query terms in these nodes are higher than their frequencies in the title node. Queries *Beautiful Mind, Pearl Harbor, Momento, The Watchmen, Basic Instinct*, and *Return of the Mummy* have the same problems.

Since the title of a movie is a semi-key but there are many production company nodes with the same content, the NTC of *title* is higher than that of *production company*. Therefore CR returns the movie with the title *Fight Club* as the first result for the query *Fight Club*. The other methods do not find the proper ranking for this query.

The NTC of a subtree containing only one actor node is higher than the NTC of a subtree containing two actor nodes. Therefore, CR places the subtree with one actor node at the top of the result list for the query *Edward Norton*. Also, since the NTC of an actor node is higher than the NTC of a producer node, CR ranks the producer nodes containing *Edward Norton* after the actor nodes. This ranking was the desired ranking for our users, who were not interested in the producer nodes.

Since the *location* node has less diverse values than the *keyword* node in IMDB, the NTC of title and keyword is less than the NTC of title and location. Therefore CR ranked the subtree containing title and location above the subtree containing title and keyword in the result list, for the query *Lara Croft Cambodia*. The same happens for the subtree containing the year and producer nodes versus the subtree containing the year and actor nodes, when CR answers the query *Jackie Chan 2007*. The NTC of the former is higher than the NTC of the latter, so CR ranks the movies that Jackie Chan acted in in 2007 higher than those produced by him in 2007. This ranking was what the users wanted. Because the number of producers is lower than the number of actors in each movie subtree, heuristics such as join selectivity [151] give higher rank to the movies that Jackie Chan produced in 2007.

IR-style ranking helped with some queries, such as *Artificial Intelligence*. Multiple movies include those words in their titles. Pivoted normalization correctly penalized the titles that were longer than the desired answers, and returned the desired answer at the beginning of the list.

We consider an example of how CR determines users' intentions when the query terms are ambiguous. For query *CCS 2008* on DBLP, our users wanted information about the 2008 ACM Conference on Computer and Communications Security (CCS). However, many 2008 papers have *CCS* (change control system) in their titles. CR successfully ranked the title of the proceedings of CCS 2007 first, as both words occurred in the title. XSearch ranked the irrelevant papers higher and XReal filtered out the proceedings node, which had very low query term frequencies. CR, however, ranked the subtree containing the title and year of the proceedings fourth, after two articles about CCS, but above the conference papers about change control systems.

For the query *Maude* on DBLP, since the NTC of *title* is higher than that of *author*, CR correctly ranked papers whose titles mention Maude above papers written by authors named *Maude*. XSearch ranks the latter first. XReal provides almost the same ranking as CR for this query. Nevertheless, it misses many relevant answers due to its pruning strategy.

Although CR provided a ranking close to users' expectations, it could not deliver exactly the ranking that the user wanted for some queries. We discuss each of these queries here.

The user who issued the query *Dakota Fanning Animation* was looking for all the animated movies voiced by Dakota Fanning. The word *Animation* matched both the *special effects company* and *genre* nodes. The desired answer is the subtree containing the *actress* and *genre* nodes. However, *genre* has less diverse values in IMDB than *special effects company*. Therefore CR chooses the subtree including the actress and the special effects company as its top answer, and ranks the subtree containing the actress and genre nodes second. One potential solution to this problem would be to exploit the NTC of individual words in the final ranking formula. The next chapter investigates this approach.

As another problematic query, the user who submitted *Pearl Harbor* was looking for the most recent movie with this name. CR ranked the most recent movie fourth, as it could not distinguish which was the most recent. This problem also occurred for the queries *True Dreams*, *Social Visualization*, and *Stereo Vision* in DBLP and IMDB. Addressing this issue is beyond the scope of this thesis; either a domain-specific hard-coded heuristic or a heuristic learned from click-through behavior could be used to help CR break ranking ties between movies of different vintages. A similar problem arose with *Basic Instinct*, where the user was looking for the first and most famous movie of the series, and marked the others as irrelevant. CR returns this movie second in the result list, rather than first. The IMDB database does not have enough information to determine popularity measures for the movies, though such information could be learned by click-through behavior.

In the query *Authenticated Dictionary* over DBLP, the user was looking for the titles where these two words occur next to each other. CR does not consider word proximity in its ranking formula, but it can be addressed by incorporating more advanced IR-style rankings [89].

Chapter 4

Using Structural Information in Keyword Search over Semi-structured Information with Long Text Fields

4.1 Background

In Chapter 3, we empirically showed how to use ranking approaches based on statistical measures of the correlation between schema elements [76, 67] to improve the effectiveness of keyword query interfaces over databases. Intuitively, answers that involve closely correlated schema elements should be ranked higher than less correlated answers. Such schema-correlation-based measures have been shown to be effective for ranking query answers for data-oriented XML without long text fields, but they do not handle long text fields such as paper abstracts or movie plot summaries appropriately. Long text fields are proliferating in data sets due to the increasing popularity of constructing and querying annotated corpora [15], so we address their special needs in this chapter.

In a nutshell, the problem is that the measures used in Chapter 3 consider two field values to be completely different, even if they differ in only one word. For example, consider a movie database that includes information about writers, movie plot lines, and movie tag lines (lines from trailers). Each movie has its own unique tag lines and plot lines, so existing correlation-based measures find that writers' names are as highly correlated with tag lines as with plots. In practice, however, a writer's plots tend to involve similar situations or characters, while the tag lines of his or her shows have little similarity. Thus intuitively, writers should be more correlated with plots than with tag lines. Overgeneralizing slightly, any correlation measure that cannot capture this intuition will tend to rank a query answer that includes a writer and a phrase from a tag line higher than an answer including a writer and a phrase from a plot line, which is undesirable in practice.

To address this problem, in this chapter, we introduce *term based correlation* measures as an effective way to capture our intuition about correlations between similar pieces of text, while at the same time exploiting all other available structural information in an XML database. Our contributions:

- We introduce a particular form of term based correlation called **normalized term presence correlation** (NTPC) and show how to incorporate NTPC into an XML query system. We show how to perform a one-time computation of NTPCs for schema elements, using a populated database instance. For all subsequent queries, the

Portions of this work has appeared in [127] and [128].

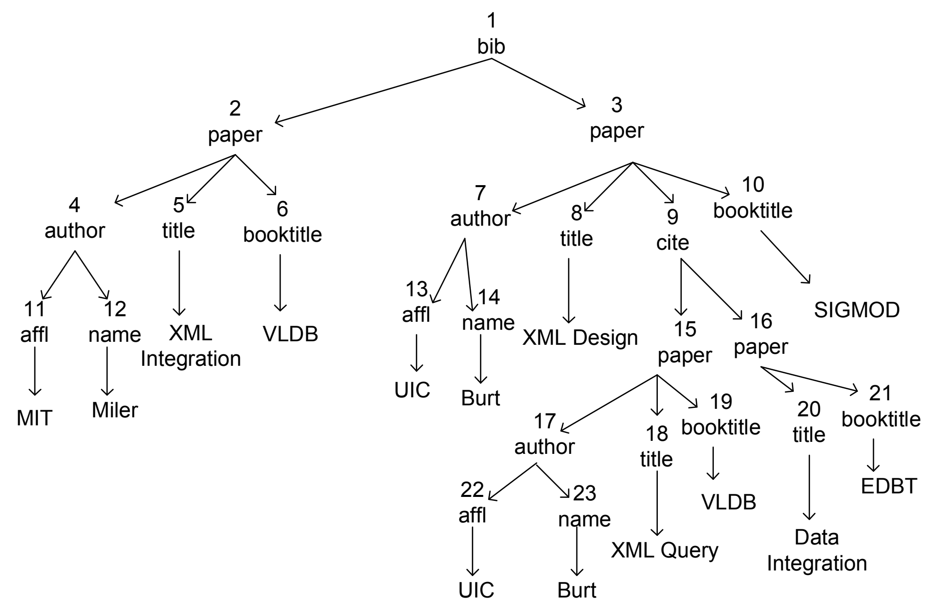


Figure 4.1: DBLP database fragment

precomputed NTPCs are used to rank candidate answers based on their structure. NTPCs only need to be recomputed if structural changes introduce a new type of schema element. We also show how to combine NTPCs with traditional IR ranking methods, so that query answer rankings consider both content and structure.

- Through an extensive user study with two real-world data sets, we show that the precision, recall, and ranking quality of a NTPC-based approach to query answering is always at least as high that of as seven other previously proposed approaches to XML keyword search. We also show that NTPC provides better ranking than previous approaches when the database includes long fields.

- The straightforward approach to computing NTPCs is prohibitively slow for large data sets. We present novel optimizations that allow NTPCs to be computed for .5-1.1 GB data sets in .5-2.5 days, which is reasonable for a one-time setup cost.

In the reminder of the chapter, Section 4.2 elaborates more on the motivation. Section 4.3 defines NTPC. Section 4.4 presents optimization techniques and algorithms for computing NTPC. Section 4.5 describes our query system, Section 4.6 presents empirical results.

4.2 Motivation

Consider the Citeseer fragment from *citeseer.ist.psu.edu/* in Figure 4.3. The best answer to query *Burt Mining* is rooted at node 3. Since *desc* is a long text field, two different papers will almost never have the same description. However, a

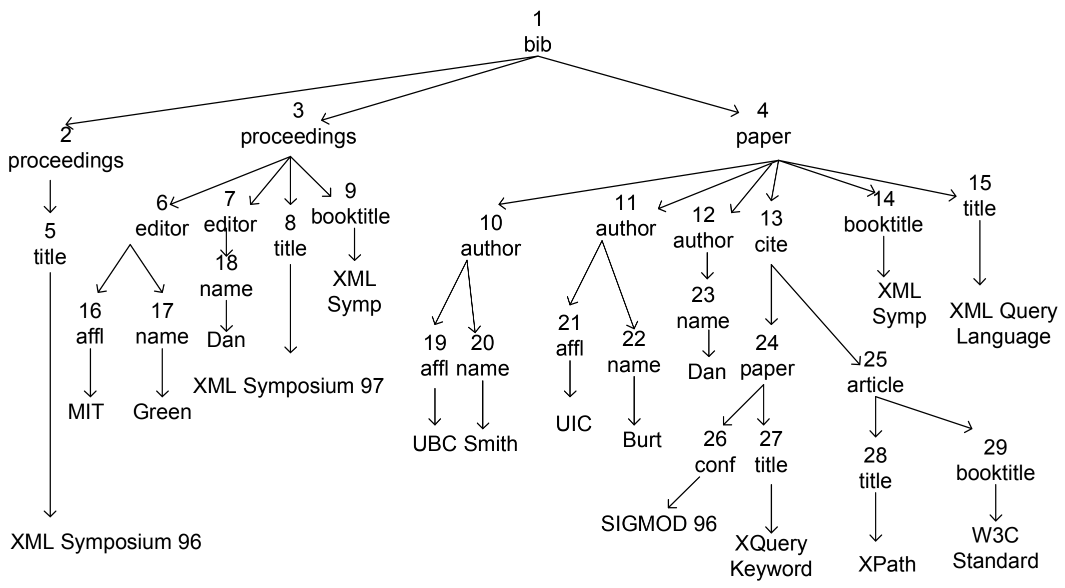


Figure 4.2: DBLP database fragment

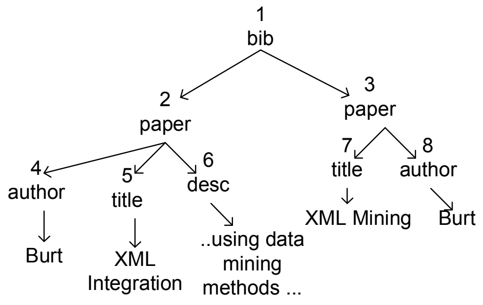


Figure 4.3: Citeseer database fragment

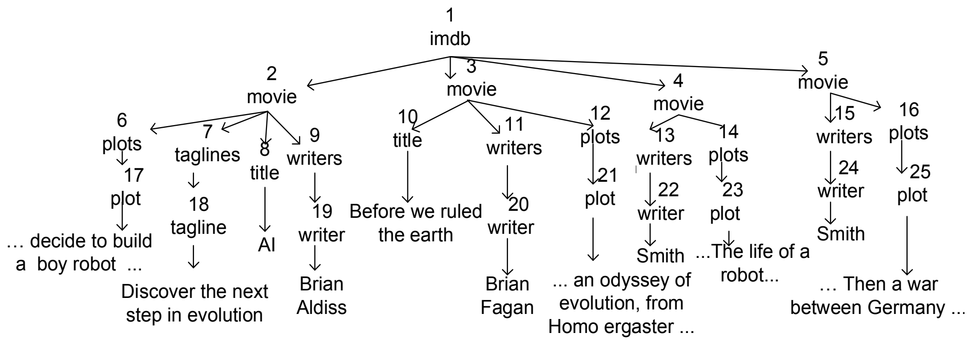


Figure 4.4: IMDB database fragment

few papers have similar titles but different authors. Thus, the NTC of the values of *paper author -1 title -1* is less than that of *paper author -1 desc -1*, and CR inappropriately returns node 2 as the best answer.

IR-style heuristics and penalties for long fields will not solve this problem. For instance, consider the IMDB fragment from *www.imdb.com* in Figure 4.4. Because tag lines are less indicative of a movie's content than plot lines, the best answer to query *Evolution Brian* is the subtree rooted at node 3. Different movies have different plot lines and tag lines. In the original IMDB, on average each movie has more *plot* nodes than *tagline* nodes (taglines are the famous sentences in the movies' trailers), so the NTC of *movie taglines tagline -1 -1 writers write -1 -1* is 1.48 and the NTC of *movie plots plot -1 -1 writers write -1 -1* is only 1.37. Thus, CR ranks node 2 above node 3. As the *plot* field is longer than the *tagline* field, penalizing long fields will not solve the problem.

The problem can occur for short text fields as well. Consider query *XML Dan* for Figure 4.2. The best answer is at node 4, which gives papers about XML written by Dan. The node 3 answer should be ranked second, as it is less interesting for most users. But since each proceedings has a different title, the NTC of *proceedings title -1 editor name -1 -1* is the same as the NTC of *paper title -1 author name -1 -1* in this fragment. NTC does not consider the fact that the proceedings titles for nodes 2 and 3 are almost identical; if it did, the correlation of the first pattern would drop and we would get the desired ranking.

Thus, intuitively, we should consider the individual components (words) of each value when computing correlations, both for long and short fields. For instance, the words in movie tag lines are not as representative of the movie's subject as the words in its plot lines. Thus in Figure 4.4, the terms in the values of the field *writer* are more correlated with those of *plot* than *tagline*.

Correlation mining is an active research area in data mining and databases [10, 76, 94]. But most research has focused on finding the correlations between data items themselves, rather than computing the collective correlation between their columns or nodes. Even the systems in the latter category [67, 76, 80] consider the values of each column as a whole, so we cannot apply them to our ranking problem.

4.3 Term Based Correlation

4.3.1 Preliminaries

We adopt the terminology of the previous chapter, with a few extensions. A **root-subtree** is a subtree whose root is the root of the XML DB and whose leaves are the parents of content nodes. For instance, *1 2 5 -1 6 -1 -1* is a root-subtree in Figure 4.1. If the root-subtree is a path, we call it a **root-path**. Each root-subtree contains at least one root-path. Every maximal set of isomorphic root-subtrees in a tree T corresponds to a pattern. The **size of a root-subtree pattern** is the number of root-path patterns it contains. From now on, the only patterns we consider are those of root-subtrees.

Definition 4.3.1. $W(r_1 : w_1, \ldots, r_n : w_n)$ *is a* ***term*** *of root-subtree* T *containing root-paths* $r_1, \ldots, r_n$*, with* ***value*** $(r_1 : v_1, \ldots, r_n : v_n)$*, if* $w_1, \ldots, w_n$ *are non-stop words that occur in values* $v_1, \ldots, v_n$*, respectively.*

The **terms** of pattern p containing root-path patterns $(p_1, \ldots, p_n)$ are the union of the sets of terms of its instances.

We write p's terms as $W(p_1 : w_1, \ldots, p_n : w_n)$, reflecting which root-path pattern contains which term. For instance, (p_1 : *Design*) is a term of the root-path pattern *bib paper title -1 -1* and p_1 : *Design,* p_2 : *SIGMOD* is a term of the root-subtree pattern t_1 = *bib paper title -1 booktitle -1 -1* in Figure 4.1.

Each term $W(p_1 : w_1, \ldots, p_n : w_n)$ is associated with 2^n possible **events**. Each event takes the form $E(p_1 : f(w_1), \ldots, p_n : f(w_n))$, where each $f(w_i)$ is either w_i or $\bar{w_i}$, depending on whether w_i does or does not occur in p_i.

Definition 4.3.2. *The* ***occurrence probability*** *of an event $E(p_1 : f(w_1), \ldots, p_n : f(w_n))$ for term W in a root-subtree pattern q is $O(E) = \frac{|t(E)|}{|t(q)|}$, where $|t(E)|$ is the number of terms where p_i contains w_i, if $f(w_i) = w_i$; or where p_i does not contain w_i, if $f(w_i) = \bar{w_i}$ $(1 \leq i \leq n)$. $|t(q)|$ is the total number of terms of q in the DB.*

In Figure 4.1, if q = *bib paper title -1 -1*, $O(p_1 : \text{“Design”}) = \frac{1}{4}$, as this pattern has 4 terms.

Definition 4.3.3. *The* ***presence probability*** *of an event $E(p_1 : f(w_1), \ldots, p_n : f(w_n))$ for term W in a root-subtree pattern q is $P(W) = \frac{|E|}{|q|}$, where $|E|$ is the number of instances of q where p_i contains w_i, if $f(w_i) = w_i$; or p_i does not contain w_i, if $f(w_i) = \bar{w_i}$ $(1 \leq i \leq n)$. $|q|$ is the number of instances of q in the DB.*

In Figure 4.1, if q = *bib paper title -1 -1*, $P(p_1 : \text{Design}) = \frac{1}{2}$, as the pattern has two instances and “Design” occurs in one.

A root-subtree pattern represents the joint distribution of the root-paths it contains. For instance, the root-subtree pattern t_1 = *bib paper title -1 booktitle -1 -1* represents an association between root-path patterns p_1 = *bib paper title -1 -1* and p_2 = *bib paper booktitle -1 -1.*

Intuitively, the entropy of a random variable indicates how predictable it is [28]. We define the entropy of term W:

Definition 4.3.4. *Given term W of pattern p of size n whose events $E_1, \ldots E_{2^n}$ have presence probability $P(E_1)$, $\ldots, P(E_{2^n})$ respectively, the* ***presence entropy*** *of p is*

$$H_p(W) = \sum_{1 \leq i \leq 2^n} P(E_i) \lg(1/P(E_i)).$$

For instance, consider the term (p_1 : *robot,* p_2 : *Aldis)* in the pattern *imdb movie plots plot -1 -1 writers writer -1 -1 -1*, where p_1 is *imdb movie plots plot -1 -1 -1* and p_2 is *imdb movie writers writer -1 -1 -1* in Fig 4.4. The term has four events. Considering only DB fragment in the figure, we can compute their presence probabilities to find the the presence entropy for the term: $2 * 1/4 \lg(4) + 2/4 \lg(2) = 1.5$.

The **occurrence entropy** $H_o(W)$ of term W can be defined similarly.

Definition 4.3.5. *Given the pattern p containing terms $W_1, \ldots, W_N$ with occurrence probabilities $O(W_1), \ldots, O(W_N)$*

respectively, the ***collective entropy*** *of p is*

$$H(p) = \sum_{1 \le i \le N} O(W_i) \lg (1/O(W_i)).$$

As opposed to the presence and occurrence entropies, collective entropy is defined over all terms of a pattern. Notice that we cannot define the collective entropy based on presence probability, as the sum of the presence probabilities of the terms of a pattern could exceed one. If the pattern has more than one path, we also call its entropy *joint entropy*, as it is defined over a joint distribution.

4.3.2 Correlation Measures

Total correlation [143] is closely related to mutual information [28]; it measures the correlation between random variables. It is defined over random variables $A_1, \ldots, A_n$ as

$$I = \sum_{1 \le i \le n} H(A_i) - H(A_1, \ldots, A_n). \tag{4.1}$$

The greater the value of I, the more correlated the variables are. If the variables are independent, the value of I will be zero. Since each w_i of term $W(p_1 : w_1, \ldots, p_n : w_n)$ is a term itself, we can extend the definition of total correlation as follows:

Definition 4.3.6. *The* ***total presence correlation*** *(TPC) of term $W(p_1 : w_1, \ldots, p_n : w_n)$ of pattern q is:*

$$I_p(W) = \sum_{1 \le i \le n} H_p(w_i) - H_p(W). \tag{4.2}$$

In the same setting as the last example, the TPC of term *(p_1 : evolution, p_2 : Fagan)* is 0.61 and the TPC of term *(p_1 : robot, p_2 : Aldiss)* is 0.31. In this fragment, from the writer's name *Fagan*, we can predict that the movie is about *evolution* and vice versa. However, knowing the movie is about *robot* does not necessarily mean that its writer is *Aldiss*, so they are not as correlated as the previous term. Thus, the TPC of a term reflects the correlation between its components. In the above definition, the instances of path p_i in the DB are a superset of its instances that are subtrees of the instances of q. Thus, we compute $H_p(w_i)$ considering only the instances of p_i that are subtrees of the instances of q. The **total occurrence correlation** (TOC) of a term can be defined similarly.

We can measure the correlation of a pattern by averaging the sum of TPCs for all terms in the pattern. However, there are many weakly correlated terms in patterns. For instance, the original IMDB has many terms in the title, tag line, or plot that are not correlated or are weakly correlated with the terms in the name of the movie writers or the terms in other fields. Such words usually have high frequency in the DB. Hence, patterns that intuitively should have different correlations may look very similar if we average over all their terms. To prevent this problem, we average over the top-k correlated terms, where k is reasonably large. If k is too small, we face the same problem, as there are always some terms that are highly correlated in most patterns. Through empirical evaluation, we found that a value of 50-100 is appropriate for a large database such as DBLP or IMDB.

TOC and TPC both measure the collective correlation of a pattern. However, their different properties make them appropriate for different applications. Consider the term $W(p_1 : w_1, p_2 : w_2)$ in pattern q, where the values of p_2 are relatively long. With TOC, w_1 is associated with many terms in p_2 where one of them is w_2. Thus, TOC finds the term W to be relatively weakly correlated. Since TPC views the presence of the components of the term W in an instance of p as an association between them, TPC gives W a higher correlation. As mentioned before, a good correlation measure must work equally well for long and short fields. Hence, we choose TPC for our application, keyword search.

Among patterns of size 1, those with more variety are more helpful for users. For instance, in Figure 4.1, *title* is more helpful for most users and queries than *booktitle*. Thus if both match the input query, the system should rank *title* first. Since the terms of highly repetitive fields such as *booktitle* are quite evenly distributed in the database and there are so few of them, averaging over their top-k entropies returns a relatively high value that does not reflect the true variety of their information. Therefore, we use collective entropy from Definition 4.3.5 to rank patterns of size one.

Since users prefer smaller patterns, we penalize larger patterns in the final formula for TPC:

$$NTPC(W) = g(n) \times \frac{TPC(W)}{H(W)}, n > 1. \tag{4.3}$$

Here, $g(n)$ is a function that penalizes larger patterns; $g(n) = n^2/(n-1)^2$ performs well in practice. Exploring options for $g(n)$ is an interesting area for future work.

The values of NTPC and collective entropy for all patterns in the DB should be computed in a separate phase before the first queries are submitted to the system. If the DB does not undergo drastic structural changes that introduce new node types and patterns, this computation need never be repeated. For example, the tightness of the relationship between paper titles and their authors will not change, no matter how many years go by or new conferences are introduced. However, if a new field *paper-body* is introduced, we must compute the NTPCs for candidate answers that include that new field.

To answer a query, we find all candidate answers and their associated patterns, and look up those patterns in a table of precomputed NTPCs. We rank answers according to their NTPC values. We choose to omit answers with zero NTPC, as they are especially irrelevant.

NTPC-based ranking successfully handles all the examples described earlier. For instance, the NTPC of *proceedings title -1 editor name -1 -1* in the full DBLP is 1.41, and the NTPC of *paper title -1 author name -1 -1* is 1.73. In the full IMDB, the NTPC of *movie taglines tagline -1 -1 writers write -1 -1* is 1.25, while the NTPC of *movie plots plot -1 -1 writers write -1 -1* is 1.49. Section 4.6 contains a detailed evaluation of the effectiveness of NTPC-based ranking.

4.4 Computing NTPC

4.4.1 Optimization Techniques

Researchers have experimented with techniques for efficiently computing correlations. Unfortunately, these techniques rely on domain characteristics that are not applicable for keyword search over XML with long fields. For example, in keyword search, many strongly correlated terms are not frequent, so we cannot consider only frequent terms [10]. For keyword search, the minimum interesting value of NTPC is too small for it to be helpful as a cutoff during correlation computation [100]. Nor is the number of distinct values for a field necessarily much less than its total number of values, which would allow us to use sampling [67]. Nor can we adopt the techniques for precomputing NTC quickly, as NTC computes the correlation for entire values, not for the terms of a pattern. However, as in *CR*, we can assume that the maximum number of keywords in a query is relatively low (2.5 on average according to IR studies [145]). Thus, the size of the patterns we seek does have a domain-dependent upper bound $MCAS$ (maximum candidate answer size). For instance, empirical studies suggest that 4 is a reasonable $MCAS$ value for bibliographic DBs.

The following lemma reduces the number of patterns for which we must compute NTPC. We call two events **presence independent** if their presence probabilities are independent of one another.

Lemma 4.4.1. *Consider the term $W(p_1 : w_1, \ldots, p_n : w_n)$, $n > 1$, of the pattern q, with events $E(p_1 : f(w_1), \ldots, p_n : f(w_n))$. If the root of q is the DB root, then all components $p_1 : f(w_1), \ldots, p_n : f(w_n)$ are presence independent.*

Proof. We show the property for two arbitrary components of the term itself. The proof is similar for other numbers of components and other events. For every $1 \leq i, j \leq n$ we have:

$$\begin{aligned} P(p_i : w_i | p_j : w_j) &= \frac{P(p_i : w_i \cup p_j : w_j)}{P(p_j : w_j)} \\ &= \frac{|w_i||w_j|/|p_i||p_j|}{|w_j|/|p_j|} \\ &= \frac{|w_i|}{|p_i|} \\ &= P(p_i : w_i). \end{aligned} \tag{4.4}$$

□

We can view the components of a term's events as a random variable that assumes value 0 when $f(w_i) = w_i$ and 1 when $f(w_i) = \bar{w_i}$. Since all the events of these random variables are independent, they are independent, too. Therefore, we have:

Corollary 4.4.2. *The NTPC of patterns rooted at the root of the database is zero.*

Hence, when computing NTPCs we ignore all patterns rooted at the root of the database tree.

The number of pattern instances containing a term is the **frequency** of the term. From the properties of total correlation, it follows that infrequent terms have relatively low NTPC. The frequency of term $W(p_1 : w_1, \ldots, p_n : w_n)$ is less than the frequencies of its components $w_i, \ldots, w_n$. As explained later in this section, we compute the NTPC of a pattern using the information from its root-path patterns. Before computing the NTPC of patterns of size $n > 1$, we remove all terms in root-path patterns p whose frequencies are less than $\epsilon|p|$, where $0 < \epsilon < 1$. Similarly, for terms of very high frequency, we use an upper frequency cutoff threshold $1 - \epsilon$, and remove components with relative frequency above that limit. The appropriate value of ϵ depends on the number of root-path instances in the DB. In our experiments we used ϵ = 0.01, with two exceptions. First, a root-path pattern will have members with very low frequency if the entity represented by the root-path pattern is a key or semi-key and each value of the pattern has only one or two terms, such as for ISBN numbers in a bibliographical database. We do not remove any term from such patterns, as that would not leave any term in the pattern. Some entities have terms that are very frequent, such as *payment-methods* with values *cash, check, credit card.* We do not remove any terms from such patterns, either.

4.4.2 NTPC Computation Algorithms

The ComputeNTPC algorithm in Figure 4.5 gives an overview of how to compute NTPC and collective entropy. First, ComputeNTPC reads the XML data set in a depth first manner, finds the root-path patterns, and creates a **compressed index** (CI) for each root-path pattern. Each entry in a CI contains a root-path instance and the terms in its value. We define a 32 bit key for each term, and store the key instead of the term in the index. We do not need the terms to compute the collective entropies and NTPCs. Each root-path instance is represented in the CIs by the Dewey code [125] of its leaf node. Every Dewey code is stored in a bitmap to save space. These optimizations reduce the space requirements and enable ComputeNTPC to keep the CIs in main memory. For instance, for the roughly 1 GB IMDB data set, the average CI size was 4MB. As we use only one instance of CI for each root-path pattern throughout ComputeNTPC, this ensures modest space overhead and drastically reduces run time. The root-path instances in each CI entry are sorted according to their depth-first traversal order in the DB.

```
Input: XML data file $data$
Input: Maximum size $MCAS$ of patterns to compute NTPC for
Input: Minimum term frequency $\epsilon$
Output: Table $CT$ of NTPC for $data$
/* Find the root-paths and build their CIs */
1  $invdx$ = Create_CI($data$);
/* Compute the collective entropies */
2  forall $p \in invdx$ do
3      clt_Entropy($p$);
       /* Prune frequent and infrequent terms */
4      prune($p$,$\epsilon$);
/* Initialize the set of prefix classes */
5  $pfxSet \leftarrow \emptyset$;
/* Add all root-path patterns as one prefix class */
6  $pfxSet$.add($invdx$);
7  for $k = 2$ to $MCAS$ do
8      $nextPfxSet \leftarrow \emptyset$;
9      last = ();
10     forall $pfx \in pfxSet$ do
11         forall $p \in pfx$ do
               /* Compute all prefix classes with prefix $p$ */
12             $nextPfx \leftarrow \emptyset$;
13             forall $q \in pfx$ do
14                 $Jnt \leftarrow$ join_Pattern($p, q$);
15                 forall $r \in Jnt$ do
16                     if subTrees($r$) $\not\subset pfxSet$ then
17                         continue;
                       /* Find the root-paths and join levels for the new pattern */
18                     $rp \leftarrow$ rootPaths($r$);
19                     $jl \leftarrow$ joinLevels($r$);
                       /* Join the CIs */
20                     $jTable \leftarrow$ join_CIs($rp$,$jl$);
21                     if $jTable.freq = 0$ then
22                         continue;
                       /* Compute the NTPC */
23                     CT[$r$] $\leftarrow$ NTPC($jTable$);
24                     if $k \neq MCAS$ then
25                         $nextPfx$.add($r$);
26             if $k \neq MCAS$ then
27                 $nextPfxSet$.add($nextPfx$);
28     $pfxSet \leftarrow nextPfxSet$
29 return $CT$;
```

Figure 4.5: ComputeNTPC: algorithm to compute NTPCs

```
Input: List $rp$ of root-path patterns to join
Input: List $jl$ of join levels
Output: Join table $jTable$
/* root-paths to CI mapping                                        */
$path2Ind$ = pathIndexMap($rp$);
/* Distinct CIs                                                    */
$indcs$ = indexes($path2Ind$);
/* Group levels for each root-path                                 */
$gLvs$ = group_Levels($rp$,$jl$);
for $i = 0$ to $indcs.size - 1$ do
    $nextInd$.add($i$);
repeat
    /* Read the terms from the CI entries                          */
    for $i = 0$ to $indcs.size - 1$ do
        if $i \notin nextInd$ then
            continue;
        $buf[i]$.clear();
        $buf[i]$.add($nextBuf[i]$);
        $nextBuf[i]$ = NULL;
        repeat
            $ent$ = $indcs$[i].next();
            /* No more entries                                     */
            if $ent$ = NULL then
                break;
            /* If entries of the same index can be grouped         */
            if grp(buf[i][0],ent,gLvs[i]) then
                $buf$[i].add($ent$);
            else
                $nextBuf[i]$ = $ent$;
                break;
        until false ;
    /* Join groups of nodes                                        */
    for $i = 0$ to $buf[0].size - 1$ do
        $resBuf$[i][0] $\leftarrow$ $buf$[0][i];
    for $i = 1$ to $rp.size - 1$ do
        forall $rent \in resBuf$ do
            forall $ent \in buf[path2Ind[i]]$ do
                if !eql(ent,rent) and jin(ent,rent.last,jl[i-1]) then
                    $tmp$.add($rent$,$ent$);
        $resBuf \leftarrow tmp$;
        $tmp$.clear();
    /* Update the join table                                       */
    $jTable$.add($resBuf$);
    /* Find the next indexes to read                               */
    $nextInd \leftarrow$ nextGrp();
until !nextInd.empty() ;
return $jTable$;
```

Figure 4.6: JoinCIs: Algorithm to join compressed inverted indexes

```
Output: The entries of the next groups
1 for $i = 0$ to $nextInd$.size-1 do
2     if $nextBuf$ = NULL then
3         $nextInd$.clear();
4         return;
5 $nextInd$.clear();
6 if $resBuf$.size ¿ 0 then
      /* move over all CIs */
7     for $i = 0$ to $indcs$.size-1 do
8         $nextInd$[i] = i;
9 else
      /* Find the groups with the smallest LCAs */
10    $node$ ← MAX_CODE;
11    for $i = 0$ to $indcs$.size-1 do
12        if grpLCA($buf$[i][0],$gLvs$[i]) ¡ $node$ then
13            $nextInd$.clear();
14            $nextInd$.add(i);
15            $node$ = $buf$[i][0];
16        else if grpLCA($buf$[i][0]) = $node$ then
17            $nextInd$.add(i);
```

Figure 4.7: NextGrp: algorithm to find the next group to join

```
Input: Nodes $n$ and $m$ to group
Input: Join level $l$
Output: true if $n$ and $m$ can be grouped
1 if $n$ = NULL or $n$.ancestor($l$) = $m$.ancestor($l$) then
2     return true;
3 return false;
```

Figure 4.8: Grp: algorithm for the grouping test

The next step in computing NTPCs is to find all patterns in the DB. We primarily use techniques from Chapter 3 to generate these patterns efficiently, and discuss only the differences here. After creating CIs, ComputeNTPC computes the collective entropy for each root-path pattern in line 3. In line 4, ComputeNTPC prunes terms whose relative frequencies are less than ϵ or more than $1-\epsilon$. In addition to lines 1-4, ComputeNTPC is different from the algorithms in Chapter 3 in its join operation at lines 18-20 and computing the correlation at line 23. After generating a pattern, in line 18 ComputeNTPC finds its root-paths, and then finds the levels of the LCAs of the root-paths. We call these levels **join levels**. Each entry in a CI represents an instance of a root-path p_i containing a root-path term w_i. Thus, joining the entries of root-paths $p_1, \ldots, p_n$ of pattern q produces terms such as $W(p_1 : w_1, \ldots, p_n : w_n)$ of the pattern q.

Figure 4.6 shows the join algorithm, JoinCIs. Many patterns have duplicates of the same root-path. For instance,

```
Input: Nodes $n$ and $m$ to join
Input: Join level $l$
Output: true if $n$ and $m$ can be joined
1 if $n$.ancestor($l$) = $m$.ancestor($l$) and $n$.ancestor($l+1$) != $m$.ancestor($l+1$) then
2     return true;
3 return false;
```

Figure 4.9: Jin: test whether nodes can be joined at a particular level

the pattern *bib paper author name -1 -1 author name -1 -1 -1* has the root-path *bib paper author name -1 -1 -1* two times. These root-paths share the same CI and our goal is to use only one CI for each root-path. Thus, the join algorithm finds the mapping from root-paths to unique CIs in line 1. The loop from line 6-34 moves down the entries of the CIs and joins them if possible. For example, assume JoinCIs wants to join the instances of root-path p_i with root-path p_j at level l. An instance of p_i can join with an instance of p_j if they have a common ancestor at level l or above. For instance, we can join the root-path *bib paper author name -1 -1 -1* with the root-path *bib proceedings title -1 -1* at level 0 to create instances of the pattern *bib paper author name -1 -1 proceedings title -1 -1* in Figure 4.2. The path instances *1 4 10 20 -1 -1 -1* and *1 4 11 22 -1 -1 -1*, whose LCA (node 4) is at level 1, can both join with the instance *1 3 8 -1 -1*. We call all instances of a root-path that have an LCA at level l or lower a **group**.

The Grp function shown in Figure 4.8 checks whether two instances are in the same group. Since each root-path in a pattern is adjacent to two other root-paths (except the first and the last root-paths in the traversal of the pattern, which are only adjacent to one), the root-path has two candidate group levels. As the groups of this root-path must join to the groups of all its adjacent root-paths, the pseudocode chooses the lowest level of the two as the grouping level. Consider a pattern q that consists of root-paths p_i, p_j, and p_i, with join levels l_1 and l_2, where $l_1 < l_2$. To perform the join, we must group all instances of p_i at l_1 to form group g_1, and group its instances at level l_2 to form group g_2. However, according to the definition of a group, g_2 is a subset of g_1. Thus, we need only the group at level g_1. We use this technique to make the join operation faster. If a root-path occurs more than once in the same pattern, we create only one group for its instances. The group levels are computed at line 3 of JoinCIs. Every instance of a root-path belongs to only one group in each join operation. Thus, JoinCIs reads each member of a CI entry only once, and finds its group at lines 13-22.

JoinCIs then joins the groups in lines 25-31. The Jin method in Figure 4.9 checks if two root-path instances can join at a given level. When two root-paths are the same, JoinCIs performs a self-join between the members of a group. Line 28 checks whether a root-path instance is different from the other root-path instances in a joint pattern instance. After joining the groups of the root-paths, JoinCIs inserts the keys of the joint terms in the **join table** at line 32. The join table is a hash table that maps the terms W of the produced pattern to the number of times $|W|$ they appear in the pattern. Line 33 finds the CIs whose groups are the next ones to read. The NextGrp algorithm in Figure 4.7 shows how to find these CIs. Every root-path instance belongs to only one group. Also, the entries in each CI are sorted. Thus, if the join operation was successful (lines 6-8), NextGrp must advance the cursor over all CIs. Otherwise, it must advance the cursor over the groups whose LCAs have the smallest Dewey codes among the current groups (lines 11-17). For the reasons mentioned earlier, these groups cannot join with any other group. Lines 1-4 of NextGrp check whether we have finished with any CIs. The time complexity of joining CIs is linear in the number of groups in each CI. If the join operation produces at least one pattern instance, line 22 of ComputeNTPCs computes the NTPCs using the join table.

4.5 Using NTPC at Query Time

We keep the collective entropy and NTPC for each pattern in a hash table in main memory during query processing. Our query processing system, *SA3*, finds each candidate answer. It then extends the candidate answer to be a root-subtree, by adding the path from the root of that answer to the root of the DB. SA3 looks up the NTPC of the root-subtree pattern of the candidate answer, and ranks the answer based on its NTPC. We use a modified version of the query processing algorithm from *CR* (call it SA2), which is in turn an extension of the SA algorithm from [61]; we refer the reader to those papers for details and a performance analysis. In the remainder of this section, we focus on SA3's unique features.

SA and SA2 use an inverted index to find nodes that contain the query terms. SA does not present answer values to the user; SA2 finds them by a sequential scan of the original XML files, which is slow. SA3 incorporates a new storage strategy and indexes to eliminate this problem.

At startup, SA3 reads the XML data set in a depth first fashion and stores it in a NODES table in Berkeley DB [8]. Each node is described in a separate tuple of NODES. The key of each tuple is the node's Dewey code and the remainder of the tuple contains its tag name, its parent's Dewey code, and its value if it is a content node.

SA3 builds an inverted index for the text information in the NODES table. Each entry in the inverted index for a term w contains two separate lists. The first list describes the leaf nodes whose values contain w. Each list entry contains the Dewey code of the node and the number of times w occurs in its value. The second list stores the number of times w occurs in the values of each node type (all the nodes with the same tag name constitute a node type). This information is required to compute the IR based ranks, as explained below. We store the inverted index as a table MATCHES in Berkeley DB.

SA2 returns only the answer subtrees. However, for many queries the user wants to also see the siblings of the leaf nodes, which can satisfy the user's underlying information need. For instance, for the query *XML Integration* in Figure 4.1, returning the subtree rooted at node 5 is not useful. The user already knows the title, and probably wants

	$\epsilon = 0$	$\epsilon = 0.001$	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$
DBLP	20.3	17.9	13.7	12.1	
IMDB	43.5	40.8	34.9	33.0	31.8
XMark	56.5	48.5	41.7	35.0	31.6

Table 4.1: NTPC computation time in hours, for different choices of ϵ

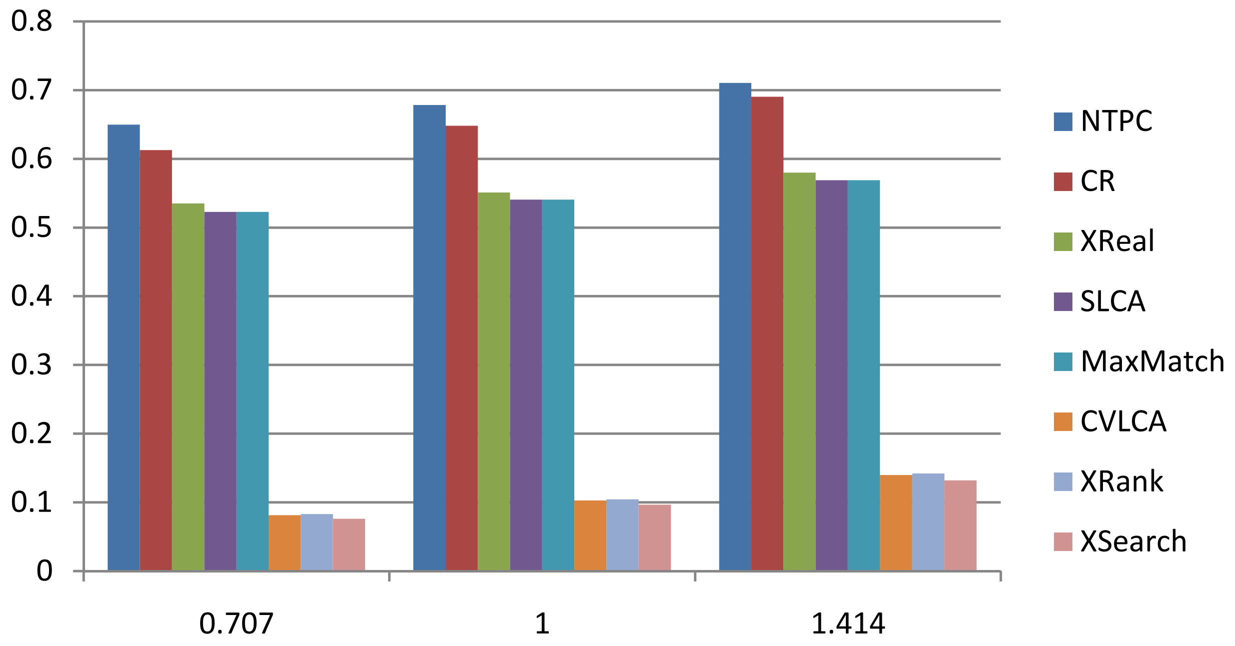


Figure 4.10: Average F-measure for IMDB queries

	NTPC	CR	XReal	SLCA	MaxMatch	CVLCA	XSearch	XRank
Precision	0.611	0.599	0.566	0.566	0.545	0.048	0.046	0.050
Recall	0.985	0.965	0.918	0.798	0.798	0.975	0.976	0.975

Table 4.2: Average precision and recall for IMDB queries

	NTPC	CR	XSearch	XReal	PN	XRank
IMDB	0.701	0.510	0.612	0.587	0.478	0.431
DBLP	0.834	0.834	0.794	0.790	0.621	0.591

Table 4.3: Mean average precision (MAP) for DBLP and IMDB queries

to learn the author and/or booktitle of the paper. To address this need, SA3 presents the full subtree to the user. To this end, SA3 builds an additional index CHILDREN on the parental information of the DB nodes, and stores it in Berkeley DB. Given the Dewey ID of a node, a lookup in CHILDREN finds its children. For each candidate LCA, SA3 queries CHILDREN to reconstruct the full subtree.

4.6 Evaluation

We evaluated ComputeNTPC and SA3 on a Linux machine with 2GB memory and a 2.13 GHz Intel Xeon CPU. We used two real-world and one synthetic data set: DBLP (529.9 MB, max depth 5), IMDB (994.8 MB, max depth 7), and XMark (1172.3 MB, max depth 11) [120]. We use XMark only to evaluate the scalability of ComputeNTPC, as there is no way to evaluate ranking quality for queries over a nonsense database. IMDB includes long text fields such as *plot*, *quote*, *crazy-credit* (interesting facts about movies' credits), *tagline*, *goofs* (mistakes in the movies), *trivia*, and others; these fields have not been included in any previous study of the effectiveness of XML keyword query processing strategies (including *CR*), and serve to illustrate the effectiveness of NTPC in the presence of long text fields. DBLP does not have long text fields; we use DBLP to show that NTPC also works well when the DB does not have very long fields, and to demonstrate ComputeNTPC's scalability.

We set $MCAS$ to 5 for DBLP and IMDB, which is a generous setting for both domains.

4.6.1 NTPC Computation Efficiency

Table 4.1 shows the time to load data, build supporting indexes, and compute NTPC, for different values of ϵ. IMDB is close to twice as large as DBLP and is more nested, so it has more patterns and more CI indexes than DBLP, and IMDB's CIs are larger on average than DBLP's. IMDB also has longer text fields than DBLP, so its join tables are larger. Thus, with $\epsilon = 0$, ComputeNTPC takes a bit more than twice as long for IMDB as for DBLP. The average length of XMark fields is less than for IMDB, but XMark is about 20% larger and is more nested, so ComputeNTPC takes about 25% longer for XMark than for IMDB, with $\epsilon = 0$.

The NTPC for a pattern is a characteristic of the underlying domain. Once NTPCs are computed for a representative instance, they do not need to be recomputed until a structural update introduces new schema element types. Thus NTPC calculations will be rare for a populated DB, and the .5-2.5 day running times shown in Table 4.1 are reasonable for a rare task.

Larger values for ϵ reduce the preprocessing time considerably. Run times drop off sharply at first as ϵ increases, then begin to level off, for a total drop of roughly 1/3 before ϵ grows too large. DBLP has more relatively infrequent terms than IMDB does. Thus, pruning infrequent terms reduces DBLP processing time the most. XMark content is randomly selected from a specific text collection, so XMark has relatively few infrequent terms and relatively many very frequent terms, compared to DBLP and IMDB. Thus, removing the terms with relative frequency greater than $1 - \epsilon$ helps to reduce the processing time for XMark. If ϵ grows too large, highly correlated terms can be pruned, which will change NTPCs enough to change the ranking of query answers. For DBLP, this happens for $\epsilon > .02$. Since IMDB has longer fields than DBLP, this happens at $\epsilon > .03$ for IMDB.

The final table of NTPCs was under 2MB for both DBs. Thus NTPCs can reside in main memory at query time.

4.6.2 Ranking Effectiveness

We used the IMDB and DBLP query workload from Chapter 3 to evaluate NTPC ranking quality. The queries came from 15 users who did not participate in the research. Each user submitted up to 5 queries to IMDB and DBLP, resulting in 40 queries for IMDB and 25 for DBLP. The exact workload can be found in Chapter 3. Users were given query results through a GUI interface where they can score each result as being relevant or irrelevant. We used the NTPCs computed for $MCAS = 5$ and $\epsilon = 0.02$ for DBLP, and $\epsilon = 0.03$ for IMDB. No candidate answer had more than 4 leaf nodes, which shows that $MCAS = 5$ was reasonable.

We first compared precision, recall, and F-measure [96] of NTPC against current methods: SLCA [149], MaxMatch [90], XRank [57], CVLCA [85], XSearch [26], XReal [6], and CR. *Recall* gives the fraction of the relevant candidate answers that are included in the actual answer returned to the user. *Precision* gives the fraction of the returned answers that are relevant. The *F-measure* shows the tradeoff between precision and recall; it is computed as:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}. \tag{4.5}$$

Setting $\beta = 1$ weights precision and recall equally. Values of $\beta < 1$ emphasize precision, while $\beta > 1$ emphasizes recall.

For DBLP, NTPC produced the exact same ranking for every query as did CR. Chapter 3 showed that CR performed as well as or better than each of the six other methods on the same workload and DB instance, in terms of precision, mean average precision (for approaches that rank their answers), recall, and F-measure.This shows that NTPC maintains CR's good performance when the data set contains short text fields. DBLP text fields are relatively short, and none of the 25 queries in the DBLP workload demonstrated NTPC's potential advantage over CR when fields have similar text, such as "EDBT 2009" and "EDBT 2010". Since the performance of CR and the other six approaches for the DBLP instance and workload was discussed in detail in Chapter 3, we focus on IMDB in the remainder of this section.

Table 4.2 summarizes the recall and precision of all 8 methods on all 40 IMDB queries by averaging over all queries in the workload. NTPC had higher recall on IMDB queries than other methods including CR. The recall of the other approaches is lower, due to imperfect pruning heuristics. For example, SLCA and MaxMatch showed even lower recall on IMDB than they did for DBLP and for the data-oriented version of IMDB used in Chapter 3, which had its long text fields removed. Many relatively unimportant short and long text fields such as *crazy-credit* and *quote* are in the lower levels of the XML tree. Some important fields like *title* appear high in the tree. SLCA and MaxMatch remove answers whose roots are ancestors of other candidate answers. Thus, they omit many relevant answers. For instance, they do not return the obvious answers to *Crime The Godfather* and *High School Musical*; instead they return the plots of some crime movies, and only the plot for the "High School Musical" movie.

XReal also delivers lower recall for IMDB, compared to DBLP and the data-oriented version of IMDB. Since many TV shows, stored in *show* elements, do not have long text fields, their values are shorter. Thus they have fewer occurrences of query keywords, compared to *movie* elements. Therefore XReal filters out TV shows. However, our users wanted to see TV shows as well as movies, for example in the queries *Pearl Harbor* and *Christian Bale*. Even with long fields not a factor, XReal filters out all *show* elements for *Christian Bale*, as he appears mostly in movies. The relative precision and recall of XRank, CVLCA, and XSearch were the same, as they do not filter based on the content or the level of the root of the candidate answers.

Keyword search approaches that do not rank their answers are frustrating to use; for example, *High School Musical* returns over 100 answers under the baseline approach. We used *mean average precision* (MAP) to compare the ranking quality of all current XML keyword search approaches that do rank their answers: XRank, XSearch, XReal, CR, and NTPC. MAP shows how many of the relevant answers appear near the beginning of the returned list [96]. To compute MAP, we first consider each query Q separately. We compute the precision of all returned answers for Q, up to and including the ith relevant answer, for each value of i. The average of these precisions is called the *average precision* for Q. The MAP is the mean of the average precisions for all queries in the workload.

To compare NTPC with the content-based ranking of XSearch, XReal, and CR, we combined NTPC with pivoted normalization (PN) [96], an IR-style content ranking formula that we customized for XML. We control the relative weight of NTPC and PN as follows:

$$r(t) = \alpha NTPC(t) + (1 - \alpha) ir(t), \tag{4.6}$$

where $ir(t)$ is the content score of the candidate answer, computed based on the classical PN formula. α is a constant that controls the relative weight of structural and contextual information in ranking. If α is set to 1, the formula uses only structural information. If $\alpha = 0$, we have pure PN. Based on our empirical evaluation, we set the value of α to 0.8 when combining it with NTPC.

Table 5.3 shows the MAPs of the methods. XRank shows a relatively low MAP, as the movie domain and IMDB are not appropriate for PageRank-like heuristics. PN delivers a low MAP as well. Generally, PN ranks smaller fields and fields with more query keyword occurrences higher. There are many fields of average length but different importance in IMDB, such as *title*, *crazy-credit*, and *tagline*, which have many words in common. For instance, for the query *Artificial Intelligence*, PN ranked some science fiction movies that have the terms *Artificial Intelligence* in their *tagline* and *plot* first, but the desired answer was the movie “Artificial Intelligence”. XReal has the same problem, as it uses IR techniques. XReal assumes that fields with more occurrences of a query keyword are more important. For instance, the keywords of query *Edward Norton* appear more often in field *actor* than field *producer*. Thus, XReal ranks movies where Edward Norton acted higher than the movies he produced. Unfortunately, this heuristic works poorly for long fields, as they contain many words found elsewhere in the DB, and the unimportant (for the query) long fields may contain more of the query keywords than important fields do. For example, for query *Beautiful Mind*, XReal concludes that *tagline* is more important than *title*, because *tagline* has more occurrences of the word *beautiful* than *title* does, and ranks the desired answer with title “Beautiful Mind” low. XReal also ranks subtrees higher if they have many occurrences of the query keywords. Each movie has many long text fields in IMDB. Therefore, movies with similar subjects tend to have almost the same number of query keyword occurrences in their subtrees. For instance, all sequels to “The Mummy” have the same number of occurrences of *Mummy* and *Return*. However, the best answer to the query *Return of the Mummy* is the one with these words in its title, and XReal does not recognize this. This situation is common in data sets with long text fields.

XSearch uses an IR-style formula to rank its results. Thus, it has the same precision problems as PN. Moreover, it ranks the smaller subtrees higher. Thus, for the query *Crime The Godfather*, it ranks a movie whose keywords include *crime* and *Godfather* first, instead of the movie “The Godfather”, which was the desired answer. XSearch has better MAP than PN and XRank, almost as good as that of XReal. CR performs only slightly better than PN, because CR does not work well for long text fields, as expected. NTPC correctly recognizes the important fields and patterns, and delivers better MAP than all other methods. For instance, for *Artificial Intelligence* and *Return of the Mummy*, NTPC ranks the desired movies first.

Fig 4.10 shows the F-measure of the 8 methods on IMDB; higher F-measures are better. NTPC’s handling of long fields allows it to surpass CR, which in turn is significantly better than the other methods.

The MAP of NTPC is high but not perfect; NTPC does not return the perfect ranking for every query. The first reason is that some important fields have low collective entropy, such as *Genera* in IMDB. NTPC does not realize how important they are. The second reason is that our users preferred the most popular/famous papers and movies. IMDB and DBLP do not provide popularity information, although some related fields are present, such as *award* in IMDB. Resolving these issues is left as future work.

Chapter 5

Design Independent Keyword Query Systems

5.1 Background

In this chapter, we explore the effect of data organization modifications on query answers. We consider a slightly more general form of keyword queries in this chapter, by focusing on *schema free query interfaces* (SFQIs) [57, 26, 87, 149, 90, 6]. As usual, a query is a sequence of terms $Q = t_1 \cdot \cdots \cdot t_q$; however, now each term is either a keyword (intended to match a leaf node in the database, as usual) or the label of an attribute (intended to match a non-leaf node). The exact definition of a query differs slightly for different SFQIs. In some SFQIs, users can specify the selection attributes, i.e., the attributes that contain the keywords; and the projection attributes, i.e., the attributes whose values users want to see in the output [26, 87]. In pure keyword query interfaces [57, 26, 87, 149, 90, 6], the query system has to figure out this information and return the values of selection and projection attributes. Our work in this chapter is orthogonal to the format of the query, as long as the query does not contain any information about the relationship between attributes in the schema.

Our contributions in this chapter are as follows:

- We explore and formally define the similarity between answers to the same query across different organizations of the same data, from a user's perspective.

- We introduce and formally define the design independence property and explore its benefits for SFQI users. We introduce *value structure preserving* database transformations, which preserve schema information and the database content. We prove that if an SFQI is built properly, it will be design independent under value structure preserving transformations.

- We analyze the design independence property for current XML SFQIs and argue that all except one, CR (introduced in Chapter 3), are not design independent.

- Design independence requires the original and redesigned databases to have the same content, so it does not cover database redesigns that introduce or eliminate data redundancy. Since database transformations such as

Portions of this work appeared in [130] and [131].

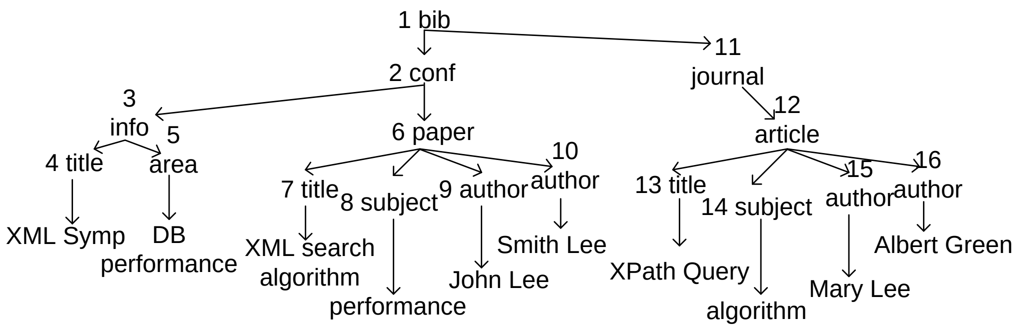


Figure 5.1: A bibliographic database

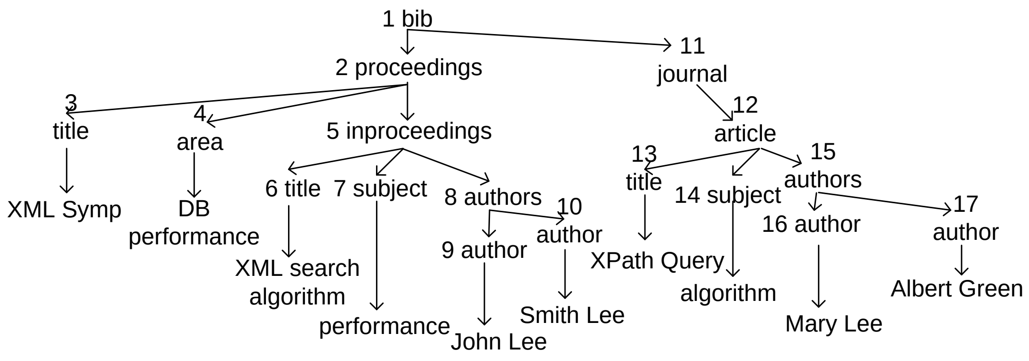


Figure 5.2: A bibliographic database

normalization affect redundancy, we introduce a new property called *weak design independence* for the case where the new database has more or less redundant data than the old database.

- No current SFQI is weakly design independent. We define a new SFQI called **Duplication Aware Coherency Ranking** (DA-CR) that is weakly design independent. DA-CR identifies desirable query answers by exploiting the information-theoretic relationships between the elements in the database, combined with a novel content scoring formula.

- We provide an extensive empirical study to evaluate the average case design independence of current and new SFQIs over three real-world data sets. We show that CR and DA-CR have considerably better average design independence than other SFQIs when the new database design does not introduce or remove data redundancy. We also show that DA-CR has higher average design independence than other methods if the new designs create or eliminate data redundancy. Through an extensive user study, we show that DA-CR delivers the same or better ranking than other methods. Thus, its design independence does not reduce its effectiveness.

In the reminder of the chapter, Section 5.2 presents basic definitions. Section 5.3 defines design independence and

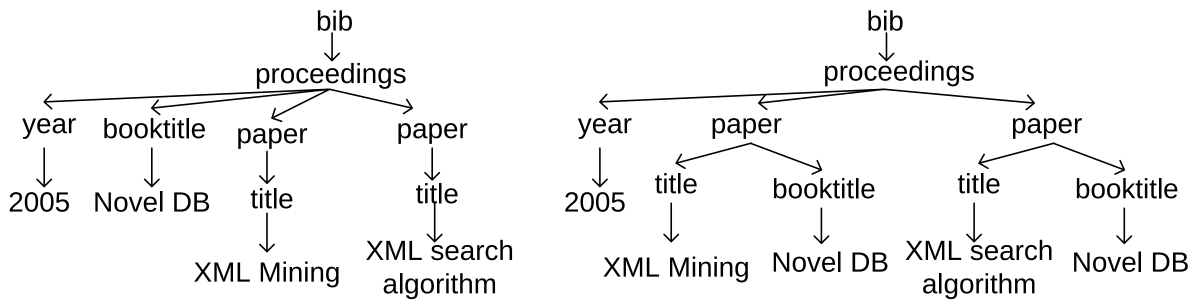


Figure 5.3: Normalization of a bibliographic database

explores its properties. Section 5.4 analyzes this property for current SFQIs. Section 5.5 defines and analyzes weak design independence and introduces DA-CR. Section 5.6 discusses experimental results for the average case design independence and weak design independence for all discussed methods, and analyzes the effectiveness of DA-CR using real world databases.

5.2 Basic Definitions

Given a query Q, a subtree S of the database is a candidate answer for Q iff each of its content nodes either contains at least one instance of each keyword term in Q, or the attribute of the content node contains one label term in Q. We consider only candidate answers that contain all terms of the input query. As usual, the root of a candidate answer is the *lowest common ancestor* (LCA) of its content nodes. For example, the subtree with LCA 3 in Figure 5.1 is a candidate answer for query Q_1: *title performance*. We consider the path from the LCA of a candidate answer to the root of the database as part of the candidate answer, as it simplifies our analysis.

As usual, the baseline SFQI returns all candidate answers to a query. As discussed in previous chapters, many candidate answers are unhelpful. For instance, consider query Q_2: *algorithm Lee* on the database fragment shown in Figure 5.1. candidate answer a_1: *1 2 6 7 “XML search algorithm” -1 -1 -1 -1 11 12 15 “Mary Lee” -1 -1 -1 -1* and candidate answer a_2: *1 2 6 9 “John Lee” -1 -1 -1 -1 11 12 14 “algorithm” -1 -1 -1 -1* are unhelpful answers for Q_2. The only relationship their attributes have is that they occur in the same database, which is not very interesting. As discussed in previous chapters, some SFQIs filter out candidate answers they deem unhelpful [87, 149, 90, 85], and others return a ranked list of candidate answers. For instance, one approach filters out every candidate answer whose root is an ancestor of the root of another candidate answer [149, 90]; the LCAs of the remaining candidate answers are called the *smallest* LCAs (SLCAs). The SLCA approach relies on the intuitively appealing heuristic that far-apart nodes are not as tightly related as nodes that are closer together. SLCA removes a_1 and a_2, as their roots (node 1) are ancestors of node 6, which is the root of another candidate answer, a_3: *6 7 “XML search algorithm” -1 -1 9 “John*

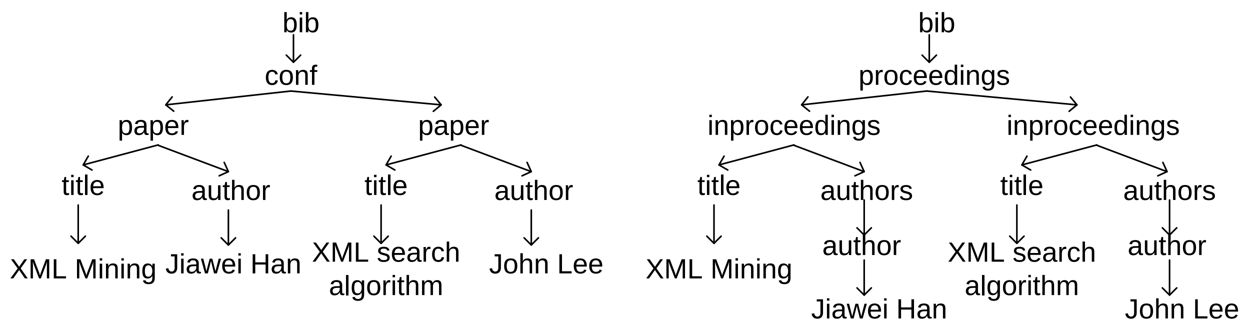


Figure 5.4: Answers to Q_4 over original and transformed databases

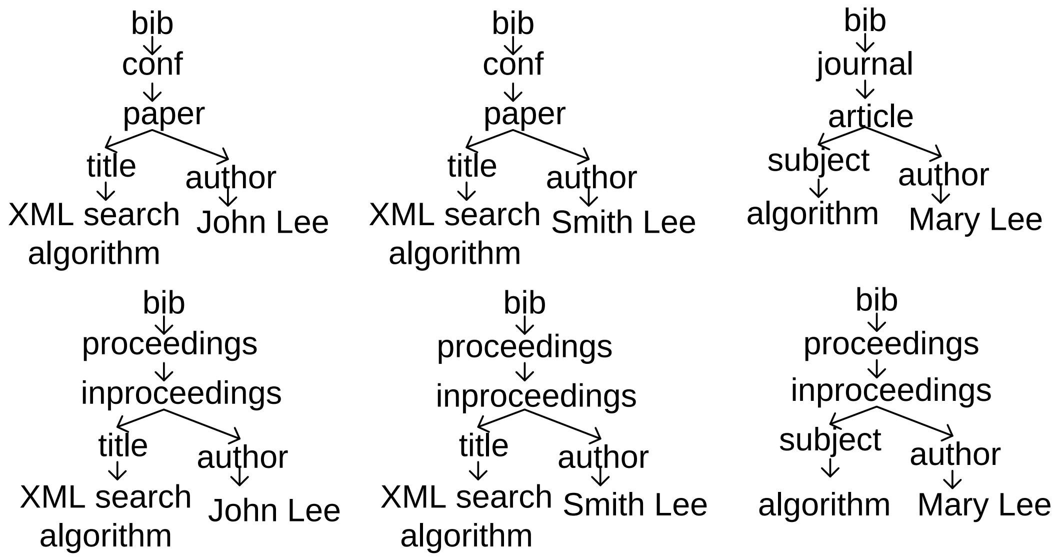


Figure 5.5: Q_2 answers over original and transformed databases

Lee' -1 -1. If an SFQI only filters out irrelevant candidate answers, it outputs a *multiset of candidate answers* (multiset for short) for a given query, rather than a list.

It is possible to develop a domain-independent SFQI and then leverage additional domain- and database-specific knowledge to help identify desirable answers. For example, most IMDB users may be more interested in new releases than older movies. In this thesis we focus on domain-independent techniques, to minimize the need for manual addition of domain-specific heuristics.

5.3 Design Independence

A **transformation** T over database D is a function that modifies D to generate a new database $T(D)$ [66, 45]. We assume that the query system is unaware of the differences between the two databases, or the transformation used to map between them.

If two candidate answers are isomorphic and their corresponding leaf nodes contain the same content, they are *label-content isomorphic*. Label-content isomorphic answers represent the same information. Hence, we intuitively consider an SFQI to be design independent if it returns the same list (multiset) of (label-content isomorphic) answers for every query over the original and modified databases. However, when a database is modified, an SFQI may not be able to return exactly the same list of candidate answers for a query over the old and new databases. For instance, Figure 5.2 is the result of a transformation of the fragment shown in Figure 5.1. Consider Q_3: *Query Green* over Figure 5.1, where its candidate answer is the subtree whose LCA is node 12 and contains nodes 13 and 16, and Figure 5.2, where its candidate answer is the subtree whose LCA is node 12 and contains nodes 13 and 17. Since the candidate answer for Q_3 over Figure 5.2 contains a new node *authors*, the candidate answers of Q_3 over Figure 5.1 and Figure 5.2 are not isomorphic. However, they both represent the authorship relation between *Albert Green* and *XPath Query*. Thus, we argue that users gain the same information from two candidate answers that express *equivalent relationships* between data items that have *the same content*. Therefore, we also consider an SFQI to be design independent if it returns essentially the same information for every query over the original and transformed databases. We will precisely define what it means for answers over the original and transformed databases to convey the same information to users.

5.3.1 Preserving Content

Answers that convey the same information must contain the same content. For instance, if the content of node 13 in Figures 5.1 and 5.2 were different, users would consider the candidate answers to Q_3 over these data fragments to be different. Consider a transformation over the data fragment in Figure 5.1 that removes *author* and *title* nodes that are children of the same *paper* node and creates a single new child node for each *paper* node, called *paperInfo*, that contains the merged content of the eliminated *author* and *title* nodes. The candidate answers for Q_3 over the original and transformed databases have similar content. Nevertheless, the candidate answer of the original database represents the author and the title of the paper in separate nodes and the candidate answer of the transformed database shows the author and the title of the paper in one content node. By looking at the candidate answer of the transformed database, some users may not be able to distinguish the tokens that represent the title from the tokens that represent the author.

Definition 5.3.1. *A bijective mapping M from pattern instance i_1 to pattern instance i_2 is* ***value preserving*** *iff it maps each member of the value of i_1 to an equal member of the value of i_2.*

We consider two value members equal if they are lexicographically equal, i.e., they have the same length and contain the same characters in the same positions. We define value preserving mappings between candidate answers similarly.

Definition 5.3.2. *Pattern instances i_1 and i_2 are* ***value equivalent*** *iff there is a value preserving mapping from i_1 to i_2.*

We can similarly define value equivalent candidate answers. Since users observe all candidate answers that an SFQI returns for a query, we must define value equality between the lists or multisets for the same query over different databases.

Definition 5.3.3. *A bijective mapping M from list l_1 to list l_2 is **value preserving** iff it maps each candidate answer $s \in l_1$ to a value equivalent candidate answer $M(s) \in l_2$, where the rank of s in l_1 is equal to the rank of $M(s)$ in l_2.*

Definition 5.3.4. *Lists l_1 and l_2 are **value equivalent** iff ere is a value preserving mapping from l_1 to l_2.*

Definition 5.3.4 requires the value equivalent lists to have the same number of candidate answers. Similarly, a bijective mapping M from multiset u_1 to multiset u_2 is **value preserving** iff it maps each candidate answer $s \in u_1$ to a value equivalent candidate answer $M(s) \in u_2$. Hence, two multisets are value equivalent if we can define a value preserving mapping between them. There may be more than one value preserving mapping for value equivalent multisets.

If a transformation manipulates the value of a content node, no SFQI will be able to return value equivalent lists (or multisets) for queries whose candidate answers contain that content node. For instance, if a transformation updates the value of node 9 in Figure 5.1 to "Kate Lee", there is no value equivalent mapping between the candidate answers of Q_2 over the original and transformed databases. Since SFQIs do not know about transformations, they cannot return value equivalent lists (multisets) to Q_2. Thus, we must find the properties of transformations that allow SFQIs to be design independent.

Definition 5.3.5. *A bijective mapping M from pattern p_1 to pattern p_2 is **value preserving** iff it maps each instance of p_1 to a value equivalent instance of p_2.*

Patterns p_1 and p_2 are **value equivalent** iff there is a value preserving mapping from p_1 to p_2.

Definition 5.3.6. *A transformation T over database D is **value preserving** iff it maps each pattern p in D to exactly one value equivalent pattern $T(p)$ in database $T(D)$, where each pattern $T(p)$ is mapped to by only one pattern p in D.*

Given a query, an SFQI filters or ranks the candidate answers for the query. Hence, for an SFQI to be design independent, a transformation must map every possible candidate answer for every query over the original database to a value equivalent candidate answer over the transformed database, and vice versa.

Proposition 5.3.7. *Given a value preserving transformation T over database D, the multisets of candidate answers to every query q over D and $T(D)$ are value equivalent.*

Proof. According to Definition 5.3.6 if q does not have any candidate answer over D, it will not have any candidate answer over $T(D)$. In that case, the multisets of answers are value equivalent. If q has at least one candidate answer

```
<!ELEMENT bib (conf*, journal*)>
<!ELEMENT conf (info, paper*)>
<!ELEMENT info (title, area)>
<!ELEMENT paper (title, subject, author*)>
<!ELEMENT journal (article*)>
<!ELEMENT article (title, subject, author*)>
```

Figure 5.6: Schema of the bibliographic database in Figure 5.1

```
<!ELEMENT bib (proceedings*, journal*)>
<!ELEMENT proceedings (title, area, inproceedings*)>
<!ELEMENT inproceedings (title, subject, authors)>
<!ELEMENT authors (author*)>
<!ELEMENT journal (article*)>
<!ELEMENT article (title, subject, authors)>
```

Figure 5.7: Schema of the bibliographic database in Figure 5.2

over D, then each candidate answer is an instance of a pattern. Since all pattern instances of D and $T(D)$ are value equivalent, each candidate answer of q over D is value equivalent to one and only one candidate answer of q over $T(D)$. Thus, the multisets for q over D and $T(D)$ are value equivalent. $\square$

As noted, if a transformation introduces new terms into a database or eliminates some terms from the database, then every SFQI will return different answers over the original and transformed databases for all queries that contain the added or removed terms. One may argue that it may be possible to develop an SFQI that returns similar results under a transformation that modifies only the number of instances of some terms in the content nodes of a database. Researchers have considered a similar problem in the context of retrieval over noisy document collections [161]. They have shown that even slight modifications in the number of instances of query terms in the documents of a collection results in considerably different rankings for some queries, when using any reasonably effective retrieval method. We have to consider the properties of the content nodes in the retrieval formula in order to develop a reasonably effective SFQI. Since each content node is a small document, it will not be possible to have an SFQI that returns similar results under this type of transformation.

5.3.2 Preserving Structure

Non-content nodes represent structural relationships between content nodes. If a transformation renames or removes schema elements or introduces new schema elements, it may change the structural relationship between content nodes. Hence, SFQIs cannot always deliver answers with exactly the same structure over the original and new database. Suppose that users can mentally translate the structure of the old answers to the structure of the new answers. Then, the SFQI returns structurally similar answers over both databases. Since an SFQI uses schema information to rank and/or filter candidate answers, we must only consider transformations that do not lose any information of the schema of the original database. (We do not consider complex consistency or integrity constraints as part of the schema information.)

Let S be the schema of XML database D (e.g., its DTD) and let $I(S)$ be the set of all databases with schema S. Given schemas S_1 and S_2, if we can find an invertible function T: $I(S_1) \rightarrow I(S_2)$, then S_2 carries at least as much information as S_1 [66, 45]. In other words, we can reconstruct the information available in any database over the original schema S_1, given the transformed database. This definition is suitable for data exchange and schema mapping, where the target schema may contain more information than the original schema. Therefore, it allows T to be a partial function. Since we would like to have similar answers over the original and transformed databases, we need T to cover all databases of a given XML schema and not add any new information to the original schema. Given XML schemas S_1 and S_2, transformation $T : I(S_1) \rightarrow I(S_2)$ is **bijective** if it is a bijective mapping.

Figures 5.6 and 5.7 show the schema of the databases excerpted in Figures 5.1 and 5.2, respectively. There is a bijective transformation between these schemas, with each database under the schema of Figure 5.6 mapped to one and only one database under the schema of Figure 5.7. Assume a transformation U over the schema of Figure 5.6 that removes *journal*, *article*, *info*, *conf*, and *paper* and connects all attribute nodes to the *bib* node. Since we defined an XML database as an unordered tree, U maps more than one database of the schema shown in Figure 5.6 to one database of the new schema. Thus, U is not bijective and loses schema information.

We must also consider the limitations of domain-independent SFQIs. If two database nodes have the same label, domain-independent query interfaces, and SFQIs in particular, consider them to be instances of the same entity type. Consider a version of the data fragment in Figure 5.1 that contains additional *article*s written by different authors. Assume a transformation removes node 15 from this database and maps node 12 to *Mary-Lee-article*. Users may recognize that the resulting candidate answer for Q_3 carries the same information as the candidate answer over the original database. Nevertheless, we do not expect SFQIs to draw such a conclusion, because values and labels play distinct roles in query interfaces, even in SFQIs [57, 26, 6, 129, 85]. Thus as far as they are concerned, the new schema introduces a new entity type that is different from *article* entities and must be treated differently. Moreover, domain independent SFQIs treat values as uninterpreted bags of words (or objects) [57, 26, 87, 149, 90, 6, 129, 85]. Therefore, they cannot comprehend that the information added to the label carries the removed content information.

As another example, consider a transformation that groups all *article*s about the same *subject* under a new node labeled *article-subject* that is a child of *journal* in the original database whose fragments are shown in Figure 5.1. An SFQI may reasonably rank candidate answers containing articles under the same *article-subject* node higher than candidate answers containing articles under different *article-subject* nodes. This is because the first group of candidate answers represents a more interesting relationship between articles than the second group. However, since SFQIs consider the values as uninterpreted objects and different from labels, they return the same ranking for candidate answers in both groups over the original database.

Users expect candidate answers with equal content nodes to also have similar structural information. Figure 5.4 shows a candidate answer for Q_4: *search Han mining John* from a database whose schema is in Figure 5.6. Figure 5.4 also shows the value equivalent answer from a database whose schema is in Figure 5.7. *John Lee* and *Jiawei Han* have the same path in the candidate answer on the left. This indicates that they have similar structural roles in the original and transformed databases. Thus, users expect their equivalent content nodes in the other answer to have similar structural roles. The path of a content node encodes its structural role in a database. Hence, if two content nodes have the same path in every candidate answer a, their mapped content nodes in the value equivalent candidate answer of a must have the same paths. In addition to the same structural role for each content node, users should see similar structural relationships between mapped content nodes in candidate answers. For instance, the authorship of a paper whose title is *XML Mining* by *Jiawei Han* is represented using *bib conf paper title -1 author -1 -1 -1* in the left candidate answer in Figure 5.4. This pattern also represents the relationship between *XML search algorithm* and *John Lee* in the same candidate answer. Since the patterns of content nodes mapped to $\{$*XML Mining*, *Jiawei Han*$\}$ and $\{$*XML search algorithm*, *John Lee*$\}$ in the right candidate answer in Figure 5.4 are isomorphic, the candidate answers convey similar structural information about the relationships of these content nodes.

Definition 5.3.8. *A bijective mapping M from candidate answer s_1 to candidate answer s_2 is* ***value structure preserving*** *(VS preserving for brevity) iff for all subpattern instances i_1 and i_2 of s_1:*

- *i_1 and $M(i_1)$ are value equivalent.*
- *The patterns of i_1 and i_2 are isomorphic iff the patterns of $M(i_1)$ and $M(i_2)$ are isomorphic.*

Definition 5.3.9. *candidate answers s_1 and s_2 are* ***value structurally equivalent*** *(VS equivalent) iff there is a VS preserving mapping from s_1 to s_2.*

According to the first constraint in Definition 5.3.8, a VS preserving mapping is value preserving. Hence, VS equivalent candidate answers are also value equivalent.

Similar to the case for value equivalence, users consider all candidate answers in the list (multiset) of candidate answers to a query when judging the similarity of two lists (multisets). Figure 5.5 shows some candidate answers for Q_2: *algorithm Lee* over Figure 5.1 in the first row. It also shows some candidate answers over a transformation of Figure 5.1 in the second row, each of which is value equivalent to the one above it in the first row. The first two candidate answers in the first row have the same pattern and represent the same structural relationship between *XML search algorithm* and *John Lee* and *XML search algorithm* and *Smith Lee*, respectively. Thus, users will expect the first two candidate answers in the second row to have the same pattern, which they do. Since *Mary Lee* is the author of an article and *Smith Lee* is the author of a paper, they have different paths in the list of candidate answers for Q_2 over Figure 5.1. Hence, users expect them to have different paths in the list of candidate answers for Q_2 over the transformed database, but they do not. Thus, we consider the candidate answer lists in the first and second row of Figure 5.5 to be structurally dissimilar. We define the subpattern instances of a list (multiset) of candidate answers as subpattern instances of its candidate answers.

Definition 5.3.10. *A bijective mapping M from list l_1 to list l_2 is* **VS preserving** *iff for all subpattern instances i_1 and i_2 of l_1:*

- *i_1 and $M(i_1)$ are value equivalent.*
- *The candidate answer that contains i_1 and the candidate answer that contains $M(i_1)$ have equal ranks in l_1 and l_2, respectively.*
- *The patterns of i_1 and i_2 are isomorphic iff the patterns of $M(i_1)$ and $M(i_2)$ are isomorphic.*

Definition 5.3.11. *A list of candidate answers l_1 is* ***VS equivalent*** *to a list of candidate answers l_2 iff there is a VS preserving mapping from l_1 to l_2.*

Based on the first two conditions in Definition 5.3.10, a VS preserving mapping between lists of candidate answers is also value preserving. Thus, two VS equivalent lists of candidate answers are also value equivalent. VS equivalence is defined similarly for two multisets of candidate answers. There may be more than one VS equivalent mapping for two multisets of candidate answers. VS equivalent lists (multisets) of answers for a query *preserve* the contents of structural relationships between the content nodes; therefore, users get essentially the same information on the relationships between content nodes in the original and new databases.

If the candidate answers for every query q over a database are VS equivalent to the candidate answers for q over the transformed database, an SFQI can provide VS equivalent answers for q over the old and new databases. Thus, we define the required conditions for such transformations.

Definition 5.3.12. *A bijective mapping M from pattern p_1 to pattern p_2 is* ***VS preserving*** *iff for all subpattern instances s_1 and s_2 of p_1:*

- *s_1 and $M(s_1)$ are value equivalent.*
- *The patterns of s_1 and s_2 are isomorphic iff the patterns of $M(s_1)$ and $M(s_2)$ are isomorphic.*

Definition 5.3.13. *Patterns p_1 and p_2 are **VS equivalent** iff there is a VS preserving mapping from p_1 to p_2.*

Similar to value preserving transformations, we have:

Definition 5.3.14. *A transformation T over database D is **VS preserving** iff it maps each pattern p in D to exactly one VS equivalent pattern $T(p)$ in database $T(D)$, and each pattern $T(p)$ is mapped to by only one pattern p in D.*

For instance, the mapping from the fragment in Figure 5.1 to the fragment in Figure 5.2 is VS preserving.

Theorem 5.3.15. *If T is VS preserving for D, then any query's candidate answers for D and $T(D)$ are VS equivalent.*

Proof. By Proposition 5.3.7, candidate answers over D and $T(D)$ are value equivalent. Since each subpattern of the pattern of a candidate answer is a pattern in the original database, by Definition 5.3.14, the multiset of candidate answers for q over D and $T(D)$ are VS equivalent. □

Definition 5.3.16. *An SFQI is **design independent** if for each database D, query q, and VS transformation T for D, the answer to q from D and from $T(D)$ are VS equivalent.*

An SFQI may use some information in the database that may not be in the candidate answers to rank and/or filter the candidate answers. Thus, a VS preserving transformation must not lose any schema information of the original database, so that SFQIs will be able to use this information. Patterns of a schema are shared between all instances of the schema. Hence, we can define VS transformations over all databases of a given schema. Given XML schemas S_1 and S_2, we define a VS preserving transformation as a mapping from $I(S_1)$ to $I(S_2)$.

Theorem 5.3.17. *Given XML schemas S_1 and S_2, every VS preserving transformation $T : I(S_1) \rightarrow I(S_2)$ is bijective.*

Proof. Assume that there are two databases D_1 and E_1 with schema S_1 that map to the same database D_2 of S_2 using T. According to Definition 5.3.14, we can define a bijective mapping that maps every instance of each pattern from D_1 to an isomorphic pattern instance from E_1 with equal values. Thus, D_1 and E_1 represent XML trees with equal patterns and values. □

We can, however, find a bijective transformation that is not VS preserving. For instance, assume the data fragment shown in Figure 5.1 has more than one article in each journal and each article has only one *subject*. Assume that transformation T groups all the *article*s in a journal that have the same subject under a new parent node *article-subject* that is the child of *journal*. This transformation is bijective, as it provides a bijective mapping over the instances of the two schemas. Nevertheless, it does not provide a bijective mapping over the patterns of the original and transformed databases.

Since a VS preserving transformation may change the labels of the leaf nodes, a query that contains label terms might have different numbers of candidate answers in the original and transformed databases. If an SFQI allows its queries to contain label terms, we restrict VS preserving transformations so that they preserve the labels of the leaf nodes in the original database.

5.4 Design Independence of SFQIs

The baseline technique for answering queries, called the LCA method, returns all candidate answers. Based on Theorem 5.3.15, the LCA method is design independent. However, as mentioned in Section 5.2, this approach returns all the non-relevant candidate answers. Hence, it has the lowest precision. Moreover, it does not rank the candidate answers. Thus other methods use the properties of candidate answers to improve its effectiveness. They filter out irrelevant candidate answers via *filtering methods* [57, 26, 87, 149, 85], and/or rank the answers, via *ranking methods* [26, 6, 129]. There are two categories of filtering techniques: distance based and label based. Distance based methods deem a candidate answer to be irrelevant if its subtree is relatively large. ELCA [57] and MLCA [87] assume that only the closest nodes are meaningfully related. If candidate subtree t_1 shares a leaf node with another candidate answer t_2 and the LCA of t_1 is an ancestor of the LCA of t_2, they filter out t_1. For instance, for query Q_5: *DB XML* over Figure 5.1, we have two candidate answers: a_1^5 whose LCA is node 2 and contains nodes 4 and 7 and a_2^5 whose LCA is node 3 and contains nodes 4 and 5. ELCA and MLCA filter a_1^5 as it shares a leaf node with a_2^5 and its LCA is an ancestor of the LCA of a_2^5. Q_5 has also two candidate answers over Figure 5.2: a_3^5, whose LCA is node 2 and contains nodes 4 and 6; and a_4^5, whose LCA is node 2 and contains nodes 3 and 4. Since the LCAs of these candidate answers are the same node, ELCA and MLCA return both candidate answers. Thus, ELCA and MLCA return different results for these two fragments.

Proposition 5.4.1. *The ELCA and the MLCA methods are not design independent.*

Generally, VS preserving transformations that change the structure of a database by adding or removing non-leaf nodes change the results of ELCA and MLCA. These changes are quite frequent in XML database design. For instance, one designer may decide to put *title* and *area* of a conference into a new node to increase readability of the database and answers, while another designer may decide to remove these nodes to save space and increase performance.

As mentioned in Section 5.2, the SLCA method [149, 90] filters out every candidate subtree whose LCA is an ancestor of the LCA of another candidate subtree. SLCA returns two candidate answers for query Q_6: *performance XML* over Figure 5.1: the one whose LCA is node 3 and contains nodes 4 and 5 and the one whose LCA is node 6 and contains nodes 7 and 8. However, SLCA returns only one candidate answer for Q_6 over Figure 5.2, the one whose LCA is node 5 and contains nodes 6 and 7. The candidate answer whose LCA is node 2 is filtered out, as node 2 is an ancestor of node 5. Thus, SLCA-based methods return different results under VS preserving transformations. Similar to ELCA and MLCA, VS preserving transformations that add or remove non-leaf nodes change the results of SLCA.

Proposition 5.4.2. *The SLCA method is not design independent.*

Label based techniques such as XSearch [26] and CVLCA [85] remove every candidate subtree having two non-attribute nodes with the same label. The idea is that non-leaf nodes are instances of the same entity type if they have duplicate labels, and there is no interesting relationship between entities of the same type. For example, Q_7: *Lee XML* over the leftmost data fragment in Figure 5.4 has two candidate answers: a_1^7, whose LCA is node *bib*; and a_2^7, whose LCA is node *paper*. These methods filter out a_1^7 because it contains duplicate labels (*paper*). They return three candidate answers for Q_8: *Green Lee* over Figure 5.1: two whose LCAs are node 1 and one whose LCA is node 12. However, they return only one candidate answer to the same query over Figure 5.2. They filter out the candidate answers whose LCAs are node 1 as they contain duplicate label *authors*.

Proposition 5.4.3. *The XSearch and CVLCA methods are not design independent.*

In general, label based methods are sensitive to schema transformations that group similar nodes under a new node. For instance, in Figure 5.2, a designer has decided to group all authors under a new grouping node *authors*, to make the database more usable. Label based methods are also sensitive to renaming of schema elements.

Ranking techniques include using depth and the number of nodes in candidate subtrees, extensions of the PageRank technique, and statistical information about candidate patterns. For example, XSearch ranks the candidate subtrees according to the number of nodes in the candidate subtree [26]. However, as mentioned earlier, VS preserving transformations can add or remove nodes from a database. Thus, they can change the number of nodes for different candidate answers if they are not instances of the same pattern. This will change the rank of the candidate answers over the original and transformed databases.

XReal uses the depths of the LCAs and attribute nodes of candidate subtrees to find and rank the candidate answers [6]. Since VS transformations can add or remove nodes from the database, they can change the depth of LCA and attribute nodes. Again, if the candidate answers do not belong to the same pattern, VS transformations can change the depth of their LCA and attributes unequally, resulting in a different ranking over the original and transformed databases.

XRank extends the PageRank formula to XML databases, where nodes replace web pages and edges replace the links in the original PageRank technique. Thus, the importance of a node is proportional to the number of edges connected to it. VS transformations can change the number of edges connected to a node through removing its children or grouping its children under new nodes. For instance, the number of edges connected to node *conf* in Figure 5.1 is different from node *proceedings* in Figure 5.2. Changes to authoritative nodes such as *conf* alter the overall ranking of the nodes considerably [46].

As discussed in previous chapters, CR and NTPC rank candidate answers according to the correlations of their patterns. They use values of candidate patterns to compute correlations. The more correlated the values of a pattern are, the stronger their relationship is. Consider the original database excerpted in Figure 5.1, where each conference has many papers. Knowing the title of a paper gives a considerable amount of information about the author of the paper in the database. In other words, given a value of *title* in *bib conf paper title -1 author -1 -1 -1*, one can determine the value of *author* with high probability, as each paper does not have many authors. However, knowing *title* in *bib conf info title -1 -1 paper author -1 -1 -1* does not narrow down *author* very much, as each conference has many paper *author*s. CR and NTPC exploit this fact, and return a candidate answer if its pattern's correlation exceeds a given low threshold ϵ. They rank the answers in descending order of their patterns' correlations, computed using an extended version of mutual information. We focus on CR here, as the design independence properties of NTPC can be shown in the same manner.

Before analyzing CR's design independence, we briefly review the concepts associated with entropy in XML. The probability of value a in pattern p is $P(a) = \frac{1}{n} count(a)$, where $count(a)$ is the number of instances of p with value a and n is the total number of instances of p in the database. Intuitively, the entropy of a random variable indicates how predictable it is. The *entropy* of a pattern p having values $a_1, \ldots, a_n$ with probabilities $P(a_1), \ldots, P(a_n)$ respectively is $H(p) = \sum_{1 \leq j \leq n} P(a_j) \lg (1/P(a_j))$. Normalized total correlation (NTC) measures the correlation of a pattern; its value for a pattern t with root-paths $p_1, \ldots, p_n$, $n > 1$, is $NTC(q) = 1 - \frac{H(q)}{\sum_{1 \leq i \leq n} H(p_i)}$. If the size of a pattern is one, CR sets the value of $NTC(q)$ to $H(q)$.

Proposition 5.4.4. *Given VS transformation T that maps database D to $T(D)$, the NTCs of each pattern $p \in D$ and $T(p) \in T(D)$ are equal.*

Proof. Since T is VS preserving, it maps every pattern p in D to exactly one pattern $T(p)$ in $T(D)$, where $T(p)$ is mapped to by only pattern p in D. Also, T maps each instance s of p to exactly one instance $T(s)$ of $T(p)$, where s and $T(s)$ are value equivalent. Since $T(s)$ is not mapped to by any other instance except for s, p and $T(p)$ will have the same values. Thus, they have the same NTC. □

CR, similar to XSearch [26] and XReal [6], considers each candidate subtree as a small document and uses variations of TF-IDF formulas to leverage the values of candidate answers in delivering the final ranking. Each term k_i in the input query has a document frequency (DF) that is the number of attribute nodes of the same label in the database that contain k_i. The larger this number is, the less important the term becomes. If a candidate subtree contains more instances of important terms in a query, it has higher term frequency (TF) and gets higher rank. Since the content of candidate answers is the same over VS transformed databases, the TF remains the same under VS transformations.

Assume that a VS transformation does not change the label of the attributes. Since VS transformations do not change the number of instances of each path of a database, they do not change the number of instances for attribute nodes with the same label. Thus, the DF will be equal over the original and transformed databases. If a VS transformation changes the names of attributes, DF may not be equal for the original and VS transformed databases. In this case, we can measure DF over paths instead of attributes and make the TF-IDF formula design independent. CR computes a linear combination over NTC and TF-IDF scores to deliver the final ranking. Since both parts of CR are design independent, we have:

Corollary 5.4.5. *The CR method is design independent.*

We can prove that NTPC is design independent similarly.

5.5 Weak Design Independence

The definition of a VS transformation ensures that we have the same number and similar answers for a query over the original and transformed schemas. Therefore, VS transformations do not include transformations that preserve schema information but do not deliver exactly the same number of similar answers for each query. In particular, VS transformations require both databases to have the same number of equal values. Figure 5.3 depicts a non-VS transformation over a bibliographic database fragment that moves *booktitle* nodes from *proceedings* and duplicates them under all *paper*s of the same *proceedings*. The transformation denormalizes the data fragment and creates redundancy [4]. Since it is bijective, since it maps each database using the schema of the database whose fragment is shown on the left to one database using the schema of the database whose fragment is shown on the right. However, Q_9: *Novel 2005* has one candidate answer over the data fragment on the left and two candidate answers over the data fragment on the right. The candidate answers over the right data fragment are VS equivalent. Thus, we may consider them to be duplicates. If users can recognize and ignore duplicate candidate answers, we can consider both candidate answers as a single candidate answer. Then the results of Q_9 over the two databases are VS equivalent.

It is not unrealistic to expect users of a search system to recognize duplicate answers [96]. Consider Q_{10}: *Novel DB*. It has one candidate answer over the left fragment a_1^{10} whose LCA is *booktitle*, but three candidate answers on the right data fragment: a_2^{10} and a_3^{10}, whose LCAs are *booktitle* nodes; and a_4^{10}, whose LCA is node *proceedings*. a_1^{10} and a_2^{10} are VS equivalent. a_4^{10} is not VS equivalent to any candidate answers for the right data fragment, but conveys the same information as the others and we can consider it a duplicate.

Definition 5.5.1. *Pattern instance S_1 is* ***almost equal*** *to pattern instance S_2 iff there is an onto relation R between every path p of S_1 and every path $p' \in R(p)$ of S_2, such that p and p' are isomorphic and have equal values.*

For instance, candidate answers a_2^{10} and a_4^{10} are almost equal. In this section, we consider the values of a pattern to be a set.

Definition 5.5.2. *Pattern p_1 is a **duplicate** of pattern p_2 iff there is an onto relation R between every instance i of p_1 and every instance $i' \in R(i)$ of p_2, such that i and i' are almost equal.*

For instance, pattern b_1: *bib proceedings paper title -1 -1 paper booktitle -1 -1 -1* is a duplicate of pattern b_2: *bib proceedings paper title -1 booktitle -1 -1 -1* in the right data fragment in Figure 5.3. candidate answers are *duplicate*s of each other if they are almost equal and their patterns are duplicates of each other. For example, the candidate answers of Q_{11}: *Mining DB* over the right data fragment in Figure 5.3 are duplicates of each other.

We define the LCA of a pattern similarly to the LCA of its instances. Since the duplicate relationship between patterns is symmetric and reflexive, patterns can be divided into equivalence classes based on this relationship. For each equivalence class, we choose the pattern with the fewest paths to be its representative element. If more than one pattern qualifies, we choose the first one in lexicographic order. The representative pattern does not have two distinct isomorphic paths.

Definition 5.5.3. *A transformation T over database D is **weak VS (WVS) preserving** if it maps each equivalence class of duplicate patterns g to exactly one equivalence class of duplicate patterns $T(g)$, such that the representative pattern of g is VS equivalent to the representative pattern of $T(g)$ and each equivalence class $T(g)$ is mapped to by only one equivalence class g.*

The transformation shown in Figure 5.3 is WVS preserving. Each VS preserving transformation is WVS preserving, but there are some WVS preserving transformations that are not VS preserving, such as the transformation shown in Figure 5.3. We relax the condition on VS equivalence to define WVS equivalence for lists or multisets of candidate answers. We consider the output of an SFQI as a set, where all duplicate candidate answers in the same list or multiset are considered as equal. Also, we consider the position of each class of duplicate candidate answers in a list as the rank of the candidate answer in the class with the lowest rank.

Proposition 5.5.4. *Given a WVS preserving transformation T over database D, the candidate answers for every query q over D and $T(D)$ are WVS equivalent.*

Proof. Since the candidate answers of the representative patterns in both databases are VS equivalent, the candidate answers are WVS equivalent. □

As discussed in Section 5.3.2, SFQIs may use some schema information that is not in the candidate answers of input queries. Thus, we must make sure that WVS transformations do not add or lose any schema information. In other words, they must be bijective over the instances of the original and transformed schemas, as defined in Section 5.3.2.

Theorem 5.5.5. *Given XML schemas S_1 and S_2, every WVS preserving transformation $T : I(S_1) \rightarrow I(S_2)$ is bijective.*

Proof. Each database pattern belongs to an equivalence class of duplicate patterns. We can define a bijective mapping between each equivalence class and its representative pattern. According to Theorem 5.3.17, each VS preserving transformation is bijective. Each WVS transformation defines a VS transformation between the representative patterns of equivalence classes in the original and the transformed databases. Hence, each WVS transformation is bijective. □

Should an ideal SFQI return similar results under *all* bijective transformations that preserve database content? For example, suppose we map *article*s written by *Mary Lee* in Figure 5.1 to *Mary-Lee-article*. As noted in Section 5.3.2, SFQIs treat the labels of structural nodes and values of content nodes differently. It is not reasonable to expect SFQIs to return the same answers for this transformation, or for any others that use the *values* of a pattern instance in D to decide how to transform D's structure.

What about the remaining bijective transformations T that do not modify database content and use only D as input? All remaining transformations use the labels and the ancestor-descendant relationships between non-leaf nodes of D in their predicates. Hence, they modify the instances of isomorphic sub-patterns similarly. According to Definition 5.5.3, these transformations are WVS preserving. We call a method **weakly design independent** if it returns a WVS equivalent list or a set of candidate answers over a WVS transformation.

5.5.1 Duplicate Aware CR

Since VS transformations are also WVS preserving, methods that are not design independent will not be weakly design independent. With a WVS transformation, VS equivalent patterns may have different numbers of almost equal instances. Thus, if we consider the values of a pattern to be a multiset, their NTC values can be different in the old and new databases. Hence, CR does not provide design independence over WVS transformations. This means that patterns' correlations should be measured by statistical properties of their instances that will provide effective ranking *and* do not change under WVS transformations. Researchers have shown that the more distinct values an attribute has in the database, the more important the information usually is that it stores [154]. As WVS transformations do not change the number of distinct values of a pattern in a database, NTC simply needs to be changed to ignore duplicates.

Definition 5.5.6. *Given pattern t with paths $p_1, \ldots, p_n$, $n \geq 1$, p's **normalized set total correlation** (NSTC) is the NTC of its set of pattern values.*

As we will show in Section 5.6, patterns' NSTC and NTC tend to be very close in practice, so both estimate pattern correlation very well. From a more analytical point of view, their similarity follows from Zipf's law, which ensures that database attributes often have many rare values. For example, approximately 65% of DBLP (*dblp.uni-trier.de*) authors have published only one paper [41]. Zipf's law is not the only source of rare attribute values. Entities typically have keys and often have semi-keys, such as *title* in Figure 5.1. These attributes' values are highly selective, i.e., rare. Further, a pattern will be highly selective if it has at least one highly selective path. For example, *bib/conf/title* is not highly selective, but almost all values of *bib conf title -1 paper title -1 -1 -1* occur only once in the database excerpted in Figure 5.1. Hence NSTC and NTC tend to be close for a pattern with many instances. We compute the values of NSTC as for NTC [129], in a preprocessing stage.

As mentioned in Section 5.4, SFQIs adapt TF-IDF methods for IR-style ranking. Assume a WVS transformation maps path p in D to $T(p)$ in $T(D)$. Since there could be different numbers of instances of p and $T(p)$, the DF of the terms in the values of p and $T(p)$ will be different. Hence, current TF-IDF methods are not design independent under WVS transformations. Thus *we redefine the DF of a term w in pattern p to be the number of distinct values of p that contain w*. With the redefined DF, we extend the pivoted normalization method [96] to determine the contextual rank $ir(t, Q)$ of a candidate answer t with pattern p for query q:

$$ir(t, Q) = \sum_{w \in q,t} \frac{1 + \ln(1 + \ln(tf(w)))}{(1-s) + s(el_t/avel_p)} \times qtf(w) \times \ln\left(\frac{N_p + 1}{df_p}\right). \tag{5.1}$$

Here, $tf(w)$ and $qtf(w)$ are the number of occurrences of w in t and the qs, respectively. el_t is the total length of the content of t, and $avel_p$ is the average length of the distinct values of p. N_p is the count of distinct values of p, and df_p is the number of distinct values of p that contain w. s is a constant; the IR community has found that 0.2 is the best value for s [96]. We combine ir and NSTC on a sliding scale as:

$$r(t, Q) = \alpha NSTC(p) + (1-\alpha) ir(t, Q), \tag{5.2}$$

where p is t's pattern and α is a constant that determines the relative weight given to structural versus contextual information. We determined the best value of α empirically.

The proposed ranking scheme has two problems. First, the user may have to scan through many duplicate patterns in the list of answers. Second, the IR formula may rank larger patterns in the same equivalence class higher, as they have more paths and therefore may contain more query terms. To address these problems, we group patterns with equal values for NSTC and the same set of paths before query time. After finding the candidate answers at query time, we find the equivalence class of the pattern of each candidate answer and consider only candidate answer(s) with the smallest patterns in each class. If there is more than one such pattern, we break the ties arbitrarily. We call the new approach *Duplicate Aware CR (DA-CR)*.

Theorem 5.5.7. *DA-CR is weakly design independent.*

Proof. Let T be a WVS transformation from database D to database $T(DB)$. We must prove that given query q, the lists of candidate answers returned by DA-CR for q over D and $T(DB)$ are WVS equivalent. Hence, according to formula (5.2), we must show that the values of NSTC and ir for the candidate answers whose patterns are the representatives of their equivalence classes in D and $T(D)$ are equal. All patterns in the same equivalence class in D have equal NSTC. Similarly, all patterns that belong to a equivalence class in $T(D)$ have equal NSTC. Let p be the representative pattern of equivalence class g in D and $T(p)$ be the representative pattern of equivalence class $T(g)$ in $T(D)$. Since p and $T(p)$ are VS equivalent, they have equal NSTC values. Since WVS transformations do not merge content nodes, p and $T(p)$ have the same size. Hence, the NSTC value of candidate answers of p is equal to the NSTC value of the candidate answers of $T(p)$. Since p and $T(p)$ are VS equivalent, they have equal $avel_p$ and N_p. As candidate answer t of p and candidate answer $T(t)$ of $T(p)$ are value equivalent, we have $el_t = el_{T(t)}$. Similarly, t and $T(t)$ have equal $tf(w)$ for all $w \in q$. Thus, t and $T(t)$ have equal ir values. As the candidate answers of p and $T(p)$ have equal NSTC and ir values, they will have the same ranking score. As D and $T(D)$ have the same number of candidate answers, the results of DA-CR for q over D and $T(D)$ are WVS equivalent. □

To compute ir in Formula (5.2), for every term w and pattern p up to a given size, we must compute $df_p(w)$, the number of distinct values of p that contain w; and N_p, the number of distinct values of pattern p. We also must compute $avel_p$, the average length of all distinct values in p. Since the number of patterns and terms in a large database is huge, exact computations of these values are prohibitively expensive. Furthermore, this information occupies a lot of space on disk. Thus, we estimate them by assuming that the terms occur in the paths of a pattern independently. Assume that p contains paths $q_1, \ldots, q_n$ and $p_w(q_i)$ shows the probability of term w appearing in distinct values of q_i:

$$\frac{df_p(w)}{N_p + 1} \approx \frac{df_p(w)}{N_p} \approx 1 - \prod 1 \leq i \leq n(1 - p_w(q_i)). \tag{5.3}$$

Similarly, we estimate $avel_p$ as $\sum_{1 \leq i \leq n} avel_{q_i}$. Spark [93] used the same idea to estimate IR-style statistics for relational data, and found the average error rate to be around 30%. We computed exact values for $avel_p$ and $\frac{df_p(w)}{N_p+1}$ for all patterns up to size 3 for DBLP and got an average error rate of 32%, which subsequent experiments show to be acceptable for ranking purposes.

5.6 Experiments

We performed an extensive empirical study to measure the average case design independence of the SFQIs discussed in Section 5.4 on three real world databases: IMDB (205 MB, 35 schema elements, max depth 7), DBLP (102 MB, 31 schema elements, max depth 7), and Mondial's geographical information about countries (*dbis.informatik.uni-*

goettingen.de/Mondial) (1.5 MB, 53 schema elements, max depth 5). Since the structure of DBLP was originally relatively flat and similar to Mondial, we transformed it by putting papers and articles under their associated journals and proceedings to get a more nested structure.

We asked undergraduate CS majors who were not conducting this research and were familiar with the concepts and issues of database design and XML to create new designs for each database. We explained to them the properties of a WVS transformation, so that they created a new database that was a WVS transformation of the original database. They were required to provide an acceptable objective for each redesign. For instance, if they created a new entity, they had to explain how the new entity helps users understand the data and answers better (usability). For instance, one designer decided to create a new entity called *crew-info* that contains the information on *actor*, *actress*, *director*, *writer*, and other movie crew members. He also created another new entity *production-info* that contained all information on the production and distribution process of a movie, such as *location*, *production-company*, and *distributor*. Inside this new entity, the business information such as the box office of a movie was grouped under another entity called *overall-business*. The objective of this design was to increase the usability of the data because IMDB has more than 40 attributes for each movie and most attributes can have over 30 instances. Thus, it is very hard for users to find the information they want among all these attributes. Similarly, if designers merged some entities or denormalized the database, they had to explain how it would improve query processing performance. For instance, one designer moved *year* and *booktitle* from *proceeding*s or *journal*s to *paper*s in DBLP. This way the system needs to perform fewer structural joins between these nodes and other nodes from a *paper*, such as *title* and *author* [87].

We collected three redesigns for each database. Since Mondial has a relatively simple and flat schema, our designers could not find a WVS transformation that involves duplication. Furthermore, we also collected a different design for each dataset from a real database in the same domain. We used the design of SIGMOD Record at *www.dia.uniroma3.it/Araneus/Sigmod* as an alternative design for DBLP, the design of the movie data set from Metacritic at *www.metacritic.com* as an alternative design for IMDB, and the design of the geographic database from the CIA World Factbook at *www.cia.gov/library/publications/the-world-factbook* as an alternative design for Mondial. Since some of the schema elements in our data are not presented in their schemas, we added some new schema elements to these designs in order to make their transformations VS preserving. The designers wrote one XSL script for each new design to transform the database instance. We name each design by its objective.

We collected 80 keyword queries for DBLP, 79 keyword queries for IMDB, and 60 keyword queries for Mondial, submitted by 16 users. The query length ranged from 2 to 5. A complete list of the queries can be found in Table 5.4.

As mentioned in Section 5.4, SFQIs filter out candidate answers and/or rank candidate answers. We implemented filtering methods SLCA, CVLCA, XSearch, XReal, and ELCA, and ranking methods CR, XSearch, XReal, and XRank. Accordingly, we used two metrics to compare the result lists or multisets delivered by a method over different

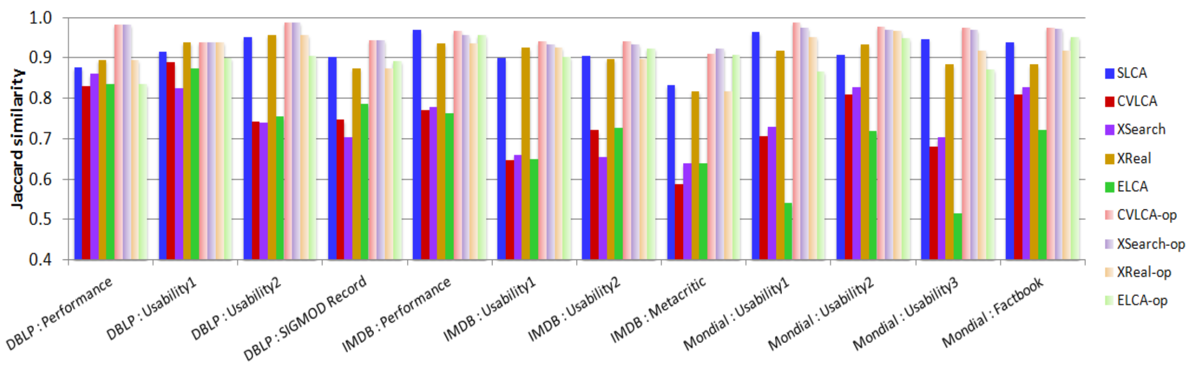


Figure 5.8: Design independence of filtering methods

designs of a database. For filtering methods, we adapted the Jaccard index to measure the similarity between their results, by counting the number of different candidate answers in the multiset results, and normalizing it by dividing by the total number of candidate answers from both results together. If the results have exactly the same candidate answers, the value is 1; if they do not overlap, the value is 0.

For rank-based comparison, we used Kendall's tau metric [77], which penalizes a list if a candidate answer occurs in a position different from its position in the other list. We normalized Kendall's tau by dividing it by its maximum value, $n(n-1)/2$, where n is the total number of elements in the list. If the list of results from two databases is identical, then the measurement is 1, and if the ranked result from one database is the reverse of the other's, then the measurement is 0.

Kendall's tau is designed to compare lists with the same number of elements. As ranking methods such as XSearch and XReal do both ranking and filtering, they can produce ranked lists of different lengths for the same query over different databases. Hence, we do multiset comparisons of these methods to find the difference between the multisets of elements in both lists. Then, we assume that the missing elements from a list are ranked at the end of the list, and compute Kendall's tau to find the difference between the rankings in the lists.

Normally, ranking methods return a very large number of candidate answers for an input keyword query, but users focus primarily on the top ranked results. Hence, we also compute the average design independence for ranking methods over only the top 20 candidate answers for each query.

5.6.1 Average Design Independence

The first five bars (solid colors) for each data set in Figure 5.8 show the design independence of the filtering methods discussed in Section 5.4 over DBLP, IMDB, and Mondial, averaged over the different designs. Since IMDB's schema is more complex than DBLP's and Mondial's, all methods except ELCA have their lowest average design independence for IMDB. IMDB has more paths and patterns than DBLP and Mondial, making its candidate answers more

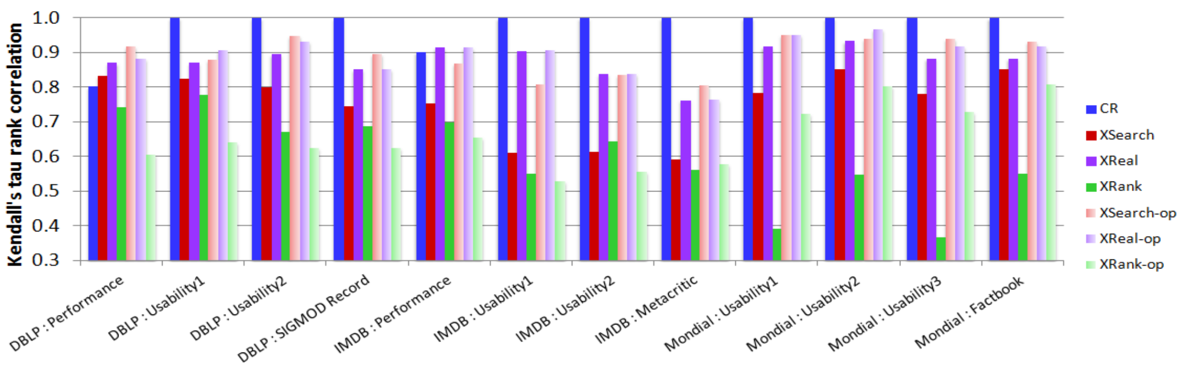


Figure 5.9: Design independence of ranking methods

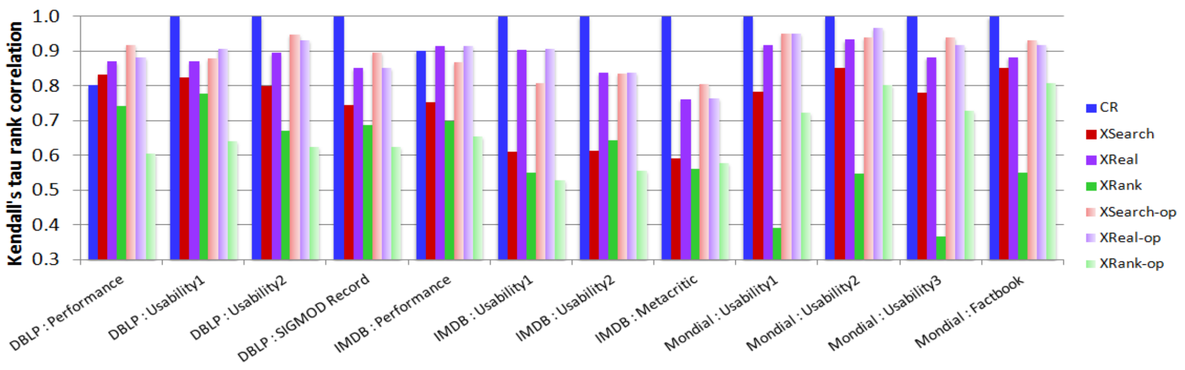


Figure 5.10: Design independence of ranking methods for top 20 answers

DB	CVLCA	XSearch	XReal
DBLP	11	11	1
IMDB	0	0	0
Mondial	26	26	33

Table 5.1: Average number of queries whose answers' LCA is the database root

DB	CVLCA	XSearch	XReal	Op.CVLCA	Op.XSearch	Op.XReal
DBLP	3	3	3	14	14	4
IMDB	0	0	11	0	0	11
Mondial	0	0	5	26	26	6

Table 5.2: Average number of queries returning no results

diverse. ELCA has its lowest average design independence for Mondial because ELCA is a distance based method, and all transformations over Mondial change the distance between leaf nodes. Since Mondial was originally very flat, a small change in distances between nodes makes a relatively large difference in query results.

The first four bars (solid colors) for each data set in Figure 5.9 show the design independence of the ranking methods in Section 5.4 for designs of DBLP, IMDB, and Mondial, respectively. Similarly, Figure 5.10 shows the average design independence of these methods considering only the top 20 answers. Interestingly, the resulting values for top 20 answers are quite close to the values computed over all ranked answers, except for a noticeable difference in IMDB, which we discuss later. DA-CR delivers perfect design independence over all three data sets (not shown in the figures). XSearch and XReal deliver their lowest average design independence over IMDB. CR and XRank, however, show their lowest design independence for DBLP and Mondial, respectively, for reasons explained later. All ranking methods except DA-CR deliver their lowest average design independence over IMDB when considering only the top 20 candidate answers, as shown in Figure 5.10. Since Mondial has the simplest schema, all filtering and ranking methods except for DA-CR generally have their highest average design independence for Mondial. Since all the transformations over Mondial are VS preserving, CR provides perfect average design independence over Mondial. The other filtering and ranking methods do not provide perfect average design independence even over such a relatively simple database.

We have proved that if a method is weakly design independent, it delivers perfect design independence over all WVS transformations of a database. Also, we have proved that if a method is design independent but not weakly design independent, it will show its lowest design independence over WVS preserving transformations. Our experiments confirm these theoretical results. DA-CR delivers perfect design independence for all transformations. CR shows its lowest design independence for *Performance* transformations over DBLP and IMDB, which are WVS preserving. As DA-CR provides perfect design independence over all transformations and databases, we ignore it in the rest of our analysis of average design independence.

Since other filtering and ranking methods are not (weakly) design independent, there is no specific type of transformation where their design independence is worst. Generally, method M has lower average design independence for transformation T, if T modifies the information used by M to filter or rank candidate answers the most. Hence, methods are less design independent for transformations that extensively change the database structure, either by introducing new nodes or modifying the nodes' positions. As discussed in Section 5.4, if a method is not design independent, it returns dissimilar results under *at least one* VS preserving transformation. Interestingly, none of the design dependent methods provides perfect average design independence over *any* VS preserving transformation in our experiments. This observation underlines the need for design independent query interfaces in real-world settings.

For DBLP, all ranking methods except XSearch deliver their lowest average design independence with the *Performance* transformation, because this mapping adds many new and duplicate nodes. As discussed in Section 5.4, XSearch ranks candidate answers by their sizes. This candidate answer property is changed the most by *Usability2* and in *SIGMOD Record* transformations. Thus, XSearch delivers its lowest average design independence for these redesigns of DBLP. We observe a similar trend for filtering methods. XReal and SLCA deliver their lowest average design independence with *Performance*. They use the depth of the LCAs to filter the candidate answers, and this property is modified the most by *Performance*. *Usability2* adds more duplicate labels and changes the distance between leaf nodes the most. As other methods consider duplicate labels and the distance between leaf nodes in candidate answers, they show their lowest average design independence under this transformation.

Except for CR, in IMDB the average design independence of filtering and ranking methods for *Usability1* and *Usability2* is lower than for *Performance*. This is because only a few queries match the parts of IMDB changed by *Performance*. *Usability1* and *Usability2* extensively change the database structure at popular query nodes such as *actor*, *actress*, and *title*. The average design independence for *Usability1* and *Usability2* over IMDB is very close for the different methods. All redesigns of Mondial are VS preserving. All methods except CR have their lowest average design independence with *Usability3* and their highest with *Usability2*. This is because *Usability3* adds more new nodes and increases the depth of the paths the most, whereas *Usability1* slightly changes the database design.

Figures 5.8 shows that the structural filtering technique of SLCA and XReal provides better design independence than label based techniques such as CVLCA and XSearch. However, their higher degree of design independence is mostly because they filter out more candidate answers than other filtering methods. For instance, XReal returns on average fewer than 1/5th of the candidate answers returned by XSearch, XRank, or CVLCA over IMDB. Similarly, SLCA returns on average fewer than half of the candidate answers returned by XSearch, XRank, and CVLCA over IMDB. If an SFQI returns more candidate answers for a query, it is likely to have lower average design independence, assuming all other conditions remain the same.

All queries in our experiments are guaranteed to have at least one candidate answer whose LCA is not the root of the database. Table 5.2 shows the number of queries that return no answer at all, using filtering and ranking methods, averaged over the database designs. XReal does not return any candidate answer for many queries over all three databases. It has been shown that SLCA and XReal have lower recall than other filtering methods [129]. Hence, the relatively high design independence of these methods comes at the cost of missing relevant answers, which is not a desirable property of an SFQI. Every filtering and ranking method but SLCA, CR, and DA-CR returns some candidate answers whose LCA is the root of the database. This type of candidate answer is not relevant in a large database [129]. Table 5.1 shows the number of queries where the LCA of every returned candidate answer is the database root, for CVLCA, XSearch, and XReal, averaged over all database designs. The recall of these methods for such queries is zero.

After DA-CR, CR delivers the highest overall average design independence among the ranking methods. Although CR is not weakly design independent, its design independence is better than all ranking methods except for DA-CR and XReal for *Performance* with IMDB and DBLP. This shows that if a method does not rely on schema details, it generally provides higher average design independence. XReal has the second best average design independence for *Performance* over DBLP and IMDB after DA-CR. However, XReal returns on average fewer than 1/6th of the candidate answers retuned by CR over DBLP with *Performance*. Similar results hold for XReal answers over IMDB.

	DA-CR	CR
IMDB	0.721	0.714
DBLP	0.837	0.832
Mondial	0.881	0.881

Table 5.3: Mean average precision for DBLP and IMDB queries

XReal has considerably lower recall than CR [129], as confirmed in Tables 5.1 and 5.2. A method that never returns any answers will have perfect design independence, so it is not surprising that XReal is slightly more design independent than CR over DBLP and IMDB for *Performance*, but it misses many relevant answers. XRank's rankings depend very much on the shape of the database. Since all transformations make significant changes to the database shape, XRank has the lowest average design independent among all ranking methods.

The ranking algorithms of SLCA, CR, and DA-CR are designed so that they do not return any candidate answer whose LCA is the root of the database. To be fair to other methods, we changed the ranking algorithms of other methods so that they discard such candidate answers. The last four bars in brighter colors in Figures 5.8 show average design independence after this optimization. Generally, the design independence of all filtering methods improves dramatically after this optimization, but no method delivers perfect average design independence. Though CVLCA and XSearch improve significantly, as shown in Table 5.2, they do not return any candidate answer for a considerable number of queries. As discussed above, returning many fewer answers tends to increase design independence, but severely penalizes recall. The average design independence of XReal does not change much. ELCA improves, but is still far from delivering perfect average design independence.

The last three bars with brighter colors in Figures 5.9 show the average design independence of ranking methods over different databases, after optimization through removal of candidate answers whose LCA is the database root. The average design independence of XSearch and XReal increase significantly, but the optimized versions of XSearch and XReal do not return any candidate answer for a considerable number of queries. The average design independence of XRank decreases after this optimization, which shows that its ranking method is quite unstable. Overall, the average design independence of most ranking methods increases after this optimization. However, they still do not deliver a perfect design independence as DA-CR does on every WVS transformation, or as CR does on VS transformations.

5.6.2 Effectiveness

As discussed above, higher design independence does not necessarily mean better effectiveness, i.e., better recall and precision. However, the previous chapters have already shown that CR delivers better ranking quality than the previously proposed methods discussed in this chapter. To show that DA-CR is at least as effective than CR, we compared the ranking quality of DA-CR and CR over three real-world databases. We asked 15 users who were not conducting this research to provide up to 5 keyword queries each for IMDB and DBLP. We collected 25 queries for DBLP, 40 queries for IMDB, and 35 queries for Mondial. There are fewer queries for DBLP because some of the users are not CS researchers, and could not provide meaningful DBLP queries. A complete list of queries can be found in [129]. We developed a query processing prototype based on the baseline algorithm that returns every candidate answer for each query. Our users submitted their queries to this system and judged each candidate answer as relevant or irrelevant by selecting a checkbox in front of each candidate answer. We compared the ranking quality of CR and DA-CR using mean average precision [96]. We use the NSTC and NTC values generated by $EV = 3$ and $L = 5$ for all databases, which suffices for our queries. We set α to 0.84 for all databases, which provides the best empirical results. Table 5.3 shows that DA-CR has almost the same MAP as CR over Mondial and DBLP, and a better MAP for IMDB. The new TF-IDF formula in DA-CR delivers better ranking quality for IMDB, because it has a more nested structure. Each query over IMDB returns more candidate patterns than for the DBLP and Mondial queries.

```
<!ELEMENT dblp (proceedings*, book*, ...)>
<!ELEMENT proceedings (year, booktitle, inproceedings*, editor*, title, ...)>
<!ELEMENT book (booktitle, year, incollection*, editor*, url, ee, ...)>
<!ELEMENT inproceedings (author*, title, pages, url, ee, cite*, ...)>
<!ELEMENT incollection (author*, title, pages, url, ee, cite*, ...)>
```

Figure 5.11: Part of the DBLP database DTD

```
<!ELEMENT dblp (proceedings*, book*)>
<!ELEMENT proceedings (inproceedings*, author*, editor*, series, volume, title, url, ee, ...)>
<!ELEMENT book (incollection*, author*, editor*, url, ee...)>
<!ELEMENT inproceedings (booktitle, year, author*, title, pages, url, ee, cite*, ...)>
<!ELEMENT incollection (booktitle, year, author*, title, pages, url, ee, cite*, ...)>
```

Figure 5.12: Part of the DBLP database DTD with *Performance* transformation

```
<!ELEMENT dblp (proceedings*, book*, ...)>
<!ELEMENT proceedings (booktitle, inproceedings*, editor*, info, url, ee, ...)>
<!ELEMENT book (booktitle, year, incollection*, editor*, url, ee...)>
<!ELEMENT info (title, series, volume, year)
<!ELEMENT inproceedings (author, title, pages, url, ee, cite*, ...)>
<!ELEMENT incollection (author, title, pages, url, ee, cite*, ...)>
<!ELEMENT author (name*)>
<!ELEMENT editor (name*)>
```

Figure 5.13: Part of the DBLP database DTD with *Usability1* transformation

Num	DBLP	IMDB	Mondial
1	robert sanz	flick jessica	greek orthodox
2	dmkd models	beautiful mind	bourgogne brest
3	design languages	dreams true	8535 opolskie
4	authenticated dictionary	english ocean	luzern switzerland
5	smyrlis yiorgos	sarandon susan	kanagawa kawasaki
6	culler schauser	grand torino	languedoc rousillon
7	barbara liskov	artificial intelligence	beijing china
8	conversation web	doctor zhivago	ajaccio franche
9	william yurcik	bale christian	olomouc praha
10	chris miller	clooney george	aragon zaragoza
11	haixun wang	ray satyajit	gorzow wielkopolskie
12	transaktionsprogrammen von	advice berg	gottingen oldenburg
13	giorgio parallel	belgian detective	arkhangelsk arkhangelskaya
14	2003 jugravu	christensen hayden	chunchon kunsan
15	stereo vision	thumbs up	anatoliki thraki
16	bart goethals	edward norton	bermuda dependent
17	reid stillings	harbor pearl	dhytiki ellas
18	social visualization	basic instinct	rostock schwerin
19	decision linear trees	crime godfather	asia japan
20	architecture hennessy patterson	cheadle jemison qin	lorraine metz
21	acm discovery oliver	allen french highmore	ivanovskaya kineshma
22	altman challenges knowledge	arnold claire terminator	slaski wodzilaw
23	1999 inferring webkdd	comedy portland stone	europe spain
24	iglesia techniques victor	japanese psycho thriller	budejovice ceske jihocesky
25	path planar shortest	animation dakota fanning	coruna galicia la vigo
26	molina sigmod ullman	alcoholic ofm paul	18412 saarland sachsen
27	authentication cryptography key	comedy military world	graz styria tyrol
28	distribution portability replication	blackson courtesy world	kuala lumpur malaysia
29	cootes edwards taylor	mathematics professor usa	cantabria laguna santander
30	1999 bailey stuart	britney disney spears	augsburg erlangen regensburg
31	database elmasri navathe	clooney pitt roberts	cote dazur provence
32	database sigmod storage	1997 kate winslet	calabria di reggio
33	flow klein planar	columbia grant hugh	burgenland eisenstadt klagenfurt
34	computations matrices polynomials	brian miracle train	10391 navarre pamplona
35	databases keyword search	damon garcia reiner	heilbronn pforzheim ulm
36	fung mangasarian selection	clooney mac pitt	czech jihomoravsky republic
37	closed frequent pattern	derek null power	italy lazio padova
38	cubesort feasible jorge	beauvais dollars tippi	de hospitalet llobregat
39	graphs matching maximum	docter hanks lasseter	cancun quintana roo
40	computer notes science	drama ghost hunt	alacant madrid valencia
41	lazebnik ponce schmid	2008 millionaire slumdog	andorra la vella
42	aleksandar boosting zoran	action clooney fox	brest bretagne rennes
43	cryptography key public	colton light utah	4374 fejer hodmezovasarhely
44	build mok salvatore	earth ship tsunami	and castile leon
45	molina ullman widom	bloom jackson mckellen	berlin erlangen furth
46	communication hypermedia modelling	japanese war world	amsterdam holland netherlands noord
47	aschoff environment measurement subsystem	2005 documentary mcginn	poland river warsaw warszwaskie
48	commutative free monoids partially	drama lynch watts	arzamas belgorod belgorodskaya oblast
49	cadam db2 engineering kohlman	amy catch christopher home	bashkortostan oktyabrsky salavat sterlitamak
50	arnulf ctla heiko mester	fox julia roberts romance	huancavelica huanuco ica peru
51	commodity launching narayan thomas	blake edwards peter sellers	cdn great lake slave
52	agarwal detection object roth	cuppers doll jassie reno	50 kharkiv kharkivska ukraine
53	barrera bivariate compactly hct	44 basil joe microcosm	arequipa ayacucho cajamarca callao
54	min on portfolio wolfgang	capitol control liberal press	bihor bistrita botosani braila oradea
55	2000 banded for pavlov	extramarital idealist teenage vigil	democracy equatorial guinea malabo transition
56	explicit lebedev systems time	afternoon bregman dog laurent	geleen limburg maastricht netherlands nl
57	andronov esm resampling simulation	cox filmmaking july keith	ababa addis africa economic ethiopia
58	bisimilarity deciding history preserving	bernet foenkinos gallotte vincent	el mariy of rep rybinsk
59	podelski reactive set tacas	cfi comedy david oregon	cauca choco cordoba cundinamarca yopal
60	multiple of power reads	mummy of return the	burma magway mandalay rangoon yangon
61	agarwal dheeraj puneet sanghi	commander far master side	
62	antje bernhard sitelang thalheim	clint eastwood the unforgiven	
63	frequent han patterns sigmod	bernstein jarecki kaluska spherical	
64	finding forbidden minimal minors	drama galileo paradise trevill	
65	1987 defined objects winfried	2004 angel julie villoldo	
66	236 calculation josefsson states	avenue mitch number original	
67	chen chi convergence wnaa	bouzereau jason summers warner	
68	breakthroughs embeddings graph optimal torben	century futurama matt patric	
69	clauses cononical horn of sets	barnett brian michael prodeje west	
70	641 beyond discussions sandra tutors	collins delano egger megan olivia	
71	biswas christine fischer friedman pietras	40 aaron griffin nights vinessa	
72	ishikawa kato kubota noguchi ono	coming garnet jordon motta scarlett	
73	alan design languages mycroft richard	and of speed the time wizard	
74	102 approach metasystem paul tremblay	al charles dede martin summers	
75	collapsing computation degrees strong via	allen comedy johansson scarlett woody	
76	cubic cuts efficient graph maximal	ambition heckler lorna reporter waitress	
77	dynamics interaction mass of the	investigation parkes scenes zophres	
78	1996 automatic properties riccardo some	2005 breckin justin mark paramount	
79	acm data mining sigmod workshop	edward featured james season yost	
80	chengfei maria workflow xiaofang xuemin		

Table 5.4: DBLP, IMDB and Mondial queries

```
<!ELEMENT dblp (proceedings*, book*, ...)>
<!ELEMENT proceedings (year, booktitle, inproceedings*, editor*, series, volume, title, location, ...)>
<!ELEMENT book (booktitle, year, incollection*, editor*, location, ...)>
<!ELEMENT inproceedings (paperinfo, pages, location, cite*, ...)>
<!ELEMENT incollection (paperinfo, pages, location, cite*, ...)>
<!ELEMENT paperinfo (author*, title)>
<!ELEMENT location (url, ee)>
```

Figure 5.14: Part of the DBLP database DTD with *Usability2* transformation

Dataset:Transformation	Renamed Elements #	Added Elements #
DBLP : Performance	0	0
DBLP : Usability1	2	2
DBLP : Usability2	0	2
DBLP : SIGMOD Rec.	0	3
IMDB : Performance	0	0
IMDB : Usability1	0	25
IMDB : Usability2	0	6
IMDB : Metacritic	5	21
Mondial : Usability1	0	7
Mondial : Usability2	0	4
Mondial : Usability3	0	10
Mondial : Factbook	8	11

Table 5.5: Modified schema elements in the schema of transformed databases

```
<!ELEMENT dblp (proceedings*, book*, ...)>
<!ELEMENT proceedings (year, booktitle, inproceedings, editor*, series, volume, title, ...)>
<!ELEMENT book (booktitle, year, incollections, editor*, ...)>
<!ELEMENT inproceedings (inproceeding*)>
<!ELEMENT incollections (incollection*)>
<!ELEMENT inproceeding (authors, title, pages, cite*, url, ee, ...)>
<!ELEMENT incollection (authors, title, pages, cite*, url, ee, ...)>
<!ELEMENT authors (author*)>
```

Figure 5.15: Part of the DBLP database DTD transformed to SIGMOD Record data set design

```
<!ELEMENT imdb (show*)>
<!ELEMENT show (title, year, keywords, actor*, actress*, director*, producer*, writers, genres,
        sound_track*, sound-mix, rating, location*, country*, ...)>
<!ELEMENT sound_track (title, info*)>
<!ELEMENT rating (votes, rank)>
```

Figure 5.16: Part of the IMDB database DTD

```
<!ELEMENT imdb (show*)>
<!ELEMENT show (title, year, keywords, actor*, actress*, director*, producer*, writer*, genres,
        sound_track*, rating, location*, country*, ...)>
<!ELEMENT show (title, year, actors, release_dates, ...)>
<!ELEMENT sound_track (title, info*, sound-mix)>
<!ELEMENT rating (votes, rank)>
```

Figure 5.17: Part of the IMDB database DTD with *Performance* transformation

```
<!ELEMENT imdb (show*)>
<!ELEMENT show (title, year, keywords, casts, staffs, places, genres, sound_track*, sound-mix, rating, ...)>
<!ELEMENT sound_track (title, info*)>
<!ELEMENT rating (votes, rank)>
<!ELEMENT casts (actors, actresses)>
<!ELEMENT staffs (directors, producers, writers, ...)>
<!ELEMENT places (locations, countries)>
<!ELEMENT actors (actor*)>
<!ELEMENT actresses (actress*)>
<!ELEMENT producers (producer*)>
<!ELEMENT directors (director*)>
<!ELEMENT writers (writer*)>
<!ELEMENT locations (location*)>
<!ELEMENT countries (country*)>
```

Figure 5.18: Part of the IMDB database DTD with *Usability1* transformation

```
<!ELEMENT imdb (show*)>
<!ELEMENT show (keywords, basic_info, cast_info, production_info, misc_info, ...)>
<!ELEMENT sound_track (title, info*)>
<!ELEMENT rating (votes, rank)>
<!ELEMENT basic_info (title, year, language, genre*)>
<!ELEMENT cast_info (actor*, actress*, director*, producer*, writer*, ...)>
<!ELEMENT production_info (location*, country*, release_date, ...)>
<!ELEMENT misc_info (rating, tech_info)>
<!ELEMENT tech_info (sound_track*, sound-mix)>
```

Figure 5.19: Part of the IMDB database DTD with *Usability2* transformation

```
<!ELEMENT movies (movie*)>
<!ELEMENT movie (title, year, production_company*, summary, detail&credits, user_reviews)>
<!ELEMENT summary (plot?, genres, keywords, runtime)>
<!ELEMENT details&credits (directors, casts, producers, writers, release_date*, locations, soundtrack, ...)>
<!ELEMENT user_reviews (distribution, votes, rank)>
<!ELEMENT producers (producer*)>
<!ELEMENT casts (actor*, actress*)>
<!ELEMENT directors (director*)>
<!ELEMENT writers (writer*)>
```

Figure 5.20: Part of the IMDB database DTD transformed to Metacritic data set design

```
<!ELEMENT mondial (country*, organization*, lake*, ...)>
<!ELEMENT country (name, population?, population_growth?, gdb_total?, gdb_agri?, religions*, border*,
        province*, city*, government, ...)>
<!ATTLIST country (capital, area, ...)>
<!ELEMENT province (name, area?, population?, city*)>
<!ELEMENT city (name, longtitude?, latitude?, area?)>
<!ELEMENT organization (name, abbrev?, established?, members)>
<!ELEMENT lake (name, located?, area?, ...)>
```

Figure 5.21: Part of the Mondial database DTD

```
<!ELEMENT mondial (countries, ...)>
<!ELEMENT countries (country*)>
<!ELEMENT country (name, population?, population_growth?, gdb_total, gdb_agri?, religions-s, borders,
        provinces, cities, government, ...)>
<!ATTLIST country (capital, area, ...)>
<!ELEMENT religions-s (religions*)>
<!ELEMENT borders (border*)>
<!ELEMENT provinces (province*)>
<!ELEMENT cities (city*)>
<!ELEMENT province (name, area?, population, city*)>
<!ELEMENT city (name, longtitude?, latitude?)>
```

Figure 5.22: Part of the Mondial database DTD with *Usability1* transformation

```
<!ELEMENT mondial (country*, ...)>
<!ELEMENT country (name, population_info, gdb_info, other_info, geograpy, ...)>
<!ATTLIST country (capital, area, ...)>
<!ELEMENT population_info (population?, population_growth, ...)>
<!ELEMENT gdb_info (gdb_total, gdb_agri?, ...)>
<!ELEMENT other_info (religions*, borders*, government, ...)>
<!ELEMENT geography (city*, province*)>
<!ELEMENT province (name, area?, population?, city*)>
<!ELEMENT city (name, longtitude?, latitude?)>
```

Figure 5.23: Part of the Mondial database DTD with *Usability2* transformation

```
<!ELEMENT mondial (countries, ...)>
<!ELEMENT countries (country*)>
<!ELEMENT country (basic_info, country_description, population_dist, geography)>
<!ELEMENT basic_info (capital, area, ...)>
<!ELEMENT country_description (name, population?, population_growth?, gdb_total, gdb_agri?, government, ...)>
<!ELEMENT population_dist (religions*, ...)>
<!ELEMENT geography (borders, cities, provinces, ...)>
<!ELEMENT borders(border*)>
<!ELEMENT cities (city*)>
<!ELEMENT provinces (province*)>
<!ELEMENT province (name, area?, population?, city*)>
<!ELEMENT city (name, longtitude?, latitude?)>
```

Figure 5.24: Part of the Mondial database DTD with *Usability3* transformation

```
<!ELEMENT factbook (record*)>
<!ELEMENT record (country, geography, people, economy, cities, provinces, ...)>
<!ELEMENT geography (location, geographic_coordinates, area, land_boundaries, border_countries, ...)>
<!ELEMENT people (population, population_growth_rate, ethnic_groups, religions, languages, ...)>
<!ELEMENT government (government_type, capital, ...)>
<!ELEMENT economy (gdp, gdp_agri, inflation, ...)>
<!ELEMENT provinces (province*)>
<!ELEMENT province (name, area?, population?, city*)>
<!ELEMENT cities (city*)>
<!ELEMENT city (name, longtitude?, latitude?)>
<!ELEMENT border_countries (border*)>
<!ELEMENT religions (religion*)>
<!ELEMENT ethnic_groups (ethnic_group*)>
```

Figure 5.25: Part of the Mondial database DTD transformed to CIA Factbook data set design

Chapter 6

A Principled Language Model for Ranking the Results of Keyword Search Over Entities

6.1 Background

In this chapter, we formulate a rigorous model for determining the desirable ranking of answers to a keyword query over a set of entities, based on reasonable and widely accepted principles of information retrieval. Our contributions in this chapter:

- We propose a new formal model for ranking keyword query answers over a set of entities, called the *S*tructured Language Model (SLM). SLM adapts the IR community's language model framework to suit a more structured setting, using two widely accepted principles from statistics and information retrieval. SLA captures the unique characteristics of the structured data in a database of entities (as opposed to pure text documents and text-centric data), leveraging the database's structural information to provide more effective ranking of answers.

- We prove that SLM determines the optimal ranking of answers for every query and data set, given the information made available to it. To the best of our knowledge, no previous model for database keyword queries guarantees such a property. Our proof is based on the probability ranking principle, a well known and widely accepted principle in information retrieval [109]. We also analyze previous formal models for database keyword queries and show that they violate the probability ranking principle.

- We provide the results of extensive experiments with SLM, using two real databases, a real query log, and queries from a user study. Our results show that SLM's ranking quality exceeds that of previous approaches to keyword queries for structured data.

In the reminder of the chapter, Section 6.2 presents basic definitions and Section 6.3 explores the shortcomings of current formal models for keyword search over a set of entities. Section 6.4 defines SLM and proves its optimality. Section 6.5 provides principled methods to estimate SLM parameters in the absence of training data. Section 6.6 presents experimental results.

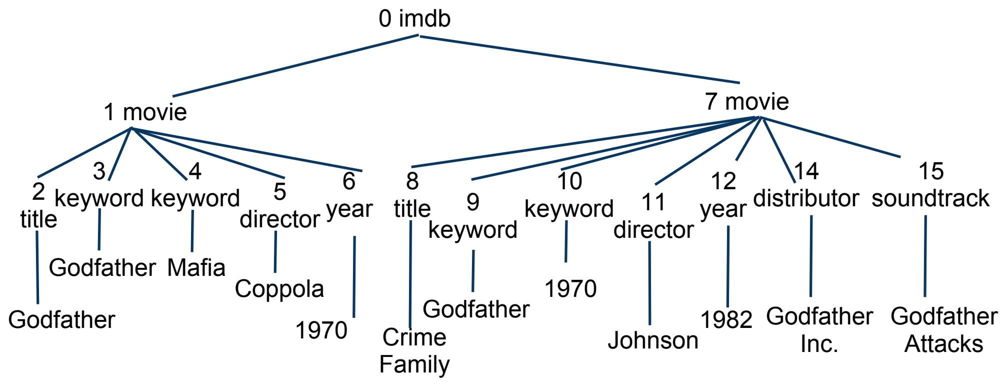


Figure 6.1: IMDB fragment

Title	Director	Keyword	Keyword	Year	···	Mapping	Relevance (R)
Godfather I	Coppola	Godfather	Mafia	1970	···	{(Godfather,**title**), (1970, **year**)}	1
Godfather I	Coppola	Godfather	Mafia	1970	···	{(Godfather, **keyword**), (1970, **year**)}	0
Crime Family	Johnson	Godfather	1970	1982	···	{(Godfather, **keyword**), (1970, **keyword**)}	0
1970	Smith	Godfather	Mafia	1991	···	{(Godfather, **keyword**), (1970, **title**)}	0
Goodfellas	Scorsese	Godfather	Mafia	1995	···	{(Godfather, **keyword**)}	0
Hanna	Allen	Family	New York	1970	···	{(1970, **year**)}	0

Table 6.1: Relevance information for Q_2 over a fragment of IMDB

6.2 Definitions

As discussed in the previous chapter, a keyword query approach that relies on database design details will rank answers differently for different designs with the same content. Thus a principled formal model that ranks answers well for all databases should not consider the details of the database design, or the data format (XML, relational, etc.). This means that our assumptions about the data model need to be so basic that any data model or database design will satisfy them.

By definition, a database has structure, so there must be at least one way to divide it into queriable units – the potential answers to queries – and the descriptions of those units. Albeit with loss of nuances, it should be possible to map any database design under any data model to a simple entity-relationship diagram containing entity sets, relationships, and attributes. We focus on building a principled model that can appropriately rank the entities in a particular entity set that match a keyword query. For example, in Figure 6.1, each subtree with a root labeled *movie* can be viewed as an entity. Given a keyword query about movies, the task is to rank the potential answers.

Since one database designer's relationship is another designer's entity, we assume that all the "interesting" relations, i.e., the ones likely to be the targets of keyword queries, have already been promoted to be entity sets in the database's entity-relationship diagram. The entity sets and their attributes are the queriable units and their descriptions, respectively. We do not consider more nuanced structural information that might be specific to a particular data model, such as how an entity is stored in an XML file or in a set of normalized relational tables. Searches that match multiple entity sets are also beyond our scope. In spite of these restrictions, many important keyword query tasks over structured databases, such as entity search [116, 49] and keyword search over XML collections [78, 33], fit this model.

The entity set being queried has an associated set of attributes, each of which represents an underlying domain, such as titles of movies or names of companies. We assume that different attributes represent different concepts. Each entity in the entity set has zero or more values for each of the set's attributes; we refer to these as the entity's *attribute values*. Each attribute value is a nonempty bag of terms, containing no stop words.

For instance, *Godfather* and *Mafia* are attribute values of the movie entity shown in the subtree rooted at node 1 in Figure 6.1. Node 2 says that *title* is an attribute of *Godfather*. *Godfather* and *Mafia* in the entity rooted at node 1 are both attribute values of the attribute *keyword*.

Two attribute values represent the same piece of information if and only if they have the same content and the same attribute. The entities rooted at nodes 1 and 7 in Figure 6.1 both have an attribute value *Godfather*. However, *Godfather* is the *title* of the entity at node 1 and a *keyword* of the other movie. Thus, they represent different information.

To simplify notation, if term w occurs in an attribute value A of attribute T belonging to entity E, we write $w \in A$, $A \in T$, $A \in E$, $T \in E$, and $w \in T$. $\mathcal{T}$ is the set of all attributes in the database.

A *keyword query* is a set $Q = \{q_1 \cdots q_{|Q|}\}$ of terms, where $|Q|$ is the number of terms in Q. Many times in keyword search the desired answers may not *exactly* contain the terms in the query. For instance, the entity under node 1 in Figure 6.1 does not contain any exact match for the term in query *gangster movie*, but it is a relevant answer to this query. Keyword query systems use a variety of methods to find approximate matches to the query terms [96]. We explain such methods in detail in Section 6.4. We consider an entity E to be an *answer* to Q iff some attribute value $A \in E$ contains exact or approximate match(es) for at least one term $q \in Q$.

The difficulties of answering a keyword query in this setting are two-fold. First, unlike queries in languages like SQL, users do not specify the desired schema element(s) for each term in their queries. For instance, Q_1: *Godfather* does not specify whether the user wants movies whose *title* is *Godfather* or movies distributed by the *Godfather* company. Thus, the query answering system must find the intended schema elements for the terms in the query. Second, keyword queries are generally underspecified, i.e., users do not give enough information to single out exactly their desired entities [96]. For instance, perhaps the user who submitted Q_1 wants to see information about movies whose *title* contains *Godfather*. Roughly *fifty* movies in IMDB satisfy this condition, but the user is most likely interested in at most a handful of them. Hence, the query answering system must find the most desired entities among all such entities, and rank them highest.

In the remainder of the chapter, DB, E, A, T, Q, q, D always refer to a database, entity in the database, attribute value of E, attribute of E and A, keyword query, term in Q, and document, respectively.

6.3 Shortcomings of Current Models

The *language model* for document retrieval ranks each document according to the probability that it *generates* the user's query [157]. That is, assuming that the query terms are conditionally independent, the language model ranks D based on:

$$P(Q \mid \theta_D) = \prod_{q \in Q} P(q \mid \theta_D), \tag{6.1}$$

where θ_D is the probability distribution of terms in document D, also known as the *language model* for D. The language model is more effective than other formal models for document retrieval when there is little or no training data available, and provides comparable results when training data is at hand [157]. Due to its solid foundation, it has been extensively used in document retrieval [157]. It has been shown that document retrieval methods based on the language model comply with the probability ranking principle [84], which we discuss later.

Researchers have adapted the language model to apply to keyword queries over documents with multiple fields and databases [104, 102, 141, 108]. A straightforward extension of the language model for entity E is to model E as a document and apply formula (6.1) directly [108]. However, this approach ignores the structural information in the entity and does not model those aspects of the information.

Since each entity consists of mutually exclusive and collectively exhaustive attribute values, some models compute $P(Q \mid \theta_E)$ as [104, 102, 141]:

$$P(Q \mid \theta_E) = \prod_{q \in Q} \sum_{A \in E} P(A \mid E) P(q \mid \theta_A), \tag{6.2}$$

where θ_A is the language model of the attribute value A and $P(A \mid E)$ is the prior probability of attribute value A. These methods compute the model for θ_A quite similarly to the approach used for computing θ_D in formula (6.1).

The model in formula(6.2) has a clear justification, but has some serious issues. The first challenging and important issue is how to set the value for $P(A \mid E)$, which signifies the desirability of attribute value A to users as a match for a query term. Some methods set these values manually based on properties of attribute value A, such as its length [104, 102]. However, it is very hard to use such a manual and ad hoc approach in practice. We could use training data to learn the values of the prior probabilities of the attribute values, but good training data is not usually available. In these situations, we would prefer to have a method that does not require manual tuning or training data.

We argue that the degree of desirability of an attribute also depends upon the particular query Q. For instance, attribute *title* in the entity rooted at node 1 in Figure 6.1 may be the desired attribute for query Q_1: *Godfather*, but it may not be the desired attribute for query Q_2: *Sony*. More formally, we may set the value of $P(A_1 \mid E)$ lower than the value of $P(A_2 \mid E)$ if the attribute value A_1 is less "interesting" than the attribute value A_2 for Q. Empirical results for collections of documents with multiple fields also show that ranking is more effective if we change the values for

$P(A \mid E)$ for each query [141].

Some researchers address this issue by choosing the values for $P(A \mid E)$ that maximize the value of $P(Q \mid \theta_E)$ in formula (6.2) [141]. It is very hard to rigorously justify this method, as according to the model in formula (6.2), the value of $P(A \mid E)$ should not depend on query Q. Moreover, the user's desired attribute to match for a query does not depend on any particular entity; for example, if the user's desired attribute for Q_1 is *title*, it will be *title* for all entities in IMDB. This method, however, sets the value of $P(A \mid E)$ to be larger for attributes with larger values for $P(q \mid \theta_A)$. Our empirical results with real world data sets show that this method fails to provide effective ranking for queries over entity sets, for this reason.

In another effort to address this issue, some methods replace $P(A \mid E)$ in formula (6.2) with the posterior probability distribution of q over $T(A)$ in the database, i.e., $P(T(A) \mid q)$, where $T(A)$ denotes the attribute of A [78, 33, 34]:

$$P(Q \mid \theta_E) = \prod_{q \in Q} \sum_{A \in DB} P(T(A) \mid q) P(q \mid \theta_A), \tag{6.3}$$

These methods usually compute the value of $P(T(A) \mid q)$ using Bayes' rule, as:

$$P(T(A) \mid q) = \frac{P(q \mid T(A)) P(T(A))}{P(q)}. \tag{6.4}$$

$P(T(A))$ represents our prior knowledge about the importance of $T(A)$, which is either assumed to have the same value across different values of $T(A)$ in the database [78, 34, 33], or is learned from training data. Some methods ignore $P(q)$ for ranking because it is the same for all attributes in the database [33, 34], and some methods set it to $\sum_{T \in DB} P(q \mid T)P(T)$ [78]. Informally, these methods assume that *the best attribute to match to a query term is the attribute where that term appears most often in the database*. In other words, the same probability distribution defines the desirability of an attribute as a match for a query term, and the distribution of that term across the attributes in the database [78, 116, 34, 33]. This heuristic has also been used in some keyword query approaches that are not based on the query generation framework [6]. We call this the *posterior distribution* heuristic.

As opposed to formula (6.2), the model in formula (6.3) is not a principled and logical derivation of the original language model in formula (6.1), and there is no reasonable justification for why formula (6.3) will deliver the desired ranking for keyword queries. More importantly, it is well known in document retrieval that the probability distribution of query terms is very different form the probability distribution of terms in the document collection [96]. There is no reason to believe otherwise for keyword queries over databases. Hence, it seems unlikely that the posterior distribution heuristic will find the desired attributes for query terms in a general setting, or that this heuristic will deliver effective rankings in any particular setting.

For instance, using all information in the IMDB data set and the computation method in [78], the value of formula

6.4 of Q_1: *Godfather* over attribute *title* is 0.003, but this value for the same query over attributes *trivia* (which contains facts about movies), *alias* (that contains aliases for the characters in movies), *keyword*, and *quote* (which contains quotes from movies) is 0.017, 0.014, 0.007, and 0.004, respectively.. Hence for Q_1 over IMDB, these models rank movies whose *trivia*, *alias*, *keyword*, or *quote* contains *Godfather* higher than movies whose *title* includes *Godfather*, given that all other conditions are the same. Intuitively, this is the wrong ranking.

Since each attribute in a database normally represents a different concept, the meaning of a term in a database depends on its attribute. For instance, in Figure 6.1, term *Godfather* under node 9 represents the subject of a movie, but the same term under node 14 represents the name of a company. It is very unlikely that these attributes both are the desired attributes for query Q_1. In general, we can argue that only a small subset of attributes in a database that represent the same or very similar concepts are the desired attributes for a query term over a database. However, formulas (6.2) and (6.3) assume that to some extent, any attribute may be the desired attribute for a query term. This assumption may be reasonable for documents with multiple fields, but is likely to produce undesirable rankings in a database context. For instance, these models may rank the entity rooted at node 7 higher than the entity rooted at node 1 in Figure 6.1 for Q_1, as the entity rooted at node 7 contains more attribute values that match term *Godfather*.

Query tagging or annotation is closely related to keyword search over databases [86, 116]. Given a keyword query, a query tagger annotates query terms with attributes and ranks these annotations. Such annotations help in finding the answers to these queries from the structured data sources. For instance, given Q_1 over Figure 6.1, a query tagger may output *movie.keyword: Godfather*. Researchers rely partially or in whole on training data to find the best annotation(s) for a keyword query [86, 116]. Since such supervised learning techniques do not scale to large data sets, some researchers also use the properties of the data set and queries to find and rank annotations. For instance, one can rank annotations for a query term q by the posterior distribution heuristic, i.e., $P(q \mid T)P(T)$ [116].

Overall, the previous work on language models for keyword queries over structured data was a step in the right direction. However, its shortcomings suggest that the language model needs significant additional adaptation to be suitable for use with databases of entities. More specifically, these approaches suffer from two important shortcomings. First, they do not rigorously justify their models and/or their parameter estimation methods. In this chapter, we prove that these models *do not* generally lead to optimal rankings for keyword search over databases. Second, they also assume that all terms in an entity represent the same or similar concepts, which the *Godfather* example shows to be unreasonable.

Researchers have worked to create effective ranking methods for keyword queries over structured and semi-structured information [9, 90, 6, 78, 102, 34, 33, 44]. Many ranking methods are not based on any formal framework, which limits their applicability to a wide variety of queries and databases, as discussed in Section 5.1 [9, 90, 6, 44]. Some of these systems use information drawn from sources other than the database or query (e.g., the importance of the web pages used to extract the database records) to rank query answers [40]. Others use clues such as the order of terms in the query to improve their effectiveness [89]. Our approach is orthogonal to these.

Researchers have developed a formal probabilistic model to rank the answers to underspecified SQL queries over a relational database [17]. Although their work targets SQL, their ideas are relevant for keyword queries. However, their approach uses query logs as training data to compute the parameters of their model, which is not always a realistic assumption in our setting. Generally, semi-structured and structured data sets are intended for queries that are too difficult to answer well when only unstructured information is available; otherwise, it would not be worth the effort of introducing structure. Because query distributions have such a long tail, it is inherently impossible for training data to include most of the long tail queries that the system will see at run time. Hence, as for queries over unstructured sets of documents, we should use principled parameter estimation methods that are effective given the data set alone, so that the system will ranks answers well for queries that were not included in the training data. We must also address the problem of query term disambiguation, which was not an issue for [17], which focused on SQL queries.

6.4 Structured Language Model

6.4.1 Probabilistic Ranking in Databases

The probability ranking principle (PRP) is a widely accepted principle for ranking documents [109, 139]. PRP says that a retrieval system is optimal, i.e., delivers the highest utility for its users, when it ranks documents in decreasing order of their probability of relevance to the query. The classical version of PRP assumes that the probabilities are computed correctly and there are no dependencies between documents [109]. There are other versions of PRP that do not make these assumptions [18, 139], but the classical version of PRP is still widely accepted and useful in developing formal probabilistic models for query answering systems [96]. We use the classical version of PRP in this chapter, and our approach can be extended for other versions [18, 139].

PRP computes the probability of relevance using a *hypothetical* complete set of relevance information for a query over the documents in the data set. Each entry in the relevance information for a given query consists of a document D and the judgment R of whether D is relevant to a particular query Q. R is 0 for non-relevant documents and 1 for relevant documents. Hence, the probability that D is relevant to Q is $P(R = 1 \mid D, Q)$.

We adapt PRP for ranking keyword queries over databases. In order to adapt PRP for keyword search over databases, we should also consider the desired concepts for query terms in the model. Generally, each attribute in a database represents a different concept. For instance, attributes *title*, *distributor*, and *soundtrack* each represent a separate concept in IMDB. For reasons such as usability or performance, database designers may use multiple attributes to describe the same aspect of an entity. For instance, in IMDB, attribute *keyword* contains short descriptions of the story, and the subject of the movie is stored in the *plot* attribute. Similar terms in these attributes represent the same concepts. Hence, we assume that it is possible for more than one attribute in the database to match the desired concept behind a query term. One may identify a group of attributes that represent the same concept manually or automatically [159]. Automatic discovery of these groups of attributes is beyond the scope of this thesis.

Let $\mathcal{P}(\mathcal{T})$ be the set of all non-empty subsets of set $\mathcal{T}$. A *mapping* M for Q is a function $Q \mapsto \mathcal{P}(\mathcal{T})$ that assigns a non-empty set of attributes $M(q)$ to each term $q \in Q$. We write each member of M as $\mu = (q, M(q))$. A mapping for query Q captures the desired concepts behind the terms in Q. As discussed in the previous paragraph, we consider only subsets of the attributes that convey similar concepts. $|M(q)|$ is the number of attributes assigned to q in M. Table 6.1 gives example mappings for Q_2: *Godfather 1970*, over part of IMDB. An entity E *satisfies* M iff for each assignment $(q, M(q))$ of M, E has attribute values with attributes $M(q)$ that exactly or approximately contain q.

Now consider a database setting, where $L(DB)$ denotes the hypothetical perfect complete set of relevance information for all possible keyword queries and answers over DB. Each entry in $L(DB)$ includes a query Q, entity E, relevance value R, and a mapping M such that E satisfies M for Q.

Table 6.1 shows the relevance information for Q_2 over part of IMDB. E appears in $L(DB)$ once for each of its mappings for Q, as in the first two rows of Table 6.1. Given E and Q, R can be different for each choice of M.

To compute relevance probabilities for the database, we use the *odds of relevance* of E and M [109]:

$$O(R \mid E, M) = \frac{P(R = 1 \mid E, M)}{P(R = 0 \mid E, M)}. \tag{6.5}$$

For brevity, we abbreviate $R = 1$ to R and $R = 0$ to $\bar{R}$.

Relevance probabilities ideally should be computed over $L(DB)$, but complete relevance information, or even sufficient samples, is rarely available. By definition, even query logs can only contain a small fraction of the long tail and difficult queries. Hence, the key challenge is to leverage the properties of Q and the database in a principled way, to deliver reasonable and effective relevance estimates.

6.4.2 Optimal Language Model

In this section, we build a language model based on PRP. We revise formula (6.5) to:

$$\frac{P(R \mid M, E)}{P(\bar{R} \mid M, E)} = \frac{P(M, E \mid R)P(R)}{P(M, E \mid \bar{R})P(\bar{R})}. \tag{6.6}$$

Applying Bayes' rule to the numerator and denominator, we have:

$$\frac{P(M, E \mid R)P(R)}{P(M, E \mid \bar{R})P(\bar{R})} \approx \frac{P(M \mid E, R)P(E \mid R)P(R)}{P(M \mid E, \bar{R})P(E \mid \bar{R})P(\bar{R})} \tag{6.7}$$

It is a widely accepted principle in information retrieval that only a very small fraction of the documents in a collection are relevant to Q [29, 96]. Hence, it is a reasonable and common practice to estimate the statistics for non-relevant documents using the set of *all* documents [29, 96]. The relative number of relevant answers is even smaller for keyword queries over large databases [91]. Thus it is reasonable to assume that almost all entities are not relevant to a given Q, even if they satisfy the desired mapping(s) for Q. Hence, we approximate the value of $P(M \mid E, \bar{R})$ as:

$$P(M \mid E, \bar{R}) \approx P(M \mid \bar{R}).$$

Using formula (6.4.2) for the denominator of the right hand side in formula (6.7), we have:

$$\frac{P(M \mid E, R)P(E \mid R)P(R)}{P(M \mid E, \bar{R})P(E \mid \bar{R})P(\bar{R})} \approx \frac{P(M \mid E, R)P(E \mid R)P(R)}{P(M \mid \bar{R})P(E \mid \bar{R})P(\bar{R})}$$

According to Bayes' Rule, we have:

$$\frac{P(E \mid R)P(R)}{P(E \mid \bar{R})P(\bar{R})} = \frac{P(R \mid E)}{P(\bar{R} \mid E)} = O(R \mid E),$$

which does not depend on Q and represents prior knowledge about the importance of E, similar to PageRank scores. Computing such a value is out of the scope of this thesis. Hence, we rank the answers using:

$$\frac{P(M \mid E, R)}{P(M \mid \bar{R})}. \tag{6.8}$$

We make the following assumptions in order to compute the probabilities in formula (6.8). Current keyword query approaches generally assume that query terms and their mappings are independent [78, 33, 116, 9, 90]. Query annotation methods make similar assumptions [116]. We do also, as otherwise computing the model will be quite complex. Given the independence assumptions, we rewrite formula (6.8) as:

$$\frac{P(M \mid E, R)}{P(M \mid \bar{R})} = \prod_{q \in Q} \frac{P(q, M(q) \mid E, R)}{P(q, M(q) \mid \bar{R})}.$$

Since $M(q)$ is a set of attributes, we have:

$$\prod_{q \in Q} \frac{P(q, M(q) \mid E, R)}{P(q, M(q) \mid \bar{R})} = \prod_{q \in Q} \frac{\sum_{T \in M(q)} P(q, T \mid E, R)}{\sum_{T \in M(q)} P(q, T \mid \bar{R})}.$$

Applying Bayes' Rule to the numerator and denominator of formula (6.4.2), it becomes:

$$\prod_{q \in Q} \frac{\sum_{T \in M(q)} P(q \mid T, E, R) P(T \mid E, R)}{\sum_{T \in M(q)} P(q \mid T, \bar{R}) P(T \mid \bar{R})}. \tag{6.9}$$

We call this formula the *Structured Language Model* (SLM).

At first glance, $P(q \mid T, E, R)$ does not seem to represent a language model. However, researchers have shown that formula (6.1) in fact *generates* all queries that users use to retrieve a document D [84]. More formally, they have argued that formula (6.1) is equivalent to $P(Q \mid D, R)$. Similar to keyword search over a document collection, users like to retrieve relevant answers when they perform keyword search over a database. Hence, we argue that $P(q \mid T, E, R)$ represents the model that generates all query terms whose concepts are T and are used to fetch E in a database setting. We call $P(q \mid T, E, R)$ the *attribute value language model.*

Using a similar argument, the probability value $P(q \mid T, \bar{R})$ in the SLM formula models the query terms that are *less* likely to be submitted by a user to retrieve entities, given that the attribute that the user desires to match is T. We call this part of the SLM formula the *non-relevant attribute language model.*

If we assume that all attributes in the database convey the same concept, the denominator of formula (6.9) becomes:

$$\sum_{T \in \mathcal{T}} P(q, T \mid \bar{R}) = P(q \mid \bar{R}),$$

which can be ignored for ranking that uses the SLM formula, as it has the same value for all entities. The value $P(T \mid E, R)$ shows our prior knowledge about the importance of attribute values of attribute T in entity E, i.e., how often users use terms within these attribute values to retrieve entity E. Thus, the SLM formula becomes probabilistically similar to the mixture model in formula (6.2).

This argument shows why we cannot use the mixture model in formula (6.2) to rank database entities. As opposed to collections of documents with multiple fields, different attributes in a database may correspond to different concepts. Hence, according to PRP, we cannot ignore the denominator in formula (6.9) when ranking over databases.

We can rewrite the denominator of the SLM formula as:

$$\sum_{T \in M(q)} P(q \mid T, \bar{R}) P(T \mid \bar{R}) = \sum_{T \in M(q)} P(T \mid q, \bar{R}).$$

Given a query, $\sum_{T \in M(q)} P(T \mid q, \bar{R})$ has different values for different $M(q)$, i.e., different sets of attributes in the database. The smaller its value is, the larger the value of formula (6.9) will be. Hence, the derivation of the SLM formula from PRP justifies that the degree of the desirability of attribute(s) depends also on the input query. SLM avoids the ranking problems that arise if the desired attribute depends only on E (e.g., [102, 141]), as noted in

Section 6.3. SLM also captures this relationship in a principled and clearly justified manner, unlike previous models (e.g., [78, 33]).

Since the probabilities in the SLM model contain the R and $\bar{R}$ variables, they have a clear interpretation. This is very important for estimating these probabilities in the absence of training data, which is the focus of Section 6.5.

For a particular entity, a query may have multiple mappings, i.e., a query term may match several different attributes, and the entity may have a different value for the SLM formula for each of these mappings. We should choose one particular mapping from among these mappings to compute the SLM formula for a query. The desired mapping for a query and an entity is the mapping that delivers the maximum value for the SLM formula.

6.5 Estimating SLM Parameters

We can compute the probabilities in the SLM formula using training data. However, such training data is not normally available. In this section, we propose methods to estimate the parameters of SLM in the absence of training data. Using training data to train SLM through methods such as (pseudo-)relevance feedback is an interesting topic for future work.

Without training data, we cannot compute the values of the query-independent terms in the SLM formula, i.e., $P(T \mid E, R)$ and $P(T \mid \bar{R})$. According to the principle of maximum entropy, we should assume that the values of these probabilities are equal for all attributes and entities in the database. We can take these values out of the summations in both the nominator and denominator. Hence, we can ignore them for ranking against the same query, as they change the estimation bias in the same way for all entities and attributes, given the query.

6.5.1 Attribute Value Language Model

As discussed in Section 6.4.2, probability $P(q \mid T, E, R)$ represents the language model for attribute value A that belongs to attribute T and entity E. Since each attribute value is a bag of terms, we can adapt current methods to compute the language models for text documents in document collections to estimate the attribute value language model, in the absence of relevant answers [157]. For brevity, we write $P(q \mid A)$ for $P(q \mid T, E, R)$.

The maximum likelihood (ML) method is an intuitive way to calculate $P(q \mid A)$. That is, given q and attribute value A, let $n(q, A)$ be the number of times q appears in attribute value A. Also, let $n(A)$ be the total number of occurrences of terms in attribute A. The maximum likelihood estimate for $P(q \mid A)$ is:

$$P_{ml}(q \mid A) = \frac{n(q, A)}{n(A)}. \tag{6.10}$$

For instance, given term *crime* and the attribute value under node 8, A_8 in Figure 6.1, the value of $P_{ml}(crime \mid A_8)$

is $\frac{1}{2}$, as the attribute value has two occurrences of terms.

Consider the query Q_5 : *Godfather sequel*. No movie in the original IMDB contains both terms, so $P(q \mid A) = 0$ for all A in the database, for one or both choices of q. The maximum likelihood estimate gives every answer the same score, 0, which is undesirable. Moreover, database entities have relatively few attributes and relatively few terms in each attribute, compared to the totality of information that one could provide for that entity and related entities. Perhaps we could estimate $P(q \mid A)$ better if we had all this information. These issues also appear in the applications of generative models in speech recognition and document retrieval [64, 157].

Researchers use *smoothing techniques* to address these issues in other domains. For instance, document retrieval research uses model θ_D to estimate the probability $P(q \mid D)$. θ_D includes D and other documents similar to D. For every query term q, $P(q \mid \theta_D) > 0$. The effectiveness of a document language model mainly depends on its smoothing method [157]. Researchers often compute θ_D using weighted combinations of D and the collection of all documents. Although the collection of all documents does not represent the set of documents that are similar to D, it delivers acceptable results [157].

A straightforward smoothing method to estimate θ_A is to combine the language model of the attributes in θ_A with the language model of all attributes of type T in DB. One popular example of these types of smoothing methods is *Jelinek-Mercer* smoothing, which is:

$$P(q \mid \theta_A) = (1 - \lambda)P_{ml}(q \mid A) + \lambda P(q \mid T), \tag{6.11}$$

where $0 \leq \lambda \leq 1$ determines the weight given to attributes of type $M(q)$ in the database. One may use other techniques to smooth the attribute value language model [160]. The issue of smoothing the attribute value language model is beyond the scope of this thesis. In our experiments, we use the same smoothing method for all models.

6.5.2 Non-Relevant Attribute Language Model

As discussed in Section 6.4.2, the value of $P(q|T, \bar{R}$ is larger for the terms that are less likely to be used in queries to access entities in the database, given that the user's intended attribute is T. In the absence of training data, we use the information about the terms in the database to estimate this probability. If users' desired attribute is T, we argue that they are less likely to use terms that appear mostly in other attributes in the database. Let $DB \setminus T$ denote all attributes in the database DB except for T. We have:

$$P(q \mid T, \bar{R}) \propto P(q \mid DB \setminus T). \tag{6.12}$$

We can view $DB \setminus T$ as a virtual document created by appending all terms in attributes such as $T' \in DB, T' \neq T$. What about the terms that appear in T in the database? We can make three different arguments about the terms that appear in T.

In absence of any information about the user's search behavior, according to the principle of maximum entropy, one should assume that all terms in T are equally likely to be used by users in their queries, provided that their desired attribute is T. Let V be the number of distinct terms in T. We assign equal probability of $\frac{1}{V}$ to all distinct terms in T and compute the value of $P(q \mid T, \bar{R})$ as:

$$P(q \mid T, \bar{R}) = \beta \frac{1}{V} + (1 - \beta) P(q \mid DB \setminus T), \tag{6.13}$$

where we have $0 \leq \beta < 1$, and the value of β may be set empirically. We refer to this version of SLM as SLM-UF.

If users perform ad-hoc keyword search, an extremely small portion of possible answers will be relevant to their information needs. For instance, if users are seeking information about a particular movie or specific actor in IMDB, at most a handful of entities in IMDB will be relevant to their queries. Such information seeking tasks appear in contexts such as Web search. In this context, if a term appears relatively more frequently in T, it is less likely that users will use this term in their queries. In other words, users are more likely to use rare terms in T to access their desired entities. The example of Q_1 : *Godfather* over IMDB data, discussed in Section 6.3, depicts such behavior: since users are looking for particular movies, they choose terms that are quite rare in the *title* attribute in IMDB. Our experiments in Section 6.6, which use queries over structured data taken from real Web search engine query logs, confirm this hypothesis. More formally:

$$P(q \mid T, \bar{R}) \propto P(q \mid T). \tag{6.14}$$

Using formulas (6.12) and (6.14), we have:

$$P(q \mid T, \bar{R}) = \beta(q \mid T) + (1 - \beta) P(q \mid DB \setminus T), \tag{6.15}$$

where $0 \leq \beta < 1$, and the value of β may be set empirically. We refer to this version of SLM as SLM-MF.

One may also argue that users are less likely to submit terms that are relatively less frequent in T, if their desired attribute is T. Such searching behavior occurs when users are performing a more exploratory search. In these settings, users do not have sufficient knowledge about their desired answers and may submit a relatively general description of their information need. The number of desired answers, at least in one interaction with the search system, is larger than for other types of information searching tasks. More formally:

$$P(q \mid T, \bar{R}) \propto 1 - P(q \mid T). \tag{6.16}$$

Using formulas (6.12) and (6.16), we have:

$$P(q \mid T, \bar{R}) = \beta(1 - P(q \mid T)) + (1 - \beta)P(q \mid DB \setminus T), \tag{6.17}$$

where $0 \leq \beta < 1$, and the value of β may be set empirically. We refer to this version of SLM as SLM-LF.

The estimation method in formula (6.17) has some similarities to the posterior distribution heuristic discussed in Section 6.3. Given that the user's desired attribute is T, both methods give larger weights to the terms that appear more frequently in attribute T. However, formula (6.17) has a more reasonable justification than the posterior distribution heuristic, and due to that justification we can predict in which settings such a heuristic is more effective. As opposed to the posterior distribution heuristic, this also justifies the need for smoothing the value of $(1 - P(q \mid T))$.

6.6 Experiments

	IMDB	Discogs
Size (GB)	29.0	9.64
Number of Entities	4,418,081	1,871,614
Number of Entity Sets	2	3
Number of Attributes	77	24

Table 6.2: Data Set Properties

6.6.1 Experiment Setting

Data Set and Query Workload: We evaluated the effectiveness of SLM over two real world databases, a standard query workload and a real world query log. We used the IMDB data set from the INEX 2010 Data-Centric Track [142]. We also used the Discogs (*www.discogs.com*) data set, which contains information about singers and music albums. Table 7.1 gives properties of the two data sets. The IMDB data set has two entity sets: **person**, which gives information about people, and **movie**, which contains information about movies and TV shows. The Discogs data set has three entity sets: **artist**, which contains information about singers, **label**, which contains information about companies that produce and distribute music albums, and **release**, which provides information about albums. We divide each data set into its separate entity sets, and evaluate them separately.

In the INEX workshop, participants were asked to submit topics that are different from typical Web queries [142]. In order to evaluate our methods over Web-like queries, we performed an additional large scale experiment. We parsed the AOL query log [106] and picked out the queries that caused the user to click through to IMDB. We considered only the queries that appeared at the end of each user session, as they indicate that the user has probably satisfied her information need in IMDB. This way, we collected about 15,000 queries. We used the IMDB data set from the INEX Data-Centric Track and filtered out the movies after 2006, which was the last date of the queries in the AOL query log.

The AOL query log does not contain the exact URL of the IMDB page that was visited by users. However, it tells which position the user clicked in the ranked result returned by the AOL search engine. The average rank of the IMDB pages clicked by users in the AOL query log was 2.71. As search engine technology has advanced significantly since 2006, we hypothesize that the ranking delivered by a popular search engine such as Bing (*www.bing.com*) today must be at least as good as the ranking delivered by the AOL search engine in 2006. Thus for each query Q, we retrieve the URLs of the IMDB pages that are in the top n links returned by Bing, where n is the rank of the desired result for Q, according to the AOL query log. If no IMDB page is in the list, we fetch the first link of IMDB in the ranked list returned by Bing. As 50 queries is normally sufficient to evaluate the effectiveness of a ranking method over a collection [96], we randomly picked 50 queries for each entity set in IMDB from this query workload, and report the results as *IMDB-AOL* in the next section.

Both the INEX and AOL query workloads use the IMDB database, and we wanted to evaluate the effectiveness of our methods over more than one database. Thus we also collected queries from the AOL query log that target the Discogs web site, which contains information about songs, singers, and label companies. Since the Discogs database is publicly available, we performed a similar experiment as for IMDB, and randomly collected 50 queries from the AOL query log whose relevant answers are in the Discogs database, for each entity set in Discogs.

Ranking Methods: We implemented SLM with three different methods of estimating the non-relevant attribute language model, as explained in Section 6.5.2. We compared SLM to two other ranking methods that are based on the language model and do not require manual tuning or extensive training. These methods are PRMS [78] and Multi-Style [141], both of which are mentioned in Section 6.3. PRMS assumes that a query term q must be mapped to an attribute that contains a relatively large number of instances of q. In other words, it uses the posterior probability heuristic [78, 116, 34, 33]. PRMS also uses a ranking formula based on the language model to compute the relevance of the content of the mapped attributes to the query. PRMS requires tuning of the value of the smoothing factor for its language model formula. It has been shown that for keyword search over databases, PRMS is more effective than the language model for document retrieval (formula (6.1)) [78] and language models that determine the weights of attributes using sufficient training [104, 102, 78] PRMS and all versions of SLM use Jelinek-Mercer smoothing, and we have to set their smoothing parameter. We estimated these parameters using five randomly selected queries for each query workload.

The Multi-Style ranking method uses an expectation maximization algorithm to tune the parameters in formula (6.2). The authors of Multi-Style have used the Multi-Style ranking method with *open vocabulary smoothing* [141]. We tried Multi-Style both with and without open vocabulary smoothing (shown as MS-OVS and MS, respectively, in our results). Multi-Style has been shown to provide better ranking than other popular methods such as BM25F [111, 141]. It has also been shown empirically that Multi-Style's effectiveness is very close to the effectiveness of a BM25F method that uses the testing data set for training [111, 141]. The methods that use the long descriptions or structural versions of keyword queries that are provided by users, or manually tune most their parameters are not comparable to our method [142].

Effectiveness Metrics: We used the popular effectiveness metrics precision at k (Prec@K) and Mean Average Precision (MAP) to measure the quality of rankings delivered by ranking algorithms [96]. Prec@K measures the precision of the returned results up through position K in the returned ranked list. Let W denote the query workload that contains $|W|$ queries Q_i, $1 \leq i \leq |L|$. Given that Q_i has relevant answers $\{E_1, \ldots, E_{n_i}\}$ and $R_{i,k}$ is the list of ranked entities up to the kth entity E_k, average precision is defined as:

$$AvgP(Q_i) = \frac{1}{|n_i|} \sum_{k=1}^{n_i} Precision(R_{i,k}).$$

Then the MAP of query workload L is:

$$MAP(L) = \frac{1}{|L|} \sum_{i=1}^{|L|} AvgP(Q_i).$$

The greater the value of MAP is, the more effective the query results will be.

Experiment Configuration: We implemented the system on a Linux server with 24 GB of main memory. SLM took roughly the same time to rank answers as all other methods in this experiment except Multi-Style.

As mentioned in the description of ranking methods, Multi-Style uses an expectation maximization (EM) algorithm to estimate its smoothing parameters and its mixture weights in its scoring formula. While the EM algorithm provides convenience for parameter tuning, the algorithm itself requires multiple re-estimations of the parameters until it converges. Each iteration of the estimation also needs a separate computation for every field in a document. With a very large dataset such as IMDB, a document may contain anywhere from from a couple to more than a thousand fields. Thus the method runs extremely slowly. This is a huge disadvantage for Multi-Style, compared to other ranking methods. For our experiments, we imposed a maximum of 20 iterations of the EM algorithm, thereby sacrificing exactness for the sake of reasonable running time. Table 6.3 shows the average processing time in scoring a single candidate answer when using SLM and Multi-Style. We see that even with the cap on EM iterations, Multi-Style is still at least 5 times slower than SLM.

	INEX'10-movie	INEX'10-person	AOL-movie	AOL-person	Discogs-Label	Discogs-Release	Discogs-Artist
SLM	3.29	13.22	2.52	9.06	0.05	0.24	0.12
Multi-Style (20)	17.16	75.44	17.01	74.73	0.38	1.01	0.64

Table 6.3: Average ranking time per candidate answer, in milliseconds

6.6.2 Ranking Quality

	INEX'10-movie	INEX'10-person	AOL-movie	AOL-person	Discogs-Label	Discogs-Release	Discogs-Artist
PRMS	0.2921	0.1353	0.0622	0.5501	0.4138	0.6397	0.5594
MS-OVC	0.4580	0.2182	0.1632	0.6459	0.5283	0.7021	0.4827
MS	0.4524	0.2151	0.1730	0.6487	0.5140	0.7154	0.4828
SLM-MF	0.4508	0.4082	0.5394	0.7024	0.7478	0.7037	0.7180
SLM-LF	0.4772	0.4069	0.3983	0.7060	0.5547	0.6986	0.6577
SLM-UF	0.4598	0.4323	0.5958	0.7332	0.7585	0.7446	0.7394

Table 6.4: The values of Mean Average Precision for different ranking methods

All methods except Multi-Style have some tuning parameters. We used 5 randomly selected queries from the IMDB query set for parameter training over the IMDB data set, and 10 queries from the Discogs query set over the Discogs data set. The training is done separately for each entity set. Table 6.4 shows the MAP of PRMS, Multi-Style (MS), Multi-Style with open vocabulary smoothing (MS-OVC), SLM-MF, SLM-LF, and SLM-UF. All methods generally deliver a better ranking for Discogs than for IMDB. This is primarily because Discogs has a simpler structure, so there is less for a method to get wrong. Also, most of the queries target an attribute that is a unique identifier for the entity in question, so the rankers can easily identify the correct answers.

On the other hand, IMDB is a larger data set with a more complicated structure, and many possible mappings for a given query. The AOL query workload is similar to that of Discogs, in the sense that each query tends to target a unique identifier for its entity. We see that most rankers also perform well with this query workload. However, the INEX query set is more exploratory. Each term in the query is intended for different fields in the document. Hence, many rankers get stuck on a sub-optimal choice for the query term mappings.

Overall, SLM delivers the best MAP over most data sets. For example, consider the query *men last stand* from the AOL query log, which has the "X-Men: The Last Stand" movie as its only relevant answer. All versions of SLM successfully map each query term to the *movie/title* attribute. On the other hand, Multi-Style is stuck on weighting the most important field to be *movielinks/link*, as many of its top ranked documents contain terms in this attribute instead. PRMS places more importance on the field that contains the most query terms, which does not match the user's intention.

For Discogs, SLM outperforms other rankers. For example, consider the query *our last goodbye killing joke*. The correct mapping attribute of this query is *artist/name*, which is the artists' group name. Only SLM got the correct result. Other rankers tended to instead choose the *artist/group/name* attribute, which gives the name of the group the artist is associated with.

For INEX's movie set, SLM and Multi-Style deliver similar ranking quality. In fact, more queries from the INEX movie set are exploratory than for any other data set. For example, consider the query *avatar james francis cameron*, where all methods except PRMS perform well, with an average precision of around 0.9. The result from PRMS has an average precision of only 0.07. PRMS puts more weight on the attribute with the most frequent query term, */actor/name*, as the query *james* is more common in this attribute. However, the user intends other attributes for this query: director, writer and movie alias. Because many other queries in the INEX movie query set are exploratory in nature, SLM-LF is more effective there than other methods.

Interestingly, SLM-UF tends to perform better than other versions of SLM, except for the INEX movie set. Formula (6.13) in Section 6.5.2 tries to balance between having too many and too few occurrences of a particular term. For example, SLM-UF performs better than other SLM versions for query *mission impossible* in the AOL query set. In this case, the relevant entity has the query term mainly in the attributes *movie/title* and *movie/url*. SLM-LF chose a wrong mapping because of the high frequency for query terms in other attributes. Similarly, SLM-MF found that these query terms are rare in *keyword*, and so it mistakenly weighted the importance of this field above others. SLM-UF captures the importance of this query term in the right attribute, as it does not overestimate or underestimate the importance of term occurrences in a particular attribute. Overall, we observe that users' search behavior is heterogeneous, even for a single entity set.

Even though SLM delivers good ranking quality, it may perform badly if it cannot find a mapping that is close to the user's intention. Consider the query *may force be with you* from INEX 2010, which relates to the attribute *plot* of the movie. SLM is misled by the term *may* and picks *releasedate* as the best mapping, and this mapping score overwhelms the score of the mapping of other query terms, leading to a poor ranking.

Multi-Style also performs quite well over our data sets. The advantages of the language model help make Multi-Style's ranking method relatively flexible, so that it is less likely to misinterpret the user's search intention. For example, for the query *avatar james francis cameron* discussed previously, Multi-Style puts appropriate mixture weights on attributes such as *director/name*, *movie/title*, and *writer/name*, which are the most desirable attributes for this query. However, Multi-Style's running time is extremely slow, making it unfair to compare Multi-Style to other ranking methods. Interestingly, we do not see a significant improvement when using the open-vocabulary smoothing technique with Multi-Style in this experiment. This is because each document attribute uses a similar vocabulary set. Hence, the vocabulary entropy between structures is negligible and thus the smoothing value is very close to the default without smoothing.

PRSM, on the contrary, is shown to perform the worst in this experiment. As we have given many examples that there are many queries that are not satisfied by the posterior probability heuristic. It is, therefore, not too unexpected that PRSM does not perform as well as other methods. On a side note, one may notice an extremely low MAP of the results for PRSM on AOL-movie query workload. We suspected that the problem is caused by the bad training samples. This training shows the best smoothing parameter of PRSM to be 0.9 which is completely opposite to the usual very small parameter. We then perform a new training using another samples, and get the best smoothing parameter of 0.1. The MAP of the running of PRSM with a new parameter is 0.4082. However, with this special treatment, PRSM still could not surpass the effective quality of SLM.

Chapter 7

Difficult Queries Over Semi-structured Data

7.1 Background

In this chapter, we analyze the characteristics of difficult keyword queries over semi-structured data and propose a novel method to detect such queries. We take advantage of the structure of the data to gain insight about the degree of the difficulty of a query given the database. We implemented some of the most popular and representative algorithms for keyword search on databases and used them to evaluate our techniques on both the INEX and SemSearch benchmarks. The results show that we can predict the degree of difficulty of a query efficiently and effectively.

We make the following contributions:

- We introduce the problem of predicting the degree of the difficulty for queries over structured data, and analyze the reasons that keyword queries may be difficult to answer well.
- We propose the *structured robustness (SR)* score, which measures the difficulty of a query based on the differences between the rankings of the same query over the original and a noisy (corrupted) version of the same database, where the noise spans both the content and the structure of the result entities.
- We introduce efficient algorithms to compute the SR score. This is important because such a measure is only useful when it can be computed with a small overhead compared to the query execution cost.
- We present the results of extensive experiments using two standard data sets and query workloads, from INEX and SemSearch. Our results show that the SR score effectively predicts the ranking quality of representative ranking algorithms, and outperforms non-trivial baselines introduced in this chapter. Also, the time spent to compute the SR score is negligible compared to the query execution time.

In the remainder of this chapter, Section 7.2 presents basic definitions. Section 7.3 explains the ranking robustness principle and analyzes the properties of difficult queries over databases. Section 7.4 presents concrete methods to compute the SR score in semi-structured data. Section 7.5 describes the algorithms to compute the SR score. Section 7.6 contains the experimental results.

Portions of this work appeared in [20].

7.2 Data and Query Models

We model a database as a set of entity sets, each of which is a collection of entities. For instance, *movies* and *people* are two entity sets in IMDB. Figure 3.6 depicts a fragment of a data set, where each subtree whose root's label is *movie* represents an entity. Each entity E has a set of attribute values A_i, $1 \leq i \leq |E|$. Each attribute value is a bag of terms. Following current approaches for answering keyword queries over unstructured and semi-structured approaches, we ignore stop words that appear in attribute values, although this is not necessary for our methods. Every attribute value A belongs to an *attribute* T, written as $A \in T$. For instance, *Godfather* and *Mafia* are two attribute values in the movie entity shown in the subtree rooted at node 1 in Figure 3.6. Node 2 depicts the attribute of *Godfather*, which is *title*.

The above is an abstract data model that ignores the physical representation of data. That is, an entity could be stored in an XML file or a set of normalized relational tables. This model has been widely used in works on *entity search* [116, 49] and *data-centric XML retrieval* [142], and has the advantage that it can be easily mapped to both XML and relational data. Further, if a KQI method relies on the intricacies of the database design (e.g., deep syntactic nesting), it will not be robust and will have considerably different degrees of effectiveness over different databases [130]. Since our goal is to develop principled formal models that cover reasonably well all databases and data formats, we do not consider the intricacies of the database design or data format in our model of a database.

A *keyword query* is a list $Q = \{q_1 \cdots q_{|Q|}\}$ of terms, where $|Q|$ is the number of terms in Q. An entity E is an *answer* to Q iff at least one of its attribute values A contains a term q_i in Q, written $q_i \in A$. Given database DB and query Q, retrieval function $g(E, Q, DB)$ returns a real number that reflects the relevance of entity $E \in DB$ to Q. Given database DB and query Q, a keyword search system returns a ranked list $L(Q, g, DB)$ of entities in DB, where entities E are listed in decreasing order of the value of $g(E, Q, DB)$.

7.3 Ranking Robustness Principle for Databases

In this section we present the *Ranking Robustness Principle*, which postulates that there is a negative correlation between the difficulty of a query and its ranking robustness in the presence of noise in the data. Section 7.3.1 discusses how this principle has been applied to unstructured text data. Section 7.3.2 presents the factors that make a keyword query on structured data difficult, which explain why we cannot apply the techniques developed for unstructured data. The latter observation is also supported by our experiments in Section 7.6.2 on the *Unstructured Robustness Method* [161], which is a direct adaptation of the Ranking Robustness Principle for unstructured data.

7.3.1 Background: Unstructured Data

Mittendorf has shown that if a text retrieval method effectively ranks the answers to a keyword query in a collection of text documents, it will also perform well for that query over the version of the collection that contains some errors, such as repeated terms [99]. In other words, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the original and the corrupted versions of the collection. We call this observation the *Ranking Robustness Principle*. Zhou and Croft [161] have applied this principle to predict the degree of the difficulty of a query over free text documents. They compute the similarity between the rankings of the query over the original and the artificially corrupted versions of a collection to predict the difficulty of the query over the collection. They deem a query to be more difficult if its rankings over the original and the corrupted versions of the data are less similar. They have empirically shown their claim to be valid. They have also shown that this approach is generally more effective than using methods based on the similarities of probability distributions, which we reviewed in Chapter 2. This result is especially important for ranking over databases. As we explained in Chapter 2, it is generally hard to define an effective and domain independent categorization function for entities in a database. Hence, we can use the Ranking Robustness Principle as a domain independent proxy metric to measure the degree of difficulty of queries.

7.3.2 Properties of Hard Queries on Databases

As discussed in Chpater 2, it is well established that *the more diverse the candidate answers of a query are, the more difficult the query is* over a collection of the text documents. We extend this idea for queries over databases and propose three sources of difficulty for answering a query over a database, as follows:

1. The more entities match the terms in a query, the more diverse the candidate answers for the query are and it is harder to answer the query well. For example, there is more than one person named *Ford* in the IMDB data set. If a user submits query Q_2: *Ford*, a query system must determine which *Ford* will satisfy the user's information need. As opposed to Q_2, Q_3: *Spielberg* matches a smaller number of people in IMDB, so it is easier for the query system to return relevant results.

2. Each attribute describes a different aspect of an entity. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in the database. For instance, some candidate answers for query Q_4: *Godfather* in IMDB contain its term in their *title* and some contain its term in their *distributor*. For this example, we temporarily ignore the other attributes in IMDB. A query system must identify the desired matching attribute for *Godfather* to find its relevant answers. As opposed to Q_4, query Q_5: *taxi driver* does not match any instance of attribute *distributor*. Hence, a query system already knows the desired matching attribute for Q_5 and has an easier task to perform. The kind of difficulty introduced by this type of

diversity is different from the kind of difficulty created by having a large number of matched entities. Assume that Q_4 and Q_5 have almost equal numbers of answers. Some of the movies whose titles match Q_5 are related, e.g., there are three documentaries in IMDB whose titles match *taxi driver*, which are about making the well-known movie *Taxi Driver* directed by *Martin Scorsese*. They may partially or fully satisfy the information need behind Q_5. However, the candidate answers whose *title* attribute matches Q_4 and the candidate answers whose *distributor* attribute matches Q_4 are not generally related.

3. Each entity set contains information about a different type of entities. Hence, if a query matches entities from more entity sets, it will have more a diverse set of candidate answers. For instance, IMDB contains information about movies in an entity set called *movie* and information about the people involved in making movies in another entity set called *person*. Consider query Q_6: *divorce* over the IMDB data set whose candidate answers come from both entity sets. A query system has a difficult task to do, as it has to identify whether the information need behind this query is to find people who got divorced or movies about divorce. In contrast to Q_6, Q_7: *romantic comedy divorce* matches only entities from the *movie* entity set. It is less difficult for a KQI to answer Q_7 than Q_6, as Q_6 has only one possible desired entity set. This kind of diversity poses a new type of difficulty. Since the candidate answers of Q_6 are romantic comedy movies about divorce, they have some similarities to each other. However, movies about divorce and people who get divorced cannot both satisfy the information need of query Q_6. Given that Q_6 and Q_7 have almost the same number of candidate answers and matching attributes, it is likely that more candidate answers of Q_6 are relevant to its users' information need than the candidate answers to Q_7.

The aforementioned observations show that we may use the statistical properties of the query terms in the database to compute the diversity of its candidate answers and predict its difficulty. One idea is to count the number of possible attributes, entities, and entity sets that contain the query terms and use them to predict the difficulty of the query. The larger this value is, the more difficult the query will be. We show empirically in Section 7.6.2 that such an approach predicts the difficulty of queries quite poorly. This is because the distribution of query terms over attributes and entity sets may also impact the difficulty of the query. For instance, assume database DB_1 contains two entity sets *book* and *movie*, and database DB_2 contains entity sets *book* and *article*. Let term *database* appear in both entity sets in DB_1 and DB_2. Assume that there are far fewer movies that contain term *database* compared to books and articles. A query system can leverage this property and rank books higher than movies when answering query Q_8: *database* over DB_1. However, it will be much harder to decide the desired entity set in DB_2 for Q_8. Hence, a difficulty metric must take into account the skew of the distributions of the query term in the database as well. In Section 7.4, we discuss how these ideas are used to create a concrete noise generation framework that considers attribute values, attributes and entity sets.

7.4 A Framework to Measure Structured Robustness

In Section 7.3, we presented the Ranking Robustness Principle and discussed the specific challenges in applying this principle to structured data. In this section, we show concretely how this principle is quantified in structured data. Section 7.4.1 discusses the role of the structure and content of the database in the corruption process, and presents the robustness computation formula given corrupted database instances. Section 7.4.2 provides the details of how we generate corrupted instances of the database. Section 7.4.3 suggests methods to compute the parameters of our model. In Section 7.4.4 we show real examples of how our method corrupts the database and predicts the difficulty of queries.

7.4.1 Structured Robustness

Corruption of structured data. The first challenge in using the Ranking Robustness Principle for databases is to define data corruption for structured data. For that, we model a database DB using a generative probabilistic model based on its building blocks, which are terms, attribute values, attributes, and entity sets. A corrupted version of DB can be seen as a random sample of such a probabilistic model. Given a query Q and a retrieval function g, we rank the candidate answers in DB and its corrupted versions $DB', DB'', \ldots$ to get ranked lists L and $L', L'', \ldots$, respectively. The less similar L is to $L', L'', \ldots$, the more difficult Q will be.

According to the definitions in Section 7.2, we model database DB as a triplet $(\mathcal{S}, \mathcal{T}, \mathcal{A})$, where $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{A}$ denote the sets of entity sets, attributes, and attribute values in DB, respectively. $|\mathcal{A}|$, $|\mathcal{T}|$, $|\mathcal{S}|$ denote the number of attribute values, attributes, and entity sets in the database, respectively. Let V be the number of distinct terms in database DB. Each attribute value $A_a \in \mathcal{A}$, $1 \leq a \leq |\mathcal{A}|$, can be modeled using a V-dimensional multivariate distribution $X_a = (X_{a,1}, \cdots, X_{a,V})$, where $X_{a,j} \in X_a$ is a random variable that represents the frequency of term w_j in A_a. The probability mass function of X_a is:

$$f_{X_a}(\vec{x}_a) = Pr(X_{a,1} = x_{a,1}, \cdots, X_{a,V} = x_{a,V}), \tag{7.1}$$

where $\vec{x}_a = x_{a,1}, \cdots, x_{a,V}$ and $x_{a,j} \in \vec{x}_a$ are non-negative integers.

Random variable $X_{\mathcal{A}} = (X_1, \cdots, X_{|\mathcal{A}|})$ models attribute value set $\mathcal{A}$, where $X_a \in X_{\mathcal{A}}$ is a vector of size V that denotes the frequencies of terms in A_a. Hence, $X_{\mathcal{A}}$ is a $|\mathcal{A}|\times V$ matrix. The probability mass function for $X_{\mathcal{A}}$ is:

$$f_{X_{\mathcal{A}}}(\vec{x}) = f_{X_{\mathcal{A}}}(\vec{x_1}, \cdots, \vec{x_{|\mathcal{A}|}}) = Pr(X_1 = \vec{x_1}, \cdots, X_{|\mathcal{A}|} = \vec{x_{|\mathcal{A}|}}), \tag{7.2}$$

where $\vec{x_a} \in \vec{x}$ are vectors of size V that contain non-negative integers. The domain of $\vec{x}$ is all $|\mathcal{A}|\times V$ matrices that contain non-negative integers, i.e., $M(|\mathcal{A}|\times V)$.

We can similarly define $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ to model the set of attributes $\mathcal{T}$ and the set of entity sets $\mathcal{S}$, respectively. The random variable $X_{DB} = (X_{\mathcal{A}}, X_{\mathcal{T}}, X_{\mathcal{S}})$ models corrupted versions of database DB. In this thesis, we focus only on the noise introduced in the content (values) of the database. In other words, we do not consider other types of noise such as changing the attribute or entity set of an attribute value in the database. Since the membership of attribute values in their attributes and entity sets remains the same across the original and the corrupted versions of the database, we can derive $X_{\mathcal{T}}$ and $X_{\mathcal{S}}$ from $X_{\mathcal{A}}$. Thus, a corrupted version of the database will be a sample from $X_{\mathcal{A}}$; note that the attributes and entity sets play a key role in the computation of $X_{\mathcal{A}}$, as we discuss in Section 7.4.2. Therefore, we use only $X_{\mathcal{A}}$ to generate the noisy versions of DB, i.e., we assume that $X_{DB} = X_{\mathcal{A}}$. In Section 7.4.2 we present in detail how X_{DB} is computed.

Structured Robustness calculation. We compute the similarity of the answer lists using the Spearman rank correlation [53]. It ranges between 1 and -1, where 1, -1, and 0 indicate perfect positive correlation, perfect negative correlation, and almost no correlation, respectively. Equation 7.3 computes the Structured Robustness score (SR score), for query Q over database DB given retrieval function g:

$$\begin{aligned} SR(Q, g, DB, X_{DB}) &= \mathtt{E}\{Sim(L(Q, g, DB), L(Q, g, X_{DB}))\} \\ &= \sum_{\vec{x}} Sim(L(Q, g, DB), L(Q, g, \vec{x})) f_{X_{DB}}(\vec{x}) \end{aligned} \tag{7.3}$$

where $\vec{x} \in M(|\mathcal{A}| \times V)$ and Sim denotes the Spearman rank correlation between the ranked answer lists.

7.4.2 Noise Generation in Databases

In order to compute Equation 7.3, we need to define the noise generation model $f_{X_{DB}}(M)$ for database DB. *We will show that each attribute value is corrupted by a combination of three corruption levels: on the value itself, its attribute and its entity set.* Now the details: since the ranking methods for queries over structured data do not generally consider the terms in V that do not belong to query Q [62, 78], we consider their frequencies to be the same across the original and noisy versions of DB. Given query Q, let $\vec{x}$ be a vector that contains term frequencies for terms $w \in Q \cap V$. Similarly to [161], we simplify our model by assuming the attribute values in DB and the terms in $Q \cap V$ are independent. Hence, we have:

$$f_{X_{\mathcal{A}}}(\vec{x}) = \prod_{x_a \in \vec{x}} f_{X_a}(\vec{x_a}) \tag{7.4}$$

and

$$f_{X_a}(\vec{x}_a) = \prod_{x_{a,j} \in \vec{x}_a} f_{X_{a,j}}(x_{a,j}) \tag{7.5}$$

where $x_j \in \vec{x}_i$ depicts the number of times w_j appears in a noisy version of attribute value A_i and $f_{X_{i,j}}(x_j)$ computes the probability of term w_j to appear in A_i x_j times.

The corruption model must reflect the challenges discussed in Section 7.3.2 about search on structured data, where we showed that it is important to capture the statistical properties of the query keywords in the attribute values, attributes and entity sets. We must introduce content noise (recall that we do not corrupt the attributes or entity sets but only the values of attribute values) to the attributes and entity sets, which will propagate down to the attribute values. For instance, if an attribute value of attribute *title* contains keyword *Godfather*, then *Godfather* may appear in any attribute value of attribute *title* in a corrupted database instance. Similarly, if *Godfather* appears in an attribute value of entity set *movie*, then *Godfather* may appear in any attribute value of entity set *movie* in a corrupted instance.

Since the noise introduced in attribute values will propagate up to their attributes and entity sets, one may question the need to introduce additional noise in attribute and entity set levels. The following example illustrates the necessity of generating such noise. Let T_1 be an attribute whose attribute values are A_1 and A_2, where A_1 contains term w_1 and A_2 does not contain w_1. A possible noisy version of T_1 will be a version where A_1 and A_2 both contain w_1. However, the aforementioned noise generation model will not produce such a version. Similarly, a noisy version of entity set S must introduce or remove terms from its attributes and attribute values. According to our discussion in Section 7.3, we must use a model that generates all possible types of noise in the data.

Hence, we model the noise in a DB as a *mixture* of the noises generated in the attribute value, attribute, and entity set levels. Mixture models are typically used to model how the combination of multiple probability distributions generates the data. Let $Y_{t,j}$ be the random variable that represents the frequency of term w_j in attribute T_t. Probability mass function $f_{Y_{t,j}}(y_{t,j})$ computes the probability of w_j to appear $y_{t,j}$ times in T_t. Similarly, $Z_{s,j}$ is the random variable that denotes the frequency of term w_j in entity set S_s and probability mass function $f_{Z_{s,j}}(z_{s,j})$ computes the probability of w_j to appear $z_{s,j}$ times in S_s. Hence, the noise generation models attribute value A_i whose attribute is T_t and entity set is S_s:

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \gamma_A f_{X_{a,j}}(x_{a,j}) + \gamma_T f_{Y_{t,j}}(x_{t,j}) + \gamma_S f_{Z_{s,j}}(x_{s,j}), \tag{7.6}$$

where $0 \leq \gamma_A, \gamma_T, \gamma_S \leq 1$ and $\gamma_A + \gamma_T + \gamma_S = 1$. $f_{X_{a,j}}$, $f_{Y_{t,j}}$, and $f_{Y_{s,j}}$ model the noise in the attribute value, attribute, and entity set levels, respectively. Parameters γ_A, γ_T and γ_S have the same values for all terms $w \in Q \cap V$ and are set empirically.

Since each attribute value A_a is a small document, we model $f_{X_{i,j}}$ as a Poisson distribution:

$$f_{X_{a,j}}(x_{a,j}) = \frac{e^{-\lambda_{a,j}} \lambda_{a,j}^{x_{a,j}}}{x_{a,j}!}. \tag{7.7}$$

Similarly, we model each attribute T_t, $1 \leq t \leq |\mathcal{T}|$, as a bag of words and use a Poisson distribution to model the noise generation at the attribute level:

$$f_{Y_{t,j}}(x_{t,j}) = \frac{e^{-\lambda_{t,j}} \lambda_{t,j}^{x_{t,j}}}{x_{t,j}!}. \tag{7.8}$$

Using similar assumptions, we model the changes in the frequencies of the terms in entity set S_s, $1 \leq s \leq |\mathcal{S}|$, using a Poisson distribution:

$$f_{Z_{s,j}}(x_{s,j}) = \frac{e^{-\lambda_{s,j}} \lambda_{s,j}^{x_{s,j}}}{x_{s,j}!}. \tag{7.9}$$

In order to use the model in Equation 7.6, we have to estimate $\lambda_{A,w}$, $\lambda_{T,w}$, and $\lambda_{S,w}$ for every $w \in Q \cap V$, attribute value A, attribute T and entity set S in DB. We treat the original database as an observed value of the space of all possible noisy versions of the database. Thus, using the maximum likelihood estimation method, we set the value of $\lambda_{A,w}$ to the frequency of w in attribute value A. Assuming that w are distributed uniformly over the attribute values of attribute T, we can set the value of $\lambda_{T,w}$ to the average frequency of w in T. Similarly, we set the value of $\lambda_{S,w}$ as the average frequency of w in S. Using these estimations, we can generate noisy versions of a database according to Equation 7.6.

7.4.3 Smoothing The Noise Generation Model

Equation 7.6 overestimates the frequency of the terms of the original database in the noisy versions of the database. For example, assume a bibliographic database of computer science publications that contains attribute T_2 =*abstract* which constitutes the abstract of a paper. Apparently, many abstracts contain term w_2 =*algorithm*, therefore, this term will appear very frequently with high probability in the f_{T_2,w_2} model. On the other hand, a term such as w_3 = *Dirichlet* is very likely to have very low frequency in the f_{T_2,w_3} model. Let attribute value A_2 be of attribute *abstract* in the bibliographic DB that contains both w_2 and w_3. Most likely, term $algorithm$ will appear more frequently than $Dirichlet$ in A_2. Hence, the mean for f_{A_2,w_2} will be also larger than the mean of f_{A_2,w_3}. Thus, a mixture model of f_{T_2,w_2} and f_{A_2,w_2} will have much larger mean than a mixture model of f_{T_2,w_3} and f_{A_2,w_3}. The same phenomenon occurs if a term is relatively frequent in an entity set. Hence, a mixture model such as Equation 7.6 overestimates the frequency of the terms that are relatively frequent in an attribute or entity set.

Researchers have faced a similar issue in language model smoothing for speech recognition [75]. We use a similar approach to resolve this issue. If term w appears in attribute value A, we use only the first term in Equation 7.6 to model the frequency of w in the noisy version of database. Otherwise, we use the second or third terms if w belongs to T and S, respectively. Hence, the noise generation model is:

```
<movie id= ‘‘1025102’’>
<title>rome ...</title>
<keyword>ancient-world</keyword>
<keyword>ancient-art</keyword>
<keyword>ancient-rome</keyword>
<keyword>christian-era</keyword>
</movie>

<movie id=‘‘1149602’’>
<title>Gladiator</title>
<keyword>ancient-rome</keyword>
<character>Rome ...</character>
<person>... Rome/UK)</person>
<trivia>"Rome of the imagination... </trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ... Rome ... </quote>
</movie>
```

Figure 7.1: Ranking of the original results of $Q11$

```
<movie id=‘‘1149602’’>
<title> Gladiator rome</title>
<keyword>ancient-rome rome</keyword>
<character>Rome ...</character>
<person> ... Rome/UK)</person>
<trivia>of the imagination ...</trivia>
<goof>Rome vs. Carthage ...</goof>
<quote>... enters Rome like a ... Rome ...</quote>
</movie>

<movie id= ‘‘1025102’’>
<title>rome ...</title>
<keyword>ancient-world ancient</keyword>
<keyword>-art</keyword>
<keyword>ancient ancient</keyword>
<keyword>christian-</keyword>
</movie>
```

Figure 7.2: Ranking of the corrupted results of $Q11$

$$\hat{f}_{X_{a,j}}(x_{a,j}) = \begin{cases} \gamma_A f_{X_{a,j}}(x_{a,j}) & \text{if } w_j \in A_a \\ \gamma_T f_{Y_{t,j}}(x_{t,j}) & \text{if } w_j \notin A_a, w_j \in T_t \\ \gamma_S f_{Z_{s,j}}(x_{s,j}) & \text{if } w_j \notin A_a, T_t, w_j \in S_s \end{cases} \tag{7.10}$$

where we remove the condition $\gamma_A + \gamma_T + \gamma_S = 1$.

```
<movie id=‘‘1492260’’>
<title>The Legend of Mulan (1998) (V)</title>
<genre>Animation</genre >
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998)</link>
<link>Mulan (1999)</link>
<link>The Secret of Mulan (1998)</link>
</movie>

<movie id=‘‘1180849’’>
<title>Hua Mulan (2009)</title>
<character>Hua Hu (Mulan}’s father)</character>
<character>Young Hua Mulan</character>
<character>Hua Mulan</character>
</movie>
```

Figure 7.3: Original results of $Q9$

```
<movie id=‘‘1492260’’>
<title>The Legend of Mulan (1998) (V) mulan mulan</title>
<genre></genre >
<link>Hua Mu Lan (1964)</link>
<link>Hua Mulan cong jun</link>
<link>Mulan (1998) mulan</link>
<link> (1999)</link>
<link>The Secret of Mulan (1998) mulan </link>
<movie>

<movie id=‘‘1180849’’>
<title>Hua (2009) hua</title>
<character>Hua Hu (Mulan’s father)</character>
<character>Young Hua Mulan mulan mulan hua</character>
<character>Mulan</character>
</movie>
```

Figure 7.4: Corrupted results of $Q9$

7.4.4 Examples

We illustrate the corruption process and the relationship between the robustness of the ranking of a query and its difficulty using INEX queries $Q9$: *mulan hua animation* and $Q11$: *ancient rome era*, over the IMDB data set. We set $\gamma_A = 1$, $\gamma_T = 0.9$, $\gamma_S = 0.8$ in Equation 7.10. We use the XML ranking method proposed in [78], called PRSM, which we discussed in the previous chapter and explain in more detail in Section 7.5. Given query Q, PRSM computes the relevance score of entity E based on the weighted linear combination of the relevance scores of the attribute values of E.

Example of calculation of $\lambda_{t,j}$ for term t =ancient and attribute T_j =plot in Equation 7.8: In the IMDB data set, *ancient* occurs in attribute *plot* 2132 times in total, and the total number of attribute values under attribute *plot* is 184,731, so $\lambda_{t,j} = 2132/184731$ which is 0.0115. Then, since $\gamma_T = 0.9$, the probability that *ancient* occurs k times in a corrupted *plot* attribute is $\frac{0.9e^{-0.0115}(0.0115)^k}{k!}$.

Q11: Figure 7.1 depicts two of the top results (ranked as 1^{st} and 12^{th} respectively) for query $Q11$ over IMDB. We omit most attributes (shown as elements in XML lingo in Figure 7.1) that do not contain any query keywords. Figure 7.2 illustrates a corrupted version of the entities shown in Figure 7.1. The new keyword instances are underlined. Note that the ordering changed according to the PRSM ranking. The reason is that PRSM believes that *title* is an important attribute for *rome* (for attribute weighting in PRSM see Section 7.6.1) and hence having a query keyword (*rome*) there is important. However, after corruption, query word *rome* also appears in the *title* of the other entity, which now ranks higher, because it contains the query words in more attributes.

Word *rome* was added to the *title* attribute of the originally second result through the second level (attribute-based, second branch in Equation 7.10) of corruption, because *rome* appears in the *title* attribute of other entities in the database. If no *title* attribute contained *rome*, then it could have been added through the third level corruption (entity set-based, third branch in Equation 7.10) since it appears in attribute values of other *movie* entities.

The second and third level corruptions typically have much smaller probability of adding a word than the first level, because they have much smaller λ; specifically, λ_T is the average frequency of the term in attribute T. However, in hard queries like $Q11$, the query terms are frequent in the database, and also appear in various entities and attributes, and hence λ_T and λ_S are larger.

In the first *keyword* attribute of the top result in Figure 7.2, *rome* is added by the first level of corruption, whereas in the *trivia* attribute, *rome* is removed by the first level of corruption.

To summarize, $Q11$ is *difficult* because its keywords are spread over a large number of attribute values, attributes and entities in the original database, and also most of the top results have a similar number of occurrences of the keywords. Thus, when the corruption process adds even a small number of query keywords to the attribute values of the entities in the original database, it drastically changes the ranking positions of these entities.

Q9: $Q9$ (*mulan hua animation*) is an *easy* query because most its keywords are quite infrequent in the database. Only term *animation* is relatively frequent in the IMDB data set, but almost all its occurrences are in attribute *genre*. Figures 7.3 and 7.4 present two ordered top answers for $Q9$ over the original and corrupted versions of IMDB, respectively. The two results are originally ranked as 4^{th} and 10^{th}. The attribute values of these two entities contain many query keywords in the original database. Hence, adding and/or removing some query keyword instances in these results does not considerably change their relevance score, and they preserve their ordering after corruption.

Since keywords *mulan* and *hua* appear in a small number of attribute values and attributes, the value of λ for these terms in the second and the third level of corruption is very small. Similarly, since keyword *animation* only appears in the *genre* attribute, the value of λ for all other attributes (second level corruption) is zero. The value of λ for *animation* in the third level is reasonable, 0.0007 for *movie* entity set, but the noise generated in this level alone is not considerable.

7.5 Efficient Computation of SR Score

A key requirement for this work to be useful in practice is that the computation of the SR score incurs a minimal time overhead compared to the query execution time. In this section we present efficient SR score computation techniques.

7.5.1 Basic Estimation Techniques

Top-K results: In general, the basic information units in structured data sets – attribute values – are much shorter than text documents. Thus, a structured data set contains a larger number of information units than an unstructured data set of the same size. For instance, each XML document in the INEX data-centric collection contains hundreds of elements with textual content. Hence, computing Equation 7.3 for a large database is so inefficient as to be impractical. Hence, similar to [161], we corrupt only the top K entity results of the original data set. We re-rank these results and shift them up to be the top K answers for the corrupted versions of the database. In addition to the time savings, our empirical results in Section 7.6.2 show that relatively small values for K predict the difficulty of queries better than large values. For instance, we found that $K = 20$ delivers the best performance prediction quality in our data sets. We discuss the impact of different values of K on query difficulty prediction quality further in Section 7.6.2.

Number of corruption iterations (N): Computing the expectation in Equation 7.3 for all possible values of $\vec{x}$ is very inefficient. Hence, we estimate the expectation using $N > 0$ samples over $M(|\mathcal{A}| \times V)$. That is, we use N corrupted copies of the data. Obviously, a smaller N is preferred for the sake of efficiency. However, if we choose very small values for N, the corruption model becomes unstable. We further analyze how to choose the value of N in Section 7.6.2.

We can limit the values of K or N in any of the algorithms described below.

7.5.2 Structured Robustness Algorithm

Algorithm 7.5 shows the Structured Robustness Algorithm (SR Algorithm), which computes the exact SR score based on the top K result entities. Each ranking algorithm uses some statistics about query terms or attributes values over the whole content of DB. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the DB or total number of attribute values in each attribute and entity set. These global statistics are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode.

Figure 7.5: $CorruptTopResults(Q, L, M, I, N)$

```
1 Input: Query $Q$, top $K$ list $L$ of $Q$ under ranking function $g$, metadata $M$, inverted indexes $I$, corruption
  iterations $N$.
  Output: $SR$ score for $Q$.
   1: $SR \leftarrow 0$; $C \leftarrow \{\}$; //$C$ caches $\lambda_T$, $\lambda_S$ for keywords in $Q$
   2: FOR $i = 1 \rightarrow N$ DO
   3:    $I' \leftarrow I$; $M' \leftarrow M$; $L' \leftarrow L$; //Corrupted copy of $I$, $M$ and $L$
   4:    FOR each result $R$ in $L$ DO
   5:       FOR each attribute value $A$ in $R$ DO
   6:          $A' \leftarrow A$; //Corrupted versions of $A$
   7:          FOR each keywords $w$ in $Q$ DO
   8:             Compute # of $w$ in $A'$ by Equation 7.10; //If $\lambda_{T,w}$, $\lambda_{S,w}$ needed but not in $C$, calculate and cache them
   9:             IF # of $w$ varies in $A'$ and $A$ THEN
  10:                Update $A'$, $M'$ and entry of $w$ in $I'$;
  11:          Add $A'$ to $R'$;
  12:       Add $R'$ to $L'$;
  13:    Rank $L'$ using $g$, which returns $L$, based on $I'$, $M'$;
  14:    $SR$ += $Sim(L, L')$; //$Sim$ computes Spearman correlation
  15: RETURN $SR \leftarrow SR/N$; //AVG score over $N$ rounds
```

The SR Algorithm generates noise in the DB on-the-fly during query processing. Since it corrupts only the top K entities, which are always returned by the ranking module, it does not perform any extra I/O access to the database, except to look up some statistics. Moreover, it uses information which is already computed and stored in inverted indexes and does not require any additional index.

Nevertheless, our empirical results, reported in Section 7.6.2, show that the SR Algorithm increases query processing time considerably. The reasons for SR Algorithm inefficiency include the following: First, Line 5 in the SR Algorithm loops over every attribute value in each top K result and tests whether it must be corrupted. As noted before, one entity may have hundreds of attribute values. The attribute values that do not contain any query term still must be corrupted (Line 8-10 in the SR Algorithm) for the second and third levels of corruption defined in Equation 7.10. This is because their attributes or entity sets may contain some query keywords. This will greatly increase the number of attribute values to be corrupted. For instance, for IMDB, which has only two entity sets, the SR Algorithm corrupts all attribute values in the top K results for all query keywords. Second, ranking algorithms for databases are relatively slow. The SR Algorithm has to re-rank the top K entities N times, which is time consuming.

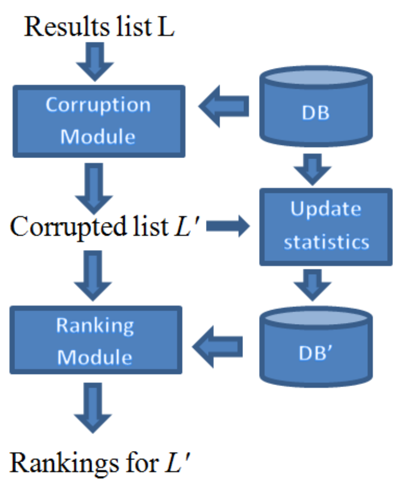


Figure 7.6: SR Algorithm

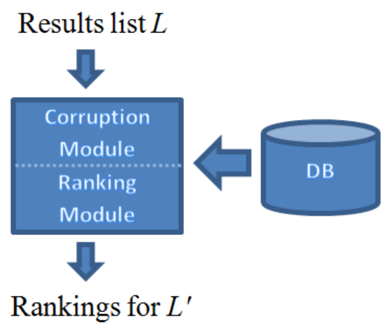


Figure 7.7: SGS-Approx Algorithm

7.5.3 Approximation Algorithms

In this section, we propose approximation algorithms to improve the efficiency of the SR Algorithm. Our methods are independent of the underlying ranking algorithm.

Query-specific Attribute values Only Approximation (QAO-Approx): QAO-Approx corrupts only the attribute values that match at least one query term. This approximation algorithm leverages the following observations:

Observation 1: The noise in the attribute values that contain query terms dominates the corruption effect.

Observation 2: The number of attribute values that contain at least one query term is relatively much smaller than the number of all attribute values in each entity.

Hence, we can significantly decrease the time spent on corruption if we corrupt only the attribute values that contain query terms. We add a check before Line 7 in the SR Algorithm to test whether A contains any term in Q.

Hence, we skip the loop in Line 7. The second and third levels of corruption (on attributes and entity sets, respectively) corrupt a smaller number of attribute values, so the time spent on corruption becomes shorter.

Static Global Stats Approximation (SGS-Approx): *SGS-Approx* uses the following observation:

Observation 3: Given that only the top K result entities are corrupted, the global DB statistics do not change much.

Figure 7.6 shows the execution flow of the SR Algorithm. Once we get the ranked list of top K entities for Q, the corruption module produces corrupted entities and updates the global statistics of DB. Then, the SR Algorithm passes the corrupted results and updated global statistics to the ranking module to compute the corrupted ranking list.

The SR Algorithm spends a high percentage of the robustness calculation time on the loop that re-ranks the corrupted results (Line 13 in SR Algorithm), by taking into account the updated global statistics. Since the value of K (e.g., 10 or 20) is much smaller than the number of entities in the database, the top K entities constitute a very small portion of the database. Thus, the global statistics largely remain unchanged or change very little. Hence, we can use the global statistics of the original version of the database to re-rank the corrupted entities. If we refrain from updating the global statistics, we can combine the corruption and ranking module. This way, re-ranking is done on-the-fly during corruption. *SGS-Approx* is illustrated in Figure 7.7.

We use the ranking algorithm proposed in [78], called PRMS, to better illustrate the details of our approximation algorithm. PRMS employs a language model approach to search over structured data. It computes the language model of each attribute value smoothed by the language model of its attribute. It assigns each attribute a query keyword-specific weight, which specifies its contribution to the ranking score. It computes the keyword-specific weight $\mu_j(q)$ for attribute values whose attributes are T_j and query q as $\mu_j(q) = \frac{P(q|T_j)}{\sum_{T \in DB} P(q|T)}$. The ranking score of entity E for query Q, $P(Q|E)$ is:

$$P(Q|E) = \prod_{q \in Q} P(q|E) = \prod_{q \in Q} \sum_{j=1}^{n} [\mu_j(q)((1-\lambda)P(q|A_j) + \lambda P(q|T_j))], \tag{7.11}$$

where A_j is an attribute value of E, T_j is the attribute of A_j, $0 \leq \lambda \leq 1$ is the smoothing parameter for the language model of A_j, and n is the number of attribute values in E. If we ignore the change of global statistics of DB, then the μ_j and $P(q|T_j)$ parts will not change when calculating the score of corrupted version of E or E', for q. Hence, the score of E' will depend only on $P(q|A'_j)$, where A'_j is the corrupted version of A_j. We compute the value of $P(q|A'_j)$ using only the information of A'_j as (# of q in A'_j / # of words in A'_j). SGS-Approx uses the global statistics of the original database to compute μ_j and $P(q|T_j)$, in order to calculate the value of $P(q|E)$. It re-uses them to compute the score of the corrupted versions of E. However, the SR Algorithm has to finish all corruption on all attribute values in top results to update the global statistics and re-rank the corrupted results. Similarly, we can modify other keyword query ranking algorithms over databases that use query term statistics to score entities.

	INEX	SemSearch
Size	9.85 GB	9.64 GB
Number of Entities	4,418,081	7,170,445
Number of Entity Sets	2	419,610
Number of Attributes	77	7,869,986
Number of Attribute values	113,603,013	114,056,158

Table 7.1: INEX and SemSearch data sets characteristics

Combination of QAO-Approx and SGS-Approx: QAO-Approx and SGS-Approx improve the performance of robustness calculation by approximating different parts of the corruption and re-ranking process. Hence, we can combine these two algorithms to further improve the performance of query difficulty prediction.

7.6 Experiments

7.6.1 Experiments Setting

Data sets: Table 7.1 shows the characteristics of two data sets used in our experiments. The INEX data set is from the INEX 2010 Data Centric Track [142] discussed in Section 7.1.The INEX data set contains two entity sets: *movie* and *person*. Each entity in the *movie* entity set represents one movie with attributes like *title*, *keywords*, and *year*. The *person* entity set contains attributes like *name*, *nickname*, and *biography*.

The SemSearch data set is a subset of the data set used in the Semantic Search 2010 challenge [133]. The original data set contains 116 files with about one billion RDF triplets. Since the size of this data set is extremely large, it takes a very long time to index and run queries over this data set. Hence, we have used a subset of the original data set in our experiments. We first removed duplicate RDF triplets. Then, for each file in the SemSearch data set, we calculated the total number of distinct query terms in the SemSearch query workload in the file. Out of the 116 files, we selected the 20 files that contain the largest number of query keywords for our experiments. We converted each distinct RDF subject in this data set to an entity whose identifier is the subject identifier. The RDF properties are mapped to attributes in our model. The values of RDF properties that end with substring "#type" indicate the type of a subject. Hence, we set the entity set of each entity to the concatenation of the values of the RDF properties of its RDF subject that end with substring "#type". If the subject of an entity does not have any property that ends with substring "#type", we set its entity set to "UndefinedType". We added the values of other RDF properties for the subject as attributes of its entity. We stored the information about each entity in a separate XML file. We removed the relevance judgment information for the subjects that do not reside in these 20 files. The sizes of the two data sets are quite close; however, SemSearch is more heterogeneous than INEX as it contains a larger number of attributes and entity sets.

Query Workloads: Since we use a subset of the dataset from SemSearch, some queries in its query workload may not contain enough candidate answers. We picked the 55 queries from the 92 in the query workload that have at least 50 candidate answers in our dataset. Because the number of entries for each query in the relevance judgment file has also been reduced, we discarded another two queries (Q6 and Q92) without any relevant answers in our data set, according to the relevance judgment file. Hence, our experiments is done using 53 queries (2, 4, 5, 11-12, 14-17, 19-29, 31, 33-34, 37-39, 41-42, 45, 47, 49, 52-54, 56-58, 60, 65, 68, 71, 73-74, 76, 78, 80-83, 88-91) from the SemSearch query workload. 26 query topics are provided with relevance judgments in the INEX 2010 Data Centric Track. Some query topics contain characters "+" and "-" to indicate conjunctive and exclusive conditions. In our experiments, we do not use these conditions and remove the keywords after character "-".

Generally, keyword query systems for databases return candidate answers that contain all terms in the query [9, 62, 130]. However, queries in the INEX query workload are relatively long (normally over four distinct keywords). If we retrieve only the entities that contain all query terms, there will not be sufficient (in some cases none) candidate answers for many queries in the data. Hence, for every query Q, we use the following procedure to get at least 1,000 candidate answers for each query. First, we retrieve the entities that contain $|Q|$ distinct terms in query Q. If they are not sufficient, we retrieve the entities that contain at least $|Q| - 1$ distinct query keywords, and so on until we get 1000 candidate answers for each query.

Effectiveness Metrics: We used the popular effectiveness metrics Average Precision and Mean Average Precision (MAP) to measure the quality of rankings delivered by ranking algorithms for a query and a set of queries over a database, respectively [96].

Ranking Algorithms: To evaluate the effectiveness of our model for different ranking algorithms, we evaluated the query performance prediction model with two representative ranking algorithms: PRMS [78] and IR-Style [62]. Many other algorithms are extensions of these two methods (e.g., [93, 33]).

PRMS: We explained the idea behind the PRMS algorithm in Section 7.5. We adjust parameter λ in PRMS in our experiments to get the best MAP and then use this value of λ for query performance prediction evaluations. Varying λ from 0.1 to 0.9 with 0.1 as the test step, we have found that different values of λ change MAP very slightly on both data sets, and generally smaller λs deliver better MAP. We use λ = 0.1 on the INEX data set and 0.2 for the SemSearch data set.

IR-Style: We use a variation of the ranking model proposed in [62] for the relational data model, referred as IR-Style ranking. Given a query, IR-Style returns a minimal join tree that connects the tuples from different tables in the DB that contain the query terms, called $MTNJT$. However, our data sets are not in relational format and the answers in their relevance judgment files are entities, not $MTNJT$s. Hence, we extend the definition of $MTNJT$ to be the minimal subtree that connects the attribute values containing the query keywords in an entity. The root of this subtree is the root of the entity in its XML file. If an entity has multiple $MTNJT$s, we choose the one with the maximum

	INEX		SemSearch	
training set	(γ_A, γ_T, γ_S)	correlation	(γ_A, γ_T, γ_S)	correlation
1	(1, 0.9, 0.8)	0.689	(1, 0.1, 0.6)	0.744
2	(1, 0.9, 0.8)	0.777	(1, 0.1, 0.6)	0.659
3	(1, 0.9, 0.8)	0.695	(1, 0.1, 0.8)	0.596
4	(1, 0.9, 0.8)	0.799	(1, 0.1, 0.6)	0.702
5	(1, 0.8, 0.3)	0.540	(1, 0.1, 0.6)	0.597

Table 7.2: Training and testing of (γ_A, γ_T, γ_S), for K=20.

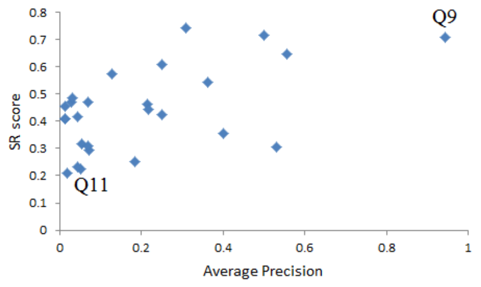


Figure 7.8: Average precision versus SR score for queries on INEX using PRMS, for K=20.

K	10							20						
Method	SR	URM	CR	iAA	iAES	iAE	iAS	SR	URM	CR	iAA	iAES	iAE	iAS
INEX	0.471	0.247	0.379	0.299	n/a	0.111	0.143	0.556	0.311	0.391	0.370	n/a	0.255	0.292
SemSearch	0.486	0.093	0.091	0.066	0.052	0.040	-0.043	0.564	0.177	0.09	0.082	0.068	0.056	-0.046

Table 7.3: Correlation of average precision against each metric of difficulty.

score, as explained below.

Let M be a $MTNJT$ tree of entity E and let $\mathcal{A}_{\mathcal{M}}$ be the attribute values in M. The score of M for query Q is $\frac{IRScore(M,Q)}{size(M)}$, where $IRScore(M, Q)$ is the score of M for query Q based on an IR ranking formula. If we use a vector space model ranking formula as in [62] to compute the $IRScore(M, Q)$, we get very low MAP (less than 0.1) for both data sets. Hence, we compute it using a language model ranking formula with Jelink-Mercer smoothing [158], which is shown in Equation 7.12. We set the value of smoothing parameter α to 0.2, as it returns the highest MAP for our data sets.

$$IRScore(M, Q) = \prod_{q \in Q} \sum_{A \in \mathcal{A}_{\mathcal{M}}} ((1 - \alpha)P(q|A) + \alpha P(q|T)) \tag{7.12}$$

Configuration: We performed our experiments on an AMD Phenom II X6 2.8 GHz machine with 8 GB of main memory that runs 64-bit Windows 7. We used Berkeley DB 5.1.25 to store and index the XML files and implemented all algorithms in Java.

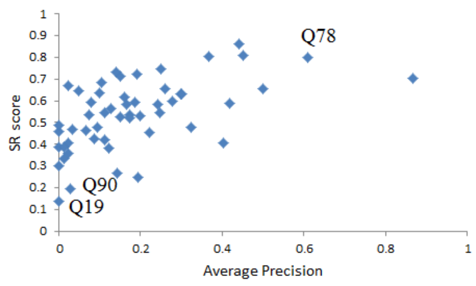


Figure 7.9: Average precision versus SR score for queries on SemSearch using PRMS, for K=20.

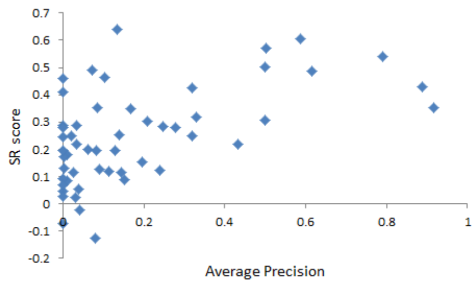


Figure 7.10: Average precision versus SR score using IR-Style over SemSearch with K=20 and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6).

K	10	20
Correlation score	0.565	0.544

Table 7.4: Effect of K on Pearson's correlation of average precision and SR score using IR-Style ranking on SemSearch

	Avg Q-time (ms)	Avg SR-time (ms)
INEX	24,177	(88,271 + 29,585)
SemSearch	46,726	(11,121 + 12,110)

Table 7.5: Average query processing time and average robustness computation for K=20, N=250 on SemSearch and N=300 on INEX.

7.6.2 Quality Results

In this section, we evaluate the effectiveness of the query quality prediction model computed using the SR Algorithm. We use Pearson's correlation between the SR score and the average precision of a query to evaluate the prediction quality of SR score.

Setting the value of N: Let L and L' be the original and corrupted top K entities for query Q, respectively. The SR score of Q in each corruption iteration is the Spearman's correlation between L and L'. We corrupt the results N times to get the average SR score for Q. In order to get a stable SR score, the value of N should be sufficiently large, but this increases the computation time of the SR score. We chose the following strategy to find the appropriate value of N: We progressively corrupt L 50 iterations at a time and calculate the average SR score over all iterations. If the last 50 iterations do not change the average SR score over 1%, we terminate. N may vary for different queries in query workloads. Thus, we set it to the maximum number of iterations over all queries. According to our experiments, the value of N varies very slightly for different value of K. Therefore, we set the value of N to 300 on INEX and 250 on SemSearch for all values of K.

Different Values for K: The number of interesting results for a keyword query is normally small [96]. Hence, it is reasonable to focus on small values of K for query performance prediction. We conduct our experiments for K=10 and K=20. Both values deliver reasonable prediction quality (i.e., the robustness of a query is strongly correlated with its effectiveness). We achieved the best prediction quality using K=20 for both data sets, with different combinations of γ_A, γ_T, and γ_S, which we will introduce later.

Training of γ_A, γ_T, and γ_S: We denote the coefficients combination in Equation 7.10 as (γ_A, γ_T, γ_S) for brevity. We train (γ_A, γ_T, γ_S) by 5-fold cross validation. After preliminary experiments, we found that large γ_A is effective. Hence, to reduce the number of possible combinations, we fix γ_A as 1, and vary the other two during the training process to find the highest correlation between average precision and SR score. We computed the SR score for γ_T and γ_S from 0 to 3 with step 0.1 for different values of K. Table 7.2 shows the training of γ_T and γ_S, and the correlation between average precision and SR score on the testing sets. It shows that for different training sets, the (γ_A, γ_T, γ_S) is quite stable for getting the best correlation score. Thus in the following results, we set (γ_A, γ_T, γ_S) to be (1, 0.9, 0.8) on INEX and (1, 0.1, 0.6) on SemSearch.

Figures 7.8 and 7.9 depict the plot of average precision and SR score for all queries in our query workload on INEX and SemSearch, respectively. In Figure 7.8, we see that $Q9$ is easy (has high average precision) and $Q11$ is relatively hard, as discussed in Section 7.4.4. As shown in Figure 7.9, query $Q78$: *sharp-pc* is easy (has high average precision), because its keywords appear together in few results, which explains its high SR score. On the other hand, $Q19$: *carl lewis* and $Q90$: *university of phoenix* have a very low average precision as their keywords appear in many attributes and entity sets. Figure 7.9 shows that the SR scores of these queries are very small, which confirms our model.

Baseline Prediction Methods

Clarity score (CR) [132] and Unstructured Robustness Method (URM) [161] are two popular query difficulty prediction techniques over text documents. The prediction quality (i.e., the correlation between average precision of queries and the metric) of clarity score and URM are 0.21 - 0.51 and 0.30 - 0.61, respectively [161]. We use these methods as well as the prevalence of query keywords as baseline query difficulty prediction algorithms for databases.

URM and CR: Our goal in this experiment is to find how accurately URM can predict the effectiveness of queries over a database. We concatenate the XML elements and tags of each entity into a text document and assume all entities (now text documents) belong to one entity set. The values of all μ_j in PRMS ranking formula are set to 1 for every query term.Hence, PRMS becomes a language model retrieval method for text documents [96]. Similar to URM, we implement CR by treating each entity in database as a text document. We have used similar parameters as [132, 161] to compute the CR for queries over our data sets.

Prevalence of Query Keywords: As we argued in Section 7.3.2, if the query keywords appear in many entities, attributes, or entity sets, it is harder for a ranking algorithm to locate the desired entities. Given query Q, we compute the average number of attributes ($AA(Q)$), average number of entity sets ($AES(Q)$), and the average number of entities ($AE(Q)$) where each keyword in Q occurs. We consider each of these three values as an individual baseline difficulty prediction metric. We also multiply these three metrics (to avoid normalization issues that summation would have) and create another baseline metric, denoted $AS(Q)$. Intuitively, if these metrics for query Q have higher values, Q must be harder and have lower average precision. Thus, we use the inverse of these values, denoted as $iAA(Q)$, $iAES(Q)$, $iAE(Q)$, and $iAS(Q)$, respectively.

Comparison to Baseline Methods: Table 7.3 shows the prediction accuracy (correlation between average precision and each metric) for SR, URM, CR, iAA(Q), iAES(Q), iAE(Q), and iAS(Q) methods for different K over both datasets. These results are based on all queries in the query workloads without distinguishing between training and testing sets as in Table 7.2. The *n/a* value appears in the table because all query keywords in our query workloads occur in both entity sets in the INEX data set. The correlation values for the SR Algorithm are significantly higher than the correlation values of URM and CR, on both data sets. This shows that our prediction model is more effective than URM and CR over databases. Metric iAA provides a more accurate prediction than all other methods over INEX. This indicates that one of the main causes of difficulty for the queries over the INEX data set is to find their desired attributes, which confirms our analysis in Section 7.3.2. SR also delivers far better prediction quality than iAA(Q), iAES(Q), iAE(Q), and iAS(Q) metrics over both data sets. Hence, SR effectively considers all causes of the difficulties for queries over databases.

IR-style ranking algorithm: The best value of MAP for the IR-Style ranking algorithm over INEX is 0.105 for K=20, which is very low. Note that we tried both Equation 7.12 as well as the vector space model originally used in [62]. Thus, we do not study the quality performance prediction for IR-Style ranking algorithm over INEX. On the other hand, the IR-Style ranking algorithm using Equation 7.12 delivers larger MAP value than PRMS on the SemSearch dataset. Due to the lack of space we cannot present all experiments under IR-Style ranking. Hence, we only present results on SemSearch. Table 7.4 shows Pearson's correlation of SR score with the average precision for different values of K, for N=250 and (γ_A, γ_T, γ_S) = (1, 0.1, 0.6). Figure 7.10 plots SR score against the average precision when K=20. On SemSearch, we also tried IR-Style ranking algorithm using vector space model ranking, but it achieves very low MAP (0.08) so we do not report results.

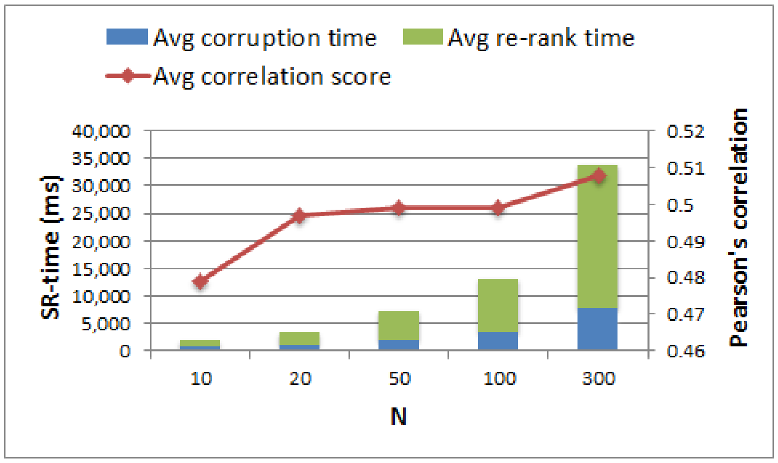


Figure 7.11: Approximations on INEX using QAO-Approx

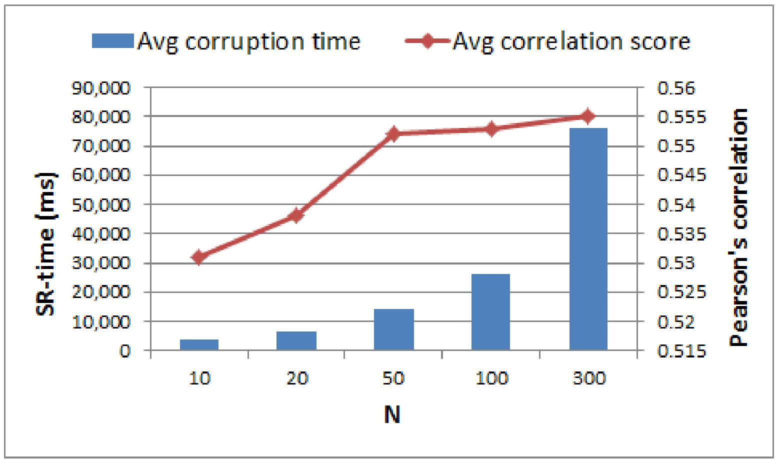


Figure 7.12: Approximations on INEX using SGS-Approx

7.6.3 Performance Study

In this section we study the efficiency of the SR score computation algorithms.

SR Algorithm: We report the average computation time of an SR score (SR-time) using the SR Algorithm and compare it to the average query processing time (Q-time) using PRSM, for the queries in our query workloads. These times are presented in Table 7.5 for K=20. SR-time mainly consists of two parts: the time spent on corrupting K results and the time to re-rank the K corrupted results. We have reported SR-time using a (corruption time + re-rank time) format. We see that the SR Algorithm incurs a considerable time overhead on query processing. This overhead is higher for queries over the INEX data set, because there are only two entity sets, (*person* and *movie*), in the INEX data set, and all query keywords in the query load occur in both entity sets. Hence, according to Equation 7.10, every attribute value in the top K entities will be corrupted, due to the third level of corruption. This does not happen for SemSearch, since SemSearch contains far more entity sets and attributes than INEX.

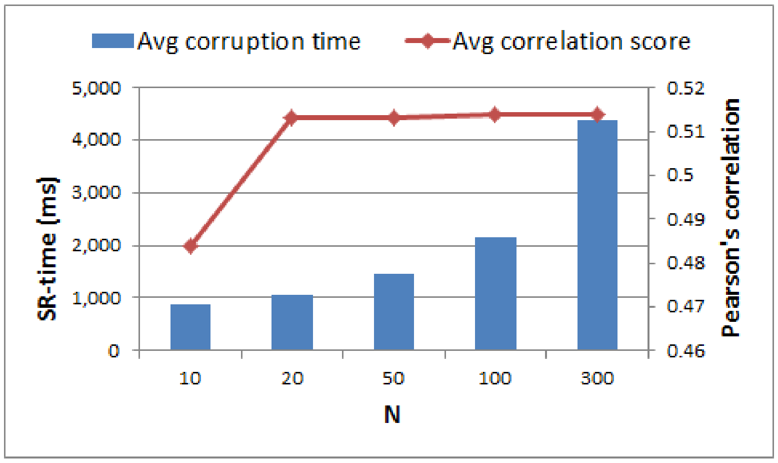


Figure 7.13: Approximations on INEX using a combination of QAO and SGS

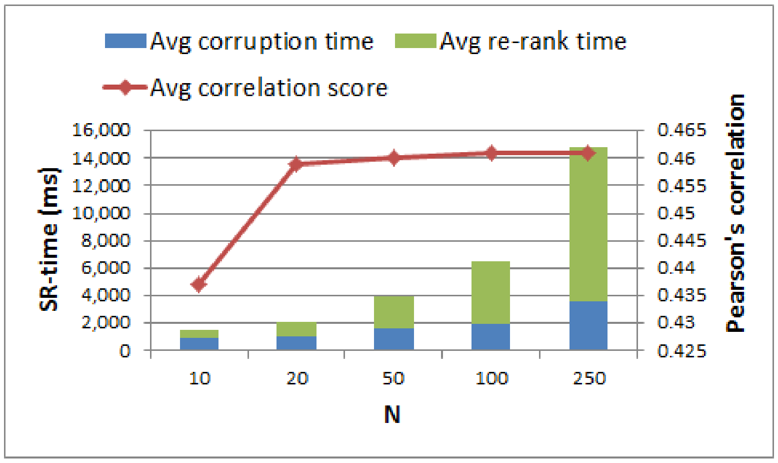


Figure 7.14: Approximations on SemSearch using QAO-Approx

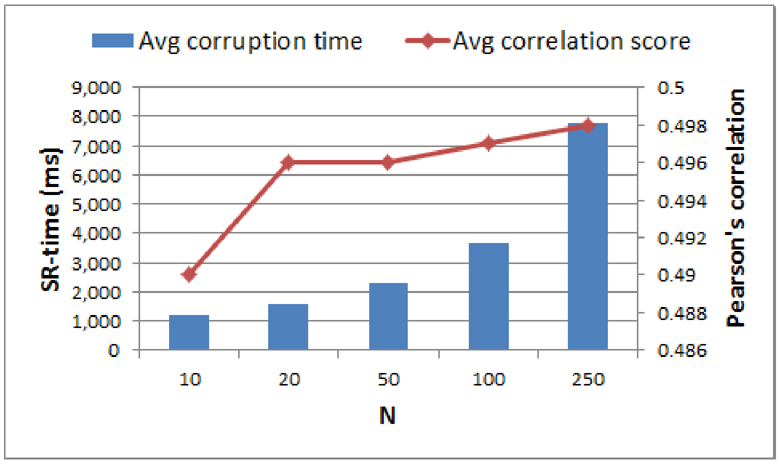


Figure 7.15: Approximations on SemSearch using SGS-Approx

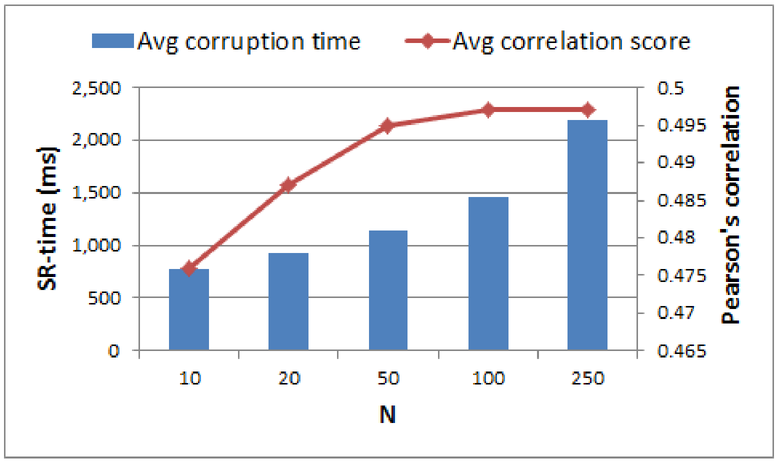


Figure 7.16: Approximations on SemSearch using a combination of QAO and SGS

QAO-Approx: Figures 7.11 and 7.14 show the results of using QAO-Approx on INEX and SemSearch, respectively. We measure the prediction effectiveness for smaller values of N using the average correlation score. The QAO-Approx algorithm delivers acceptable correlation scores and corruption times of about 2 seconds for N=10 on INEX and N=20 on SemSearch. Comparing to the results of the SR Algorithm for N=250 on SemSearch and N=300 on INEX, the correlation score drops, because less noise is added by second and third level corruption. These results show the importance of these two levels of corruption.

SGS-Approx: Figures 7.12 and 7.15 depict the results of applying SGS-Approx on INEX and SemSearch, respectively. Since re-ranking is done on-the-fly during the corruption, SR-time is reported as corruption time only. As shown in Figure 7.12, the efficiency improvement on the INEX data set is slightly worse than QAO-Approx, but the quality (correlation score) remains high. SGS-Approx outperforms QAO-Apporx in terms of both efficiency and effectiveness on the SemSearch data set.

Combination of QAO-Approx and SGS-Approx: As noted in Section 7.5, we can combine QAO-Approx and SGS-Approx algorithms to achieve better performance. Figures 7.13 and 7.16 present the results of the combined algorithm for the INEX and SemSearch databases, respectively. Since we use SGS-Approx, the SR-time consists only of corruption time. Our results show that the combination of two algorithms works more efficiently than either of them alone with the same value for N.

Summary of the Performance Results: According to our performance study of QAO-Approx, SGS-Approx, and the combined algorithm over both data sets, we observe that the combined algorithm delivers the best balance of improvement in efficiency and decrease in effectiveness for both data sets. On both data sets, the combined algorithm can achieve high prediction accuracy (correlation score about 0.5) with SR-time around 1 second. On INEX when the value of N is set to 20, the correlation is 0.513 and the time is decreased to about 1 second using the combined algorithm. For the SR Algorithm on INEX, when N decreases to 10, the correlation score is 0.537, but SR-time is over 9.8 seconds, which is not ideal. If we use the combined algorithm on SemSearch, the correlation score is 0.495 and SR-time is about 1.1 seconds when N=50. However, to achieve similar efficiency, SGS-Approx needs to decrease N to 10, where SR-time is 1.2 seconds and correlation is 0.49. The SR Algorithm spends over 2.2 and 6 seconds of SR-time on the same data set when N is set to 10 and 50, respectively. Thus, the combined algorithm is the best choice to predict the difficulties of queries both efficiently and effectively.

Chapter 8

Toward a Cost Effective Structure for Semi-Structured Data

8.1 Background

Imposing structure on unstructured data sets improves the effectiveness of query answering systems [23, 14, 39, 36, 135]. For instance, suppose that a user wants to find information about the athlete Michael Jordan, and submits keyword query Q_1: *Michael Jordan* to Wikipedia, a fragment of which is shown in Figure 1.1. Many Wikipedia articles contain Q_1's two terms, and the search system does not have enough evidence to determine which article best satisfies the information need behind Q_1. Thus, the search system may give a high rank to articles that do not satisfy the information need behind Q_1, such as an article whose *url* is *wiki/Michael_I._Jordan* rather than *wiki/Michael_Jordan*. Figure 1.2 depicts a more structured representation of the information shown in Figure 1.1. A search system can use the added structure to improve users' experience as they search and explore the data.

However, developing programs to annotate the instances of a concept over a very large data set can be an extremely time-consuming and difficult task [21]. Developers have to identify and implement hundreds of rules or design and train complex machine learning based modules to annotate instances of each concept. Developers also have to maintain the modules and update them as new information is added to the database [105]. These efforts may be even more costly for concepts that are defined in specific domains, such as medicine and the law, as they require collaboration between developers and domain experts.

Since annotation modules are very complex, it can take a very long time and a great deal of computational resources to execute and re-execute them over a very large database [1, 70]. Since the budget and resources of any enterprise are limited, it is vital to identify the concepts whose annotations will most benefit the overall effectiveness of the query system.

In this chapter, we make the following contributions.

- We propose a formal framework that models the improvement in the quality of search over data sets as we add particular pieces of additional semantic information to the data.
- We formally define the problem of cost effective concept annotation and show that it is NP-hard in the general

```
<article>
<url>wiki/Michael_Jordan</url>
<body>
  Michael Jeffrey Jordan
  (born February 17, 1963) is a
  former American professional
  basketball player, ... .
  Michael Jordan had spent
 his entire career with the Chicago Bulls ... .
</body>
</article>

<article>
<url>wiki/Michael_Jordan_statue</url>
<body>
   The Michael Jordan statue, officially
   known as The Spirit, ... .
</body>
</article>

<article>
<url>wiki/Michael_I._Jordan</url>
<body>
    Michael I. Jordan is a leading
    researcher .... . He is currently a full
    professor at the University of California,
    Berkeley... .
</body>
</article>
...
</collection>
```

Figure 8.1: Wikipedia article excerpts

case. We show that some widely used heuristics, such as annotating the most popular concepts, are not optimal in general.

- We propose approximation algorithms that choose which concepts to annotate in pseudo-polynomial time.
- We validate our approach through extensive experiments over standard query workloads and a real-world data set.

8.2 Basic Definitions

A *collection* is a set of unstructured or semi-structured documents. Each document in the collection is identified by a unique identifier, called its *URI*. An *ontology*, $\mathcal{C}$, is a collection of *concepts*. Each concept is a set of named entities

```
<collection>
<article>
<url>wiki/Michael_Jordan</url>
<body>
  <athlete> Michael Jeffrey Jordan </athlete>
   (born
   <birthdate> February 17, 1963 </birthdate>
   ) is a former
   <nationality> American </nationality>
   professional basketball player, ... .
   <athlete> Michael Jordan </<athlete>
   had spent his entire career with the
   <club> Chicago Bulls </club>
    ... .
</body>
</article>

<article>
<url>wiki/Michael_Jordan_statue</url>
<body>
  <art-form> The Michael Jordan statue </art-form>
  , officially known as
  <art-form> The Spirit </art-form>
  ... .
 </body>
</article>

<article>
<url>wiki/Michael_I._Jordan</url>
<body>
  <scientist> Michael I. Jordan </scientist>
   is a leading researcher .... . He is currently a
   <position> full professor </position>
   at the
   <school> University of California, Berkeley </school>
   ... .
</body>
</article>
...
</collection>
```

Figure 8.2: A more structured version of Wikipedia article excerpts

which are its *instances*. Similar to previous work, we refrain from rigorous definition of concepts [32]. Some examples of concepts are *person*, *location*, and *organization*. *Abraham Lincoln* is an instance of concept *person* and *Lincoln Square* is an instance of concept *location*.

Concepts C_1 and C_2 are *mutually exclusive* iff when a named entity e belongs to C_1, e does not belong to C_2 and vice versa. For instance, concepts *person* and *location* are *mutually exclusive*, as no named entity is both a person and a location. In this chapter, we assume that all concepts in an ontology are mutually exclusive. Our models can be extended to other types of relationships such as a concept that is a subclass of another concept. We focus on this type of relationship for two reasons. First, this type of relationship is the basic building block of categorization in ontologies. Second, as we will show later, annotating the instances of the concepts that have this type of relationship is more useful than annotating other types of concepts, for improving the effectiveness of the keyword queries. Modeling all types of relationships between concepts is an interesting subject for future work.

An *annotator* for concept C takes a collection and annotates all instances of C in the collection. $Cost(C)$ is a real-valued function that reflects the amount of resources used to annotate all instances of concept C. These resources could include the amount of time and money spent on developing or training an annotator for c, or the computational resources and time to run the annotator over the collection.

Given keyword query Q, a query system returns a ranked list of documents, where the documents that are deemed to be more relevant to the information need behind Q reside at the top of the list. In this chapter, we consider queries that explicitly or implicitly refer to a named entity that belongs to a concept in our ontology. Users may refer to different named entities by the same name in their queries. For instance, users may use term *Lincoln* to refer to both *Lincoln Square* and *Abraham Lincoln*. Queries that contain such terms are ambiguous and will benefit from concept annotation and extraction. In this chapter, we focus on such queries. To answer queries over an annotated collection, the query system will determine which concept matches the user's information need, for each of the entities referred to in the queries. In other words, the query system will *annotate* the queries.

A query system may use other types of disambiguation when answering a query. For instance, a query system may use entity resolution techniques to handle different ways of referring to the same named entity. These types of disambiguation are not the subject of this chapter.

8.3 Modeling the Benefit of Concepts

8.3.1 Basic Model

The probability ranking principle (PRP) is a widely accepted principle for ranking documents [109, 139]. PRP says that a retrieval system is optimal, i.e., delivers the highest utility for its users, when it ranks documents in decreasing order of their probability of relevance to the query. We used the classical version of PRP, which assumes that the probabilities are computed correctly and there are no dependencies between documents [109], in previous chapters and we also use it here.

Each entry in the relevance information for a given query consists of a document D and the judgment R of whether D is relevant to a particular query Q. R is 0 for non-relevant documents and 1 for relevant documents. The probability that D is relevant to Q is $P(R = 1 \mid D, Q)$, thus, a keyword query system ranks the document in the collection according to $P(R = 1 \mid D, Q)$. For brevity, we abbreviate $R = 1$ to R and $R = 0$ to $\bar{R}$. The effectiveness of a query interface depends on how well it estimates the probability $P(R \mid D, Q)$. How to estimate this probability is beyond the scope of this chapter; our approach works with any reasonable estimate of this probability.

As discussed in above, users may use the same query to refer to named entities that belong to different concepts. Since probability $P(R \mid D, Q)$ does not contain any information regarding the concepts behind queries, it will not precisely reflect the relevance of documents to queries in these settings. To include this information, we rewrite the probability of relevance for document D to query Q which refers to concept C as:

$$P(R \mid D, Q)P(C|D, Q), \tag{8.1}$$

where $P(C|D, Q)$ is the probability that D and Q contain information about the named entities of concept C.

If we annotate the named entities in concept C in documents and queries, without any error, the value of $P(C|D, Q)$ will be 1 iff the named entities referred to in Q and D belong to the same concepts. However, we would like to estimate the effectiveness of a ranker over a collection where the named entities of concept C have not been annotated or organized. In other words, the query interface is not aware of the concepts referred to in D and Q.

It has been well established that most information needs over annotated and organized collections are precision-oriented [36, 23, 7]. Thus, we model the amount of improvement in the ranking quality of the query interface over a set of queries over an annotated and organized collection. We choose *precision at k ($P@k$)*, which is an standard metric for ranking quality of the results of queries, as the objective function in our model [96]. The value of $P@k$ for query Q is the number of relevant answers in the top k answers returned by the query interface for Q, divided by k. We use $P@k$ because it accurately reflects a user's experience. Users usually look at only the very top answers in keyword search. Thus, in empirical studies the value of k is normally small, e.g., 5 or 10. When the value of k is sufficiently small, our results will be independent of the value of k chosen for the $P@k$ metric.

Given a query workload $\mathcal{Q}$, we compute the expectation of $P@k$ as:

$$\mathtt{E}[P@k] = \sum_{C \in \mathcal{C}, Q \in \mathcal{Q}} u(C, Q) \frac{1}{k} \sum_{\mathcal{K}} P(R \mid D, Q)P(C|D, Q) \times R, \tag{8.2}$$

where $\mathcal{C}$ is the ontology, $u(C, Q)$ is the probability that query Q refers to entities of concept C in $\mathcal{Q}$, $\mathcal{K}$ is the set of top k documents returned by the query interface, and R is equal to one.

Probability $P(R \mid D, Q)$ models the relevance of D and Q given features and evidence which are not related to the concepts of the named entities in Q and D. This evidence may be traditional TF-IDF features or features learned from the query log [96]. Since we focus on ambiguous queries, it is reasonable to assume that there are at least a relatively small number of documents that have large and almost equal values for $P(R \mid D, Q)$. Otherwise, annotations about the concepts referred to in queries or documents may not be needed, as other features can identify the relevant documents for Q. Thus, since the value of k is normally very small, we can assume that for all documents in $\mathcal{K}$:

$$P(R \mid D, Q) \approx p.$$

We rewrite formula 8.2 as:

$$\begin{aligned}\mathtt{E}[P@k] &= \sum_{C\in\mathcal{C},Q\in\mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} pP(C|D,Q)\\ &= p\sum_{C\in\mathcal{C},Q\in\mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} P(C|D,Q).\end{aligned}$$

Since the query answering system does not have any information regarding the concepts in documents, it is reasonable to assume that it randomly picks documents that contain information about the various concepts corresponding to the named entity mentioned in Q. We estimate the value of $P(C|D,Q)$ as $d(C)$, which is the probability of randomly selecting a document that contains information about the named entity from concept C. We rewrite the expected value of $P@k$ accordingly as:

$$\begin{aligned}\mathtt{E}[P@k] &= p\sum_{C\in\mathcal{C},Q\in\mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} d(C) \\ &= p\sum_{C\in\mathcal{C},Q\in\mathcal{Q}} u(C,Q)d(C)\\ &= p\sum_{C\in\mathcal{C}} d(C)\sum_{Q\in\mathcal{Q}} u(C,Q)\\ &= p\sum_{C\in\mathcal{C}} u(C)d(C).\end{aligned} \tag{8.3}$$

Assume that we annotate all entities of concepts in set $\mathcal{S} \subseteq \mathcal{C}$ in the collection and the queries in $\mathcal{Q}$. The query interface will answer the annotated and unannotated keyword queries differently. If a query is annotated with concepts in S, since the query interface knows the documents that contain information about these concepts, it will process the query over only these documents. We can rewrite equation 8.2 for annotated queries as:

$$\begin{aligned}\mathtt{E}[P@k] &= \sum_{C\in\mathcal{S},Q\in\mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} P(R \mid D,Q)\times 1\times 1 \\ &= \sum_{C\in\mathcal{S},Q\in\mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} p\\ &= p\sum_{C\in\mathcal{S}} u(C)\end{aligned} \tag{8.4}$$

For queries that are not annotated, we have:

$$\begin{aligned}
\mathtt{E}[P@k] &= \sum_{C \notin \mathcal{S}, Q \in \mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} P(R \mid D,Q)P'(C|D,Q) \times 1 \\
&= \sum_{C \in \mathcal{S}, Q \in \mathcal{Q}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} pP'(C|D,Q) \\
&= p\sum_{C \in \mathcal{S}} u(C,Q)\frac{1}{k}\sum_{\mathcal{K}} pP'(C|D,Q),
\end{aligned}$$

where $P'(C|D,Q)$ is the probability of selecting a document that contains information about concept $C \notin \mathcal{S}$. Since text documents are coherent, they do not usually contain information about named entities with the same name but from different and mutually exclusive concepts. For instance, it is very unlikely to find a document that contains information about both *Jaguar* the vehicle and *Jaguar* the animal. Therefore, we assume that if a document contains information about the desired entity of a query, it will not contain any information about other entities with the same name but from other concepts.

Since the concepts in $\mathcal{S}$ and the concepts that are not in $\mathcal{S}$ are mutually exclusive, the query system can ignore documents that match the named entity mentioned in the query if their value of $P(R|D,Q)$ is relatively large, but their matched named entities are annotated as instances of concepts of S. Hence, we estimate the value of $P'(C|D,Q)$ as the probability of randomly selecting a document that contains information about the entity mentioned in Q, which belongs to C, from the set of all documents that match the entity in Q and refer to concepts that are not in S, i.e., are not annotated instances. Thus, we have:

$$P'(C|D,Q) \approx \frac{d(C)}{\sum_{C \notin \mathcal{S}} d(C)}.$$

Therefore, the expectation of $P@k$ over both annotated and unannotated queries and documents is:

$$\mathtt{E}[P@k] = p(\sum_{C \in \mathcal{S}} u(C) + \frac{\sum_{C \notin \mathcal{S}} u(C)d(C)}{\sum_{C \notin \mathcal{S}} d(C)}). \tag{8.5}$$

Our goal is to explore the improvement in the expected value of $P@k$ for annotations of different sets of concepts and queries, and find the annotation that delivers the largest improvement. Since the value of p does not change across annotations of different sets of concepts, we ignore it in our analyses and algorithms.

Definition 8.3.1. *Given ontology $\mathcal{C}$, collection L, query workload $\mathcal{Q}$, and a set of concepts $\mathcal{S} \subseteq \mathcal{C}$, the annotation benefit of S is:*

$$AB(\mathcal{S}) = \sum_{C \in \mathcal{S}} u(C) + \frac{\sum_{C \notin \mathcal{S}} u(C)d(C)}{\sum_{C \notin \mathcal{S}} d(C)}. \tag{8.6}$$

The value of an annotation benefit reflects the improvement in the expected value of $P@k$ achieved by annotating all instances of concepts in set $S \subseteq \mathcal{C}$.

One of the main questions about the extraction is when it is helpful to annotate documents for a concept. The following proposition states that annotation always increases the expected value of precision at k, under the assumption that annotation is error-free.

Proposition 8.3.2. *The annotation benefit for any non-empty set of concepts, $\mathcal{S} \neq \emptyset$, is greater than the annotation benefit for an empty set of concepts.*

Proof. In formula (8.4) (ignoring p), the value of $u(C)$, $C \in \mathcal{S}$ is always multiplied by a number less than 1. However, in formula (8.6), the same value is multiplied by 1. For concepts $C' \notin S$, the value of their multipliers in formula (8.6) is greater than the value of their multipliers in formula (8.4). Hence, we have $AB(\mathcal{S}) > AB(\emptyset)$ □

8.3.2 Errors in Annotating

Annotators generally make mistakes in finding the concepts in the text of documents in a collection. We should consider these errors to provide a realistic estimate of the impact of annotation on the effectiveness of queries. For every $C \in \mathcal{S}$, let $|C|$, $TP(C)$, and $FP(C)$ denote the total number of instances, the number of true positive annotations, and the number of false positive annotations for concept C in the collection, respectively. If the query system correctly detects the desired concept behind query Q whose entity belongs to C, then the probability of finding the desired entity for Q in the annotated instances of C is:

$$\frac{TP(C)}{|C|} \times \frac{TP(C)}{TP(C) + FP(C)} + (1 - \frac{TP(C)}{|C|}) \times 0. \tag{8.7}$$

The *recall* of the annotation of concept C, denoted as $r(C)$, is $\frac{T-P(C)}{|C|}$. Similarly, $p(C)$ denotes the *precision* of the annotation for concept C, which is $\frac{TP(C)}{TP(C)+FP(C)}$. We rewrite formula (8.7) based on the value of recall and precision of C as $r(C) \times p(C)$. Hence, the annotation benefit for the annotated concepts will be $\sum_{C \in \mathcal{S}} u(C)r(C)p(C)$.

Since the annotation processes for the concepts in S might not identify and annotate all instances of these concepts, some of these instances end up in the unannotated collection. We denote the probability of having such instances in the collection as $fn(\mathcal{S})$. Similarly, the annotation programs for the concepts in $\mathcal{S}$ might mistakenly include some entities that belong to concepts such as $C' \notin \mathcal{S}$ in the annotated collection. The probability of having these instances in the collection is $fp(C')$. Considering these factors, we rewrite formula (8.6) as:

$$AB(\mathcal{S}) = \sum_{C \in \mathcal{S}} u(C)r(C)p(C) + \frac{\sum_{C \notin \mathcal{S}} u(C)d(C)fp(C)}{\sum_{C \notin \mathcal{S}} d(C) + fn(\mathcal{S})}. \tag{8.8}$$

The value of $fn(\mathcal{S})$ is:

$$fn(\mathcal{S}) = \sum_{C \in \mathcal{S}} (1 - r(C))d(C).$$

The process of entity annotation and extraction is generally noisy but informative, and the recall values for annotation are relatively large [38]. Each collection may contain instances of a large number of concepts, but the available annotation programs can cover a limited number of such concepts. In other words, the value of $\sum_{C \notin \mathcal{S}} d(C)$ is larger than $\sum_{C \in \mathcal{S}} d(C)$. Therefore, the value of $fn(\mathcal{S})$ is generally very much smaller than the value of $\sum_{C \notin \mathcal{S}} d(C)$. In order to simplify the model, we ignore the $fn(\mathcal{S})$ in our model.

The portion of mistakenly annotated instances of concept $C \in \mathcal{S}$ is $(1 - p(C))d(C)$, and its total value for all concepts in $\mathcal{S}$ is $\sum_{C \in \mathcal{S}} (1 - p(C))d(C)$. It is reasonable to assume that the portion of these instances whose concept is $C' \notin \mathcal{S}$ is:

$$d(C') \sum_{C \in \mathcal{S}} (1 - p(C))d(C).$$

Thus, we have:

$$fp(C') = 1 - \sum_{C \in \mathcal{S}} (1 - p(C))d(C).$$

Using a similar argument as for $fn(\mathcal{S})$, we ignore the value of $fp(C)$ in our model.

8.4 The Optimal Concept Annotation Problem

Since the resources available to develop, maintain, and execute concept annotators are limited, our goal is to find a set of concepts $\mathcal{S}$ such that annotating them in the queries and collection will maximize the annotation benefit. We define the cost of annotation for concept $C \in \mathcal{C}$ to reflect the amount of resources spent on annotating instances of concept C in the collection. Let a fixed number $B > 0$ denote the amount of resources available to perform the annotation. Annotating a set of concepts $\mathcal{S}$ is feasible if $\sum_{C \in \mathcal{S}} w(C) \leq B$. We formally define the problem of *optimal concept annotation* as follows:

Problem 8.4.1. *Given a collection of concepts $\mathcal{C}$, where each concept has cost of annotation $w(C)$, precision $p(C)$, and recall $r(C)$, we aim to annotate a set of concepts $\mathcal{S} \subset \mathcal{C}$ such that the value of annotation benefit $AB(\mathcal{S})$ is maximized, subject to the constraint that $\sum_{C \in \mathcal{S}} w \leq B$, $B > 0$,* The obvious first step toward a solution for the optimal concept annotation problem is to adopt a greedy solution, i.e., pick the concepts with the largest value of $u(C)$. However, the following example shows this idea is misleading.

Example 8.4.2. *Consider three concepts C_1, C_2, C_3 with perfect precision and recall and the following u and d values:*

	C_1	C_2	C_3
u	*0.5*	*0.4*	*0.1*
d	*0.8*	*0.15*	*0.05*

Although C_1 has both greater u and d value, annotating C_2 is more helpful.

$$\begin{aligned} E_1 &= 0.5 + \frac{1}{1-0.8}(0.4 \times 0.15 + 0.1 \times 0.05) \\ &= 0.825 \end{aligned}$$

By annotating concept C_3, the value of the annotation benefit becomes:

$$\begin{aligned} E_2 &= 0.4 + \frac{1}{1-0.15}(0.5 \times 0.8 + 0.1 \times 0.15) \\ &= 0.876 \end{aligned}$$

The justification could be that the order of d for C_2 and C_3 is the same. Nevertheless, C_1 has considerably higher d than the other two. After extracting C_1, C_2 and C_3 would still be indistinguishable, while annotating C_2 helps C_1 to be easily accessible as well.

8.4.1 NP Hardness

We prove that the optimal concept annotation problem is NP-hard by a reduction from an NP-complete variant of the partition problem [81]:

Problem 8.4.3. *Given $2m$ positive integers $a_1, \ldots, a_{2m}$ that sum up to $2A$, such that $\frac{A}{m+1} < a_k < \frac{A}{m-1}$, $1 \leq k \leq 2m$, decide whether there exists an index set K such that $\sum_{k \in K} a_k = A$ holds.*

Theorem 8.4.4. *Problem 8.4.3 is reducible to the optimal concept annotation problem in polynomial time.*

Proof. Given an instance of problem 8.4.3, we construct an instance of the optimal concept annotation problem with $2m$ concepts, where for each concept $C_i \in \mathcal{C}$, $1 \leq i \leq 2m$, $w_i = u_i = a_i$, $d_i = 1$, and $B = A$. Let $\mathcal{S}$ be the desired answer to our optimal concept annotation problem instance and $S = \sum_{C_i \in \mathcal{S}} u_i$. Since the budget limit for our instance is A and $w_i = u_i = a_i$, we have $S \leq A$. Since for all u_is, we have $u_i > \frac{A}{m+1}$, then $|\mathcal{S}| \leq m$, where $|\mathcal{S}|$ is the size of set $\mathcal{S}$. We can write the objective function of our optimal concept annotation problem instance as $AB(\mathcal{S}) = S + \frac{2A-S}{2m-|\mathcal{S}|}$. Since $AB(\mathcal{S})$ is an increasing function of S and $|S|$, it obtains its maximum value when $S = A$ and $|\mathcal{S}| = m$. Since we have $\frac{A}{m+1} < u_i = a_i < \frac{A}{m-1}$, given $\sum_{C \in \mathcal{S}} u = A$, we have $|\mathcal{S}| = m$. Hence, this solution is a feasible one for the instance of the optimal concept annotation problem. This implies that the instance of the partition problem will have a solution iff the corresponding instance of the optimal concept annotation problem has a solution of value $A(1 + 1/m)$. □

Since problem 8.4.3 is NP-complete, the optimal concept annotation problem is NP-hard.

8.4.2 Special Cases

It is helpful to get a sense of the optimal solution by examining some special cases. In this section, we consider the case where all concepts have equal cost of annotation, and have perfect precision and recall. We propose exact algorithms for some of these cases. Since all concepts have equal annotation costs, the goal is to find the m concepts whose annotations will maximize the value of the annotation benefit over the collection.

One of the cases that might occur is the situation where a user accesses different concepts with equal likelihood. In this setting, the following proposition holds.

Proposition 8.4.5. *Given the values of $u(C)$ are equal for all $C \in \mathcal{C}$, annotating any set of concepts of size m will maximize the annotation benefit value.*

Proof. Let $u(C) = p$, $C \in \mathcal{C}$, where p is a fixed constant. Before annotation, the value of the annotation benefit is:

$$AB = \sum_{C \in \mathcal{C}} u(C)d(C) = p \sum_{C \in \mathcal{C}} d(C) = p.$$

After annotating an arbitrary set $\mathcal{S}$ of concepts, the value of the annotation benefit will be equal to:

$$\begin{aligned} AB(S) &= \sum_{C \in \mathcal{S}} u(C) + \frac{\sum\limits_{C \notin \mathcal{S}} u(C)d(C)}{\sum\limits_{C \notin \mathcal{S}} d(C)} \\ &= mp + \frac{p}{\sum\limits_{C \notin \mathcal{S}} d(C)} \sum_{C \notin \mathcal{S}} d(C) \\ &= (m+1)p. \end{aligned}$$

□

This means that in this case, the annotation benefit depends only on the number of annotated concepts, m. We can justify this fact by considering two effects of extraction on the annotation benefit. When concept C is annotated, it helps in answering the queries whose entities belong to C. Furthermore, after annotating entities of concept C, we will increase the value of $d(C')$ for all concepts $C' \in \mathcal{C}$, $C' \neq C$. If $d(C)$ is large, the most of the benefit of the annotation comes from helping other concepts. If $d(C)$ is low, the amount of benefit we get out of annotation is due to the improvement in the effectiveness of queries whose entities belong to concept C.

Another case of interest is when the values of d are the same for all concepts in $\mathcal{C}$. This means that all concepts have the same number of instances in the collection.

Proposition 8.4.6. *Given that the values of $d(C)$ are equal for all $C \in \mathcal{C}$, choosing m concepts with the largest values of u provides the maximum annotation benefit.*

Proof. Let $d(C) = p$, $C \in \mathcal{C}$, where p is a fixed constant. Before annotation, the value of the annotation benefit is:

$$AB = \sum_{C \in \mathcal{C}} u(C)d(C) = p.$$

After annotating an arbitrary set $\mathcal{S}$ of concepts, the value of the annotation benefit will be equal to:

$$\begin{aligned} AB(S) &= \sum_{C \in \mathcal{S}} u(C) + \frac{\sum\limits_{C \notin \mathcal{S}} u(C)d(C)}{\sum\limits_{C \notin \mathcal{S}} d(C)} \\ &= \sum_{C \in \mathcal{S}} u(C) + \frac{p}{1 - mp} \sum_{C \notin \mathcal{S}} u(C) \\ &= u(\mathcal{S}) + \frac{p}{1 - mp}(1 - u(\mathcal{S})) \\ &= u(\mathcal{S})(1 - \frac{p}{1 - mp}) + \frac{p}{1 - mp}. \end{aligned}$$

So, the greater $u(\mathcal{S})$ is, the greater annotation benefit we get. If we want to annotate a set of concepts $\mathcal{S}$ of size m from $\mathcal{C}$ to maximize the annotation benefit, we should choose the concepts with the m highest u values. □

8.4.3 Choosing Between Two Concepts

Now, we examine carefully how the annotation benefit behaves in the general case, without any assumption regarding $u(C)$ and $d(C)$, when the cost of annotation is the same for all concepts and we are only allowed to anotate *one* concept. By annotating documents with the concept C_s, we have:

$$\begin{aligned}
AB(C_s) &= u(C_s) + \sum_{C \neq C_s} u(C)d'(C) \\
&= u(C_s) + \frac{1}{1-d(C_s)} \times \sum_{C \neq C_s} u(C)d(C) \\
&= u(C_s) + \frac{M - u(C_s)d(C_s)}{1-d(C_s)} \\
&= M \times \frac{1 - \frac{u(C_s)}{M}d(C_s)}{1-d(C_s)} + u(C_s)
\end{aligned}$$

where $M = \sum_{C \in \mathcal{C}} u(C)d(C)$. Hence, we can state:

Proposition 8.4.7. *For two concepts C_1 and C_2 in $\mathcal{C}$, where $d(C_1) = d(C_2) = p$ and $u(C_1) > u(C_2)$, annotating C_1 is more beneficial than annotating C_2.*

Proof. The annotation benefit of C_1 is

$$\begin{aligned}
AB(C_1) = {} & u(C_1) \\
& + \frac{1}{1-p}(\sum_{C \notin \{C_1, C_2\}} u(C)d(C) + u(C_2)d(C_2)),
\end{aligned}$$

and for the concept C_2 is:

$$\begin{aligned}
ABC_2 = {} & u(C_2) \\
& + \frac{1}{1-p}(\sum_{C \notin \{C_1, C_2\}} u(C)d(C) + u(C_1)d(C_1))
\end{aligned}$$

So, by letting $A = \sum\limits_{C \notin \{C_1, C_2\}} u(C)d(C)$, we can write $AB(C_1) - AB(C_2)$ as the following:

$$\begin{aligned}
AB(C_1) - AB(C_2) &= u(C_1) + \frac{1}{1-p}(A + u(C_2)p) \\
&\quad - u(C_2) - \frac{1}{1-p}(A + u(C_1)p) \\
&= u(C_1)(1 - \frac{p}{1-p}) - u(C_2)(1 - \frac{p}{1-p}) \\
&= u(C_1) - u(C_2))(\frac{1-2p}{1-p}) \\
&> 0
\end{aligned}$$

□

Proposition 8.4.8. *For two concepts C_1 and C_2 where $u(C_1) = u(C_2) = p$ and $d(C_1) > d(C_2)$, depending on the value of M, one of the following cases holds:*

- $p \leq M$. $AB(C_1) \geq AB(C_2)$.
- $p \geq M$. $AB(C_1) \leq AB(C_2)$.

Proof. The annotation benefit of C_1 is

$$AB(C_1) = p + \frac{1}{1-d(C_2)}(\sum_{C \notin \{C_1, C_2\}} u(C)d(C) + pd(C_2)),$$

and for the concept C_2 is:

$$ABC_2 = p + \frac{1}{1-d(C_1)}(\sum_{C \notin \{C_1, C_2\}} u(C)d(C) + pd(C_1)).$$

So, by letting $A = \sum_{C \notin \{C_1, C_2\}} u(C)d(C)$, we can write $AB(C_1) - AB(C_2)$ as:

$$\begin{aligned}
AB(C_1) - AB(C_2) &= p + \frac{1}{1-d(C_1)}(A + d(C_2)p) \\
&- p - \frac{1}{1-d(C_2)}(A + d(C_1)p) \\
&= \frac{1}{(1-d(C_1))(1-d(C_2))} \times \{A(d(C_1) \\
&- d(C_2)) + p(d(C_2) - d(C_1) + d(C_1)^2 - d(C_2)^2)\} \\
&= \frac{d(C_1) - d(C_2)}{(1-d(C_1))(1-d(C_2))} \\
&\times (A + p(d(C_1) + d(C_2)) - p) \\
&= \frac{d(C_1) - d(C_2)}{(1-d(C_1))(1-d(C_2))} \times (M - p)
\end{aligned}$$

So, $AB(C_1) > AB(C_2)$ iff $M > p$. □

We observe an unintuitive result here, because we expect that a higher u value always guarantees a greater annotation benefit. Consider this example:

Example 8.4.9. *Assume three different concepts $\{C_1, C_2, C_3\}$ with the following u and d values. The goal is to annotate documents of a concept to get the highest annotation benefit.*

	C_1	C_2	C_3
u	*0.2*	*0.4*	*0.4*
d	*0.3*	*0.3*	*0.4*

At first look, one may consider C_3 to be the best choice, since both its u and d are dominant. But this is not correct, and we need to examine it more carefully. The annotation benefit C_2 is:

$$\begin{aligned} AB(C_2) &= 0.4 + \frac{1}{1-0.3}(0.2 \times 0.3 + 0.4 \times 0.4) \\ &= 0.4 + \frac{0.22}{0.7} \\ &> 0.4 + 0.3 = 0.7 \end{aligned}$$

By extracting concept C_3, the value of the expected precision at k becomes,

$$\begin{aligned} AB(C_3) &= 0.4 + \frac{1}{1-0.4}(0.2 \times 0.3 + 0.4 \times 0.3) \\ &= 0.4 + \frac{0.18}{0.6} \\ &= 0.4 + 0.3 = 0.7 \end{aligned}$$

This holds because the current initial value of expected precision at k, M, is $0.2 \times 0.3 + 0.4 \times 0.3 + 0.4 \times 0.4 = 0.32 < u(C_2) = u(C_2) = 0.4$. *In fact, it can be derived from Proposition 8.4.7.*

8.5 Efficient Algorithms

8.5.1 Largest-U Greedy Algorithm

One greedy algorithm we can apply to solve the optimal annotation problem is *Largest-U* (LU) greedy. LU sorts concepts in non-increasing order of their u values, and checks them one by one. For each concept C_i, if by picking C_i the total cost of the selected concepts does not exceed B, it picks C_i and proceeds to the next concept; otherwise, it omits C_i and goes on to the next concept. We can show that LU has a reasonable approximation ratio for some variants of the problem.

Theorem 8.5.1. *The LU algorithm is an approximation for the optimal concept annotation problem with ratio* $\frac{w_{min}}{w_{max}} \times \frac{B}{B+w_{min}}$.

Proof. We use the method of Gal et al. [48] to analyze the algorithm. Let w_{max} and w_{min} be respectively the maximum and minimum value of $w_i, i \leq n$. It is clear that LU at least picks the $\frac{B}{w_{max}}$ concepts with largest u. On the other hand, the optimal solution cannot pick more than $\frac{B}{w_{min}}$ concepts of $\mathcal{C}$. If the optimal solution picks l concepts, we will have $l+1$ terms in the annotation benefit formula (l for extracted concepts and one for the mixed one). The total benefit of these terms is less than the sum of the $l+1$ largest values of u. Since the annotation benefit of the output of LU is more than the sum of $\frac{B}{w_{max}}$, the concepts with the largest value of u, $AB(S_{LU}) > \frac{\frac{B}{w_{max}}}{\frac{B}{w_{min}}+1}\text{OPT} = \frac{w_{min}}{w_{max}} \times \frac{B}{B+w_{min}}\text{OPT}$. □

Corollary 8.5.2. *For the case that the annotation cost of all concepts is equal, LU is an $\frac{B}{B+1}$-approximation algorithm.*

Furthermore, we can show that the obtained ratio for LU in Theorem 8.5.1 is sharp. Consider the following example, similar to Gal et al. [48]. A set $\mathcal{C}$ of $n+1$ concepts with the following u_i and d_i is given:

- $u_i = 1-(i-1)\epsilon, 1 \le i \le n; u_{n+1} = 0,$
- $d_i = (1-\epsilon-\epsilon^2)/(m-1), 1 \le i \le n-1; d_n = \epsilon^2; d_{n+1} = \epsilon,$

Here, ϵ is a small positive number. We set w_i equal to one for all concepts in $\mathcal{C}$. For $B = n$, the value of LU is close to n, while the optimal solution that selects concepts $C_2, \cdots, C_n$ achieves a benefit of about $n+1$.

8.5.2 Ratio Greedy Algorithm

Consider the following greedy algorithm for the optimal concept annotation problem. Pick concepts according to the ratio $r_i = \frac{u_i}{w_i}$, from the largest to the smallest. If r_m is the first place where the total cost of the selected concepts exceeds the budget B, we compare the annotation benefit of the set $\{r_1, \cdots, r_{m-1}\}$ to the concept with maximum value of u, u_{max}, and return the set with greater annotation benefit. We call this algorithm Largest Ratio (LR) greedy.

Remark. The algorithm does not really stop at the first place that the total cost of the selected concepts exceeds the budget. It proceeds with the concepts until it traverses all concepts.

Theorem 8.5.3. *LR is a $\frac{1}{3}$-approximation algorithm for the annotation benefit optimization problem.*

Proof. After annotating documents of l concepts, we have two types of concepts: *extracted* and *mixed*. By extracted concepts, we mean the set of concepts we have selected. The phrase *mixed concepts* refers to the set of concepts that remain after removing the selected concepts. We have two types of benefits; one is related to extracted concepts and the other is obtained by the mixed part. Suppose C_m is the first concept that LR cannot pick. It is clear that the benefit of the extracted part is less than $\sum_{i=1}^{m} u_i$, and we know that in all cases, the benefit of the mixed part is less than u_{max}. So, $\sum_{i=1}^{m-1} u_i + u_m + u_{max} >$ OPT. Since $u_m < u_{max}$, $\sum_{i=1}^{m-1} u_i + 2u_{max} >$ OPT. Hence, $\max(\sum_{i=1}^{m-1} u_i, u_{max}) > \frac{1}{3}$OPT. This shows that LR is a $\frac{1}{3}$-approximation algorithm. □

We are required to sort the concepts by their ratio; then we can pick the concepts by traversing them in linear time. Thus the running time of LR is $O(n \log n)$, where n is the number of concepts.

	T_1	T_2
u	$M_1, \cdots, M_r$	$m_1, \cdots, m_s$
d	$D_1, \cdots, D_r$	$d_1, \cdots, d_s$
w	$W_1, \cdots, W_r$	$w_1, \cdots, w_s$

Table 8.1: The example for the worst case approximation ratio of LR

Furthermore, we can show that the worst case approximation ratio of LR is no more than $1/2$. Consider a set of concepts $\mathcal{C} = \{C_1, \ldots, C_{m+1}\}$ such that $u_1 = d_1 = w_1 = m$ and for all $i > 1$, $u_i = 1 - \varepsilon$ and $d_i = w_i = 1$. LR will pick C_1 and its benefit is $m + 1 - \varepsilon$. But the optimal solution can be obtained by picking $C_2, \ldots, C_{m+1}$, giving $m(2 - \varepsilon)$. By choosing m to be arbitrarily large, the approximation ratio of the LR algorithm is $1/2$. We can generalize this case as follows. Given the input concepts that are partitioned in mutually exclusive sets T_1 and T_2 as shown in Table 8.1, let $B = \sum_{i \le r} W_i = \sum_{j \le s} w_j$ and $m_j/w_j > M_i/W_i$ for $j \le s, i \le r$. Furthermore, $m_j \ll M_i$ for $j \le s, i \le r$. Thus, we have: $r \ll s$. LR picks concepts of type T_1 for this example, and its annotation benefit is $r \times M + m$. We obtain the annotation benefit of about $s \times m + M$ by annotating concepts of type T_2. The approximation ration of this example is $(r \times M + m)/(s \times m + M) \approx r/(r+1) + m/((r+1)M) \approx r/(r+1)$. By setting r to a sufficiently small value, the approximation ratio of LR becomes sufficiently small.

8.5.3 Psuedo-polynomial Time Algorithm

In this section, we present a pseudo-polynomial time algorithm for the optimal benefit annotation problem. An algorithm is pseudo-polynomial if its running time is polynomial in the size of the unary encoding of the input. Our algorithm takes advantage of the dynamic programming method for the 0-1 knapsack problem. **Without loss of generality, we assume that $u(C_i)$ and $d(C_i)$ are positive integers for all concepts**. We introduce a two-variable function to solve the problem. We define function $f(D, \mathcal{C}_S)$ as follows:

$$f(D, \mathcal{C}_S) = \frac{1}{D} \times (D \sum_{C_i \in \mathcal{C}_S} u(C_i) + \sum_{C_i \notin \mathcal{C}_S} u(C_i) d(C_i))$$

For a set of concepts $\mathcal{C}_S$ such that $\sum_{C_i \notin \mathcal{C}_S} d(C_i) \le D$, $f(D, \mathcal{C}_S) \le AB(\mathcal{C}_S)$. We define the domain of f to be the set of concepts $\mathcal{C}$ such that $\sum_{d_i \notin \mathcal{C}} d(C_i) \le D$. For a fixed value of D, f is a separable function. Thus it is straightforward to maximize f for a fixed D. The following lemma shows that the maximum value of $f()$ over all possible values of D is the same as the maximum value of $AB()$.

Lemma 8.5.4. *The maximum value of function $f()$ on its domain and $AB()$ on all possible sets is the same.*

Proof. Suppose that function $AB()$ obtains its maximum value at $\mathcal{C}_{S^*}$. Let $D^* = \sum_{C \in \mathcal{C}_{S^*}} d(C_i)$. Then, $f(D^*, \mathcal{C}_{S^*}) = AB(\mathcal{C}_{S^*})$. Thus, $\max_{D,\mathcal{C}} f(D, \mathcal{C}) \ge \max_{\mathcal{C}} AB(\mathcal{C})$. For the other direction, we know that $f(D, \mathcal{C}) \le AB(\mathcal{C})$ for all

$\mathcal{C}$ in the domain of $f()$ with respect to D. This implies that $\max_{D,\mathcal{C}} f(D,\mathcal{C}) \leq \max_{\mathcal{C}} AB(\mathcal{C})$.

So, $\max_{D,\mathcal{C}} f(D,\mathcal{C}) = \max_{\mathcal{C}} AB(\mathcal{C})$. It also implies that the set $\mathcal{C}$ that achieves the maximum value of AB obtains the maximum value of f with proper choice of D, as well. $\square$

For maximizing function f we can apply dynamic programming. Let $V_{init}(D) = \sum_{i \leq n} u(C_i)d(C_i)/D$ be the value of function f before annotating any concept ($\mathcal{C}_S = \emptyset$). We can rewrite the formulation of f as

$$f(D,\mathcal{C}_S) = \sum_{C_i \in \mathcal{C}_S} v(C_i) + V_{init},$$

where $v(C_i) = u(C_i)(1 - d(C_i)/D)$. In our algorithm we should take care of two constraints, $\sum_{C_i \in S} d(C_i) \geq D_{total} - D$ and $\sum_{C_i \in S} w(C_i) \leq B$, where $D_{total} = \sum_{i \leq n} d(C_i)$.

We define $M[i, P, X]$ to be the minimum required cost we should pay to obtain a benefit of at least P of $f(D,\mathcal{C})$, with respect to constraint $\sum_{C_i \in S} d(C_i) > D'$, by using concepts $C_1, \ldots, C_i$. For a fixed value of D, we can introduce a recursive definition of $M[i, P, X]$ as follows:

1. $M[0, P, 0] = 0$ for all $P \leq V_{init}(D)$
2. $M[0, P, X] = \infty$ if $X > 0$ or $P > V_{init}(D)$
3. $M[i, P, X] = \min(M[i-1, P, X],$
   $M[i-1, P - v(C_i), X - d(C_i)] + w(C_i))$

Then we need to traverse $M[n, P, D_{total} - D]$ to find the maximum value of P such that $M[n, P, D_{total} - D] \leq B$.

The running time of the dynamic program for a fixed D is $O(nVD)$, where V is the maximum possible benefit. Thus the overall running time (considering all possible values of D) is $O(nPD_{total}^2)$. This algorithm will be significantly faster than the brute force algorithm when the number of concepts is relatively large. The optimal answer to the optimal concept annotation problem will remain the same if we scale the costs and budget limit. Therefore, we can rescale these values to improve the running time of the pseudo-polynomial algorithm.

8.6 Experiments

8.6.1 Experiment Setting

We used INEX-2009 and INEX-2010 adhoc track data sets, query workloads, and query assessments to evaluate the quality of our algorithms [52, 51]. These benchmarks use a semantically annotated version of the Wikipedia collection [43], which is 50.7 GB and contains 2,666,190 articles. The semantically annotated version of the Wikipedia collection

Query ID	Original Query ID	Query	Query with Annotated Concepts
1	2009002	best movie	best award:movie
2	2009036	notting hill film actors	movie:notting movie:hill film actors
3	2009042	sun java	company:sun java
4	2009055	european union expansion	economy:european economy:union expansion
5	2009066	folk metal groups finland	music_genre:folk music_genre:metal groups country:finland
6	2009069	singer in Britain's got talent	singer in series:Britain's got series:talent
7	2009074	web ranking scoring algorithm	invention:web ranking scoring algorithm
8	2009089	world wide web history	invention:world invention:wide invention:web history
9	2009110	paul is dead hoax theory	person:paul is theory:dead hoax theory
10	2009113	Toy Story Buzz Lightyear 3D rendering Computer Generated Imagery	movie:Toy movie:Story series:Buzz series:Lightyear 3D rendering Computer Generated Imagery
11	2010003	monuments of india	monuments of country:india
12	2010027	paul auster	person:paul person:auster
13	2010033	collapse of housing bubble usa	collapse of agency:housing bubble country:usa
14	2010045	messi argentine	person:messi country:argentine
15	2010050	novels that won booker prize	novels that won award:booker award:prize
16	2010054	indian railways	company:indian company:railways
17	2010056	cold war	cold conflict:war
18	2010075	volcanic ash cloud flight disruption	volcanic ash cloud accident:flight disruption

Table 8.2: Selected queries from the INEX-2009 and INEX-2010 query workloads

uses a schema based on the concepts from WordNet, from *wordnet.princeton.edu*. The query workload consists of keyword queries.

To evaluate our algorithms, we should identify the concepts behind the query terms in keyword queries. We observed that the terms in most queries appear in more than one concept in the relevant documents. In order to simplify our evaluation, we annotated each query term with its dominant concept in the relevant document, and created an annotated version for each keyword query. Since our focus is on a set of mutually exclusive concepts, we picked the queries whose concepts are mutually exclusive. It is well established that the effectiveness of some queries do not improve by semantic annotation [112]. For instance,, if the terms in a query not do appear in any concept in the relevant results for a query, the query will not benefit from annotation. We compared the ranking quality of the queries over annotated and unannotated versions of Wikipedia collection, and selected only those queries whose ranking qualities have improved. Tables 8.2 shows the selected keyword queries and their corresponding annotated versions. Some terms in these queries appear very rarely and do not appear in any concept in the relevant answers, thus, we did not annotate these terms and submitted them to the collection as free text.

We indexed the annotated and unannotated versions of Wikipedia collection using the Lucene inverted index (*lucene.apache.org*), and used the Lucene ranker to rank the queries over the collection. We performed our experiments on a Linux server with 24 GB of main memory and two quad core processors.

8.6.2 Experimental Results

Figures 8.4 and 8.3 show the probability distributions of concepts over query workloads $u(C)$ and the complete Wikipedia collection $d(C)$, respectively. We do not show the concepts that do not appear in query workloads, as they do not have any impact in our analysis. Since the number of concepts in the query workloads is very much smaller than the number of concepts in the collection, the probability values for the concepts over the query workloads are generally larger than the probability values of the concepts over the collections. We observe that the probability distribution on the collection does not reflect the probability distribution on the query workload.

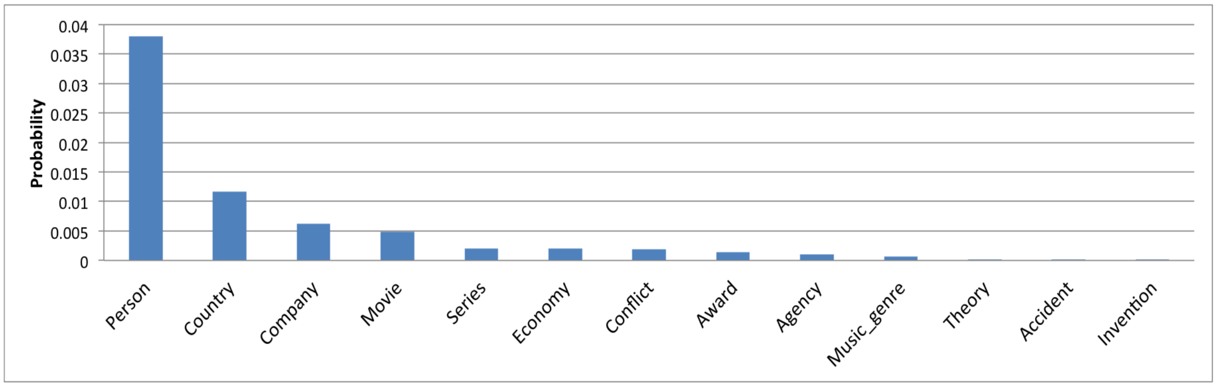


Figure 8.3: Probability distribution of concepts over Wikipedia collection

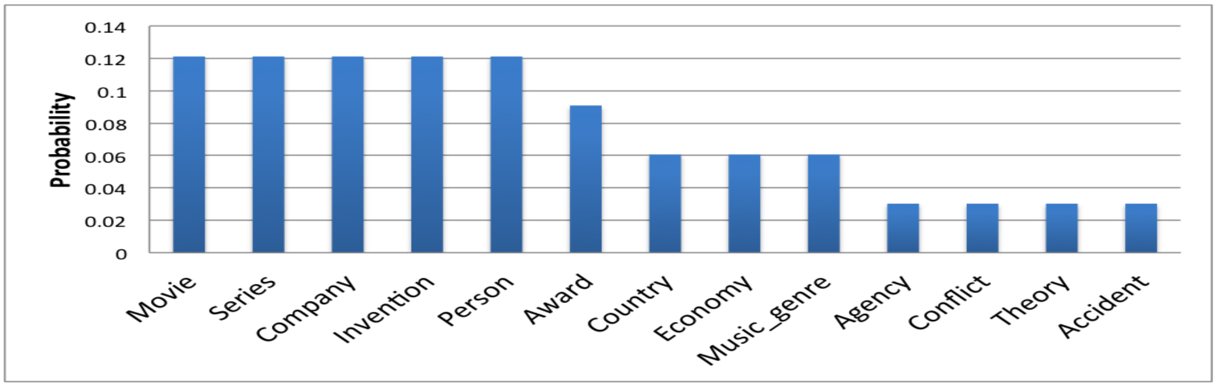


Figure 8.4: Probability distribution of concepts over used queries

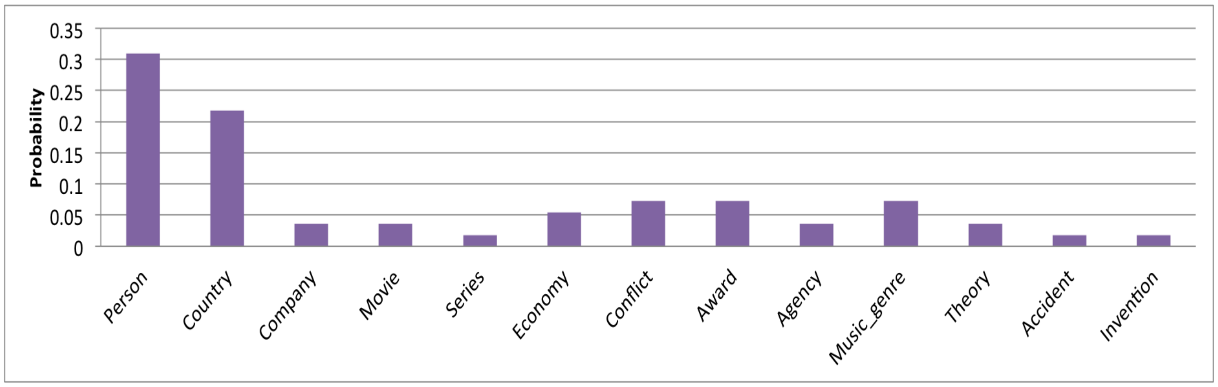


Figure 8.5: Probability distribution of concepts over 50 randomly selected queries

We also randomly selected 50 queries from INEX 2009 and INEX 2010 query workloads to explore the difference between the probabilities of concepts over the collection and query workload. Figures 8.5 shows that probability distribution of concepts over the selected 50 queries. We see that the top two dominant concepts, *person* and *country*, in the collection and query workloads are the same. However, this is not true in general. For instance, concept *award* is more frequent in the query workload than many other concepts, but has a smaller probability than many of those

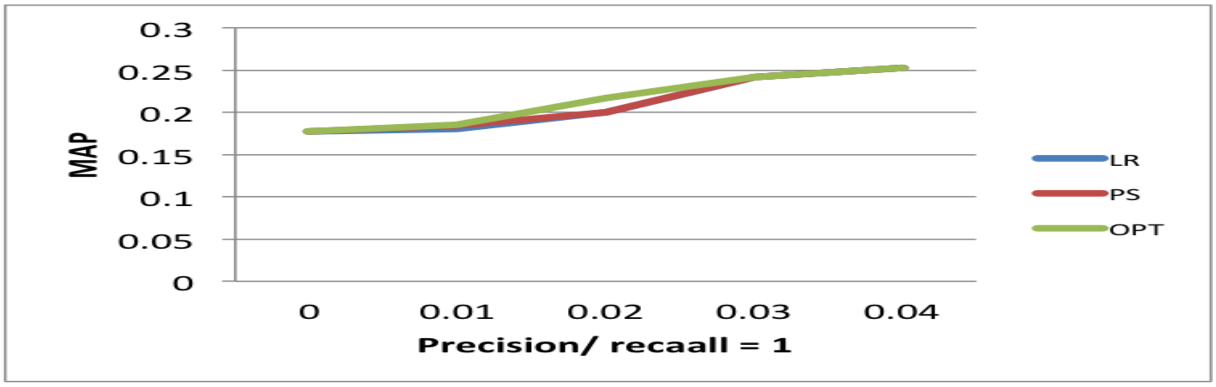


Figure 8.6: MAP of queries with perfect precision and recall in annotation

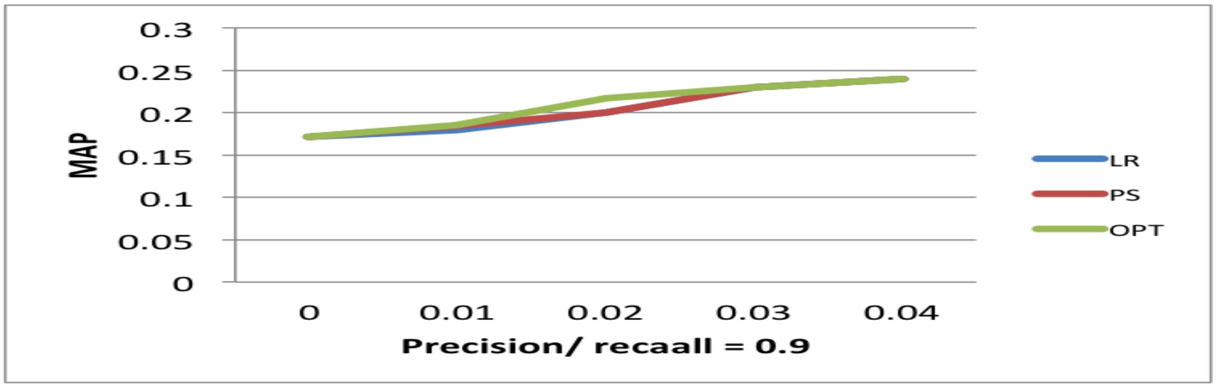


Figure 8.7: MAP of queries with precision and recall values of 0.9 in annotation

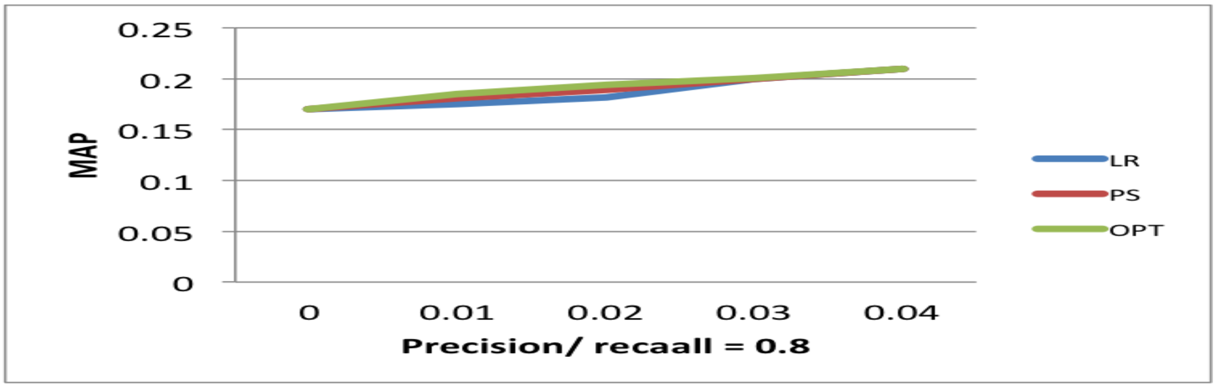


Figure 8.8: MAP of queries with precision and recall values of 0.8 in annotation

concepts in the collection. This is because there are queries in the query workloads about awards and people who won awards. Interestingly, there are not as many documents about these awards in the collection. Hence, it is not generally optimal to rely only on the collection statistics to determine the popularity of the concepts and their benefits for users.

We observed that semantic annotation improves the precision of top-K answers of different queries at different values of K. For instance, the value of P@5 for query *volcanic ash cloud flight disruption* increased from 0.60 to 0.80, but the value of P@10 and P@20 stayed the same for this query. On the other hand, the value of P@5 remained the same for query *cold war*, but its P@10 and P@20 improved from .30 to 0.50 after semantic annotation. Thus, we used the *average precision* metric in our experiment, which measures the improvement in ranking over all possible position [96]. We measured the amount of improvement for the query workload using Mean Average Precision (MAP) [96].

We used the macro model from [24] to introduce error into the annotation of our concepts in the collection. Given the overall values for recall and precision of the annotation process for all concepts, this model computes the most probable false positives and false negatives in annotating the concepts in the collection. We updated the Lucene index by adding and/or removing new annotations, based on the model for different overall values of recall and precision for the annotation. We also computed the new value for precision and recall of each concept after updating the index. We tried different values for overall precision and recall from 0.80 to 1.0, which are reasonable values for precision and recall of current annotators.

The larger the value of $d(C)$ is for concept C, the more computational power we need to annotate instances of C. Similarly, we need more resources to develop a more precise annotator for a concept. Hence, we set the cost of annotating a concept C as $d(C) \times r(C) \times p(C)$ in our experiments.

We used the ratio-greedy and pseudo polynomial algorithms to select concepts in our experiments. The average running time of the ratio-greedy and pseudo polynomial algorithms over different values of the budget limit were 13.6 and 824.7 milliseconds.

Figures 8.6, 8.7, and 8.8 show the improvement in the value of MAP for different values of the budget limit and annotation recall/precision for queries, after annotating the output concepts of LR and pseudo polynomial (PS) algorithms. We have also tried annotating all possible subsets of concepts whose costs do not exceed the value of the given budget limit, and report the largest resulting value of MAP from annotating one of these subsets as optimal (OPT).

Generally, for large values of the budget limit, all three algorithms retrieve almost the same set of concepts. This is because all algorithms have high flexibility to choose the desired concepts. For small values of the budget limit, the LR algorithm provides the lowest value for MAP as it returns a different set of concepts than the optimal algorithm. This difference is greater when the algorithm does not have sufficient resources. Since almost all $u()$ values are small and close to each other, ratio-greedy works generally well. We observed that in the cases that the LR algorithm does not work well, it has ignored some of the concepts with high $u()$ values such as *company*. This is the inherent problem with the LR algorithm. Sometime it is better to consider some concepts with high $u()$ values as well. The pseudo polynomial algorithm returns a more effective set of concepts and provides a very close MAP value to the optimal solution. As the values of precision and recall for annotation degrade, the amount of improvement for all algorithms reduces. The order of the effectiveness of the algorithms remains almost the same over all levels of precision and recall.

Chapter 9

Conclusions

9.1 Concluding Remarks

In this dissertation, we analyzed the effectiveness, efficiency, flexibility, and cost-benefit tradeoffs for keyword query answering approaches over large scale databases. We showed that when these query systems are based on ad hoc heuristics, they are not able to satisfy users' information needs and expectations. We set forth a series of novel, formal, and rigorous frameworks to model these requirements and developed new query answering approaches that satisfy these models. We showed the validity of our models and the usability of our query answering approaches through extensive user studies over real-world large-scale data sets and query workloads. We summarize the main contributions of this dissertation as follows:

- We proposed *coherency ranking*, a new ranking method for keyword search over semi-structured data that ranks candidate answers based on statistical measures of their cohesiveness. Coherency ranks are determined during a one-time precomputation phase that exploits the structure of the data set. For each type of subtree in the data set, the precomputation phase computes its *normalized total correlation* (NTC), a new statistical measure of the tightness of its relationship. NTC is invariant under equivalence-preserving schema transformations, thus avoiding overreliance on shallow structural details. As it is too expensive to compute NTCs for all subtrees of an XML data set, we developed approximation and optimization methods to make precomputation affordable. For query answering, we extended the SA algorithm to rank candidate answers based on the precomputed NTCs. For user-supplied queries over two real-world data sets, coherency ranking showed considerable improvements in precision, mean average precision and recall, compared to six previously proposed methods.

- We addressed a new challenge for keyword queries over semi-structured data: how to provide high-quality, well-ranked query answers when the data is highly structured but also contains long text fields such as paper abstracts or movie plot lines. Previous approaches to semi-structured keyword search do not rank query answers well for such databases. We proposed an approach to measure the correlation between both short and long text fields in such data sets, based on the new concept of normalized term presence correlation (NTPC). After a one-time computation of NTPCs, candidate answers for all subsequent queries are quickly ranked, based on the NTPC of their structure. We performed a user study to evaluate the effectiveness of an NTPC-based keyword query system for data-oriented XML with and without long text fields. The study showed that the NTPC-based approach works as well as NTC for data sets without long fields, and performs better than any previous approach for data sets that include long fields.

- We introduced a highly desirable property for keyword queries, called design independence. We formalized this property and analyzed and compared the design independence of current keyword search and schema free query interfaces. We differentiated between design independence and weak design independence; the latter allows for data duplication and denormalization. We showed analytically and empirically that among current methods, only CR, our approach based on NTC, is design independent. Since CR is not weakly design independent, we provided a novel weakly design independent method, DA-CR. Our empirical studies showed that the average case design independence of other methods is lower than CR's, which in turn is lower than DA-CR's when the new DB design affects data redundancy. Finally, we found that the ranking quality of DA-CR is generally at least as good as that of CR.

- We introduced the novel problem of predicting the effectiveness of keyword queries over databases. We set forth a principled framework and proposed novel algorithms to measure the degree of the difficulty of a potentially underspecified query over a database, using the ranking robustness principle. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low error and negligible time overhead.

- We introduced SLM, a novel formal model for answering keyword queries over structured data effectively. We rigorously justified SLM based on the *probability ranking principle*, which is a widely accepted principle for ranking in search systems effectively. We used principles and heuristics in statistics and information retrieval to compute the parameters of the model when little or no training data is available. Due to its principled foundations, SLM does not require extensive manual training and will be reasonably effective over all types of structured data sets and query workloads given sufficient information. We performed extensive experimental studies using standard benchmarks, real query logs, and real-world data sets and showed that SLM provides more effective ranking that other keyword query ranking methods over structured data.

- Imposing additional structure on a data set can improve the effectiveness of keyword queries, but requires significant resources. We proposed a novel formal model for analyzing the cost-benefit tradeoff of adding more structure to a data set. Our model shows that current perceptions about the impact of additional structure on the effectiveness of queries are not generally true. We proposed novel and efficient optimization algorithms to find the optimal organization for a data set, given the costs and budget for data organization.

9.2 Future Directions

We plan to extend our research results in the following directions.

Value-aware Information Management: Given finite resources, a value-aware approach can tell us where to concentrate our efforts to give the most value for the effort and/or money. We plan to expand our current work on value-aware data organization to encompass all stages in the life cycle of information management systems, from information acquisition and data cleaning to data integration.

Social Information Communication Paradigms: Users dynamically generate a great deal of unstructured information on social platforms. The handful of more structured operations that are popular, such as the *like* operation in Facebook and the # tag in Twitter, have had a huge impact on those platforms, because the information created by these operations is much easier to explore, analyze, and aggregate than pure text. This suggests that the introduction of additional structured operations can potentially bring great benefits as well, to users at large but perhaps also in specific domains such as scientific collaborations. However, a more structured communication paradigm must have compelling benefits and be extremely easy to use, or users will not adopt it. We are interested in determining what these structured operations could be. Such a platform can also be very beneficial for crowd sourced data processing and management.

Integrated Search, Analysis, and Decision Making: The models and systems for search, analysis, and decision making are not usually integrated. Information about the decision spaces could help search and analysis systems to satisfy users' needs. For instance, knowing that the motivation behind searching for a product is to buy a set of related products, a search interface can provide a much more useful ranking for queries over a product data set. By making search interfaces more expressive, users can use search interfaces to perform complex data analysis tasks, such as *What were the hot topics in operating systems 10 years ago and how have they evolved?* We are interested in developing models, algorithms, and systems that unify these elements.

Reliable Data Search and Exploration: One aspect of the value of the data is its trustworthiness. Not all information sources and data sets are equally trustworthy. For instance, many Web sites contain inaccurate and invalid information and many scientific data sets contain noisy and unreliable data. We are interested in principled models and systems to determine the trustworthiness of information and provide both effective and reliable search and exploration results.

References

[1] E. Agichtein and L. Gravano. Querying Text Databases for Efficient Information Extraction. In *ICDE*, 2003.

[2] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-based Search over Relational Databases. In *Proceedings of the Eighteenth International Conference on Data Engineering*, 2002.

[3] Timos Antonopoulos, Wim Martens, and Frank Neven. The Complexity of Text-Preserving XML Transformations. In *PODS*, 2011.

[4] M. Arenas and L. Libkin. A Normal Form for XML Documents. *TODS*, 29(1):195–232, 2004.

[5] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based Keyword Search in Databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, 2004.

[6] Zhifeng Bao, Tok Wang Ling, Bo Chen, and Jiaheng Lu. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE*, 2009.

[7] Paul N. Bennett, Krysta Svore, and Susan T. Dumais. Classification-Enhanced Ranking. In *WWW*, 2007.

[8] Berkeley DB. http://www.oracle.com/technology/products/berkeley-db, 2008.

[9] G. Bhalotoa, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in databases using BANKS. In *ICDE*, 2002.

[10] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. In *Proceedings of the Twenty Third ACM SIGMOD International Conference on Management of Data*, 1997.

[11] Sergey Brin and Larry Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh World Wide Web Conference*, 1998.

[12] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *Proceedings of the Twenty Eighth ACM SIGMOD International Conference on Management of Data*, 2002.

[13] Pablo Castells, Miriam Fernandez, and David Vallet. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. *TKDE*, 14:215–216, 1963.

[14] Soumen Chakrabarti, Kriti Puniyani, and Sujatha Das. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In *WWW*, 2007.

[15] Soumen Chakrabarti, Kriti Puniyani, and Sujatha Das. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In *Proceedings of the Sixteenth International World Wide Web Conference*, 2007.

[16] Soumen Chakrabarti, Sunita Sarawagi, and S. Sudarshan. Enhancing Search with Structure. *IEEE Data Engineering Bulletin*, 33(1), 2010.

[17] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic Ranking of Database Query Results. In *VLDB*, 2004.

[18] Harr Chen and David R. Karger. Less is More: Probabilistic Models for Retrieving Fewer Relevant Documents. In *SIGIR*, 2006.

[19] Yi Chen, Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. Keyword Search on Structured and Semi-structured Data. In *SIGMOD*, 2009.

[20] Shiwen Cheng, Arash Termehchy, and Vagelis Hristidis. Predicting the Effectiveness of Keyword Queries on Databases (Extended Version). In *Technical Report, University of Illinois at Urbana-Champaign*, 2012.

[21] Laura Chiticariu, Yunyao Li, Sriram Raghavan, and Frederick Reiss. Enterprise information extraction: recent developments and open challenges. In *SIGMOD*, 2010.

[22] B. Choi. What are real DTDs like? In *Fifth International Workshop on the Web and Databases*, 2002.

[23] Jennifer Chu-Carroll et al. Semantic search via xml fragments: a high-precision approach to ir. In *SIGIR*, 2006.

[24] Jennifer Chu-Carroll and John Prager. An experimental study of the impact of information extraction accuracy on semantic search performance. In *CIKM*, 2007.

[25] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *CACM*, 13(6):377–387, 1970.

[26] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A Semantic Search Engine for XML. In *VLDB*, 2003.

[27] Kevyn Collins-Thompson and Paul N. Bennett. Predicting query performance via classification. In *ECIR*, pages 140–152, 2010.

[28] Thomas Cover and Joy Thomas. *Elements of Information Theory*. Wiley, 1983.

[29] Bruce Croft and David Harper. Using Probabilistic Models of Document Retrieval without Relevance Information. *Journal of Documentation*, 35(4):285–295, 1979.

[30] Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. Graceful Database Schema Evolution: The Prism Workbench. In *VLDB*, 2008.

[31] Mehmet M. Dalkilic and Edward L. Robertson. Information Dependencies. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2000.

[32] Nilesh Dalvi et al. A Web of Concepts. In *PODS*, 2009.

[33] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. DivQ: Diversification for Keyword Search over Structured Databases. In *SIGIR*, 2010.

[34] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. IQP: Incremental Query Construction, a Probabilistic Approach. In *ICDE*, 2010.

[35] Ludovic Denoyer and Patrick Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 40(1):64–69, 2006.

[36] Stephen Dill et al. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *WWW*, 2003.

[37] AnHai Doan et al. Information Extraction Challenges in Managing Unstructured Data. *SIGMOD Record*, 2008.

[38] Doug Downey, Oren Etzioni, and Stephen Soderland. A probabilistic model of redundancy in information extraction. In *IJCAI*, 2006.

[39] Ofer Egozi, Shaul Markovitch, and Evgeniy Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Transactions on Information Systems*, 29(2):1–34, 2011.

[40] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow, and Gerhard Weikum. Language-model-based Ranking for Queries on RDF-Graphs. In *CIKM*, 2009.

[41] Ergin Elmacioglu and Dongwon Lee. On Six Degrees of Separation in DBLP-DB and More. *SIGMOD Record*, 2005.

[42] Oren Etzioni, Michele Banko, Stephen Soderland, , and Daniel S. Weld. Open Information Extraction from the Web. *Communications of the ACM*, 51(12):68–74, 2008.

[43] Ralf Schenkel Fabian, M. Suchanek, and Gjergji Kasneci. YAWN: A Semantically Annotated Wikipedia XML Corpus. In *BTW*, 2007.

[44] Ronald Fagin, Benny Kimelfeld, Yunyao Li, Sriram Raghavan, and Shivakumar Vaithyanathan. Understanding Queries in a Search Database System. In *PODS*, 2010.

[45] Wenfei Fan and Philip Bohannon. Information Preserving XML Schema Embedding. *TODS*, 33(1), 2008.

[46] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A Large-Scale Study of the Evolution of Web Pages. In *WWW*, 2003.

[47] FNV Hash Function. http://www.isthe.com/chongo/tech/comp/fnv/, 2009.

[48] Shmuel Gal and Boris Klots. Optimal Partitioning Which Maximizes The Sum of Weighted Averages. *Operations Research*, 1995.

[49] Venkatesh Ganti, Yeye He, and Dong Xin. Keyword++: A Framework to Improve Keyword Search Over Entity Databases. *PVLDB*, 2010.

[50] Hector GarciaMolina, Jeff Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, 2008.

[51] Shlomo Geva, Jaap Kamps, Ralf Schenkel, and Andrew Trotman, editors. *Comparative Evaluation of Focused Retrieval - 9th International Workshop of the Inititative for the Evaluation of XML Retrieval, INEX 2010, Vugh, The Netherlands, December 13-15, 2010, Revised Selected Papers*, volume 6932 of *Lecture Notes in Computer Science*. Springer, 2011.

[52] Shlomo Geva, Jaap Kamps, and Andrew Trotman, editors. *Focused Retrieval and Evaluation, 8th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, December 7-9, 2009, Revised and Selected Papers*, volume 6203 of *Lecture Notes in Computer Science*. Springer, 2010.

[53] J.D. Gibbons and S.Chakraborty. *Nonparametric Statistical Inference*. Marcel Dekker, New York, 1992.

[54] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword Proximity Search in Complex Data Graphs. In *Proceedings of the Thirty Fourth ACM SIGMOD International Conference on Management of Data*, 2008.

[55] Jens Graupmann, Michael Biwer, Christian Zimmer, Patrick Zimmer, Matthias Bender, Martin Theobald, and Gerhard Weikum. Compass: A concept-based web search engine for html, xml, and deep web data. In *VLDB*, 2004.

[56] P. Gulhane et al. Web-Scale Information Extraction with Vertex. In *ICDE*, 2011.

[57] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD*, 2003.

[58] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[59] Ben He and Iadh Ounis. Query performance prediction. *Inf. Syst.*, 31:585–594, November 2006.

[60] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. In *Proceedings of the Nineteenth International Conference on Data Engineering*, 2003.

[61] Vagelis Hristidis et al. Keyword Proximity Search in XML Trees. *TKDE*, 18(5):525–536, 2006.

[62] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB 2003*.

[63] Jian Huang and Cong Yu. Prioritization of Domain-Specific Web Information Extraction. In *AAAI*, 2010.

[64] X. Huang, A. Acero, , and H. Hon. *Spoken Language Processing*. Prentice Hall, 2001.

[65] Yke Huhtala, Juha Kerkkeinen, Pasi Porkka, and Hannu Toivonen. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. In *Proceedings of the Fourteenth International Conference on Data Engineering*, 1998.

[66] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SICOMP*, 15(3), 1986.

[67] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*, 2004.

[68] Panagiotis Ipeirotis, Eugene Agichtein, Pranay Jain, and Luis Gravano. To Search or to Crawl? Towards a Query Optimizer for TextCentric Tasks. In *SIGMOD*, 2006.

[69] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making Database Systems Usable. In *SIGMOD*, 2007.

[70] Alpa Jain, AnHai Doan, and Luis Gravano. Optimizing SQL Queries over Text Databases. In *ICDE*, 2008.

[71] Alpa Jain, Panagiotis Ipeirotis, and Luis Gravano. Building Query Optimizers for Information Extraction: The SQoUT Project. *SIGMOD Record*, 2008.

[72] Myung-Gil Jang, Sung Hyon Myaeng, and Se Young Park. Using Mutual Information to Resolve Query Translation Ambiguities and Query Term Weighting. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistic*, 1999.

[73] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating Query Substitutions. In *Proceedings of the 15th international conference on World Wide Web*, 2006.

[74] Pallika Kanani and Andrew McCallum. Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning. In *WSDM*, 2012.

[75] S. M. Katz. Estimation of Probabilistic from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35:400–401, 1987.

[76] Yiping Ke, James Cheng, and Wilfred Ng. Mining Quantitative Correlated Patterns Using an Information-Theoretic Approach. In *SIGKDD*, 2006.

[77] M. Kendall and J. D. Gibbons. *Rank Correlation Methods*. Edvard Arnold, London, 1990.

[78] Jinyoung Kim, Xiaobing Xue, and Bruce Croft. A Probabilistic Retrieval Model for Semistructured Data. In *ECIR*, 2009.

[79] Benny Kimelfeld and Yehoshua Sagiv. Finding and Approximating Top-k Answers in Keyword Proximity Search. In *Proceedings of the Twenty Fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2005.

[80] I. Kojadinovic. Relevance Measures for Subset Variable Selection in Regression Problems Based on K-additive Mutual Information. *Computational Statistics and Data Analysis*, 49:1205–1227, 2005.

[81] B. Korte and R. Schrader. On the Existence of Fast Approximation Schemes. *O.L. Mangasarian, R.R. Meyer, and S.M. Robinson (eds.) Nonlinear Programming, Academic Press, New York, 415437*, 1981.

[82] G. Kourtika, A. Simitsis, and Y. Ioannidis. Precis: The Essence of a Query Answer. In *Proceedings of the Twenty Second International Conference on Data Engineering*, 2006.

[83] Robert Krovetz and W. Bruce Croft. Lexical ambiguity and information retrieval. *ACM Transactions on Information Systems*, 10:115–141, 1992.

[84] John Lafferty and Chengxiang Zhai. Probabilistic Relevance Models Based on Document and Query Generation. *Language Modeling and Information Retrieval*, 13:526–533, 2003.

[85] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective Keyword Search for Valuable LCAs over XML Documents. In *CIKM*, 2007.

[86] Xiao Li, Ye-Yi Wang, and Alex Acero. Extracting Structured Information from User Queries with Semi-Supervised Conditional Random Fields. In *SIGIR*, 2009.

[87] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, 2004.

[88] Yunyao Li, Huahai Yang, and H.V. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. In *Proceedings of the Tenth International Conference on Extending Database Technology*, 2006.

[89] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search in Relational Databases. In *SIGMOD*, 2006.

[90] Ziyang Liu and Yi Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, 2008.

[91] Alexander Loeser, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Navigating the Intranet with High Precision. In *WWW*, 2007.

[92] Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In *Thirty First International Conference on Very Large Data Bases*, 2005.

[93] Yi Luo, Xumein Lin, Wei Wang, and Xiaofang Zhou. SPARK: Top-k Keyword Query in Relational Databases. In *SIGMOD 2007*.

[94] Sheng Ma and Joseph L. Hellerstein. Mining Mutually Dependent Patterns. In *ICDM*, 2002.

[95] D. Maier. *Theory of Relational Databases*. Computer Science Press, Reading, Massachusetts, 1983.

[96] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[97] W. J. McGill. Multivariate Information Transmission. *Psychometrika*, 19:97–116, 1954.

[98] R. Miller. Response time in man-computer conversational transactions. In *Proceedings of the AFIPS Fall Joint Computer Conference*, 1968.

[99] Elke Mittendorf and Peter Schauble. Measuring the Effects of Data Corruption on Information Retrieval. In *SDAIR*, 1996.

[100] Shinichi Morishita and Jun Sese. Traversing Itemset Lattices with Statistical Metric Pruning. In *SIKDD*, 2006.

[101] Arnab Nandi and H. V. Jagadish. Assisted Querying Using Instant-Response Interfaces. In *SIGMOD*, 2007.

[102] Zaiqing Nie, Yunxiao Ma, Shuming Shi, Ji-Rong Wen, and Wei-Ying Ma. Web Object Retrieval. In *WWW*, 2007.

[103] S. Nijssen and J. N. Kok. Efficient Discovery of Frequent Unordered Trees. In *First International Workshop on Mining Graphs, Trees, and Sequences*, 2003.

[104] Paul Ogilvie and James P. Callan. Combining Document Representations for Known-Item Search. In *SIGIR*, 2003.

[105] Aditya G. Parameswaran, Nilesh N. Dalvi, Hector Garcia-Molina, and Rajeev Rastogi. Optimal schemes for robust web extraction. *PVLDB*, 4(11):980–991, 2011.

[106] G. Pass, A. Chowdhury, and C. Torgeson. A Picture of Search. In *InfoScale*, 2006.

[107] Desislava Petkova, W. Bruce Croft, and Yanlei Diao. Refining Keyword Queries for XML Retrieval by Combining Content and Structure. In *Proceedings of the 31st European Conference on Information Retrieval*, 2009.

[108] Hema Raghavan, James Allan, and Andrew McCallum. An Exploration of Entity Models, Collective Classification and Relation Description. In *KDD*, 2004.

[109] S. E. Robertson. The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4):294–304, 1977.

[110] Stephen Robertson and Karen SpŁrck Jones. Relevance Weighting of Search Terms. *JASIS*, 27:129–146, 1976.

[111] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *CIKM*, 2004.

[112] M. Sanderson. Ambiguous Queries: Test Collections Need More Sense. In *SIGIR*, 2008.

[113] Mark Sanderson. Word sense disambiguation and information retrieval. In *SIGIR*, 1994.

[114] Mark Sanderson. Retrieving with good sense. *Information Retrieval*, 2:49–69, 2000.

[115] Sunita Sarawagi. Information extraction. *Found. Trends databases*, 1:261–377, March 2008.

[116] Nikos Sarkas, Stelios Paparizos, and Panayiotis Tsaparas. Structured Annotations of Web Queries. In *SIGMOD*, 2010.

[117] Anish Das Sarma, Alpa Jain, and Philip Bohannon. Building a Generic Debugger for Information Extraction Pipelines. In *CIKM*, 2011.

[118] Ralf Schenkel and Martin Theobald. Feedback-Driven Structural Query Expansion for Ranked Retrieval of XML Data. In *Proceedings of the 10th International Conference on Extending Database Technology*, 2006.

[119] A. Schmidt, M. Kersten, and M. Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *ICDE*, 2001.

[120] A. R. Schmidt et al. XMark: A Benchmark for XML Data Management. In *VLDB*, 2002.

[121] Hinrich Schutze and Jan O. Pedersen. Information retrieval based on word senses. In *DAIR*, 1995.

[122] Anna Shtok, Oren Kurland, and David Carmel. Using Statistical Decision Theory and Relevance Models for Query-Performance Prediction. In *SIGIR*, 2010.

[123] Christopher Stokoe, Michael P. Oakes, and John Tait. Word Sense Disambiguation in Information Retrieval Revisited. In *SIGIR*, 2003.

[124] C. Sun, C. Chan, and A. Goenka. Multiway SLCA-based keyword search in XML data. In *WWW*, 2007.

[125] I. Tatarinov et al. Storing and Querying Ordered XML using a Relational Database System. In *SIGMOD*, 2002.

[126] Arash Termehchy and Marianne Winslett. Effective, Design-Independent XML Keyword Search. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management, CIKM*, 2009.

[127] Arash Termehchy and Marianne Winslett. Keyword Search for Data-Centric XML Collections with Long Text Fields. In *EDBT*, 2010.

[128] Arash Termehchy and Marianne Winslett. Using Deep Structural Information in XML Keyword Search. *Proceedings of the VLDB Endowment, PVLDB*, 3(2), 2010.

[129] Arash Termehchy and Marianne Winslett. Using Structural Information in XML Keyword Search Effectively. *TODS*, 36(1), 2011.

[130] Arash Termehchy, Marianne Winslett, and Yodsawalai Chodpathumwan. How Schema Independent Are Schema Free Query Interfaces? In *ICDE*, 2011.

[131] Arash Termehchy, Marianne Winslett, Yodsawalai Chodpathumwan, and Austin Gibbons. On the Schema Independence of Query Interfaces. *IEEE Transactions on Knowledge and Data Engineering, Special Issue on the Best Papers of ICDE*, 2012.

[132] Steve Cronen Townsend, Yun Zhou, and Bruce Croft. Predicting Query Performance. In *SIGIR*, 2002.

[133] Thanh Tran, Peter Mika, Haofen Wang, and Marko Grobelnik. Semsearch'10: the 3th semantic search workshop. In *WWW*, 2010.

[134] Andrew Trotman and Shlomo Geva. Report on the SIGIR 2006 Workshop on XML Element Retrieval Methodology. *ACM SIGIR Forum*, 40, 2006.

[135] Roelof van Zwol and Tim van Loosbroek. Effective Use of Semantic Structure in XML Retrieval. In *ECIR*, 2007.

[136] Moshe Vardi. The universal-relation data model for logical independence. *IEEE Software*, 5:80–85, 1988.

[137] Yannis Velegrakis, Renee J. Miller, and Lucian Popa. Mapping Adaptation under Evolving Schemas. In *VLDB*, 2003.

[138] Ellen M. Voorhees. Overview of the TREC 2003 Robust Retrieval Track. In *TREC*, 2004.

[139] Jun Wang and Jianhan Zhu. Portfolio Theory of Information Retrieval. In *SIGIR*, 2009.

[140] K. Wang and H. Liu. Discovering typical structures of documents: A Road Map Approach. In *Proceedings of the twenty First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998.

[141] Kuansan Wang, Xiaolong Li, and Jianfeng Gao. Multi-Style Language Model for Web Scale Information Retrieval. In *SIGIR*, 2010.

[142] Qiuyue Wang and Andrew Trotman. Data-Centric Track . In *INEX Workshop*, 2010.

[143] S. Watanabe. Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development*, 4(1):66–82, 1960.

[144] William Webber. Evaluating the Effectiveness of Keyword Search. *IEEE Data Engineering Bulletin*, 33(1), 2010.

[145] J. Wen, J. Nie, and H. Zhang. Clustering User Queries of a Search Engine. In *WWW*, 2001.

[146] K. Win, E. Rosasillfiani, W. Ng, and E. Lim. A Visual Interface for Native XML Database. In *Proceedings of the Fourteenth International Conference on Database and Expert Systems Applications*, 2003.

[147] Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[148] Dietmar Wolfram. Search characteristics in different types of Web-based IR environments: Are they the same? In *WWW*, 2001.

[149] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

[150] Yu Xu and Yannis Papakonstantinou. Efficient LCA based Keyword Search in XML Data. In *EDBT*, 2008.

[151] Xiaoxin Yin, Jiawei Han, and Jiong Yang. Searching for Related Objects in Relational Databases. In *Proceedings of the Seventeenth International Conference on Scientific and Statistical Database Management*, 2005.

[152] Elad YomTov, Shai Fine, David Carmel, and Adam Darlow. Learning to Estimate Query Difficulty. In *SIGIR*, 2005.

[153] C. Yu and H. V. Jagadish. Efficient discovery of XML data redundancy. In *Proceedings of the Thirty Second International Conference on Very Large Data Bases*, 2006.

[154] Cong Yu and H. V. Jagadish. Schema Summarization. In *VLDB*, 2006.

[155] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. Keyword Search in Relational Databases: A Survey. *IEEE Data Engineering Bulletin*, 33(1), 2010.

[156] M. Zaki. Efficiently Mining Frequent Trees in a Forest. *TKDE*, 17(8):1021–1035, 2005.

[157] ChengXiang Zhai. Statistical Language Models for Information Retrieval: A Critical Review. *Foundations and Trends in Information Retrieval*, 2-3:137–213, 2008.

[158] ChengXiang Zhai and John Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *SIGIR*, 2001.

[159] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. Automatic Discovery of Attributes in Relational Databases. In *SIGMOD*, 2011.

[160] L. Zhao and J. Callan. A Generative Retrieval Model for Structured Documents. In *CIKM*, 2008.

[161] Yun Zhou and Bruce Croft. Ranking Robustness: A Novel Framework to Predict Query Performance. In *CIKM*, 2006.