USABILITY AND EXPRESSIVENESS IN DATABASE KEYWORD SEARCH: BRIDGING THE GAP

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

DOKTORIN DER NATURWISSENSCHAFTEN

Dr. rer. nat.

genehmigte Dissertation
von

M.Sc. Elena Demidova

geboren am 29. June 1974, in St. Petersburg, Russland

2013

Referent: Prof. Dr. techn. Wolfgang Nejdl
Korreferent: Prof. Dr. Heribert Vollmer
Tag der Promotion: 30 Januar 2013

ABSTRACT

With increasing availability of structured data on the Web, in organizations and enterprises end users gained direct and independent access to scientific data in a variety of domains, data warehouses in enterprises as well as services and entertainment facilities on the Web. Existing database systems enable expert users to efficiently interact with structured data, obtain information with well-defined semantics and directly use this information to perform follow-up transactions. However, database query construction is a laborious and error-prone process, which cannot be performed well by most end users. Database keyword search alleviates the usability problem at the price of query expressiveness. As keyword search algorithms do not differentiate between the possible informational needs represented by a keyword query, users may not receive adequate results. This observation motivated us to develop new approaches to enable end users going beyond the most likely interpretations. In this thesis we tackle four important usability aspects of database keyword search, namely: (i) How to enable users to incrementally refine a keyword query into the intended interpretation on the target database? (ii) How to provide database search results with an increasing level of novelty? (iii) How to enable efficient and scalable query construction solutions for large scale data?, and (iv) How to enrich a large scale database with semantic information?

We start with a presentation of IQP - a novel approach to bridge the gap between usability of keyword search and expressiveness of database queries. IQP enables end users to start with an arbitrary keyword query and incrementally refine it into a structured query through an interactive interface. In this thesis we present the detailed design of IQP, and demonstrate its effectiveness and scalability through extensive experiments and a user study. We continue with DivQ - a scheme to balance the relevance and novelty of keyword search results over structured databases. We introduce a scheme to diversify search results by re-ranking query interpretations, taking into account redundancy of query results. Then, we propose new metrics taking into account graded relevance of subtopics. Our evaluation on two real-world datasets demonstrates that search results obtained using the proposed algorithms better characterize possible answers than the results of the initial relevance ranking.

Then, we present FreeQ that further develops interactive query construction approach of IQP by incorporating a set of novel techniques to boost the scalability of query construction over large scale data. We construct an abstract ontology layer over the database schema to ensure the efficiency of user-computer interaction. We also introduce a search mechanism to enable efficient exploration of query interpretation spaces over large scale data. We show through the extensive experiments that our approach scales well on Freebase - an open database containing more than 7,000 relational tables in more than 100 domains. Finally, to provide more semantic information for Freebase, we connect it to the YAGO ontology that contains more than 360,000 classes. We analyze the structure of YAGO in more depth and show how to match YAGO categories and Freebase tables. We make our YAGO+F structure available online in the hope that it can provide a good starting point for future applications which build upon a wide variety of Freebase data clearly arranged in the semantic categories of YAGO.

Keywords: *Keyword search in relational databases, incremental query construction, diversification of search results.*

ZUSAMMENFASSUNG

Mit der zunehmenden Verfügbarkeit von strukturierten Daten im Web, Organisationen und Unternehmen, haben die Benutzer einen direkten und unabhängigen Zugang sowohl zu wissenschaftlichen Daten und universellen Datenbanken, als auch zu den webbasierten Service- und Unterhaltungsanwendungen bekommen. Bestehende Datenbankmanagementsysteme ermöglichen es den fachkundigen Benutzern mit den strukturierten Daten effizient interagieren zu können, Informationen mit eindeutig definierter Semantik zu bekommen und diese Informationen in den darauffolgenden Transaktionen zu verwenden. Gleichzeitig stellt die Formulierung komplexer Datenbanksuchanfragen eine aufwendige und fehleranfällige Aufgabe dar, die von einem Endbenutzer nicht effektiv erfüllt werden kann. Volltextsuche in relationalen Datenbanken erhöht zwar die Benutzerfreundlichkeit der Datenbankschnittstelle, jedoch wird dabei die Ausdrucksfähigkeit der Suchanfragen verringert. Um die Benutzerfreundlichkeit sowie die Ausdrucksfähigkeit der Volltextsuche in relationalen Datenbanken zu verbessern, in dieser Dissertation untersuchen wir vier Fragen, nämlich: (i) Wie können wir es einem Endbenutzer ermöglichen, eine komplexe Suchanfrage an eine relationalen Datenbank mit Hilfe der Volltextsuche zu formulieren? (ii) Wie können wir in einem Suchergebnis einen Überblick über die suchanfragerelevanten Datenbankinhalte liefern? (iii) Wie können wir eine interaktive Erstellung komplexer Suchanfragen an sehr großen relationalen Datenbanken effizient unterstützen?, und (iv) Wie können wir eine große relationale Datenbank mit semantischen Informationen anreichern?

Wir beginnen mit der Darstellung von IQP - einem Ansatz, der es den Endbenutzern ermöglicht, mit einer beliebigen Volltextsuchanfrage anzufangen, und diese in einem interaktiven Prozess in eine strukturierte Datenbankanfrage umzuwandeln. In dieser Dissertation stellen wir ein detailiertes Design von IQP vor und demonstrieren seine Effizienz anhand von umfangreichen Experimenten und einer Benutzerstudie. Desweiteren stellen wir DivQ vor - ein System, das die Relevanz und Neuigkeit der Suchergebnisse der Datenbankvolltextsuche ausbalanciert. Wir präsentieren eine Diversifikationstechnik um die Suchergebnisse unter Berücksichtigung deren Redundanz einzuordnen. Um die Qualität der Diversifikationsergebnisse zur beurteilen, schlagen wir zwei neue Maße vor, welche eine Berücksichtigung der Relevanz der Unterkategorien in einem Suchergebnis ermöglichen. Eine Auswertung mit realen Daten zeigt, dass unsere Diversifikationsergebnisse die verfügbare Datenmenge besser beschreiben, als die herkömmliche Suchergebnisse.

Danach stellen wir FreeQ vor - ein System, das den IQP Ansatz weiterentwickelt und eine Menge von Methoden zur Verfügung stellt um eine interaktive Erstellung komplexer Suchanfragen an die sehr großen relationalen Datenbanken effizient zu unterstützen. FreeQ erstellt eine Ontologie-basierte Schicht über das Datenbankschema, welche die Effizienz der Mensch-Computer-Interaktion steigert. Wir demonstrieren die Skalierbarkeit der vorgestellten Algorithmen anhand von Experimenten mit Freebase – eine Datenbank mit über 7000 relationalen Tabellen in mehr als 100 Domains. Letztendlich reichern wir die Freebase Datenbank mit der semantischen Kategorien der sehr großen YAGO Ontologie an, weiche mehr als 360 000 Klassen beinhaltet. Wir analysieren die Struktur von YAGO und Freebase und zeigen wie die beiden Schemata in einer neuen Struktur, YAGO+F benannt, miteinander verknüpft werden können. Wir hoffen, dass YAGO+F ein Ausgangspunkt für viele zukünftige Anwendungen darstellen kann, welche von der Vielfalt der Freebase Daten, die mit der semantischen YAGO Kategorien angereichert sind, profitieren können.

Schlagwörter: *Volltextsuche in relationalen Datenbanken, interaktive Erstellung komplexer Suchanfragen, Diversifizierung der Suchergebnisse.*

ACKNOWLEDGMENTS

First, I am deeply grateful to my supervisor, Prof. Dr. techn. Wolfgang Nejdl for giving me the opportunity of being part of L3S Research Center and Gottfried Wilhelm Leibniz University of Hannover, for guiding me in how to pursue excellent research, and for supporting me during these past six years.

My sincere thanks are due to Prof. Dr. Heribert Vollmer, my second supervisor, for providing very useful comments on the draft of my thesis, and to Prof. Dr.-Ing. Bernardo Wagner for being a part of the dissertation committee.

I wish to express my warm and sincere thank to Dr. Xuan Zhou for his continuous support throughout my thesis work, for his knowledge, scientific advice, fruitful discussions, and collaborations. I also would like to thank Dr. Peter Fankhauser for his collaboration and many insightful discussions and suggestions. A special thank to Prof. Marianne Winslett for her advice and fruitful collaborations starting from the very beginning of my Ph.D. studies. I also would like to thank Dr. Daniel Olmedilla for greatly supporting me at the initial stage of my work at the L3S Research Center.

I am grateful to my colleagues from the L3S Research Center and Gottfried Wilhelm Leibniz University of Hannover for providing an excellent and inspiring working atmosphere, for their support and valuable comments not limited just to this thesis. Many thanks to the colleagues working in the administrative and technical departments, especially to Anca Vais, Michaela Kleiner, Angelika van Agen, and Iris Zieseniss for their support and help.

I owe my deepest gratitude to Prof. Dr. Jürgen Biermann, one of the best Professors I met at the Osnabrück University of Applied Science who believed in me and supported my studies of Computer Science. My special thank to Prof. Roland Schwänzl, who founded the Information Engineering program and supported my studies at the University of Osnabrück.

I am also very grateful to my family. My loving thanks are due to my parents and my daughter Danielle for supporting me, and especially for standing by me along this Ph.D. I owe my loving thanks to my husband Sergej for his understanding, support, and valuable advices.

The work on this thesis was partly funded by the European Commission under the projects ARCOMEM (contract no. 270239), LivingKnowledge (contract no. 231126), and OKKAM (contract no. 215032).

FOREWORD

The algorithms presented in this thesis have been published or are under review at various conferences and journals, as follows:

In Chapter 3 we structure the presentation of incremental query construction around the following papers:

- *A Probabilistic Scheme for Keyword-Based Incremental Query Construction.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In IEEE Trans. on Knowl. and Data Eng. (TKDE), 24(3):426-439, March 2012. DOI: 10.1109/TKDE.2011.40 ©2012 IEEE. Reprinted with permission. [DZNonb].

- *IQ^P: Incremental Query Construction, a Probabilistic Approach.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 26th IEEE International Conference on Data Engineering, March 1-6, 2010, Long Beach, California, USA. DOI: 10.1109/ICDE.2010.5447929 ©2010 IEEE. Reprinted with permission. [DZNona].

The presentation of diversification of keyword search results over structured data in Chapter 4 is built upon the work published in:

- *DivQ: Diversification for Keyword Search over Structured Databases.* Elena Demidova, Peter Fankhauser, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 33rd Annual ACM SIGIR Conference, 19-23 July 2010, Geneva, Switzerland [DFZN10].

Then, Chapter 5 describes contributions that enable to increase scalability of interactive query construction to large scale data included in:

- *FreeQ: An Interactive Query Interface for Freebase.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 21st International World Wide Web Conference, 16-20 April 2012, Lyon, France [DZN12a].

- *Scaling Interactive Query Construction on a Very Large Database.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In submission [DZN12b].

Finally, Chapter 6 describes a mapping between a large scale Freebase dataset and the YAGO ontology to support interactive query construction over large scale data included in:

- *YAGO meets Freebase: Combining a Large Scale Database with an Ontology.* Irina Oelze, Elena Demidova, Wolfgang Nejdl. In submission [ODN12].

During the early stages of the Ph.D. studies I have also published a number of papers investigating interoperability and confidentiality aspects of federated keyword search. These aspects are not touched in this thesis due to space limitations. The complete list of publications follows:

Journal articles

- *A Probabilistic Scheme for Keyword-Based Incremental Query Construction.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In IEEE Trans. on Knowl. and Data Eng. (TKDE), 24(3):426-439, March 2012. ©2012 IEEE. [DZNonb]

- *Social Software for Lifelong Competence Development: Challenges and Infrastructure.* Ivana Marenzi, Elena Demidova, Wolfgang Nejdl, Daniel Olmedilla, Sergej Zerr. In: International Journal of Emerging Technologies in Learning (iJET), Vol 3 (2008). ISSN: 1863-0383. [MDN+08b]

Conference papers

- *Privacy-Aware Image Classification and Search.* Sergej Zerr, Stefan Siersdorfer, Johnathon Hare, Elena Demidova. In: Proceedings of the 35th Annual International ACM SIGIR Conference, August 2012, Portland, Oregon, USA. [ZSHD12]

- *FreeQ: An Interactive Query Interface for Freebase.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 21st International World Wide Web Conference, 16-20 April 2012, Lyon, France. [DZN12a]

- *Evaluating Evidences for Keyword Query Disambiguation in Entity Centric Database Search.* Elena Demidova, Xuan Zhou, Irina Oelze, Wolfgang Nejdl. In: Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA 2010), 30 August - 3 September 2010, Bilbao, Spain. [DZON10]

- *DivQ: Diversification for Keyword Search over Structured Databases.* Elena Demidova, Peter Fankhauser, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 33rd Annual ACM SIGIR Conference, 19-23 July 2010, Geneva, Switzerland. [DFZN10]

- *IQ^P: Incremental Query Construction, a Probabilistic Approach.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In: Proceedings of the 26th IEEE International Conference on Data Engineering, March 1-6, 2010, Long Beach, California, USA. ©2010 IEEE. [DZNona]

- *SUITS: Faceted User Interface for Constructing Structured Queries from Keywords.* Elena Demidova, Xuan Zhou, Gideon Zenz, Wolfgang Nejdl. In: Proceedings of the 14th International Conference on Database Systems for Advanced Applications (Best Demo Award), 21-23 April 2009, Brisbane, Australia. [DZZN09]

- *LearnWeb 2.0: Integrating Social Software for Lifelong Learning.* Ivana Marenzi, Elena Demidova, Wolfgang Nejdl. In: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, Vienna, Austria, 2008. [MDN08a]

- *Zerber: r-Confidential Indexing for Distributed Documents.* Sergej Zerr, Elena Demidova, Daniel Olmedilla, Wolfgang Nejdl, Marianne Winslett, Soumyadeb Mitra. In: Proceedings of the 11th International Conference on Extending Database Technology (EDBT 2008), March 25-30 2008, Nantes, France. [ZDO+08]

- *Services for Knowledge Resource Sharing & Management in an Open Source Infrastructure for Lifelong Competence Development.* Elena Demidova, Philipp Kärger, Daniel Olmedilla, Stefaan Ternier, Erik Duval, Michele Dicerto, Carlos Mendez, Krassen Stefanov. In: Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies (ICALT 2007) July 18-20 2007, Niigata, Japan, pp. 691-693, 2007, IEEE Computer Society, 978-0-7695-2916-5. [DKO+07]

Workshop papers

- *Preservation of Social Web Content based on Entity Extraction and Consolidation.* Stefan Dietze, Diana Maynard, Elena Demidova, Thomas Risse, Wim Peters, Katerina Doka, Yannis Stavrakas. In: Proceedings of the 2nd International Workshop on Semantic Digital Archives (SDA) in conjunction with the 16th International Conference on Theory and Practice of Digital Libraries (TPDL), September 2012, Pafos, Cyprus. [DMD+12]

- *deskWeb2.0: Combining Desktop and Social Search.* Sergej Zerr, Elena Demidova, Sergey Chernov. In: Proceedings of the Desktop Search Workshop, in conjunction with the 33rd Annual ACM SIGIR Conference, 23 July 2010, Geneva, Switzerland. [ZDC10]

- *Usability and Expressiveness in Database Keyword Search: Bridging the Gap.* Elena Demidova, Wolfgang Nejdl. In: Proceedings of the VLDB 2009 PhD Workshop, In conjunction with VLDB 2009, 24 August 2009, Lyon, France. [DN09]

- *Do We Mean the Same? Disambiguation of Extracted Keyword Queries for Database Search.* Elena Demidova, Irina Oelze, Peter Fankhauser. In: Proceedings of the First International Workshop on Keyword Search on Structured Data (KEYS 2009), 28 June 2009, Providence, Rhode Island, USA. [DOF09]

- *Social Software for Lifelong Competence Development: Scenario and Challenges.* Ivana Marenzi, Elena Demidova, Wolfgang Nejdl, Daniel Olmedilla. In: Empowering Learners for Lifelong Competence Development: pedagogical, organisational and technological issues. Proceedings of the 4th TENCompetence Open Workshop Madrid, Spain, April 2008. ISBN 978-90-6813-8474. [MDNO08]

- *Integration of Heterogeneous Information Sources into a Knowledge Resource Management System for Lifelong Learning.* Elena Demidova, Stefaan Ternier, Daniel Olmedilla, Erik Duval, Michele Dicerto, Krassen Stefanov, Naiara Sacristán. In Proceedings of the 2nd TenCompetence Workshop, January 11-12, 2007, Manchester, United Kingdom. [DTO+07]

- *Integrating RDF Querying Capabilities into a Distributed Search Infrastructure.* Elena Demidova, Wolfgang Nejdl. In: Proceedings of the "Web Search Technology - from Search to Semantic Search" Workshop, in conjunction with the 1st Asian Semantic Web Conference (ASWC 2006), September 3-7, 2006, Beijing, China. [DN06]

Technical reports

- *SUITS: Constructing Structured Queries from Keywords.* Xuan Zhou, Gideon Zenz, Elena Demidova, Wolfgang Nejdl. In: Technical Report at the L3S Research Center, 2008. [ZZDN08]

Manuscripts under review

- *Scaling Interactive Query Construction on a Very Large Database.* Elena Demidova, Xuan Zhou, Wolfgang Nejdl. In submission [DZN12b].

- *YAGO meets Freebase: Combining a Large Scale Database with an Ontology.* Irina Oelze, Elena Demidova, Wolfgang Nejdl. In submission [ODN12].

Contents

List of Figures

List of Tables

1 Introduction

With an increasing availability of structured data on the Web, in organizations, and enterprises, end users gained direct and independent access to scientific data in a variety of domains, data warehouses, as well as services and entertainment facilities on the Web. Existing database systems enable expert users to efficiently interact with structured data, obtain information with well-defined semantics and directly use this information to perform follow-up transactions. To this extent, database systems typically expose structured query interfaces such as SQL with a high expressive power to the expert users. On the one hand, this expressive power comes at the price of a fairly complex internal data structures and query languages which are typically beyond the expertise of end users. On the other hand, predefined query structures such as forms, typically exposed to end users, lack flexibility. In this context, database usability becomes crucial to enable novice users to take an advantage of the expressiveness of structured query languages and to effectively interact with structured data.

Database usability attracted an increased research attention over the last decade. New interfaces and algorithms such as natural language query interfaces, query autocompletion techniques, and flexible forms were developed to provide end users with a more flexible way to access structured data [And95, NJ07, JJ08]. Another recently developed technique to access structured data in a user-friendly way is database keyword search [YQC10a]. In contrast to structured queries, keyword search requires neither a-priori schema knowledge nor any specific query writing skills and can be performed efficiently by novice users. Compared with typical Web search applications, database search can provide several advantages, such as more complex query semantics, more precise and complete query answers, as well as structured results [JCE+07].

Despite the high usability, keyword queries lack expressiveness to precisely describe users' informational needs, and may return irrelevant or incomplete results. To take advantage of both, i.e. expressiveness of structured queries and usability of keyword search, some database keyword search approaches perform keyword query disambiguation, where the system first translates keyword queries against a database

into structured queries, which are likely to represent the user's informational need initially expressed by keywords [KKR+06, TL08, TCRS07, ZWX+07]. Effective keyword query disambiguation can enable end users to precisely specify their informational needs using a simple query language and obtain search results with well-defined semantics.

1.1 Problems Addressed in this Thesis

Despite many research efforts, some important usability aspects of database keyword search, and, more specifically, keyword query disambiguation, have not been sufficiently addressed by the existing work. These aspects include *incremental query construction*, *diversification of search results over structured data*, *efficient incremental query refinement for large scale databases*, and *enrichment of large scale databases with semantic information*. State-of-the-art query disambiguation approaches typically assess the likelihood of the possible structural query interpretations and select the most likely query interpretation(s) to retrieve their results from the database. While this approach performs reasonably well for the most simple and straightforward keyword queries, it does not provide end users with any means to retrieve results of the less probable interpretations or to obtain an overview of the data available in the database. This problem becomes even more critical in face of large scale datasets, where the space of the possible interpretations of a keyword query can become too big to be completely materialized and ranked. The big and flat schemas of heterogeneous large-scale databases cannot provide an informative overview of the data stored.

For solving these problems additional techniques are needed, namely *incremental query construction*, *diversification of search results over structured data*, *scalable algorithms for incremental query construction over large scale datasets*, and *enrichment of large scale databases with semantic information*. *Incremental query construction* addresses the scenario in which the keyword query interpretation intended by the user is not found within the top ranked results. This technique enables users to clarify their search intents step by step, and to obtain the intended structural interpretation even if this interpretation is less probable. *Diversification of search results over structured data* aims at minimizing the risk of users' dissatisfaction by balancing relevance and novelty of search results. Using diversification, the users can obtain a quick glance at the relevant search results within a database. *Scalable algorithms for interactive query construction over large scale datasets* are needed to ensure an efficient query construction procedure for the real-world large scale heterogeneous datasets such as, for example, Freebase [BEP+08], DBpedia [BLK+09], WikiTaxonomy [PS08], Probase [WLWZ12], and others. Finally, *enrichment of large scale databases with semantic information* can help to increase efficiency of incremental query construction over large scale data, and also provide a good starting point for future applications which can be build upon a wide variety of relational data clearly arranged in the semantic categories.

This thesis will thus aim at solving the following problems:

Problem 1. *How to enable the user to incrementally refine a keyword query into the intended interpretation on the target database?*

Existing approaches to keyword query disambiguation over structured data automatically translate a keyword query in a ranked list of the most likely structural interpretations and execute some of these interpretations to retrieve results from a database. While a query ranking approach is sufficient for the most simple and straightforward keyword queries, intended interpretations of ambiguous queries may not be found within the top ranked results. If the query interpretation intended by the user does not receive a good rank, the system should enable users to clarify their search intents step by step. Using incremental query construction, the user could construct structured queries efficiently, without necessarily knowing the database schema in advance or mastering a structured query language.

Can we offer interactive solutions addressing the shortcomings of the existing query disambiguation approaches?

Problem 2. *How to provide database search results over structured data with an increasing level of novelty?*

Keyword queries over structured data are notoriously ambiguous. No single interpretation of a keyword query can satisfy all users, and multiple interpretations may yield overlapping results. The key challenge here is to give users a quick glance of the major plausible interpretations of a keyword query in the underlying database, to enable user to effectively select the intended interpretation. To address similar problem in Information Retrieval, a technique called diversification is used. Diversification aims at minimizing the risk of user's dissatisfaction by balancing relevance and novelty of search results. Whereas diversification of search results on unstructured documents is a well-studied problem, diversification of search results over structured databases attracted much less attention.

Can we provide such search result diversification methods for structured data?

Problem 3. *How to enable efficient incremental query construction over large scale databases?*

With an increasing amount of structured data available on the Web, interactive query construction approaches mentioned above in Problem 1 need to become highly scalable to perform well on databases containing thousands of tables and millions of entities efficiently.

On the one hand, when the database schema graph is very big, the user interaction options generated by the query construction system need to become highly informative to enable an efficient reduction of the search space. On the other hand, the interpretation space of a keyword query in a very big database is usually too big

to be materialized completely, such that approaches to incremental query construction need to be adapted to materialize and explore a very large query interpretation space incrementally.

Can we offer scalable and efficient solutions to these problems?

Problem 4. *How to enrich a large scale database with the semantic categories of an ontology?*

When the schema graph of the database is big, a keyword can have a large number of occurrences spread across the database. Given a large scale database such as Freebase, it becomes crucial to provide effective and efficient structures that give users a quick and informative overview of the data available and provide a backbone for the wide variety of applications such as incremental query construction over large scale data described above in Problem 3, as well as schema summarization, question answering, and many others. Using these applications users who are not familiar with an internal database schema can efficiently narrow down the search space and retrieve the desired data quickly and accurately.

Ontologies are typically used for organizing large scale information and knowledge in a wide variety of domains. In this thesis we consider this problem at the example of the large scale Freebase dataset and the YAGO ontology. The YAGO ontology is a natural choice for organizing Freebase data, as both YAGO and Freebase share a large number of instances originating from Wikipedia. In related work, YAGO and Freebase were brought together in the context of Linked Data [BHBL09] that loosely connects their shared instances through the DBpedia references. However, the mapping of these datasets on the schema level does not exists.

Can we create such ontological layer over a large database schema?

1.2 Proposed Solution

Our proposed solutions to the above mentioned problems are based on probabilistic models, efficient algorithms, and ontologies.

In this thesis we will analyze in detail the characteristics of different database search systems and propose novel methods using probabilistic models, efficient algorithms, and ontologies. Research for efficient and effective database search is necessary for quite a lot of application environments, e.g. the World Wide Web, Enterprise Networks, Digital Libraries, Social Networks, Multimedia Repositories, and others. For all these and especially for the domain of large scale relational databases available on the Web, current query disambiguation and diversification algorithms are still rather poor or even inexistent, although at the same time they are more and more required due to the rapidly growing amount of data stored and searched for each of these particular scenarios.

The contributions of this thesis are manifold:

- First of all, we provide a detailed analysis of related research in the area of database usability in general and, in particular, database keyword search and discuss the benefits of incremental query construction over structured data.
- Then, we propose novel algorithms for incremental query construction. Using these algorithms, novice users can create structured queries over relational databases step-by-step starting from simple keywords without knowing the database schema a-priori or mastering a structured query language.
- In the next step, we propose advanced algorithms for search result diversification over structured data. This approach provides a quick overview of the variety of differently structured keyword search results available in the relational database.
- We evaluate our approaches to incremental query construction and search result diversification on real-world medium-sized datasets such as IMDB and Lyrics and show that they outperform the baseline ranking approaches in the related work and are scalable for the datasets with up to 100 tables.
- Following that, we move towards large scale data and address the problem of scalability of the initial approaches to incremental query construction in face of large scale databases containing thousands of tables. We further increase scalability and efficiency of incremental query construction proposed in this thesis to cope with the challenges resulting from the increased scale of data. We evaluate our approach on Freebase - a large scale datasets containing thousands of tables and confirm its efficiency through extensive experiments.
- Finally, in order to further increase efficiency of incremental query construction over large scale data, we enrich the wide variety of Freebase data with the semantic categories of the large scale YAGO ontology. We make the mapping between YAGO and Freebase datasets resulting from this work available to the community in the hope that this mapping can possibly facilitate many other future applications.

1.3 Thesis Structure

In **Chapter 2** we start by introducing general notions in the context of database usability and database keyword search in general and describe some of the characteristics of database keyword search systems, essential for understanding the rest of the dissertation. First of all, in Section 2.1, we review the aspects of database usability in general. Then, in Section 2.2 we focus on the various aspects of database keyword search. We start with data indexing techniques in Section 2.2.1. Following that, we discuss data-based and schema-based approaches to database keyword search in Sections 2.2.2 and 2.2.3, respectively. Then, we review ranking techniques in

Section 2.2.4. After that, in Section 2.2.5, we discuss state-of-the-art top-k query processing. In Section 2.2.6 we present current approaches of materializing and presenting search results. Then, in Section 2.2.7 we discuss aspects of query expressiveness in database keyword search. Finally, in Section 2.2.8, we present evaluation techniques typically applied in this context. More detailed reviews of specific related work are included in each of the next four chapters, centered on the four problems we aim to solve:

The first problem (**Problem 1**), namely incremental query construction, is addressed in **Chapter 3**, where we start with introducing the reader into the topic in Section 3.1. Following that, in Section 3.2 we provide a summary of contributions contained in this chapter. Then, we review relevant literature in Section 3.3. We provide an overview of the IQP system for incremental query construction in Section 3.4. In Section 3.5 we present the conceptual framework for probabilistic incremental query construction. Section 3.6 introduces a probabilistic model for assessing the likelihood of a structural query interpretation, which is required in order to enable an efficient query construction procedure. Section 3.7 presents the algorithms for generating (near-) optimal query construction plans. We evaluate the methods we introduced in Section 3.8 and discuss the results in Section 3.9.

Then, in **Chapter 4** we aim at providing database search results with increasing novelty to address **Problem 2**. After introducing the reader into the topic (Section 4.1), in Section 4.2 we provide a summary of contributions contained in this chapter. A detailed review of the literature follows in Section 4.3. In Section 4.4 we present our diversification scheme which enables obtaining relevant and diverse search results from a database. Section 4.5 introduces an adaptation of the evaluation metrics such as α-nDCG and S-recall, typically used to evaluate the quality of diversification over unstructured data, to a database scenario. Section 4.6 contains the results of our empirical investigation and confirms the quality of the proposed methods. We discuss the results presented in this chapter in Section 4.7.

Afterwards, in **Chapter 5** we move towards large scale data and present a set of techniques to boost the scalability of interactive query construction first proposed in Chapter 3 in face of a large scale dataset to address **Problem 3**. First of all, we introduce the reader into the topic in Section 5.1. Then, in Section 5.2 we provide a summary of contributions. Following that, we review related work in Section 5.3. Then, in Section 5.4, we review the query construction model that was first introduced in Chapter 3 and discuss its advantages and limitations. Section 5.5 talks about the efficiency of the query construction options and introduces an ontological layer at the top of the database schema that can significantly improve the efficiency of interactive query construction in face of a large scale dataset. Then, Section 5.6 presents efficient algorithms that enable incremental exploration of very large query interpretation spaces. Finally, we experimentally evaluate the efficiency of the proposed solution in Section 5.7 and discuss the results in Section 5.8.

Following that, in **Chapter 6** we focus on the **Problem 4** of enrichment of large

scale data with the semantic categories to advance the efficiency of query construction over large scale data presented in Chapter 5, and ensure portability of our solution to databases that are not associated with any ontological layer a-priori. We exemplify this problem using Freebase dataset and YAGO ontology. We introduce the reader into the topic and motivate our approach in Section 6.1. Following that, we summarize the contributions of this chapter in Section 6.2. Then, in Section 6.3 we provide a brief overview of the specific background from the area of schema matching. Following that, in Section 6.4 we perform a detailed analysis of the concept and instance distribution in the YAGO ontology. Then, in Section 6.5, we present matching techniques used to align YAGO and Freebase. In Section 6.6 we analyze the concept and instance distribution in the resulting YAGO+F structure combining YAGO and Freebase and provide an evaluation of the matching quality. Lastly, we discuss the matching results in Section 6.7.

Finally, in **Chapter 7** we provide a conclusion to the thesis with an enumeration of the contributions, while also discussing some possible future research directions and open challenges associated with these topics.

2 General Background

Databases are widely used in enterprises, organizations and on the Web to collect and disseminate scientific data, as well as information about real-world entities, such as people, products, publications and genes. In enterprises, database technology facilitates reporting and analysis; on the Web, databases enable transactions such as ticket booking and hotel reservation services as well as shopping and price comparison applications to name just a few examples. Recently, user created databases, such as DBpedia [BLK+09] and Freebase [BEP+08] gain on popularity, enabling end users to directly share structured data containing a lot of textual information in a variety of domains.

Structured queries are a powerful tool to precisely describe a user's informational need and retrieve the intended information from a database. However, manual creation of a structured query is a labor-intensive and error-prone task. This task requires exact knowledge of the database schema as well as proficiency in a query language, which are typically beyond the expertise of end users. In this context, it becomes increasingly important to create ways enabling novice users to work with structured data, taking advantage of the rich and well-defined semantics encoded in the data structures.

In this chapter we review state-of-the-art techniques aimed at enhancing database usability. We pay specific attention to database keyword search that became especially important with increasing availability of structured data on the Web.

2.1 Database Usability - An Overview

Database usability is a long-term research issue [JCE+07]. One of the early approaches to address this problem was the Query by Example (QBE) interfaces [BCC05, Zlo75]. QBE [Zlo75] provides a way for a user to perform queries without knowing a query language. The user of a QBE interface formulates the query by filling in the ap-

propriate skeleton tables with an example of a possible answer. XQBE [BCC05] is a visual interface to create XQuery expressions for XML data inspired by QBE. Although Query by Example interfaces free the user from learning the syntax of a query language, they still require users to comprehend schema information presented by the system. To address a similar problem, some commercial database products, e.g. Microsoft Access, offer visual query builder interfaces, which allow users to construct a query by combining graphical elements, without writing the actual SQL query expression. However, query graphs in a typical visual query builder interface have to be created starting from scratch. The user has to study the database schema and manually put together pieces of the query graph.

More recent approaches to database usability include Natural Language Query Interfaces [And95, AME07, Blu99, LYJ06, LCY+07], query auto-completion [BCSW07, NJ07], and adaptive forms [JJ08]. Natural Language Query Interfaces [And95, AME07, Blu99, LYJ06] are intended to enable users to specify structured queries in a human language. These systems use a variety of statistical and machine learning techniques to translate a user query written in a human language into the corresponding SQL expression intended by the user. Although Natural Language Query Interfaces provide certain flexibility for database access, state-of-the-art techniques still require users to use terminology compatible with the database schema and to form grammatically well-formed sentences. Moreover, supervised machine learning techniques applied in order to segment and label the elements of the natural language query expression mostly require labeled data from the specific domain for training to achieve a high precision in translation.

Query auto-completion [BCSW07, NJ07] assists users to form structured queries, by suggesting possible structures or terms based on the already entered sub-query. The goal of auto-completion if twofold: on the one hand it reduces the typing effort of the user; on the other hand it guides the user's typing towards the schema elements and values available inside the target database. This way, auto-completion techniques enable users to create database queries without complete schema knowledge. Initial auto-completion techniques relied on the user to provide correctly spelled prefixes in order to offer completion suggestions. A more recent technique [CK09] relaxes this assumption and enables correction of mistakes in the user input. However, auto-completion techniques still require users to use a dedicated query language to form structured queries.

Forms are typically used to query databases through a pre-defined query template. Form-based interfaces enable users to query a database without mastering a query language and knowing how the data is structured. Typically, the forms are static and their expressiveness is limited by the query templates defined a-priori. Recently, adaptive forms [JJ08] were proposed to alleviate this problem. When the existing forms do not support a user query, adaptive forms can be used to perform a form modification. Another recent form-related approach is a combination of keyword search and a form interface where user keywords are used in the first step to identify

the forms relevant to the user query [CBC+09].

2.2 Keyword Search in Databases - A General Characterization

One of the most flexible techniques enabling novice users to access structured data is keyword search. Keyword search interfaces found wide adaptation on the Web in the context of Information Retrieval, where user query targets are unstructured documents ordered with respect to some objective function [MRS08]. Recently, integration of Information Retrieval and DB technologies to provide users with flexible access to structured and semi-structured data attracted a lot of research attention (e.g. [HGP03, CRW05, AYCR+05, AYS05, Wei07, CD09, CWLL09, YQC10a]). Compared with typical Web search scenarios, database keyword search can provide several advantages, such as more complex query semantics, more precise and complete answers, as well as structured results [JCE+07]. Keyword search also become a popular way to access semi-structured data such as XML [HPB03, AYS05] and RDF [KKD11]. In this section we review enabling techniques of database keyword search.

In a typical database keyword search application, user keyword queries are answered in several steps. In an initial pre-processing step, query cleaning [PY08] or query segmentation [YS09] can be applied to identify meaningful query segments. Then, the search system identifies the database units (e.g. tables, attributes or tuples) containing any valid query segments. This step is typically performed using an inverted index created a-priori. In the next step, the valid join paths connecting the keyword occurrences are enumerated. These connections can be identified in two alternative ways, namely either *data-based* or *schema-based* [YQC10b]. Finally, the search results corresponding to the identified join paths are retrieved, ranked according to some objective function and presented to the user. In what follows, we describe these steps in more details.

2.2.1 Data Indexing using an Inverted Index

In order to identify database units containing users' keywords, existing search applications use indexing techniques. They can use either a separate index on each textual database attribute (column) (e.g. in [HP02]), or an inverted index over the whole database content (e.g. in [SA02, TL08]). The index is created a-priori in a pre-processing step.

A typical inverted index consists of a dictionary and posting lists [MRS08]. The dictionary contains a set of terms extracted from the cells of the textual attributes of the database tables. Optionally, the terms can be normalized using information retrieval techniques such as stop word removal, stemming, and others [MRS08]. The granularity of the postings in an inverted index may vary dependent on the specific

keyword search application. For each term t in the dictionary, a posting list with either names of the database tables, attributes, or identifiers of the tuples (rows) containing the term t can be employed. The granularity of the postings influences the space consumption and the efficiency of the index. A comparison between the row-based and the column-based indexes can be found e.g. in DBXplorer [SA02].

Figure 2.1 exemplifies a part of an inverted index containing the postings relevant to the keyword query *"hanks terminal"*. In this example, an indexing unit is a database attribute, such that each posting represents the attribute in which the term occurs.

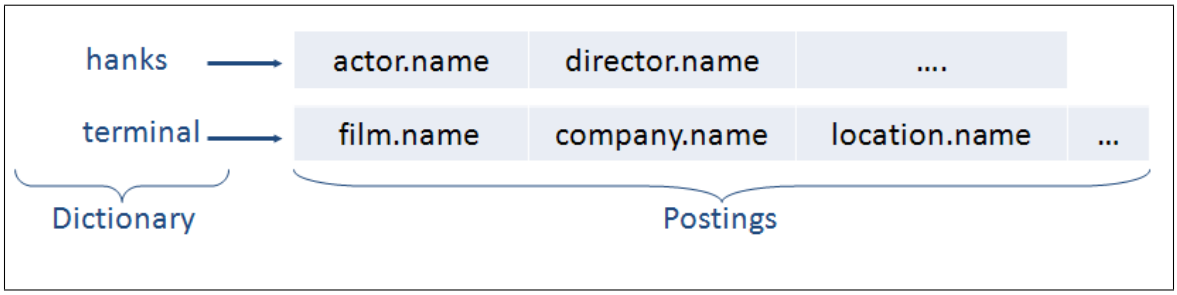


Figure 2.1 An Inverted Index Example. The figure exemplifies dictionary terms and postings relevant for the query *"hanks terminal"*. The postings represent the names of the attributes. For example, the term *"hanks"* occurs in the attributes *"actor.name"* and *"director.name"*.

As Figure 2.1 shows, the term *"hanks"* is contained in the attributes *"actor.name"*, and *"director.name"*, whereas the term *"terminal"* occurs in the columns *"film.name"*, *"company.name"*, and *"location.name"*.

2.2.2 Data-based Approaches

In the next step, single keyword occurrences identified using the inverted index are connected to the query structures that include join operations. The join structures connecting tuples that collectively contain user's keywords are known as *Join Tuple Trees (JTTs)*. The data-based approaches, such as e.g. [ABC+02, BHN+02, HWYY07, KPC+05, DXYW+07, LOF+08, TWRC09, LT11, FLW11], work on the data graph and identify search results using graph search algorithms.

The data-based keyword search approaches represent the data as a weighted directed graph $G(V, E)$. In a relational database, the nodes of the graph represent database tuples. The edges represent foreign to primary key relationships connecting tuples. The weights of the edges can reflect tuple proximity [BHN+02, DXYW+07]. The node and edge weights can be assigned using PageRank [BP98] or ObjectRank [BHP04].

The aim of keyword search is then to identify the minimal cost JTTs connecting the user's keywords. These JTTs are in fact the sub-tree(s) of the data graph. In

the literature, the minimality of JTTs is typically linked to relevance, as minimality reflects the closeness of the user's keywords in a search result. Thus the problem of finding optimal search result(s) is the *Minimum Group Steiner Tree Problem* [HRZ98] or *Top-k Group Steiner Tree Problem* [DXYW+07]. These problems are required to find the minimum cost interconnection(s) for a given set of objects. The Minimum Group Steiner Tree Problem is NP-complete [CSRL01], such that an efficient algorithm to solve this problem optimally in polynomial time is not known.

To compute search results efficiently, existing approaches relax the minimality condition [KPC+05, KS06, HWYY07, GKS08, LOF+08] and employ greedy graph search algorithms (e.g. [DXYW+07, KS06]) and index structures [HWYY07]. For example, BANKS employs [BHN+02] *Backward Expanding Search Algorithm* that creates rooted JTTs using Dijkstra's algorithm [Dij59] for shortest paths finding starting from each node containing keywords in the data graph. BANKS2 [KPC+05] uses bidirectional expansion to reduce the size of the search space and improve the efficiency.

The data-based approaches are schema-agnostic as they operate directly on the data, and thus are applicable not only to relational databases [BHN+02], but also to semi-structured data [TWRC09] as well as to the combinations of structured, semi-structured and unstructured data [LOF+08]. In case of XML, the nodes of the data graph (tree) correspond to the XML elements and the edges reflect element containment, or IDREF links [KPC+05]. The result of a keyword query over XML data is a subtree of the data graph rooted at the Lowest Common Ancestor (LCA) of a set of nodes that collectively match query keywords [BLCL09, CMKS03, GSBS03, KGM09, LYJ04, LC08, SCG07, XP05, XP08]. In RDF, the user's keywords are mapped to the nodes of the RDF graph and the neighborhood of these nodes is explored to extract subgraph(s) of the data graph containing all the user's keywords [KKD11]. Thus, the aim of keyword search over semi-structured data is to extract the minimal subtree(s) or subgraph(s) including all the user's keywords [AYS05, KKD11].

As data-based approaches operate on the data graph, the search results, i.e. JTTs, subtrees, or subgraphs become available in this step directly.

2.2.3 Schema-based Approaches

As keyword search interfaces do not offer sufficient expressiveness for the users to precisely specify their informational needs, the data-based approaches discussed above may return irrelevant or incomplete results. To cope with this limitation, schema-based approaches [SA02, HP02, HGP03, LYMC06, AME07, KKR+06, TZC+06, LLWZ07, TCRS07, ZWX+07, CBC+09, QYC09], perform keyword query disambiguation before retrieving any tuples from the database. The disambiguation process first translates a keyword query into a set of structured database queries, also known as *Candidate Networks (CNs)*, which are likely to expresses the user's informational need. Then the structured queries can be executed to retrieve the search results (i.e. JTTs) from the database.

Compared to the data-based keyword search, an advantage of the schema-based approaches is an increased expressiveness of the structured queries that can be used to interpret keywords. We discuss expressiveness of query interpretations in Section 2.2.7. The other advantage is that the optimization capabilities of the underlying database engine can be fully utilized to retrieve search results of the structural query interpretation. In this thesis, we focus on the schema-based keyword search approaches.

In the literature, a tuple is called *non-free* with respect to a keyword query K if it contains at least one keyword $k_i \in K$. A database table is called *non-free* if it contains at least one non-free tuple. Otherwise, the tuples and tables are called *free*. Join structures connecting non-free database tables are typically called *Candidate Networks (CNs)*. CNs were first discussed in DISCOVER [HP02] and DBXplorer [SA02], where they are called *join trees*. The connections between the non-free tables in a CN are established using foreign to primary key relationships as defined by the database schema. The CNs are in fact relational algebra expressions.

In order to answer a keyword query, the schema-based approaches first identify a set of valid CNs. The validation criteria for the CNs vary dependent on the expressiveness of the particular keyword search approach. For example, the valid CNs typically have to comply to the minimality condition, mining that no empty leaf nodes in the join tree are allowed. Usually, all user keywords need to be included (completeness condition). In addition, the maximal length of the joining path is normally restricted to enable efficient processing.

For example, consider the schema graph G in Figure 2.2. This graph contains five entity tables, such as *"actor"*, *"director"*, *"film"*, *"company"*, and *"location"* as well as four relational tables, such as *"acts"*, *"directs"*, *"employed_by"*, and *"situated_in"*. Based on the inverted index shown in Figure 2.1, we can associate the tables of this graph with the terms of the user's query *"Hanks Terminal"*.

We use the notation $a \bowtie b$ to denote that table a joins with table b on their primary key to foreign key relationship. In the example above, the candidate networks generated for the query *"Hanks Terminal"* include:

- *actor:"hanks"* $\bowtie$ *acts* $\bowtie$ *film:"terminal"*,
- *director:"hanks"* $\bowtie$ *directs* $\bowtie$ *film:"terminal"*,
- *actor:"hanks"* $\bowtie$ *employed_by* $\bowtie$ *company:"terminal"*, and
- *director:"hanks"* $\bowtie$ *employed_by* $\bowtie$ *company:"terminal"*.

In the small and medium-sized datasets, such as IMDB [IMD] and DBLP [Ley09], the candidate networks for a keyword query can be enumerated by performing Breadth-First-Search (BFS) on the schema graph [SA02, HP02, HGP03, LYMC06]. Qin et

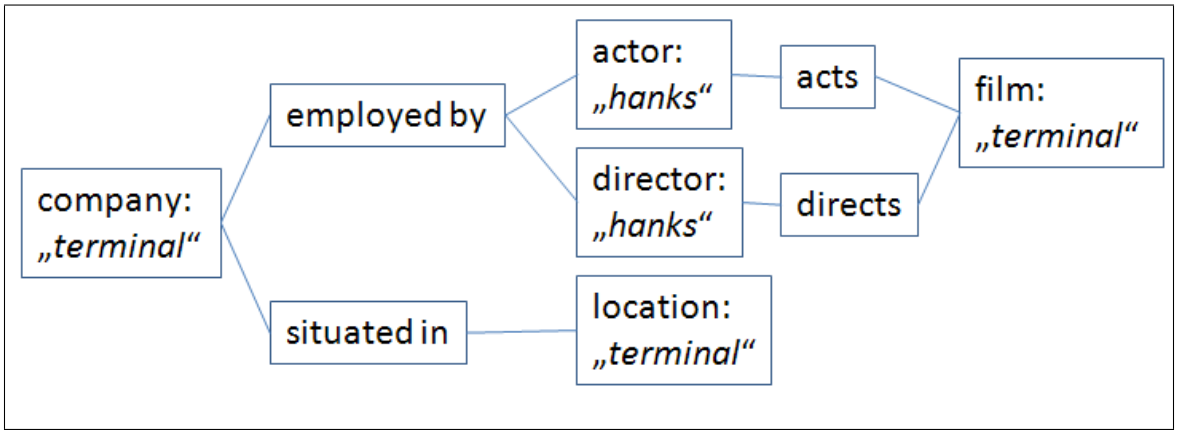


Figure 2.2 A Schema Graph Example. The figure exemplifies an undirected schema graph with 9 tables populated with keyword occurences for the query "Hanks Terminal". The nodes of the schema graph represent the database tables and the edges represent the foreign key relationships.

al. [QYC09] use SQL to compute all the interconnected tuple structures for a given keyword query.

In the schema-based approaches, the SQL statements corresponding to the identified candidate networks can be executed to retrieve search results.

2.2.4 Ranking of Queries and Search Results

Due to the ambiguity of keyword search, given a keyword query, there usually exist a large number of structural interpretations and search results in a database. With an increasing size of the data and schema, ranking became increasingly important. To this extent, existing keyword search approaches employ various scoring functions that assess the relevance of the possible structural query interpretations and search results. Using scoring functions keyword search approaches can obtain the most relevant query answers efficiently.

In practice, a variety of statistics related to data nodes, query results, and query logs can be utilized to make relevance assessment. The statistics related to the data nodes include TF-IDF (term frequency and inverse document frequency [MRS08]), PageRank [BP98], and others. The query results are often ranked based on the number of edges, weights on edges, size normalization, and redundancy penalty [BLCL09, CMKS03, DKS08, DXYW+07, GKS08, GSBS03, HWYY07, KS06, LOF+08, LFWZ08, LYMC06, LLWZ07, SLDG07, SGB+07, TL08, VOPT08, WPZ+06, WSR07, YLST07]. The query log based statistics can make use of frequencies of the query patterns found in search logs.

The weighting factors used in the state-of-the-art ranking functions can be classified as *structure-related*, taking into account structural patterns of the query interpre-

tations and results, and *keyword-related*, considering statistics of keyword occurrences.

The **structure-related weighting factors** include:

- **Tuple tree size** [SA02, HP02, LYMC06, LLWZ07]. Using this factor, the CNs and JTTs are considered as more relevant if the database elements containing keywords are closely connected. Initial approaches used a simple size normalization such as $1/size(X)$, where $size(X)$ is a number of joins in the corresponding CN or JTT.

  This factor can also be adjusted to take into account the number of free and non-free nodes separately [LLWZ07].

- **PageRank** [BHN+02, HHP08]. In Information Retrieval, PageRank considers the global importance of pages in the Web graph [BP98]. Applied to the databases, the PageRank-based techniques consider the global importance of the tables and tuples based on their connections in the schema and data graphs [YPS09].

- **Query log analysis**. The logs of a database search system is a valuable source to learn structural patterns (templates) used in user's queries. In addition, the logs can be used in order to improve the query segmentation, that, as a consequence, could improve the overall ranking [YS09]. However, the query logs are rarely available, such that the majority of the existing keyword search approaches make use of other available statistics.

The most commonly used **keyword-related weighting factors** are:

- **Relevance** [HGP03, LYMC06, MdSdM+07, LLWZ07]. In Information Retrieval, in order to estimate the relevance of a search result to a query, term frequency (TF) is typically used [MRS08]. TF is the number of times the term appears in a search result. The intuition behind TF is that the documents that are most relevant to the query should contain the query terms more often than the less relevant documents. Dependent on the document collection, this frequency can be further normalized. In database keyword search TF is used to assess the relevance of a tuple or a table attribute.

- **Document length normalization** [LYMC06]. As observed in Information Retrieval research, the keywords tend to occur more frequent in longer documents. In order not to over-estimate longer search results, document-length normalization can be applied to TF. In the database context, the length of the tuple or attribute can be used to perform such normalization.

- **Document frequency (DF)** [MdSdM+07]. In the context of a relational database, a tuple can be viewed as a document, and a set of tuples in a database table can be viewed as a document collection. Then, the document frequency

of a term in a database table is the number of tuples in this table containing the term. In case a keyword is very frequent in a database table, the match in this table can be considered as more typical.

- **Inverse document frequency (IDF)** [HGP03, LYMC06, MdSdM+07]. This factor reflects selectivity of a keyword in the database and is inversely proportional to DF. In case a keyword is very selective in a database table, the match in this table can be considered as more specific.

- **Matching schema terms** [LYMC06, TL08]. In case a keyword matches a schema term, such as an attribute or a table name, the match can be considered as more important than a match in an attribute value. For example, *schema-based document frequency* assigns a term matching a name of a text column or a table name with the largest document frequency value among all the terms in this column or table correspondingly.

- **Phrase-based ranking**. Phrase-based search has been shown to improve search effectiveness in Information Retrieval [BP98, LLYM04]. In database keyword search, modifications of term frequency values can be applied to assign higher weights to the phrase matches as opposed to the individual term matches [LYMC06].

- **Completeness** [LLWZ07]. Completeness gives higher weight to the CNs and JTTs including more user's keywords. This factor can be used to adjust semantic of the query towards either the *OR* or *AND* semantics.

In the literature, typical ranking functions for CNs and JTTs use several structure-related and keyword-related weighting factors in a combination. These ranking functions can be monotonic, e.g. [HGP03, LYMC06] and non-monotonic [LLWZ07, LWL+11].

Ranking techniques for effective database search evolved over the last decade. To rank query results, initial approaches such as DBXplorer [SA02], and DISCOVER [HP02] simply used the number of joins in the query interpretations and search results. Intuitively, the shorter joining sequences are considered to be more relevant than the longer sequences because they imply a closer association of the user's keywords [HP02]. Most of the follow-up approaches used the number of joins as one of the factors in ranking of query results and query interpretations.

Different works exploited the structure of the data and schema graphs to assess importance of the results. For example, BANKS [BHN+02] used a combination of tuple weights and edge weights in a tuple tree in a PageRank [BP98] style method to give higher weights to the highly connected tuples. In [HHP08], the authors extended PageRank for database keyword search in databases for which there is a natural flow of authority between their objects. In general, these methods consider the better connected tables and tuples within a database to be more important.

Ranking of search results has a long tradition in the field of Information Retrieval. DISCOVER2 [HGP03] first incorporated state-of-the-art Information Retrieval ranking formula in database search. The ranking formula was subsequently improved by Liu et al. [LYMC06] by using several refined weighting schemes specifically developed for the database keyword search. Compared to DISCOVER2, Liu et al. proposed normalizations of the TF-IDF weighting in the database context.

Various ranking methods can be applied to both, structured queries interpreting keywords (i.e. CNs) and search results (JTTs). Ranking search results is discussed in [HGP03, LYMC06, LLWZ07]. The approaches to keyword query disambiguation employed ranking of query interpretations (CNs) rather than JTTs. A crucial step of keyword query disambiguation is to assess the likelihood of the possible interpretations and pick the most probable ones to be executed against the database. In LABRADOR [MdSdM+07], the authors employed probabilistic models to estimate the likelihood of the possible structural query interpretations.

2.2.5 Top-k Query Processing

As an interpretation space of a keyword query in a database is typically large, it is oft not feasible to completely materialize the search space. To this extent, existing keyword search approaches employ various techniques to efficiently obtain the top-k most relevant query answers.

Li et al. considered the problem of generation of search results in the decreasing order of their individual ranks in the context of Web documents and XML [LCVA02]. In the data-based keyword search, the focus of the top-k processing is to quickly materialize the most relevant search results (JTTs) using the data graph. The algorithms for the top-k JTT generation apply heuristics to obtain the most relevant search results efficiently without the complete materialization of the search space [HGP03, LLWZ07].

In the schema-based keyword search, existing approaches assume that it is feasible to enumerate and rank the entire space of possible query interpretations (CNs) if the maximal length of the join path is restricted [LLWZ07]. However, given a large-scale database, this assumption may not hold anymore. In order to enumerate all valid CNs, DISCOVER [HP02] proposed a Breadth-First (BF) algorithm that finds all valid subgraphs of the schema graph of size n by traversal of the schema graph in a breadth first order. This technique found application in many further schema-based works, e.g. [HGP03, LYMC06, LLWZ07, LWL08].

Given a CN, one can execute the SQL query corresponding to this CN to retrieve search results. Thus, the focus of top-k query processing in the schema-based approaches is, given a ranked list of CNs, to find a query execution strategy to retrieve the overall top-k search results efficiently. Given a list of CNs, a naive solution to find top-k query answers is to issue a SQL query for each CN, to union the search results of these queries, and to sort the results according to their scores. Given a big

number of CNs and large result sizes, this approach can become inefficient.

In the literature, the top-k query processing algorithm was first introduced by Fagin [Fag99]. Given multiple lists of ranked results and a monotone score aggregating function, Fagin's algorithm and Threshold Algorithm (TA) can retrieve the top-k objects efficiently by early stopping the query execution process as soon as the top-k results were found [Fag99, Fag02]. While Fagin's algorithm evaluates the top-k objects in each list, TA uses the score upper bound to determine criteria for early stopping.

DISCOVER2 [HGP03] introduced query evaluation strategies optimized for early stopping the query execution in line with TA. The idea is to define an upper bound score, such that any potential result from the future execution of a CN will not have a higher score. Then the query execution can stop immediately as soon as k results with the scores higher than upper bound are retrieved. This optimization technique found application for other variants of top-k queries [LYMC06, CMKS03, XIG09].

Majority of the existing top-k approaches assume the ranking functions for JTTs to be monotone [Fag99, MCYC06, BMS+06, XCH06, DGKT06, NCS+01, IAE04, CH02]. In the context of keyword search monotonicity means that the higher scores of the individual tuple in a JTT will result in a higher score of the JTT as a whole [HGP03, LYMC06]. SPARK [LLWZ07, LWL+11] addresses the problem of top-k processing in presence of a non-monotone ranking function that aggregates relevance, completeness and size normalization factors in a non-linear combination.

2.2.6 Materializing and Presenting Search Results

In the majority of cases, a result of database keyword search is a joining network of tuples (JTT). A special case of JTT is a minimal total joining network of tuples (MTJNT) that does not contain any empty leaf nodes, and includes all keywords of the user query [HP02].

In the data-based approaches, such as BANKS [ABC+02], this result is directly available by search on the data graph. In the schema-based keyword search, the result needs to be materialized in an extra step. In fact, a candidate network corresponds to a single SQL statement that joins the tables as specified in the CN tree, and selects those rows that contain the keywords. The search results retrieved by this procedure can be ranked according to some objective function as discussed in Section 2.2.4.

Besides retrieval of the tuples explicitly matching user's keywords, some studies focused on identifying data that do not match keywords, but are implicitly relevant to user queries [HPB03, KSI06, LC07, LWC07, TY09].

In the context of Web search, a search result is typically presented as a snippet - a brief passage representing the content of the result. The snippets give users a possibility to get a quick glance at the returned results and judge their relevance. Generation of snippets has been considered in the context of XML keyword

search [HLC08a, HLC08b].

Another useful technique to organize database keyword search results is clustering [XP05, HKPS06, KZGM09]. The clusters help to disambiguate the query automatically, grouping together similar search results.

2.2.7 Query Expressiveness

There is a natural trade-off between the expressiveness and the usability of database search interfaces [JCE+07]. The expressiveness of query interpretations used in database keyword search approaches has evolved greatly over the last decade.

The first works on database keyword search had very limited expressiveness and focused on keyword queries in which all keywords were contained in the same tuple [GSVGM98, MV00a, MV00b]. Following that, DBXplorer [SA02], DISCOVER [HP02] and BANKS [ABC+02, BHN+02] considered join trees that included tuples from different relations and collectively contained keywords of the user query.

DBXplorer [SA02], DISCOVER [HP02] and BANKS [ABC+02, BHN+02] assumed *AND* semantics for an answer, whereas DISCOVER2 [HGP03] enabled *OR* semantics. The methods for interpreting keywords evolved from considering attribute values only (e.g. [SA02]), to include schema terms (e.g. [LYMC06]) and aggregation operators [TL08].

DBXplorer considered exact and prefix matches between keywords and attribute values. BANKS [ABC+02, BHN+02] also took the matches between keywords and metadata, such as column, table and view names, into account.

Labeled keyword search [CMKS03, LWL08, LYJ04] enabled users who are familiar with the database schema to explicitly specify the mapping between keywords and metadata labels in the query, thus mapping the keywords exclusively to the attributes complying with the specific label.

Analytical keyword queries [WSR07, TL08] enable grouping and aggregation of search results, either based on user's keywords or based on the search results.

Natural Language query interfaces such as [And95, Blu99, LYJ06, AME07, LCY+07] enable users to put structured queries complying with the database schema in a natural query language.

Finally, structured query languages such as SQL, XQuery, and SPARQL provide the most expressive querying capabilities, requiring detailed knowledge of the schema and query language from the user.

2.2.8 Evaluation Techniques

Empirical evaluation of database keyword search is typically performed using a benchmark. In the context of XML keyword search, an established benchmark is provided

by INEX (INitiative for the Evaluation of XML Retrieval) [GKST11]. In the context of database keyword search, popular datasets used for evaluation are e.g. IMDB (The Internet Movie Database) [IMD], Lyrics [LYMC06], DBLP [Ley09], and others. Still, development of benchmarks for keyword search on structured data is an open issue [CWLL09].

Another type of keyword search evaluation is a theoretical evaluation, that reasons about a keyword search approach formally [LC08, TWC11]. For example, [LC08] proposes four properties, such as data monotonicity, query monotonicity, data consistency, and query consistency that an XML keyword search strategy should ideally satisfy. Termehchy et al. [TWC11] provides a theoretical framework to measure the amount of design independence provided by a schema-free query interface.

3 Incremental Query Construction

3.1 Introduction [1]

Structured queries are a powerful tool to precisely describe a user's informational need and retrieve the intended information from a database. However, manual creation of a structured query is a labour-intensive and error-prone task. This task requires exact knowledge of the database schema as well as proficiency in a query language, which are typically beyond the expertise of end users. Keyword queries on the other hand require neither a-priori schema knowledge nor query construction skills and can be performed efficiently by novice users. However, keyword search lacks expressiveness to precisely describe a user's informational need, and may return irrelevant or incomplete results.

To take advantage of both, i.e., expressiveness of structured queries and usability of keyword search, recent schema-based approaches [KKR+06, TL08, TCRS07, ZWX+07] translate a keyword query into a ranked list of structured queries, such that the user can select the query that best represents her informational need. While a query ranking approach is sufficient for the most simple and straightforward keyword queries, intended interpretations of ambiguous queries may not be found within the top ranked results. This can happen for two reasons: first, as each keyword can potentially occur in any textual attribute of a database, the number of structural interpretations grows sharply with the complexity of the database schema and the length of the keyword query. In fact, with a complex database and a lengthy keyword query, it is computationally infeasible to materialize and sort all possible structural query interpretations

[1] Chapter 3 includes material reprinted with permission from the following papers: Elena Demidova, Xuan Zhou, Wolfgang Nejdl. *A Probabilistic Scheme for Keyword-Based Incremental Query Construction.* In IEEE Trans. on Knowl. and Data Eng. (TKDE), 24(3):426-439, March 2012, DOI: http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.40. ©2012 IEEE; Elena Demidova, Xuan Zhou, Wolfgang Nejdl. *IQ^P: Incremental Query Construction, a Probabilistic Approach.* In: Proceedings of the 26th IEEE International Conference on Data Engineering, March 1-6, 2010, Long Beach, California, USA. DOI: 10.1109/ICDE.2010.5447929. ©2010 IEEE.

in online query processing. Second, even a theoretically optimal ranking algorithm can, at best, rank the most common query interpretations highest; whereas the users with less frequent informational needs may not receive adequate results. For example, if the majority of users who issued the keyword query "London" were interested in a city guide of London, the results referring to Jack London as a book author will receive a low rank. In case a ranking function fails to identify the user intended query interpretation within the top results, the user will need to examine all interpretations prior to the intended one, which is tedious and error-prone for the user.

In this chapter we present IQP, a novel system aimed at bridging the gap between usability of keyword search and expressiveness of database queries. Using IQP a user can benefit from both, a conventional ranking interface and a more controllable query construction interface. The former allows the user to immediately identify the most common interpretation of her query. The latter enables the user to clarify her search intent step by step, which is highly helpful when the intended query interpretation is not found within the top ranked results. For instance, if a user issues a keyword query *"London Age"*, IQP would ask a small number of questions, such as *"Is London a person?"* or *"Are you looking for a city's age?"*. Based on the user's responses, IQP is able to accurately identify the structured query representing the user's informational need. Using IQP, users are able to construct structured queries efficiently, without necessarily knowing the database schema or mastering a query language.

The algorithms presented in this chapter thus focus on addressing Problem 1 presented in Chapter 1, namely enabling users to incrementally refine a keyword query into the intended interpretation on the target database. Our IQP system consists of three components: (1) a probability-based framework that formally defines the process of incremental query construction, which does not require any a-priori schema knowledge or proficiency in a query language; (2) a probabilistic model to estimate the probabilities of structural query interpretations, so as to identify meaningful items for users to interact with; (3) an algorithm for generating the optimal query construction plan, which enables a user to obtain the intended structured query within a minimal number of interactions. In this chapter, we present the detailed design of these components. We also show the effectiveness of our system through a user study and extensive experiments using real world datasets.

The rest of this chapter is organized as follows: Section 3.2 provides a summary of contributions contained in this chapter. Section 3.3 summarizes related work. Section 3.4 gives an overview of the IQP system. Section 3.5 presents the conceptual framework for incremental query construction. Section 3.6 introduces a probabilistic model for assessing the likelihood of a structural query interpretation. Section 3.7 presents algorithms for generating (near-) optimal query construction plans. Our experiment results are reported in Section 3.8. Finally, Section 3.9 provides a discussion of the results.

3.2 Summary of IQP Contributions

The IQP system proposed in this chapter mainly contributes to the areas of database usability described in Section 2.1, and database keyword search discussed in Section 2.2.

Several state-of-the-art approaches described in Section 2.1 aim at enhancing database usability and assist users in creation of structured queries. However, these approaches typically require prior knowledge of the database schema and/or a dedicated query language. These approaches include Query by Example (QBE) interfaces [BCC05, Zlo75]), visual query builder interfaces, e.g. Microsoft Access, Natural Language Query Interfaces [And95, AME07, Blu99, LYJ06], and query autocompletion [BCSW07, NJ07, CK09]. In contrast to these approaches, IQP proposed in this chapter requires neither prior knowledge of the database schema, nor a structured query language, and assists users in creation of structured queries starting from simple keywords.

In database keyword search, users' keyword queries can be very ambiguous having many different interpretations in the underlying database, leading to a big variety of differently structured search results. As state-of-the-art approaches to database keyword search mainly focus on retrieval of a few most likely structured queries and/or search results ([HGP03, LYMC06, HWYY07, ZWX+07, TL08], etc.), existing approaches can only satisfy the users issuing the most simple and straightforward keyword queries. At the same time, informational needs of the users searching for less typical interpretations of ambiguous keyword queries may remain unsatisfied, as the user intended interpretations of these queries are not always found within the top ranked results. Whereas state-of-the-art approaches to database keyword search do not offer users opportunities to clarify the semantics of their queries, IQP enables users to incrementally refine keywords into the desired structured queries. Using IQP, users can retrieve the intended search results of ambiguous keyword queries efficiently, within only a few iterations, even if these results are not very typical.

Initial approaches to incremental query construction presented in [ZZDN08, DZZN09, ZZM+09] did not take probability of query interpretations into account, such that the resulting query construction process may be not optimal. In contrast to these approaches, IQP takes probability of query interpretations into account to enable an efficient query construction process.

In summary, the IQP contributions described in this chapter include:

- A probabilistic model that enables IQP to assess probability of structured interpretations and query construction options for a given keyword query.
- A probabilistic framework for incremental query construction that empowers ordinary users to interactively guide the search process towards the intended results starting from simple keywords.

- The algorithms for generating (near-) optimal query construction plans that enable users to create desired query interpretations efficiently, within only a few iterations.

Our user study and experiments with real-world datasets have demonstrated that the proposed approach outperforms SQAK [TL08], a state-of-the art system to schema-based keyword search, if the user intended query interpretation cannot be found within the top ranked results. Our experiments have shown that the proposed probability estimates enable us to reduce the interaction cost of query construction significantly compared with a base line approach that did not take probability of query interpretations into account. Our simulations confirmed the scalability of the proposed algorithms for the medium-sized databases containing up to 100 tables and the quality of the proposed algorithms.

3.3 Specific Background

IQP supports efficient incremental construction of structured queries, which is related to databases and information retrieval. In Chapter 2 we provided an overview of the related work in the areas of database usability in general and database keyword search in particular. In this section we discuss some additional related work on faceted search and incremental query construction.

3.3.1 Faceted Search

User-driven query disambiguation has been successfully applied in Information Retrieval in the context of faceted search [MRS08]. Faceted search engines, such as the product search engine of Google and the Clusty search engine [clu], organize search results into meaningful groups, called facets, by applying some clustering or categorization algorithms. Users can easily shrink the scope of the search by focusing on a small number of facets. Several navigational techniques [Hea06, KÖ5, BRWD+08, WSR07] were proposed to support users in finding information in a hierarchy of faceted categories. The interface of IQP is similar to a faceted interface, whereas each facet corresponds to a query construction option.

3.3.2 Incremental Query Construction

In [DZZN09] and [ZZDN08] we presented SUITS, a faceted interface that enables users to interactively disambiguate keyword queries. However, SUITS lacks a theoretical foundation for verifying its effectiveness. In [ZZM+09], we extended SUITS with a formal framework for incremental query construction and applied this framework to Semantic Web data. However, as the framework of [ZZM+09] does not take

the probability of query interpretations into account, the resulting query construction process may be not optimal. Compared with [DZZN09, ZZM+09, ZZDN08] the contributions of IQP include: (1) a probabilistic framework to define the process of incremental query construction; (2) a probabilistic model to estimate the probabilities of structural query interpretations; (3) an algorithm for generating an optimal query construction plan based on Information Gain.

3.4 Overview of IQP

IQP query construction interface is designed to enhance the ranking centric approaches to database keyword search, such as [KKR+06, TL08, TCRS07, ZWX+07], by giving users control over the query disambiguation process. Fig. 3.1[2] illustrates the user interface of IQP.

The user interface of IQP consists of (1) a search field to input keyword queries, (2) a query construction window which presents query construction options, (3) a query window listing structured queries and (4) a result window for presenting search results. When a user issues a keyword query, IQP provides the user with a ranked list of structured queries (as interpretations of the keyword query) and the corresponding results, which are presented in the query and result windows respectively. If the user identifies the desired structured query, she can double click the query to retrieve its results. If the user cannot immediately find the desired structured queries or results, she can resort to the query construction window, which enables the user to refine the structured queries being suggested.

For instance, to find the movies starring Tom Hanks and Tom Cruise in 2001, the user Alice issues a keyword query "Hanks Cruise 2001" to IQP. Without knowing the exact informational need of Alice, IQP translates the keyword query into a number of structured queries, which give different interpretations to Alice's keywords. For example, one possible structured query searches for the movies of 2001 starring Hanks and entitled "Cruise", whereas another query searches for the movies starring Cruise and directed by Hanks. These possible interpretations and corresponding results are ranked and presented in the query and result windows respectively (Fig. 3.1 (3) and (4)). Simultaneously, IQP generates a set of query construction options, and presents these options in the query construction window (Fig. 3.1 (2)).

If the query interpretation desired by Alice is shown in the query window, she can double click this interpretation and obtain the results. In case Alice cannot immediately identify the desired information in the query or result windows, she can use the query construction interface to refine the presented list of queries. For instance, Alice can select a query construction option specifying that "Hanks" is

[2] Fig. 3.1 includes the following images: http://de.fotolia.com/id/25660633 ©Anatoly Maslennikov - Fotolia.com; http://de.fotolia.com/id/10056489 ©ioannis kounadeas - Fotolia.com; http://de.fotolia.com/id/9974098 ©XYZproject - Fotolia.com

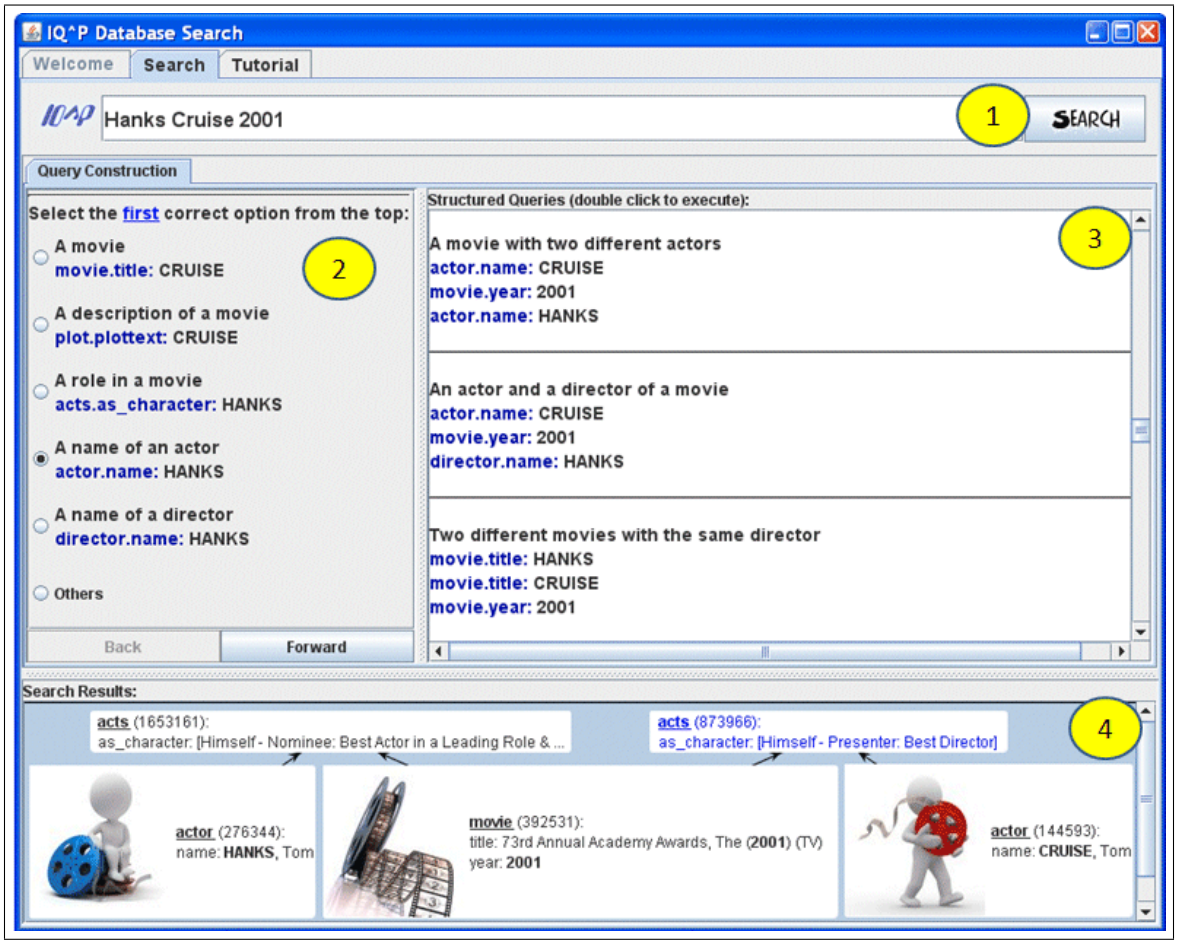


Figure 3.1 IQP User Interface. Its components include: (1) an input field for keyword queries, (2) a query construction panel, (3) top-k structured queries, and (4) query results. ©2012 IEEE.

an actor's name. Whenever Alice selects an option, the query and result windows zoom into the structured queries and results satisfying the selected options. At the same time, the set of the query construction options is updated to enable Alice to further clarify her informational need. Interaction between Alice and the system continues until Alice obtains the desired structured query or results. In fact, the query construction options shown in Fig. 3.1 are structured queries too, while they interpret only a part of keywords of the Alice's keyword query.

As the number of possible structural interpretations of a keyword query grows sharply with the number of keywords and the size of the database schema, ranking alone is not always sufficient to help users identify desired structured queries or results. While some natural language processing techniques, such as phrasing and tokenization, can be used to restrict the interpretation space, they have limited usage. For instance, although the keywords "Tom" and "Hanks" can be typically interpreted as the name of a single actor, a user may use these keywords to refer to a movie with

two different actors named "Tom Cruise" and "Colin Hanks". In contrast, a set of properly selected query construction options enables a user to exponentially reduce the interpretation space and thus obtain the intended structured query quickly. For example, if "Hanks" can be a part of a movie title or an actor's name in a movie database, Alice can select the "Hanks is an actor's name" option, and thereby implicitly eliminate all structured queries that interpret "Hanks" differently.

IQP can present structured queries either in a natural language (using e.g. techniques such as [KSI10]) or as graphs similar to the visual query builder interfaces. To generate meaningful options IQP requires appropriate naming of the schema elements in a relational database. In the following, we present the detailed techniques for enabling the above interface.

3.5 Query Construction Framework

This section presents a conceptual framework for incremental query construction. First, we show how IQP translates a keyword query to a set of structured queries of relational database. Then, we introduce the concept of query construction plan, which can guide a user through an interactive process to obtain the intended query.

3.5.1 From Keywords to Structured Queries

We first define the concepts of keyword queries and structured queries.

Definition 3.5.1. ***Keyword Query:*** *a keyword query $K = k_1, \ldots, k_m$ is a bag of words, where duplicates are allowed.*

Some examples of keyword queries are K_1="titanic 1997", K_2="actor tom hanks", K_3="movie hanks 2001", and K_4 ="number of movies with tom hanks".

Definition 3.5.2. ***Structured Query:*** *a structured query Q in IQP is an expression of relational algebra which is composed of tables and the following operators and predicates:*

Operator: *The operators for forming structured queries include selection (σ) and natural join ($\bowtie$).*

Predicate: *A predicate is in the form $k_1, \ldots, k_m \subset A$, indicating that the bag of keywords $k_1, \ldots, k_m$ is contained in the value of the attribute A.*

Some examples of structured queries are as follows:

- $\sigma_{titanic \in title \cap 1997 \in year}(Movie)$:
  A movie with title "Titanic" which is produced in 1997.

- $\sigma_{\{tom,hanks\}\subset name}(Actor)$:
  An actor whose name is Tom Hanks.

- $\sigma_{hanks\in name}(Actor) \bowtie Acts \bowtie \sigma_{2001\in year}(Movie)$:
  Movies starring Hanks in 2001.

To construct a structured query from a keyword query, IQP first interprets each keyword to an element of a structured query, which is called keyword interpretation.

Definition 3.5.3. ***Keyword Interpretation:*** *a keyword interpretation is denoted by $A_i : k_i$, which maps a keyword k_i to an element A_i of a structured query. A_i can refer to a table, an operator, an attribute, or a value in a predicate. A_i is also called an interpretation of k_i.*

In IQP, a keyword is interpreted to an element of a structured query, if it matches the name or the value of that element. For example, to translate a keyword query K_2="actor tom hanks", we can make the following keyword interpretations: $Actor : actor$, $tom \in name : tom$, and $hanks \in name : hanks$, where "actor" is interpreted as a table, and "tom" and "hanks" are interpreted to values in predicates.

A set of keyword interpretations can be connected to form a structured query. For example, based on the keyword interpretations mentioned above, we can build a structured query $\sigma_{\{tom,hanks\}\subset name}(Actor)$. We call a mapping from a keyword query to a structured query a query interpretation.

Definition 3.5.4. ***Query Interpretation:*** *given a keyword query K, we say that a structured query Q is a query interpretation of K, if and only if there is a set of keyword interpretations $A_i : k_i$, where $Ai \in Q$ and $k_i \in K$, such that (1) each keyword in K is interpreted as at most one element of Q; (2) given a sub-structure Q' of Q, if after removing Q' the remaining structure of Q is also a structured query, then there is at least one keyword in K that is interpreted as an element of Q'.*

If $A_i : k_i$ contains the keyword interpretations for all the keywords in K, we call Q a complete interpretation of K. Otherwise, we call Q a partial interpretation of K.

The first condition in Def. 3.5.4 guarantees that each keyword has a unique interpretation w.r.t. the structured query. This reflects the intuition that users typically assign one specific meaning to a keyword. The second condition ensures that a query interpretation does not contain any redundant parts, i.e., no part can be excluded from a query such that remaining query still contains the same number of keywords. This corresponds to the minimality condition introduced in [HP02, TL08]. For instance, $\sigma_{\{tom,hanks\}\subset title}(Movie)$ and $\sigma_{\{tom,hanks\}\subset name}(Actor)$ are both valid interpretations of K_1="tom hanks", and $\sigma_{\{tom,hanks\}\subset name}(Actor) \bowtie Acts \bowtie \sigma_{2001\in year}(Movie)$ is a valid interpretation of K_2="movie tom hanks 2001". However, $\sigma_{\{tom,hanks\}\subset name}(Actor) \bowtie Acts \bowtie (Movie)$ is not a valid interpretation of K_3="actor tom hanks", as $\bowtie Acts \bowtie (Movie)$ does not contain any interpretation of a keyword in K_3 and thus violates Def. 3.5.4(2).

In our query construction framework, each complete interpretation of a keyword query can be a possible structured query desired by the user. Each partial interpretation can be used as a query construction option. As the number of possible complete interpretations of a single keyword query is usually large, the objective of query construction is to quickly identify the interpretation desired by the user.

Definition 3.5.5. ***Interpretation Space:*** *Given a keyword query K, the interpretation space of K is the entire set of complete interpretations of K.*

Our current implementation of IQP considered only a subset of operators and predicates in relational algebra. Involvement of other operators, such as projection, has been considered in the related work [LC07]. It is an interesting direction for our future research. Currently, a structured query created by IQP returns the values of all referred attributes which corresponds to "SELECT *" in SQL.

3.5.2 Query Interpretation Generation

As creation of the entire interpretation space of a keyword query is expensive, IQP uses a set of pre-computed *query templates* to accelerate this process. A query template is a pattern typically used to issue structured queries.

Definition 3.5.6. ***Query Template:*** *a query template T is a structured query whose predicates do not contain any keywords, but variables.*

For instance, $T_1 = \sigma_{?\in name}(Actor) \bowtie Acts \bowtie \sigma_{?\in year}(Movie)$ is a frequently used query template, as users often look for a movie starring a certain actor in a certain year. Other examples of query templates include $\sigma_{?\in name}(Director) \bowtie Directs \bowtie \sigma_{?\in year}(Movie)$ looking for movies directed by specific person and $\sigma_{?\in name}(Actor) \bowtie Acts_1 \bowtie Movie \bowtie Acts_2 \bowtie \sigma_{?\in name}(Actor)$ searching for a movie with two actors.

A structured query (query interpretation) is a composition of a query template and a set of keyword interpretations. IQP uses the following process to create an interpretation of a keyword query K:

- Find an interpretation for each keyword in K;
- Pick an available query template T;
- Combine T and the keyword interpretations into a query interpretation.

For example, given a query K_3="movie hanks 2001", we first find a set of keyword interpretations $Movie$: $movie$, $\sigma_{hanks \in name}$: $hanks$, $\sigma_{2001 \in year}$: 2001 and a query template $T_1 = \sigma_{?\in name}(Actor) \bowtie Acts \bowtie \sigma_{?\in year}(Movie)$, and then combine them into a structured query $\sigma_{hanks \in name}(Actor) \bowtie Acts \bowtie \sigma_{2001 \in year}(Movie)$. This process can be repeated to generate the entire interpretation space of a keyword query.

As the expressiveness of IQP is bounded by the pre-computed query templates, it is essential to generate query templates that cover as many informational needs of the users as possible. IQP generates query templates using three approaches.

- Firstly, query templates can be automatically generated by exploring possible join paths of the database schema within a predefined length. The methods for automatic template generation can be found in some existing work, e.g. [CGRM06, HGP03, HP02]. Using this method, IQP can achieve similar expressiveness as the ranking approaches that generate interpretations automatically.
- Secondly, if a database is used frequently, its query log should contain a large number of structured queries. The common patterns in the query log can be used as query templates.
- Lastly, query templates can also be manually defined or adjusted by a database administrator, based on the intended application of the database.

When creating structured queries, sometimes we may not find an appropriate complete interpretation for a keyword query. In case one of the keywords is misspelled or does not exist in the target database, it is excluded from the query construction process. The user can either construct a partial interpretation without this keyword or reconsider the keywords. If the query desired by the user is beyond the expressiveness of IQP, the user has to resort to other means, e.g. creating a SQL query manually.

3.5.3 Sub-Query Relationship

During query construction, IQP utilizes the sub-query relationships between the partial and complete query interpretations to quickly reduce the interpretation space of a keyword query.

Definition 3.5.7. ***Sub-Query:*** *Given two structured queries Q and Q', we say that Q' is a sub-query of Q (or Q' subsumes Q), iff Q' is a sub-structure of Q.*

For instance, $Q = \sigma_{\{tom,hanks\}\subset name}(Actor) \bowtie Acts \bowtie \sigma_{2001\in year}(Movie)$ is an interpretation of K="movie tom hanks 2001". $Q' = \sigma_{\{hanks\}\in name}(Actor)$ is a sub-query of Q, because Q' is a substructure of Q. The sub-query relationship is transitive. Namely, if Q' is a sub-query of Q and Q'' is a sub-query of Q', then Q'' is also a sub-query of Q.

In a query construction process, the user is presented with a number of query construction options, i.e., partial interpretations of the keyword query (Fig. 3.1 (2)). When the user accepts an option, she actually confirms that the option is a sub-query of the intended structured query. When the user rejects an option, she indicates that

the option is not a sub-query of the intended structured query. In either case, she reduces the interpretation space of her keyword query.

Using sub-query relationships, we can connect all the complete and partial interpretations of a keyword query to form a Query Hierarchy. Fig. 3.2 shows a part of the query hierarchy for the keyword query K="Tom Hanks 2001".

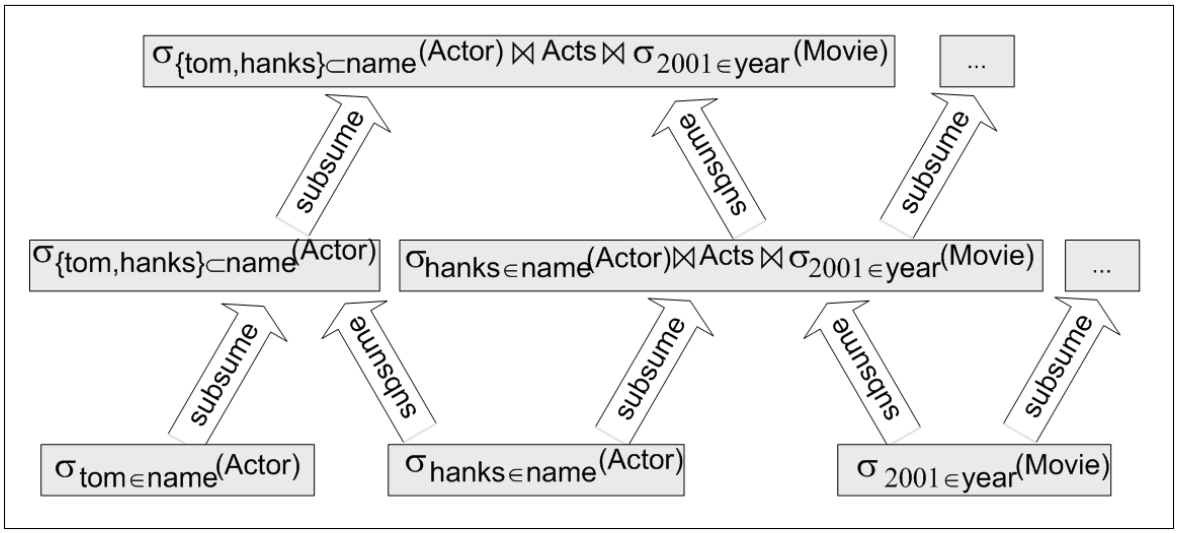


Figure 3.2 A Query Hierarchy Example. The figure presents a part of the query hierarchy for the keyword query "Tom Hanks 2001". The arrows represent subsumption relationships between the (partial) query interpretations. ©2012 IEEE.

The shape of a query hierarchy is like an upside-down trapezoid. The bottom of a query hierarchy is usually very small, as it contains only the smallest partial query interpretations. The top of a query hierarchy is much larger, as it is contains all complete interpretations. IQP utilizes the query hierarchy for an incremental generation of complete and partial query interpretations. It first uses smaller query templates to generate partial interpretations at the bottom of the hierarchy, and then gradually expand them to generate more complete interpretations. With a complex database schema or a long keyword query, IQP may have to deal with an interpretation space that is prohibitively large. As shown in Section 3.7, a query hierarchy enables IQP to generate appropriate query construction options without materializing the entire interpretation space, so as to ensure the scalability of the system.

3.5.4 Query Construction Plan

As shown in Fig. 3.1, in each query construction step of IQP, a user is presented with a list of query construction options, i.e. partial interpretations. She is supposed to select the option that correctly interprets her keywords. To simplify our presentation,

we assume that a user decides on only one option at a time. If the option is a sub-query of the intended query interpretation, the user accepts it. If the option is not a proper partial interpretation, the user rejects it. After the user accepts or rejects an option, the interpretation space of the keyword query can be reduced accordingly. The user keeps evaluating the options one after another, until only one possible query interpretation is left. Such a query construction process can be modeled as a binary decision tree, which is called *query construction plan*.

Definition 3.5.8. ***Query Construction Plan:*** *A Query Construction Plan (QCP) is a binary tree. Each node of the tree represents a set of structured queries, i.e. complete query interpretations. The left and right edges of a node represent the acceptance and the rejection of a query construction option, i.e. a partial interpretation, respectively. The tree satisfies: (1) the root represents the entire interpretation space of a keyword query; (2) each leaf node represents a single complete query interpretation; (3) given an edge* (Y, X) *(*Y *is* X*'s parent), (i) if* (Y, X) *is an acceptance of a query construction option* O*, then* X *is the subset of* Y *that subsumes* O*; (ii) if* (Y, X) *is a rejection of query construction option* O*, then* X *is the subset of* Y *that does not subsume* O*.*

A fragment of a query construction plan for the keyword query K="hanks 2001" is shown in Fig. 3.3. A query construction process is a traversal of the query construction plan. The user starts from the root of the plan. At each node, the user decides on the query construction option represented by the outgoing edges of that node. After accepting or rejecting this option, the user moves to the node pointed by the corresponding out-edge. The process continues until the user reaches a leaf of the tree, which is the structured query desired by the user.

In the user interface shown in Fig. 3.1, the user needs to decide on multiple query construction options in each round. That interface is based on an N-ary tree, which is illustrated in Fig. 3.4. Each edge in the N-ary tree represents a query construction option. In each step, the user is sup-posed to select the first option that satisfies her intent. It can be proven that the N-ary tree in Fig. 3.4 can be uniquely transformed from the binary tree in Fig. 3.3, and vice versa. To make the transformation, we traverse the binary tree in post-order. For each node NB in the binary tree, we remove the right edge (RE) and the right child (RC) of NB, and add RC's edges and children to NB. To facilitate our presentation, we always refer a query construction plan as a binary tree in the rest of this chapter.

For a single keyword query, there typically exist an arbitrary number of QCPs, whose efficiency differs significantly. Therefore, the key issue of query construction in IQP is to find an optimal QCP that enables the user to obtain the intended query interpretation as fast as possible. Each interaction between a user and a QCP always follows the same pattern, that is, the user receives a query construction option from QCP, evaluates it and chooses to accept or reject it. (In most of the cases, a user does not need to explicitly reject an option. Instead, the user proceeds with evaluating

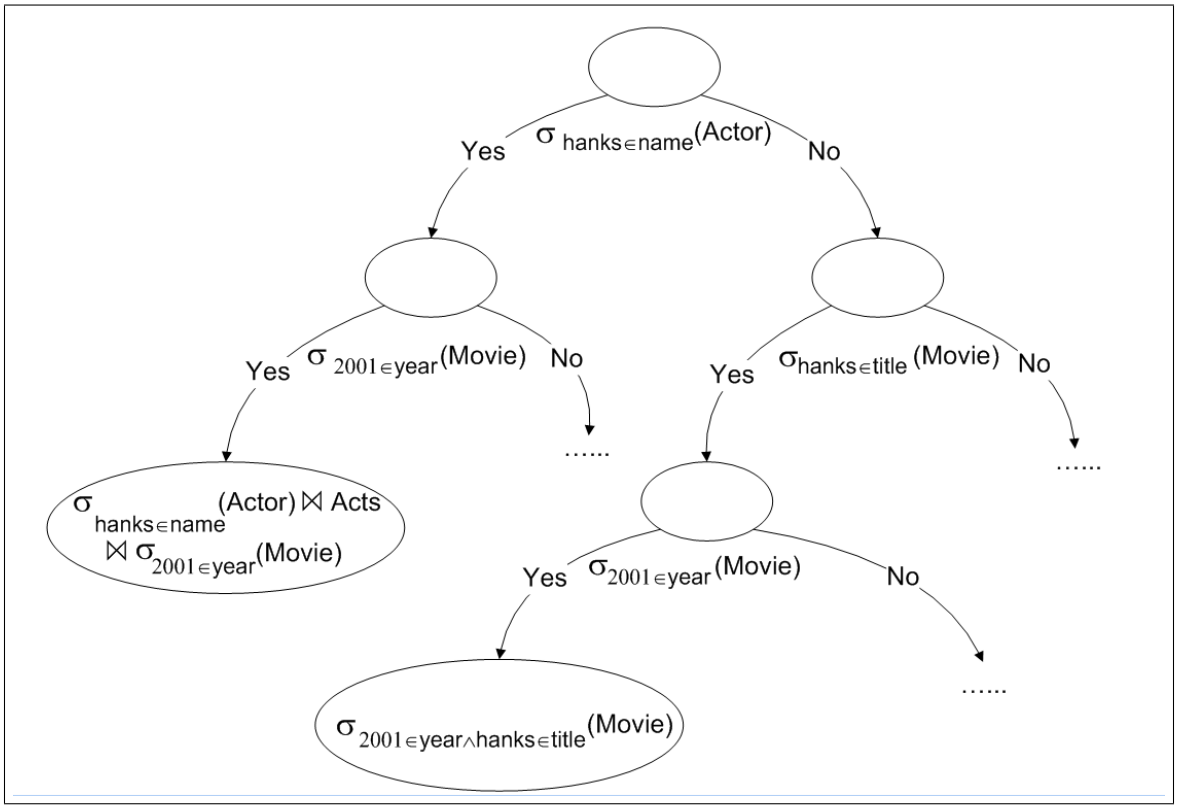


Figure 3.3 Query Construction Plan as a Binary Tree. ©2012 IEEE.

the next one). Therefore, we can measure the efficiency of a QCP by an average number of interactions between a user and the QCP until the user reaches a complete structured query at a leaf node. We call this measure interaction cost.

Definition 3.5.9. ***Interaction Cost of a Query Construction Plan:*** *We measure the interaction cost of a query construction plan by the expected number of query construction options (partial interpretations) a user has to evaluate to reach a structured query (complete query interpretation), i.e. a leaf of the binary tree. Given a structured query Q, the number of sub-queries a user needs to evaluate to construct Q is the depth of Q in the query construction plan. Therefore, the interaction cost of the plan can be calculated by:*

$$Cost(QCP) = \sum_{leaf \in QCP} depth(leaf) \times P(leaf), \tag{3.1}$$

where $depth(leaf)$ is the depth of the leaf in QCP, and $P(leaf)$ is the probability that the leaf is the user intended query interpretation.

Then, the problem of incremental query construction is reduced to the problem of finding the query construction plan with the minimum interaction cost.

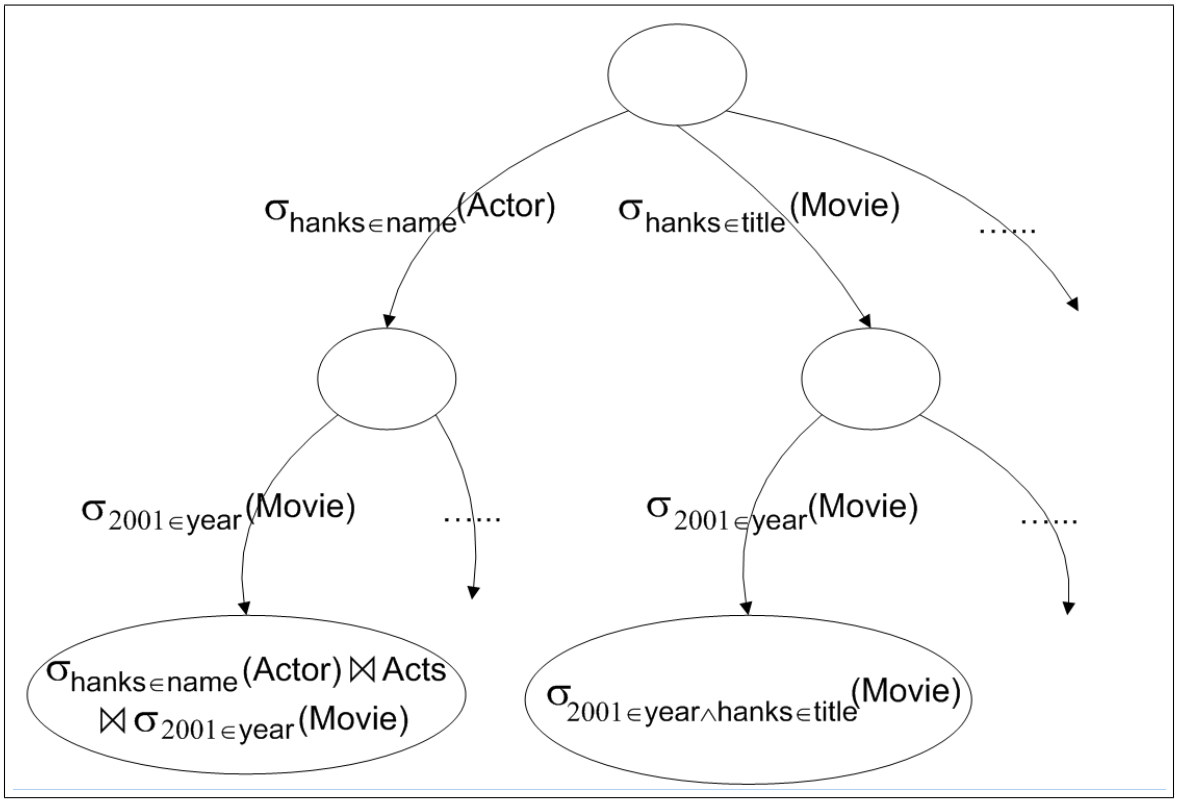


Figure 3.4 Query Construction Plan as an N-ary Tree. ©2012 IEEE.

Definition 3.5.10. ***Minimum Query Construction Plan:*** *A query construction plan QCP is a minimum query construction plan for the keyword query K, iff there is no query construction plan of K whose interaction cost is less than that of QCP.*

3.5.5 Query Construction vs. Ranking

In the ranking centric approaches [KKR+06, TL08, TCRS07, ZWX+07], the structural interpretations of a keyword query are ranked based on their probability of matching the user's intent. The user can then navigate through the ranked list to identify the desired interpretations. Such a ranked list of query interpretations is actually a special case of QCP in Def. 3.5.8. In this special QCP, a user is always presented with options corresponding to complete query interpretations. If the user accepts the option, she gets the desired complete query interpretation directly. If the user rejects the option, she traverses to the next node representing the next complete query interpretation. However, such a QCP is not necessary the optimal one. Through this QCP, the user can reach the top ranked interpretations quickly, but has to undertake a lot of interactions to find the less probable interpretations.

In general, a QCP for ranking is not balanced. It works for the case where the semantics of the keyword query is obvious, such that the majority of the probability

is assigned to a small number of query interpretations. It performs poorly in case a keyword query is ambiguous, such that the probability is more evenly distributed over a larger number of interpretations. In contrast, IQP always aims to create the optimal QCP with the minimum interaction cost. When the semantics of a keyword query is obvious, it generates a QCP similar to that of ranking. When the keyword query is ambiguous, it generates a more balanced QCP.

In Sections 3.6 and 3.7, we show how IQP generates query construction plans. We address two issues: (i) how to estimate the probability of a query interpretation, and (ii) how to generate minimum query construction plans based on the probability estimation.

3.6 Estimating Query Probability

To support efficient query construction, it is important to have an accurate assessment of the probability of whether a structured query (or a query construction option) correctly interprets a user's keyword query. In this section, we introduce a probabilistic model, which enables IQP to compute these probabilities. We also discuss possible statistics for supporting probability assessment.

3.6.1 A Probabilistic Query Interpretation Model

Given a keyword query, IQP is uncertain about the exact informational need represented by this query. We quantify this uncertainty using probability.

Definition 3.6.1. ***Probability of Query Interpretation:*** *Given a keyword query K, let the structured query Q be a complete interpretation of K, i.e., $Q : K$. Then, $P(Q|K)$ represents the conditional probability that, given K, Q is the user intended complete interpretation of K. Analogously, given a query construction option (a partial interpretation) O of K, $P(O|K)$ represents the conditional probability that, given K, O subsumes the user intended complete interpretation of K.*

$P(Q|K)$ corresponds to $P(leaf)$ in Equation 3.1. It is a crucial parameter used by IQP in creating query construction plans. If a keyword query K has been used repeatedly in a database, we can directly estimate $P(Q|K)$ using the previous interpretations of K in a database's query log. However, in a large database, it is unlikely to find sufficient number of records for a particular keyword query. To compute $P(Q|K)$, we need to resort to other statistics. In the following, we decompose $P(Q|K)$ to a set of atomic probabilities which are much easier to obtain.

A query interpretation Q is composed of a set of keyword interpretations $\{A_i : k_i\}$ and the query template T of Q. Thus, the probability $P(Q|K)$ can be transformed to:

$$P(Q|K) = P(\{A_i : k_i\}, T|K). \tag{3.2}$$

To decompose this probability, we make a number of assumptions in our probabilistic model.

Assumption 3.6.1. ***Keyword Independence:*** *First, we assume that the interpretation of each keyword in a keyword query is independent from the other keywords.*

The assumption of keyword independence is used to simplify the probability calculation, similarly to many other models in the literature (e.g. Vector Space Model in Information Retrieval and Naive Bayes in Machine Learning [MRS08]). Although the resulted probability estimation may not be very precise, our experiments in Section 3.8 show that this model provides an adequate prediction of the query relevance and significantly reduces the interaction cost. Based on Assumption 3.6.1 as well as Bayes' rule, we can transform Equation 3.2 to:

$$\begin{aligned} P(Q|K) &= \frac{P(\{A_i:k_i\}|T)\times P(T)\times P(K|\{A_i:k_i\},T)}{P(K)} \\ &= \frac{(\prod_{k_i\in K} P(A_i:k_i|T))\times P(T)\times P(K|\{A_i:k_i\},T)}{P(K)}. \end{aligned} \tag{3.3}$$

As the set of keywords K can be known from the set of keyword interpretations $\{A_i : k_i\}$, we have $P(K|\{A_i : k_i\}, T) = 1$. Therefore, we can transform Equation 3.3 to:

$$P(Q|K) = \frac{(\prod_{k_i\in K} P(A_i:k_i|T))\times P(T)}{P(K)}. \tag{3.4}$$

In this formula, $P(T)$ is the prior probability that the template T is used to form a query interpretation. $P(A_i : k_i|T)$ represents the probability that, given that T is used to form a query interpretation, A_i is used to form the query interpretation too. $P(K)$ is the prior probability that a user issues the keyword query K.

Assumption 3.6.2. ***Keyword Interpretation Independence:*** *Second, we assume that the probability of a keyword interpretation is independent from the part of the query interpretation the keyword is not interpreted to. In other words, $P(A_i : k_i|T) = P(A_i : k_i|T \cap A_i)$, where $P(A_i : k_i|T \cap A_i)$ represents the probability that, given that $T \cap A_i$ is a part of a query interpretation, A_i is also a part of the query interpretation.*

For instance, let $T = \sigma_{?\in name}(Actor) \bowtie Acts \bowtie \sigma_{?\in year}(Movie)$ be a query template searching for an actor who played in a movie from a given year. Let $A_1 : k_1 = \sigma_{hanks\in name}(Actor) : hanks$ be a keyword interpretation that interprets keyword "hanks" as a name of an actor. The probability of this keyword interpretation given the template T is computed as: $P(A_1 : k_1|T) = P(A_1 : k_1|T \cap A_1) = P(\sigma_{hanks\in name}(Actor) : hanks|\sigma_{?\in name}(Actor))$, which represents the probability

that, given that the attribute *Actor.name* is a part of the query interpretation, $\sigma_{hanks \in name}(Actor)$ is also a part of the query interpretation. We assume that the probability of $\sigma_{hanks \in name}(Actor) : hanks$ only depends on the fragment of $\sigma_{? \in name}(Actor)$ of the template.

Furthermore, as we consider different interpretations of the same keyword query K, we can skip computing $P(K)$, which is a constant for all query interpretations. Finally, we have:

$$P(Q|K) \propto (\prod_{k_i \in K} P(A_i : k_i | T \cap A_i)) \times P(T). \tag{3.5}$$

The probability of a partial interpretation O, i.e. $P(O|K)$, can be computed similarly as Formula 3.5. Suppose that O only contains the keyword interpretations of $K' \subset K$. Then,

$$P(O|K) \propto (\prod_{k_i \in K'} P(A_i : k_i | T \cap A_i)) \times P(T). \tag{3.6}$$

3.6.2 Probability Estimation

According to Formulas 3.5 and 3.6, the calculation of the probability of a query interpretation requires the estimation of $P(T)$, the prior probability of the query template T, as well as $P(A_i : k_i | T \cap A_i)$, the probability that, given that $T \cap A_i$ is a part of a query interpretation, A_i is also a part of the query interpretation.

Estimating Probability of a Template: If the database possesses a query log that is statistically representative, $P(T)$ can be estimated directly using the log. Based on the maximum likelihood model, $P(T)$ can be calculated as the frequency of T in the query log. Namely,

$$P(T) = \frac{\#occurences(T) + \alpha}{N}, \tag{3.7}$$

where $\#occurences(T)$ is the number of queries using T as a template, N is the total number of queries, and α is a smoothing parameter, which is typically set to 1. When the query log is absent or is not sufficient, we assume that all query templates are equally probable.

Estimating Probability of a Keyword Interpretation: In this work, we focus on two types of keyword interpretations defined in Def. 3.5.3. The first type interprets a keyword as part of a query template, such as a table name, an attribute name or an operator name. The second type maps a keyword to a "contains" predicate, interpreting the keyword as a value of an attribute. This interpretation has the form $\sigma_{k_i \in AT_i}(Table) : k_i$.

For the first type of interpretation, we can estimate the probability $P(A_i : k_i|T \cap A_i)$ using the query log too. If A_i is a table name, $P(A_i : k_i|T \cap A_i)$ is the frequency of using k_i to represent table A_i among the existing query interpretations containing table A_i. If A_i is an operator name, $P(A_i : k_i|T \cap A_i)$ is the frequency of using k_i to represent operator A_i among the existing query interpretations containing operator A_i. Without a query log, our system can use some empirical values set by domain experts.

In case a keyword is interpreted as an attribute value, we can hardly estimate the probability of the keyword interpretation using a query log. This is because the number of occurrences of each particular attribute value in a query log is usually insignificant. Therefore, we estimate probability of this interpretation using statistics obtained from the database instances. We model the formation of a query interpretation as a random process. For an attribute AT_i, this process randomly picks one of its instances a_j and randomly picks a keyword k_i from that instance to form the expression $\sigma_{k_i \in AT_i}(Table)$. Then, the probability of $P(\sigma_{k_i \in AT_i}(Table) : k_i|\sigma_{? \in AT_i}(Table))$ is the probability that $\sigma_{k_i \in AT_i}(Table)$ is formed through this random process. Based on maximum likelihood mode, this probability can be estimated using Attribute Term Frequency (ATF), which is defined as:

$$ATF(k_i, AT_i) = P(\sigma_{k_i \in AT_i}(Table) : k_i|\sigma_{? \in AT_i}(Table)) = TF(k_i, AT_i) + \alpha. \quad (3.8)$$

In Equation 3.8, $TF(k_i, AT_j)$ is the normalized frequency of keyword k_i in the attribute AT_i and α is a smoothing parameter, which is typically set to 1. The concept of TF closely corresponds to the Term Frequency used in Information Retrieval if we treat each attribute instance as a document [MRS08]. ATF is similar to the AF factor introduced in [MdSdM+07], which describes how typical the term is in the values of the respective attribute. While some other probabilistic models and statistics can also be used to estimate $P(Q|K)$, we show through experiments that our approach is highly effective.

3.7 Query Construction Algorithms

As mentioned in Section 3.5, given a keyword query, there typically exist multiple possible query construction plans (QCPs). While every plan can allow a user to obtain the intended structured query, these plans can differ in efficiency significantly. A less efficient plan also means reduced usability of IQP. To find an intended interpretation, a user needs to traverse a branch of the QCP (as a binary tree). This requires the QCP to be balanced. As mentioned earlier, a special case of an unbalanced QCP is a simple ranked list of all possible query interpretations [KKR+06, TL08, TCRS07, ZWX+07]. In case the intended interpretation does not appear within the top-ranked items, the user has to examine all queries prior to the intended one, which is tedious and error-

prone. Instead, a more balanced QCP tree can efficiently prune the search space and enable the user to obtain a desired query in fewer steps. In this section, we propose an algorithm to create a plan that imposes as little effort on the user as possible, i.e., a minimum query construction plan.

3.7.1 Brute-Force Algorithm

Before presenting our algorithm for creating the minimum query construction plan, we first introduce several properties of query construction plans.

Definition 3.7.1. ***Cost of Sub-plan:*** *a sub-plan is a sub-tree of a query construction plan. After a user has finished evaluating a set of query construction options, she reaches an internal node of the query construction plan. The sub-tree rooted at this node is called a sub-plan. Similar to the cost of QCP, the cost of a sub-plan measures the expected number of query construction options the user has to further evaluate until she finally reaches a complete query interpretation. Suppose the user has evaluated a set of options, in which she has rejected the set of options D and accepted the set of options A. The user reaches a sub-plan st. Then the cost of st is:*

$$Cost(st) = \sum_{leaf \in st} depth(leaf) \times P(leaf|\neg D \cap A), \tag{3.9}$$

where $depth(leaf)$ is the depth of the leaf in the query construction plan QCP (represented as a binary tree), and $P(leaf)$ is the probability that the leaf is the intended query interpretation.

Lemma 3.7.1. *Suppose QCP is a query construction plan, and QCP_1, QCP_2 are the two sub-plans rooted at the children of QCP's root. The option on the outgoing edges of QCP's root is R. Then the cost of QCP is:*

$$Cost(QCP) = P(R) \times Cost(QCP_1) + P(\neg R) \times Cost(QCP_2) + 1. \tag{3.10}$$

Applying Lemma 3.7.1, Algorithm 3.1 is able to identify the minimal query construction plan for a keyword query. The algorithm works recursively. For each node A, it enumerates all possible query construction options that can act as A's out-edges. For each option R, it computes the two minimal sub-plans of accepting R and rejecting R, and applies Lemma 3.7.1 to compute the cost of the corresponding sub-plan of A. Finally, the algorithm returns the sub-plan of A with the smallest cost.

While Algorithm 3.1 can always find the optimal query construction plan, it is expensive. With a big database schema and a long keyword query, it is infeasible to generate all possible structured queries and query construction options in the online query processing. Creation of an optimal query construction plan using the brute-force algorithm is even more impractical. It can be proven that the complexity of Algorithm 3.1 is $O(N^{log(M)})$, where N is the number of the query construction options

Algorithm 3.1: A Brute-Force Algorithm for MQCP

input :
CQ := all complete queries;
PQ := all options;
AP := empty; //accepted options
DP := empty; //rejected options

output:
QCP; //an optimal query construction plan

begin
QCP:= empty;
if $|CQ| = 1$ **then**
$QCP.root := CQ$;
$QCP.root.left\_edge := c$;
$QCP.root.right\_edge :=!c$;
$QCP.root.left\_child := c$;
$QCP.root.right\_child :=$ "$unknown$";
return QCP;
$partial\_query\ best\_r := null$;
$best\_cost := +\infty$;
tree $best\_t1 := null$;
tree $best\_t2 := null$;
for $R \in PQ$ **do**
$Accepted\_CQ = CQ.subset(R)$;
// the subset of CQ consuming R
$Denied\_CQ = CQ.subset(!R)$;
// the subset of CQ not consuming R
tree $QCP_1 := mini\_tree(Accepted\_CQ, PQ - R, AP + R, DP)$;
tree $QCP_2 := mini\_tree(Denied\_CQ, PQ - R, AP, DP + R)$;
$cost := P(R|AP \cap !DP) \times cost(QCP_1) + P(!R|AP \cap !DP) \times cost(QCP_2) + 1$;
if $cost < best\_cost$ **then**
$best\_cost := cost$;
$best\_r := R$;
$best\_t1 := QCP_1$;
$best\_t2 := QCP_2$;
$QCP.root := CQ$;
$QCP.root.left\_edge := best\_r$;
$QCP.root.right\_edge :=!best\_r$;
$QCP.root.left\_child := best\_t1$;
$QCP.root.right\_child := best\_t2$;
return QCP;

and M is the number of the complete query interpretations. As both M and N grow sharply with the size of the database schema and the length of the keyword query, this algorithm can be prohibitively costly even for a moderate database or keyword query size.

3.7.2 Greedy Algorithm

With a big database schema and a long keyword query, it is already infeasible to generate all possible structured queries and query construction options. Creation of an optimal query construction plan is even more impractical. Therefore, IQP uses a greedy algorithm to construct a near-optimal query construction plan in a top-down fashion. Instead of generating the entire plan, the algorithm generates query construction options one by one. Whenever an option is generated, it is presented to the user. After the user evaluates the option, (she either accepts it or rejects it,) it proceeds to generate the next option. The process is similar to that of the ID3 algorithm [Qui86] in creating decision trees.

As mentioned in Section 3.5.2, IQP generates query interpretations by expanding the query hierarchy in a bottom-up fashion. Instead of fully expanding the query hierarchy, the greedy algorithm stops when the size of the top level of the query hierarchy reaches a certain threshold. Then it searches for the best query construction option within the generated query hierarchy and presents the option to the user. If the user accepts the option, the algorithm keeps the part of the top level subsumed by this option and discards the rest. If the user rejects an option, the algorithm discards the part of the top level subsumed by this option. In either case, the algorithm can reduce the size of the top level of the current query hierarchy. The algorithm continues presenting query construction options to the user, until the size of the top level falls below a certain threshold. When the threshold is reached, the algorithm expands the top level of the query hierarchy again to get a new level of query interpretations. This process continues until the algorithm reaches the level of complete query interpretations and the user identifies the final intended structured query. We present the pseudo-code of the greedy algorithm in Algorithm 3.2.

As the query hierarchy is selectively expanded, we avoid generation of all possible query interpretations. In fact, the number of query interpretations generated by the algorithm is only proportional to the number of options the user needs to evaluate, i.e., the cost of the query construction plan.

Without fully expanding the query hierarchy, it is not possible to find the theoretically optimal query construction option. However, the partially expanded hierarchy is good enough to provide some near-optimal options. Our greedy algorithm tries to find a query construction option that can reveal as much information as possible about the intended structured query. We use Information Gain to measure the amount of information that can be revealed by a query construction option. We describe the computation of information gain in Section 3.7.3.

Algorithm 3.2: A Greedy Algorithm for MQCP

```
input :
        HQ := Initial Query Hierarchy;
        TQ := Top Level of HQ;
        T := Threshold;
output:
        Query c := Final Structured Query;
begin
    while true do
        if |TQ| < T then
            if HQ can be expanded then
                expand HQ;
            else if |TQ| = 1 then
                // let TQ=c
                return c;
        partial_query best_r := null;
        best_gain := +∞;
        for R ∈ HQ do
            if IG(TQ|R) < best_gain then
                best_gain := IG(TQ|R);
                best_r := R;
        present R to the user;
        if R is accepted then
            Sub(R) := all queries subsumed by R;
            TQ := TQ ∩ Sub(R);
        else if R is rejected then
            Sub(R) := all queries subsumed by R;
            TQ := TQ − Sub(R);
```

In the greedy algorithm, the query construction option that maximizes this information gain is selected and presented to the user. Our experiments (in Section 3.8.6 show that this greedy approach is efficient and is able to generate the near-optimal query construction plans.

3.7.3 Computation of Information Gain

Let $H(I)$ be an entropy of the entire interpretation space of the keyword query K. Let $H(I|O)$ be the conditional entropy of the interpretation space I, given that we know whether a query construction option O subsumes the user intended structured query. Then the information gain of the user's evaluation on O is:

$$IG(I|O) = H(I) - H(I|O). \tag{3.11}$$

The entropies $H(I)$ and $H(I|O)$ can be readily computed using the probabilities, such as $P(Q|K)$, obtained from our probabilistic model. As the greedy algorithm does not expand the query hierarchy completely, it does not have complete knowledge of the entire query interpretation space. As an alternative, we use the partial interpretations at the top level of the current query hierarchy to compute the information gain, as they are the best knowledge we have about the query interpretation space. Therefore,

$$H(I) = \sum_{Q \in Top} \frac{P(Q|K)}{\sum_{Q \in Top} P(Q|K)} \times lg \frac{P(Q|K)}{\sum_{Q \in Top} P(Q|K)}, \tag{3.12}$$

where Top represents all the query interpretations in the top level of the current query hierarchy.

$$H(I|O) = \sum_{Q \in Top \to O} \frac{P(Q|K)}{\sum_{Q \in Top \to O} P(Q|K)} \times lg \frac{P(Q|K)}{\sum_{Q \in Top \to O} P(Q|K)}, \tag{3.13}$$

where $Top \to O$ represents the query interpretations at the top of the query hierarchy that are subsumed by the option O.

3.8 Evaluation

To evaluate our query construction approach, we performed extensive experiments. First, we assessed the efficiency of IQP in helping users to construct structured queries on two real-world databases. Then, we compared the usability of incremental query construction with that of the ranking approach. Finally, we carried out simulations to study the scalability of IQP with respect to the size of the database and the length of keyword queries.

3.8.1 Datasets and Keyword Queries

In our experiments, we used two real-world datasets: a crawl of the Internet Movie Database (IMDB) and a crawl of a lyrics database from the web. The IMDB dataset contains seven tables, such as movies, actors and directors, with more than 10,000,000 records. The Lyrics dataset [LYMC06] contains five tables, such as artists, albums and songs, with around 400,000 records.

As these datasets do not have any associated query logs, we extracted the keyword queries from the query log of a major Web search engine [PCT06]. We pruned the queries based on their target URLs (www.imdb.com or any domain containing keyword "lyrics" in the URL), and obtained a few thousands of keyword queries targeted on the IMDB and lyrics domains. To assess the capability of IQP in creating complex structured queries, we further restricted the query set to the queries containing at least two attributes, such as movie-actor and artist-lyrics. This finally gave us 108 queries for IMDB and 76 queries for Lyrics. Each of these queries contains two-six terms, with an average length of four terms. For each of these queries we manually assessed its meaning and constructed the corresponding structured queries. We used IQP to generate templates for the both datasets using the automatic approach presented in Section 3.5.2. We set the maximal length of the join path to four, and obtained 74 templates for IMDB and 16 templates for Lyrics.

Finally, we manually identified a set of ambiguous keyword queries targeted at the IMDB dataset for which none of the ranking algorithms we tested produced acceptable results. We present these queries in Section 3.8.4.

We installed the datasets on a MySQL 5.0.22 database server, running on dual Xeon server with 4 GB RAM. The inverted index was constructed using Lucene [MHG09]. Our IQP system was implemented using JDK 1.5 and installed on a laptop with a 2.0 GHZ C2D and 2 GB RAM.

3.8.2 Effectiveness of the Probability Estimates

To assess how fast IQP can enable a user to construct a structured query, we measured the interaction cost of query construction, that is, the number of query construction options a user needs to evaluate to obtain the intended structured query. Our experiments were performed in an automatic way. We applied the greedy algorithm introduced in Section 3.7 to each of the keyword queries. The algorithm generates query construction options one by one. Based on the ground truth interpretations established a-priori, we let our system automatically accept the correct options and reject the incorrect options. The process of query construction stops when less than five complete query interpretations are left in the query window, in which the user is able to quickly identify the intended query. At the end of each construction process, we record the number of query construction options that have been evaluated.

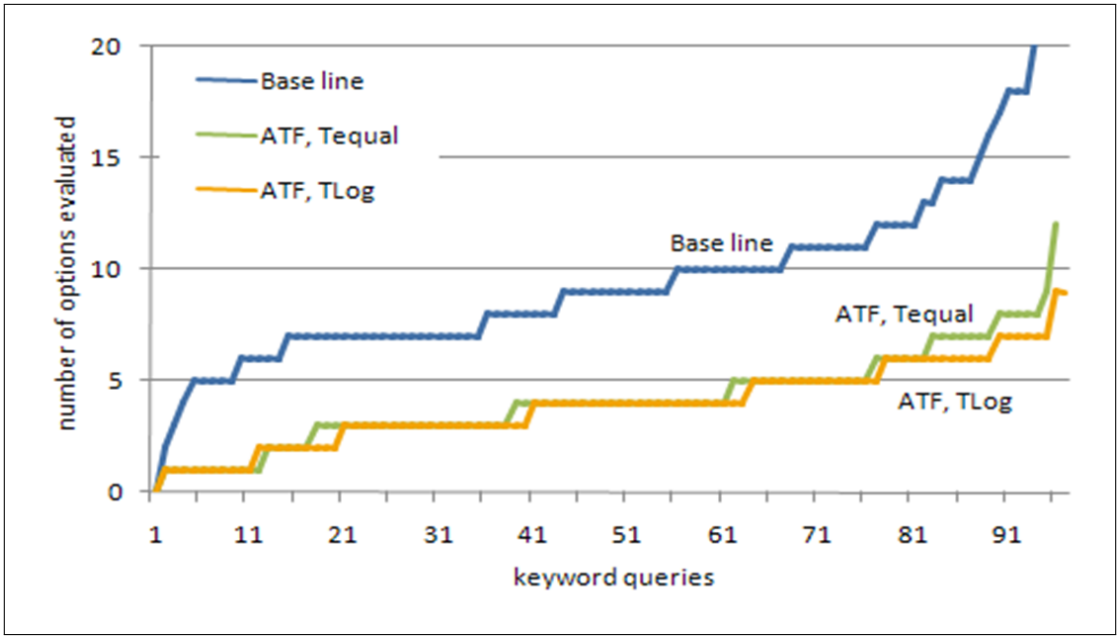
To estimate the effectiveness of the proposed query probabilistic model, we used

three variations of probability estimates. The first variation, also called base line, assumes that all structured queries and query construction options are equally likely. The second variation, referred as (ATF, Tequal), applied the probabilistic model introduced in Section 3.6 (represented by Formulas 3.5 and 3.6). It uses the Attribute Term Frequency (ATF) to estimate $P(A_i : k_i | T \cap A_i)$, but assumes equal probabilities of query templates. The third version, represented by (ATF, TLog), not only used ATF to estimate $P(A_i : k_i | T \cap A_i)$, but also used the query log to estimate the probabilities of the query templates.

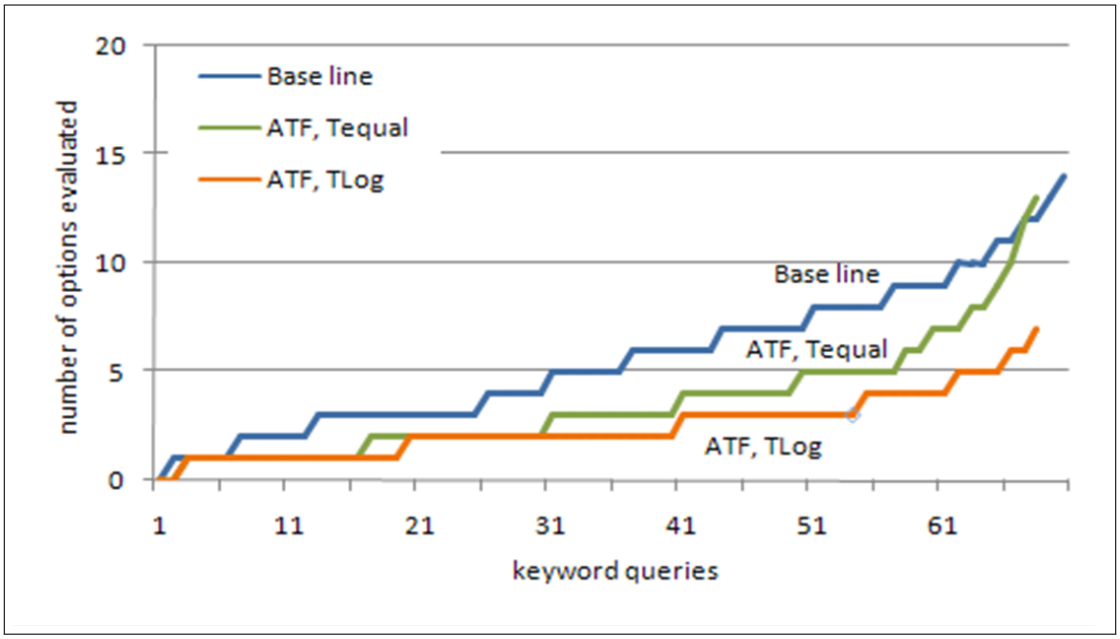
The experiment results for IMDB and Lyrics are shown in Fig. 3.5a and Fig. 3.5a respectively. In both figures, each data point on the X-axis represents a keyword query. Each Y-value presents an interaction cost, which is the number of query construction options a user needs to evaluate until she identifies the intended structured query. In this context "evaluate" means to decide whether an option correctly interprets the user's intent.

As shown in Fig. 3.5a, for the IMDB dataset, using the base line probability estimate, a user needs to evaluate one to 20 query construction options to construct a structured query. In more than 50% of the cases, the interaction cost is below ten. In more than 80% of the cases, the interaction cost is below 15. Occasionally, the interaction cost can reach 20. By estimating the probabilities of structured queries, the interaction cost can be significantly reduced. As shown by the lines of (ATF, Tequal) and (ATF, TLog), in more than 70% of the cases, a user needs to evaluate at most five options to create a structured query. In most of the cases, the interaction cost falls below ten. A similar trend can be seen in the results of the Lyrics data (Fig. 3.5b). Using the baseline probability estimation, the interaction cost ranges between zero and 15. By applying our probabilistic model, especially by using the probability estimation of (ATF, TLog), the interaction cost is reduced by around 50%.

Both Fig. 3.5a and Fig. 3.5b show that Attribute Term Frequency (ATF) is a highly effective statistics for estimating probabilities of query interpretations. It helped IQP reduce users' interaction costs significantly. The probability estimation of query templates based on usage is also useful. However, its effectiveness differed in the two data sets. In the Lyrics dataset, we observed a more significant improvement when considering the usage statistics of a template. This is because some query templates of Lyrics are used much more often than the others. For example, the most commonly used query template in Lyrics, which has the frequency of 0.85, is composed of five tables: $Song \bowtie AlbumSong \bowtie Album \bowtie ArtistAlbum \bowtie Artist$. In contrast, the usage of the IMDB query templates is more uniformly distributed. As a result, probabilities of query templates contribute less to the optimization of query construction process.



(a) IMDB



(b) Lyrics

Figure 3.5 Evaluation of the Probability Estimates: Number of Query Construction Options in IMDB and Lyrics Datasets. ©2012 IEEE.

3.8.3 Query Construction vs. Query Ranking

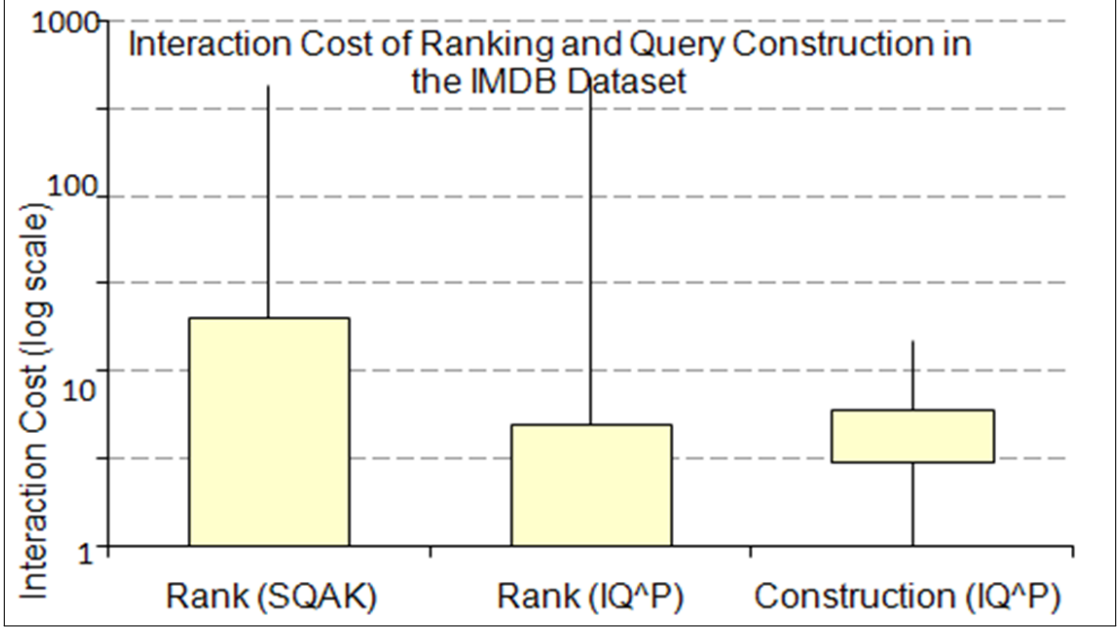
Our second set of experiments aimed to compare the interaction cost of query construction against that of query ranking. We used two query ranking functions: the ranking function of IQP and SQAK [TL08], one of the most recent query ranking functions in the related work. Query ranking approaches use different statistics from that of result ranking approaches like [HGP03], [LYMC06], as the ranking is conducted without materializing query results.

We measure the interaction cost of query ranking as the rank of the intended query interpretation in the ordered query list. This reflects the fact that the user has to evaluate each interpretation in the ranked list until she finds the intended interpretation. In case of IQP query construction we measure interaction cost as the number of options the user has to evaluate until she arrives at the intended query. In this context "evaluate" means to decide whether an option suggested by the system correctly interprets the user's intent. For the IQP ranking and incremental query construction, we considered all templates to be equally probable (ATF, Tequal), to reflect the situation in which a query log is not available.

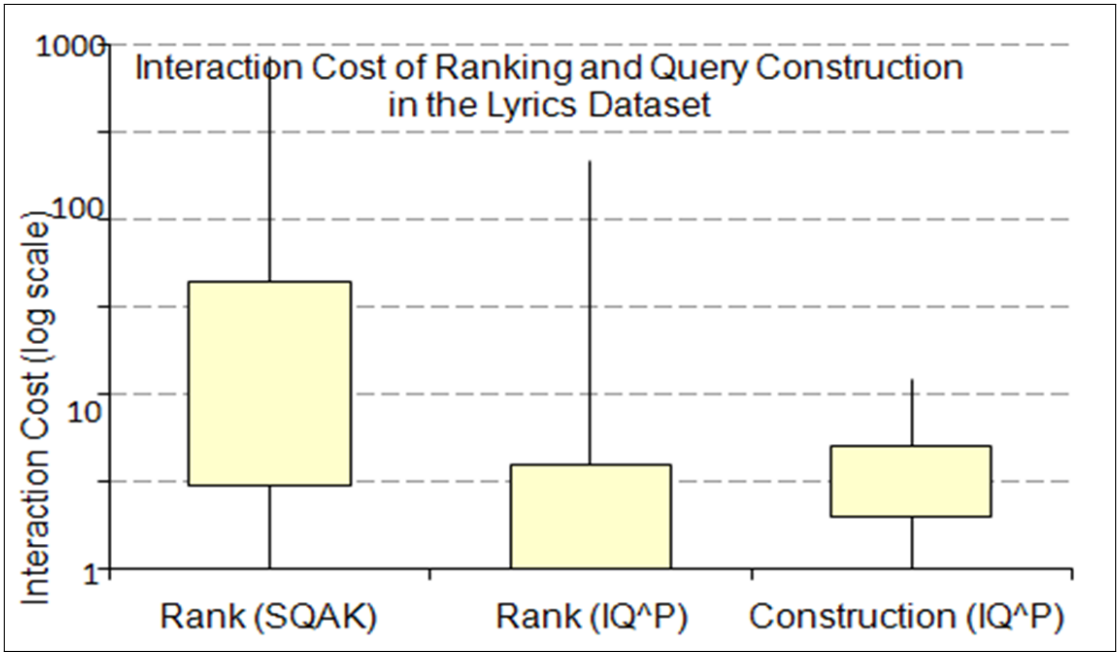
In SQAK, a query interpretation is regarded as a graph, whose score is the sum of the scores of its nodes and edges. Edges and nodes which do no contain keyword are assigned with unit scores. The score of a node containing a keyword is computed as TF-IDF of the keyword, which is then normalized using Lucene scoring function [MHG09]. The original SQAK function does not consider the case where multiple keywords are contained in a single node. We used the score of the Boolean AND query of Lucene to assess the score of the nodes containing more than one keyword.

Fig. 3.6 shows the boxplots [J. 83] of the interaction cost of ranking by SQAK and IQP as well as the interaction cost of IQP query construction. The Y-Axis of Fig. 3.6 (log scale) represents interaction cost, that is, the number of structured queries or query construction options to be evaluated by the user before the desired query can be identified. The box boundaries correspond to the upper and lower quartile of the data points, such that 50% of the data points lay inside of the boxes. "Rank (SQAK)" and "Rank (IQP)" represent the rank of the intended query using the corresponding ranking function. "Construction (IQP)" represents the number of options which needs to be evaluated to obtain the intended query using incremental query construction.

As Fig. 3.6 shows, our ranking function performs very well for the majority of the queries in the both datasets. The median value of Rank (IQP) is two for both datasets. In these cases construction is not necessary as the user can find the desired interpretation immediately within the top results. However, ranking function has a high variance, such that interpretations of ambiguous queries can receive ranks above 400. If the intended query interpretation does not receive a good rank, the user has to scan through the entire query list, which can contain 3,500 queries for our test set, until she identifies the intended structured query. The process is tedious and error-



(a) IMDB



(b) Lyrics

Figure 3.6 Interaction Cost of IQP and SQAK Ranking: Number of Query Construction Options in IMDB and Lyrics Datasets. ©2012 IEEE.

prone, as the user does not even know whether the intended query interpretation exists for the target database or if she occasionally missed the interpretation already.

In contrast, the interaction cost of incremental query construction has a much lower variance than the cost of ranking. As Fig. 3.6 shows, the cost of construction is around 3-4 options on average and 15 options in the worst case, which makes it highly helpful when user intended structured queries cannot be found within the top ranked results. Using the construction interface, a user can reach any structured query in the interpretation space in a reasonable number of steps. In addition, when using IQP, the user does not have to rely on the incremental query construction only. As IQP simultaneously presents a refined ranked list of queries in the query window, a user can short-cut the construction plan if the correct query is already presented in the top items (Fig 3.1 (3)). To estimate the scope in which construction outperforms ranking in real life, we performed a user study described in Section 3.8.4.

The experiments have shown that incremental construction is especially helpful when users' keyword queries are ambiguous. For example, in the IMDB query set, the intended interpretations of keyword queries containing more than one person name (e.g., an actor, a director or a character in a movie like "Melissa Gilbert Bruce Boxleitner") usually do not receive good ranks. Fig. 3.6 shows that IQP's query ranking outperforms SQAK's query ranking on our test set. The median interaction cost of IQP is two on both datasets, whereas the median cost of SQAK is six on IMDB and 13.5 on Lyrics.

It appears that the attribute term frequency (ATF) used by IQP is more effective for query ranking than the TF-IDF score used by SQAK. Intuitively, ATF prefers more typical interpretations, and TF-IDF prefers more distinctive interpretations. For our query set, typical interpretations are used more often. For example, "Garcia" is usually interpreted as an actor name, e.g. Andy Garcia. By using TF-IDF, it will be interpreted as movie title, as "Garcia" occurs less frequently in the movie title than in the actor name. Furthermore, for the Lyrics dataset, Steiner tree minimization used by SQAK does not provide good results for many queries. This is because the majority of Lyrics queries, such as "mariah carey emotions", requires a relatively long join $Artist \bowtie ArtistAlbum \bowtie Album \bowtie AlbumSong \bowtie Song$, whereas Steiner tree minimization prefers shorter joins such as $Artist \bowtie ArtistAlbum \bowtie Album$. Therefore, we used IQP ranking for our user study. In our future work we plan to perform a more detailed comparison between different ranking approaches.

3.8.4 Usability of Query Construction

To assess usability of IQP in real life, we conducted a user study. The user study aimed to compare usability of two user interfaces and to assess in which cases the IQP interface can enable more efficient data access than the query ranking interface. One interface is the IQP interface shown in Fig. 3.1. The other is a query ranking interface without using the query construction panel. For query ranking, we used

Table 3.1 Example Tasks for the User Study. C_1: rank of the intended query interpretation in the ordered list of possible query interpretations while using IQP ranking; C_2: an approximate number of options to be evaluated by the user in the query construction process; $|I|$: size of the interpretation space. ©2012 IEEE.

Task	C_1	C_2	$\|I\|$
Find a role of Brad Pitt in the movie directed by Steven Soderbergh in 2004.	18	6	3365
Find an actor who played in both movies: "Frida" and "Three Sisters".	48	6	268
Find a movie starring both Tom Hanks and Diego Luna.	73	7	1550
Find the role of an actor in the movie "Be Cool". The same actor played a character Sam Baily in another movie.	104	10	1470
Find a movie where Blake Blue is a director and Conners Chad is an actor.	213	7	1648

the IQP ranking function with the probability estimate (ATF, TEqual). For user convenience, the query ranking interface presented structured queries in several pages, each containing 20 queries.

Our users were 15 graduate students from the Computer Sciences Department. We selected 14 tasks for the users to perform. Each task required the user to retrieve certain information from the IMDB dataset. For each of the tasks, we proposed a keyword query. Based on the rank of the correct query interpretation, which ranged from 0 to 220, we grouped the tasks into seven complexity categories: 0, 1, 2, 3, 4, 6, and 11. Each category contained 2 tasks. Category k means that the correct query interpretation appears in the k^th page of the ranking interface. (0 means the correct interpretation is within the top-10.) Table 3.1 provides an overview over several example tasks.

We announced the user study as a time-based competition, such that our participants would solve the tasks as quickly as possible. The tasks were given to the participants in a random order. For the two tasks in each complexity category, each participant had to solve one using the IQP query construction interface and the other using the query ranking interface. Before the study, we provided the participants with a tutorial and some example tasks to practice, such that they could become familiar with the interfaces. To measure the time spent on each task, we recorded the interval between the time when a user clicked on the Search button and the time when the user executed (double clicked) the correct query interpretation. In case the participant did not manage to complete the task we set the time for this task to 10 minutes. Fig. 3.7 presents the median time used for different categories of tasks with both interfaces.

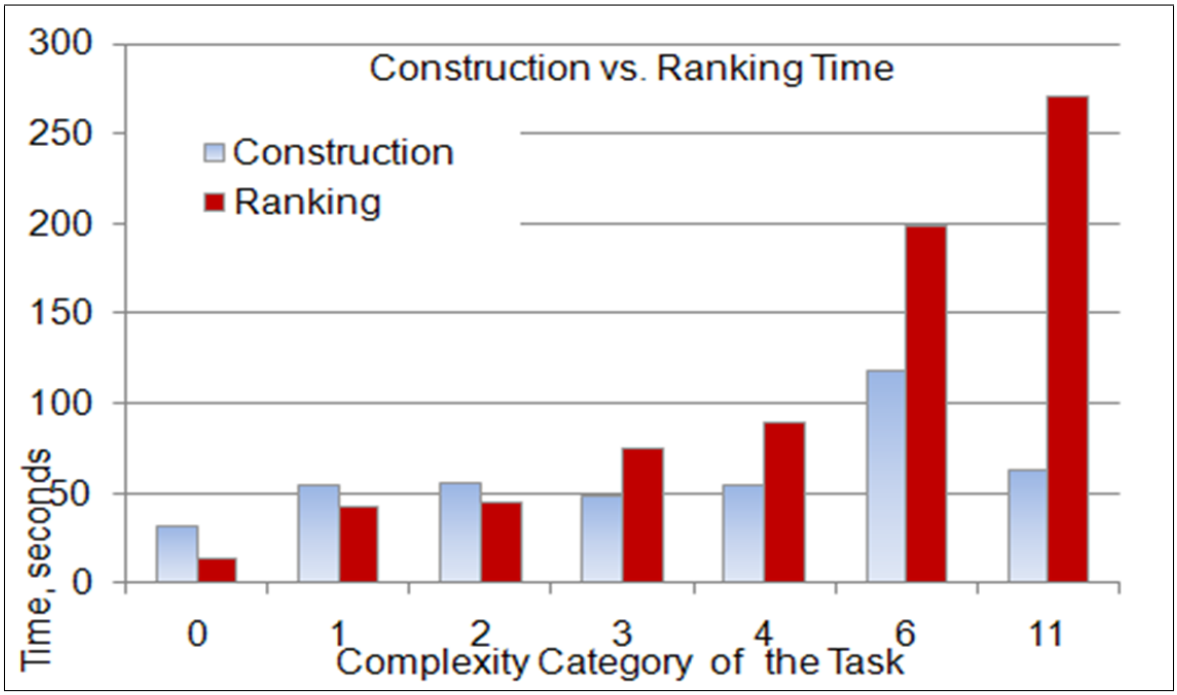


Figure 3.7 Usability of Query Construction: Median Time for the Construction Interface vs. Ranking. ©2012 IEEE.

Each data point on the X-axis of Fig. 3.7 represents complexity of the task. The Y-Axis of Fig. 3.7 represents the median time in seconds required for the user to complete a task. "Ranking" represents the results of the query ranking interface; "Construction" represents the results of the IQP construction interface. For example, for a task with complexity 11, the median time spent by a participant using construction was 63 seconds. In contrast, ranking interface in the same complexity category required 270 seconds, which is 4.3 times more.

As we can see from Fig. 3.7, the ranking interface outperforms the construction interface in the first three categories, where the ranks of the intended query interpretations are below 40. For the queries in Categories 3 and 4, where the ranks of the intended query interpretations range between 40 to 80, the IQP interface started to outperform the query ranking interface. For the queries in Category 6 and above, where the ranks of the intended query interpretations are above 120, the advantage of the IQP interface becomes very obvious.

We also asked the users to rate the percentage of queries for which they found the IQP interface to be more useful than the simple ranking interface on a 5-point Likert scale. Their answers were 70% of queries on average. Several participants pointed out, that they perceived time savings while using IQP as they proceed with tasks. We attribute this to increased familiarity with the interface.

3.8.5 Scalability

Given a keyword query, a keyword can occur in any textual attribute of a database, such that the number of possible query interpretations is polynomial in the size of the database schema and exponential in the number of keywords. To test the scalability of the query construction plan generation algorithm, we conducted a set of simulations.

In our simulations we generated a database schema as a completely connected graph of a given size, where each node represents a relational table. Based on the schema graph, we generated a set of query templates, where each template is a randomly picked connected sub-graph of the schema. Given a randomly generated keyword query, we assumed that each keyword occurs in a table with a certain probability (60% in our experiments). By combining the occurrences of the generated keywords and query templates, we obtained a number of possible structured queries. We assigned random probabilities to each table and keyword occurrence, and used the probabilistic model in Section 3.6 to estimate probabilities of structured queries and query construction options.

We believe that this simple simulation is representative, because (1) it simulates the basic structure of a general database schema; (2) it simulates trend that the number of structural interpretations of a keyword query grows polynomially with the size of the database schema and exponentially with the number of keywords.

Size of the Database Schema: To study the efficiency and scalability of the proposed greedy algorithm in generating query construction plans, we conducted two sets of experiments. In the first set of experiments, we fixed the number of terms in a keyword query to three, and varied the number of tables in the database from 5 to 80. Note that varying the number of columns gives a similar effect as varying the number of tables; therefore we report only one set of results in this chapter. In each experiment, we tuned the threshold of the greedy algorithm from 10 to 30, and simulated the construction of a randomly picked structured query. For each database size, we repeated the experiment 20 times, and recorded (1) the average number of possible structured queries that can be used to interpret a keyword query; (2) the average time for generation of a query construction option and (3) the average number of options a user needs to evaluate to obtain the intended query. The results of these experiments are shown in Table 3.2. It can be seen that for a constant number of keywords, the number of possible query interpretations grows very sharply (even polynomially) with the size of database. However, the number of options a user needed to evaluate to construct a query grows only in a similar scale as the database size.

The computation time for generating each query construction option increases with the size of the database as well, but at a low speed. This conforms to the complexity analysis performed in Section 3.7. We also observed that, with a higher threshold, the greedy algorithm can produce more efficient query construction plans. This is because a higher threshold allows the greedy algorithm to use a larger fraction

Table 3.2 Greedy Algorithm vs. Database Size. ©2012 IEEE.

# of tables	# of queries	Threshold					
		10		20		30	
		#steps	time/step	#steps	time/step	#steps	time/step
5	28	4	1ms	3	3 ms	3	3 ms
10	328	10	1 ms	8	20 ms	7	32 ms
20	1,953	13	2 ms	16	8 ms	13	30 ms
40	16,895	33	11 ms	26	12 ms	36	19 ms
80	104,962	65	43 ms	74	40 ms	70	43 ms

Table 3.3 Greedy Algorithm vs. the Number of Keywords. ©2012 IEEE.

# of keywords	# of structured queries	Threshold					
		10		20		30	
		#steps	time/step	#steps	time/step	#steps	time/step
2	48	5	1 ms	3	7 ms	4	6 ms
4	1,468	14	1 ms	12	19 ms	11	114 ms
6	30,463	19	3 ms	15	31 ms	14	252 ms
8	787,777	29	10 ms	25	33 ms	21	220 ms
10	47,859,840	40	25 ms	34	65 ms	36	239 ms

of the query hierarchy to estimate the goodness of query construction options. This improvement becomes less visible when the threshold increases to a certain value, such as 20 in our experiment. This indicates that the greedy algorithm only needs to evaluate a small fraction of the query hierarchy to achieve its optimal performance.

Size of the Keyword Query: In the second set of experiments, we fixed the database size to ten tables and varied the size of a keyword query from two to ten keywords. We repeated the experiments described above. The results are shown in Table 3.3. As expected, the number of query interpretations grows exponentially with the number of keywords. In contrast, the average number of query construction options a user needs to evaluate grows only linearly with the number of keywords. The computation time for generating a single query construction option also increases slowly. Similar to the results of the previous experiments, higher thresholds of the greedy algorithm can result in better query construction plans, and the improvement becomes insignificant when the threshold increases to 20.

3.8.6 Quality of the Greedy Algorithm

We performed experiments to compare the result quality of the brute-force algorithm and that of the greedy algorithm. As the brute-force algorithm is highly expensive, it is infeasible to test it using our simulation program. As an alternative, we created small sets of complete query interpretations and query construction options, and

Table 3.4 Result Quality of the two Algorithms. ©2012 IEEE.

# of structured queries	# of construction options	Brute force cost of the plan	Greedy cost of the plan
8	4	2.902265	2.915598
12	6	3.527421	3.547109
16	8	3.834929	3.891822
20	10	4.18971	4.210756
24	12	4.453365	4.469938

conducted the two algorithms directly on them. We varied the number of query interpretations from 8 to 24, and the number of query construction options from 4 to 12. In the simulation, each construction option subsumed a half of the query interpretations, and each query interpretation was assigned a random probability. When running the greedy algorithm, we set the threshold to 30, which is the total number of query interpretations. We repeated our experiments 20 times and recoded the average costs of the resulted query construction plans. As shown in Table 3.4, the quality of the query construction plans generated by the greedy algorithm is only slightly worse than those of the brute-force algorithm.

In summary, our simulation results show that the greedy algorithm for generating query construction plans is scalable with respect to the size of the database and the length of a keyword query. The plans it generates are near-optimal.

3.9 Discussion

Keyword search plays an increasingly important role in enhancing database usability. However, keyword queries are ambiguous and can retrieve imprecise or incomplete results. The most likely structural query interpretations as obtained by completely automatic keyword query disambiguation procedures (e.g. [KKR+06, TL08, TCRS07, ZWX+07]), can only satisfy the most simple and straightforward keyword queries, whereas user intended interpretations of ambiguous queries may not be found within the top ranked results. This observation motivated us to develop new approaches to enable users going beyond the most likely interpretations. To this end we extended previous work with an interactive probabilistic keyword query refinement approach.

In this chapter we analyzed the problem of probabilistic incremental query refinement and presented IQP - a novel system, which enables construction of structured queries from keywords. Given an ambiguous keyword query, IQP asks a minimal number of questions and enables the user to efficiently construct the desired structural query interpretation in an interactive way. We presented a conceptual framework for the incremental query construction as well as a probabilistic model, which enables consistent assessment of the probability of a query interpretation and query construc-

tion options. We presented two algorithms for generating optimal query construction plans, which enable users to obtain the intended structured query with a minimal number of interactions.

Our experimental results on two real-world datasets and a user study provided evidence for the usefulness of incremental query refinement of IQP when user intended structured queries cannot be found within the top ranked results. As a first step, we analyzed effectiveness of the proposed probability estimates for a set of real-world keyword queries. Our experiments have shown that the proposed estimates enabled us to reduce the interaction cost of query construction by around 50%. We have also shown that query ranking model of IQP outperforms that of the other recent approach to query ranking in related work called SQAK [TL08].

To assess usability of IQP in real life, we conducted a user study. The user study aimed to compare usability of two user interfaces, IQP and a simple query ranking interface, and to assess in which cases the IQP interface can enable more efficient data access than a query ranking interface. In our user study, for the queries where the ranks of the intended query interpretations ranged between 40 to 80, the IQP interface started to outperform the query ranking interface. For the queries where the ranks of the intended query interpretations were above 120, the advantage of the IQP interface became very obvious. Finally, our simulations confirmed the scalability of the proposed algorithms with respect to schema and keyword query size for the databases containing up to 100 tables. Our experiments have also shown that the quality of the query construction plans which can be generated by a brute-force algorithm was only slightly better than those of the efficient greedy algorithm we proposed.

4 Diversification of Search Results over Structured Data

4.1 Introduction

As we have seen in the previous chapter, interactive approaches can enable users to efficiently refine keyword query in the intended structural interpretation. In the present chapter we will investigate the problem of enabling user to obtain an overview of the available results, rather than constructing a particular interpretation, i.e. we will focus on solving Problem 2 from Chapter 1.

Diversification aims at minimizing the risk of user's dissatisfaction by balancing relevance and novelty of search results. Whereas diversification of search results on unstructured documents is a well-studied problem, diversification of search results over structured databases attracted much less attention. Keyword queries over structured data are notoriously ambiguous offering an interesting target for diversification. No single interpretation of a keyword query can satisfy all users, and multiple interpretations may yield overlapping results. The key challenge here is to give users a quick glance of the major plausible interpretations of a keyword query in the underlying database, to enable user to effectively select the intended interpretation.

For example, a user who issued a keyword query *"London"* may be interested either in the capital of the United Kingdom or a book written by Jack London, an American author. In contrast to document search, where data instances need to be retrieved and analyzed, rich database structures offer a more direct and intuitive way of diversification. For instance, if keyword *"London"* occurs in two database attributes, such as "location" and "name", each of these occurrences can be viewed as a keyword interpretation with different semantics offering complementary results. In addition, as in a database the query disambiguation can be performed before the actual execution, the computational overhead for retrieving and filtering redundant search results can be avoided. In the final step the database system executes only the top-ranked query interpretations to retrieve relevant and diverse results.

Applying diversification techniques for unstructured documents to keyword queries over structured databases, calls for two main adaptations: First, keyword queries need to be interpreted in terms of the underlying database, such that the most likely interpretations are ranked on top. Second, diversification should take advantage of the structure of the database to deliver more diverse and orthogonal representations of query results. In this chapter we present a novel approach to search result diversification in structured databases. We first present a probabilistic query disambiguation model to create semantic interpretations of a keyword query over a structured database. Then, we propose a diversification scheme for generating the top-k most relevant and diverse query interpretations.

Also evaluation measures need to be adapted. Conventional metrics for search result diversification such as α-nDCG and S-recall do not take into account graded relevance of subtopics in a search result, and thus are not directly applicable to structured data, where such assessment is important. We propose an adaptation of α-nDCG and S-recall to measure quality of diversification in database keyword search. These new metrics may be of independent interest. We performed a user study to assess the quality of the disambiguation model and the diversification scheme. Our evaluation results on two real world datasets demonstrates that search results obtained using the proposed algorithms are able to better characterize possible answers available in the database than the results obtained by the initial relevance ranking.

The algorithms presented in this chapter thus focus on addressing Problem 2 presented in Chapter 1, namely enabling users to obtain database search results with increasing level of novelty.

The rest of this chapter is organized as follows: In Section 4.2 we provide a summary of contributions contained in this chapter. Then, in Section 4.3 we discuss specific background. Following that, in Section 4.4 we present the diversification scheme. Then, in Section 4.5 we introduce an adaptation of α-nDCG and S-recall measures. Section 4.6 contains the results of our empirical investigation. Finally, in Section 4.7 we discuss the results.

4.2 Summary of DivQ Contributions

The challenges associated with the problem of search result diversification in database keyword search are twofold: First, diversification shall give users a quick glance of the major plausible interpretations of a keyword query in an underlying database. Second, as materialization of search results in databases is especially computationally expensive, diversification shall avoid materialization of potentially redundant search results. In contrast to the state-of-the-art diversification approaches for unstructured documents [AGHI09, CG98, CK06, CKC+08, GS09, WZ09] and database search results [CL07, VSS+08] that diversify materialized search results, DivQ proposed in this chapter performs diversification of query interpretations before any search results are

materialized. There are two advantages of our approach: Firstly, as query interpretations have clear semantics, they offer quality information for diversification. Secondly, our approach avoids the overhead of generation and filtering of redundant results and is therefore more efficient.

As results of keyword search over structured data differ from conventional documents, state-of-the-art evaluation metrics typically applied to document diversification such as α-NDCG [CKC+08] and S-recall [CK06] require adaptation. This is due to two main reasons: First, conventional metrics for search result diversification do not take into account graded relevance of subtopics, which is important in the context of structured data. Second, as different query interpretations may yield overlapping results, this overlap need to be taken into account by the evaluation measure. To this extent, we propose an adaptation of the state-of-the-art evaluation metrics taking into account subtopic relevance and result overlap.

Furthermore, we propose a similarity measure for query interpretations based on Jaccard coefficient to enable efficient diversification of search results over structured data. In addition, we further enhance the probabilistic query disambiguation model first presented in Chapter 3 to take into account dependencies between keywords in database attributes and further increase ranking effectiveness.

In summary, the DivQ contributions described in this chapter include:

- An efficient diversification approach for database keyword search performed at the query interpretation level without prior materialization of potentially redundant search results.
- An adaptation of the state-of-the-art evaluation measures for search result diversification such as α-NDCG [CKC+08] and S-recall [CK06] to take into account graded relevance of subtopics and result overlap.
- A similarity measure for query interpretations based on Jaccard coefficient to enable efficient diversification of search results over structured data.
- An enhancement of the probabilistic model proposed in Chapter 3 to take into account keyword co-occurrences and further increase ranking effectiveness.

Experiments on real-world data and a user study have demonstrated that DivQ achieves significant reduction of result redundancy, while preserving retrieval quality in the majority of the cases. This way search results obtained using the proposed algorithms also better characterize possible answers available in the database than the results obtained by the initial relevance ranking.

4.3 Specific Background

Recently, a lot of work on diversification of document search results and adaptation of the evaluation schemes in this context was performed [AGHI09, CG98, CK06, CKC+08, GS09, WZ09]. Several techniques (e.g. [CG98]) perform diversification of search results as a post-processing or re-ranking step of document retrieval. These techniques first retrieve the relevant search results and then filter or re-order the result list to achieve diversification. This approach cannot be directly applied to structured databases, where retrieval of all relevant data, which are potentially redundant, is especially computationally expensive. In contrast, DivQ performs diversification in the query disambiguation process before search results are retrieved. There are two advantages of our approach. Firstly, as the query interpretations generated within the disambiguation have a clear semantics, they offer quality information for diversification. Secondly, our approach avoids the overhead of generation and filtering of redundant results.

Another way to achieve diversification is clustering and classification of search results. Both techniques group search results based on similarity, so that users can navigate to the right groups to retrieve the desired results. Clustering and classification have been applied to document retrieval [AGHI09, MRS08], image retrieval [vLGOvZ09], recommender systems [ZMKL05], and database query results [CL07, LJ09]. On the one hand, similar to result re-ranking, clustering is usually performed as a post-processing step, and it may incur big performance overhead. Moreover, it lacks semantic interpretations, making results less understandable by end users [Hea06]. On the other hand, classification is usually pre-computed, and is not query aware. The query interpretations of DivQ can be regarded as a special kind of clusters or classes. In contrast to typical clusters and classes, query interpretations have well-defined semantics and are generated based on users' keyword queries. They are both query aware and easily understandable for end users. Most importantly, in contrast to existing work we considered the similarity between query interpretations as an important factor in diversification of search results. This enables us to further improve user satisfaction.

Chen et al. [CK06] employ pseudo-relevance feedback to achieve diversification of search results. In difference to DivQ they consider the intent of the query only tacitly. Wang et al. [WZ09] focus on the theoretical development of the portfolio theory of document ranking. They propose to select top-n documents and their order by balancing the overall relevance of the list against its risk (variance). We believe that portfolio technique can be adopted to compute diverse query interpretations in DivQ.

In contrast to document search, only few works focused on diversification of search results over structured data. In [CL07] the authors propose SQL result navigation through a set of categories, created taking into account user preferences. In [VSS+08], the authors introduce a pre-indexing approach to speed up the diversification of query

Table 4.1 Top-k Structured Interpretations for a Keyword Query "CONSIDERATION CHRISTOPHER GUEST"

Keyword query: CONSIDERATION CHRISTOPHER GUEST			
Relevance	**Top-3 Ranking**	**Relevance**	**Top-3 Diversification**
0.9	A director CHRISTOPHER GUEST of a movie CONSIDERATION	0.9	A director CHRISTOPHER GUEST of a movie CONSIDERATION
0.5	A director CHRISTOPHER GUEST	0.4	An actor CHRISTOPHER GUEST
0.8	An actor CHRISTOPHER GUEST in a movie CONSIDERATION	0.2	A plot of a movie containing CHRISTOPHER GUEST
...	...	...	...

results on relational databases. As diversification in both approaches is performed on the result level, these approaches are complementary to DivQ which conducts diversification on the interpretations of keyword queries.

Recent approaches to database keyword search [DZNona, KKR+06, TL08, TCRS07, ZWX+07] translate a keyword query into a ranked list of structured queries, also known as query interpretations, such that the user can select the one that represents her informational need. This disambiguation step is also a crucial step of DivQ. In this chapter, we build upon the probabilistic model presented in Chapter 3 for the keyword query disambiguation. However, the existing query disambiguation approaches consider only the likelihood of different query interpretations rather than their diversity. As a result, users with uncommon informational needs may not receive adequate results [CK06]. For example, if the majority of users who issued the keyword query *"London"* were interested in the guide of a city, the results referring to books written by Jack London may receive a low rank and even remain invisible to users. DivQ alleviates this problem by providing not only relevant but also diverse query interpretations.

4.4 The Diversification Scheme

DivQ translates a keyword query to a set of structured queries, also known as query interpretations. These interpretations can then be presented to the user, allowing selection of the intended interpretation. Given a keyword query, a database can offer a broad range of query interpretations with various semantics. In contrast to web search which focuses on few relevant results, to minimize the risk of user's dissatisfaction in this environment diversification needs to provide a better overview of the available results, even though they are less relevant.

Table 4.1 gives an example of the query interpretations for the keyword query *"CONSIDERATION CHRISTOPHER GUEST"*, once ranked only by relevance, and once

re-ranked by diversification. Both rankings enable the user to quickly understand the several possible interpretations of the query, so as to choose the intended one. However, ranking by estimated relevance bears the danger of redundant results. For example, the results of the partial interpretation "A director CHRISTOPHER GUEST" which is ranked second in the top-3 ranking clearly overlap with the results of the complete query interpretation ranked first. In contrast, the diversified ranking shows a set of possible complementary interpretations with increased novelty of results.

4.4.1 Bringing Keywords into Structure

As discussed in Chapter 3.1, in the context of a relational database, a structured query is an expression of relational algebra. To translate a keyword query K to a structured query Q, DivQ applies a disambiguation procedure similar to the one described earlier in Section 3.5. It first obtains a set of keyword interpretations $A_i : k_i$, which map each keyword k_i of K to an element A_i of an algebraic expression. DivQ then joins the keyword interpretations using a predefined query template T [CGRM06, HGP03], which is a structural pattern that is frequently used to query the database. We call the structured query resulting from the translation process described above a query interpretation.

For instance, "CONSIDERATION CHRISTOPHER GUEST" is first translated into a set of keyword interpretations, which are "director:CHRISTOPHER", "director:GUEST", and "movie: CONSIDERATION". Then, these keyword interpretations are connected to a template "A director X of a movie Y" to form a query interpretation "A director CHRISTOPHER GUEST of a movie CONSIDERATION". A query interpretation is complete if it contains interpretations for all keywords from the initial user query. Otherwise we talk about partial query interpretation. Given a keyword query K, the interpretation space of K is the entire set of possible interpretations of K. In this chapter, we focus on interpretations that retrieve non-empty results from the database.

4.4.2 Estimating Query Relevance

As presented earlier in Chapter 3.1, we estimate relevance of a query interpretation Q to the informational need of the user as the conditional probability $\mathrm{P}(Q|K)$ that, given keyword query K, Q is the user intended interpretation of K. A query interpretation Q is composed of a query template T and a set of keyword interpretations $I = \{A_j : [k_{j1}, k_{jn}] | A_j \in T,\ [k_j, k_{jn}] \in K,\ [k_{i1}, k_{im}] \cap [k_{j1}, k_{jn}] = \{\} \text{ for } i \neq j\ \}$. Thus, the probability $\mathrm{P}(Q|K)$ can be expressed as:

$$P(Q|K) = P(I, T|K). \tag{4.1}$$

The query disambiguation model in this chapter improves upon the initial model

presented in Chapter 3.1 by taking into account keyword dependencies in database attributes. To simplify the computation, we assume that (i) each keyword has one particular interpretation intended by the user; and (ii) the probability of a keyword interpretation is independent from the part of the query interpretation the keyword is not interpreted to. Based on these assumptions and Bayes' rule, we can transform Equation 4.1 to:

$$P(Q|K) \propto (\prod_{A \in T} P(A_j : [k_{j1}, k_{jn}]|A_j) \times \prod_{k \in K \cap k \notin Q} P_u \times P(T)), \tag{4.2}$$

where $P(T)$ is the prior probability that the template T is used to form a query interpretation. $P(A_j : [k_{j1}, k_{jn}]|A_j)$ represents the probability that, given that A_j is a part of a query interpretation, keyword interpretations $A_j : [k_{j1}, k_{jn}]$ are also a part of the query interpretation. In case a keyword $k_u \in K$ is not mapped to any keyword interpretation in Q, we introduce a smoothing factor P_u, which is the probability that the user's interpretation of keyword k_u does not match any available attribute in the database.

$P(A_j : [k_{j1}, k_{jn}]|A_j))$ can be estimated using attribute specific term frequency, i.e. the average number of occurrences of the keyword combination $[k_1, k_{jn}]$ in the attribute A_j. Note that, when $[k_1, k_{jn}]$ co-occur in an attribute A_j, the joint probability $P(A_j : [k_{j1}, k_{jn}]|A_j)$ will usually be larger than the product of the marginal probabilities $P(A_j : [k_{j1}]|A_j) \dots P(A_j : [k_{jn}]|A_j)$. Thus, query interpretations that bind more than one keyword to the same attribute, for example, a first name and a last name of a person to attribute "name", will get higher ranked than query interpretations that bind keywords to different attributes. P_u is a constant, whose value is smaller than the minimum probability of any existing keyword interpretation, such that the function assigns higher probabilities to complete query interpretations than to partial interpretations. $P(T)$ can be estimated as a frequency of the template's occurrence in the database query log. When the query log is not available, we assume all templates to be equally probable. As indicated in Section 4.4.1, query interpretations with an empty result are assigned zero probability. In this case the independence assumption (ii) used in Equation 4.2 is obviously violated, because the query interpretation maps keywords k_1 and k_2 to attributes A_1 and A_2, such that the marginal probabilities $P(A_1 : [k_1]|A_1)$ and $P(A_2 : [k_2]|A_2)$ are larger than zero, but, given the instances of the database, the joint probability $P(A_1 : [k_1], A_2 : [k_2]|A_1, A_2)$ is zero.

4.4.3 Estimating Query Similarity

As our objective is to obtain diverse query results, we want the resulting query interpretations to be not only relevant but also as dissimilar to each other as possible. Let Q_1 and Q_2 be two query interpretations of a keyword query K. Let I_1 and I_2 be the sets of keyword interpretations contained by Q_1 and Q_2 respectively. To assess similarity between the two query interpretations, we compute the Jaccard coefficient

between I_1 and I_2.

Definition 4.4.1. ***Query Similarity:*** *We define similarity between two query interpretations Q_i, and Q_j as the Jaccard coefficient between the sets of keyword interpretations they contain, that is,*

$$Sim(Q_1, Q_2) = \frac{|I_1 \cap I_2|}{|I_1 \cup I_2|}. \tag{4.3}$$

The resulting similarity value should always fall in [0, 1], where 1 stands for the highest possible similarity.

4.4.4 Combining Relevance and Similarity

To generate the top-k query interpretations that are both relevant and diverse, we employ a greedy procedure. We always select the most relevant interpretation as the first interpretation presented to the user. Then, each of the following interpretations is selected based on both its relevance and novelty. Namely, given a query interpretation Q and a set of query interpretations QI that are already presented to the user, we estimate the score of Q as its relevance score discounted by the average similarity between Q and all the interpretations in QI:

$$ScoreQ = \lambda \times P(Q|K) - (1 - \lambda) \times \sum_{q \in QI} \frac{Sim(Q, k)}{|QI|}. \tag{4.4}$$

Relevance and similarity factors are normalized to equal means before λ-weighting is applied. The interpretation with the highest score is selected as the next interpretation to be presented to the user. In Equation 4.4, λ is a parameter to trade-off query interpretation relevance against novelty. For example, with $\lambda = 1$ the score of the query interpretation takes only relevance into account; $\lambda = 0.5$ corresponds to a balance between relevance and novelty, whereas $\lambda < 0.5$ emphasizes novelty of the interpretation.

4.4.5 The Diversification Algorithm

To create a set R of the most relevant and diverse query interpretations in an efficient way we first materialize the top-k most probable query interpretations of a keyword query and sort the interpretations according to the relevance scores. Then we go through the query interpretations and output the most relevant and diverse interpretations one by one. The pseudo-code of the algorithm is presented in Algorithm 4.1. Let L be the list of top-k query interpretations sorted by probability of their relevance to the user's informational need. The process starts with the most relevant

query interpretation at the top of L. To compute the i^{th} relevant and diverse element, i.e. $R[i]$, we scan the remaining candidate elements in L, compare their scores in Formula 4.4, and add the element with the highest score to R. As the diversity value of each item is always larger than 0, it is not necessary to scan the entire L to obtain each $R[i]$. The scan stops when we are sure that the rest of L cannot possibly outperform the current optimal item, which is evaluated by $best\_score > \lambda P(L[j])$. The algorithm terminates after r elements are selected.

Algorithm 4.1: Proc Select Diverse Query Interpretations

input :
list $L[l]$ of top-k query interpretations ranked by relevance.
output:
list $R[r]$ of the relevant and diverse query interpretations.

```
begin
    R[0] = L[0];
    i = 1;
    // select the best candidate for R[i]
    while i < r do
        //less than r elements are selected
        j = i;
        best_score = 0;
        while L[j] ≠ null do
            //more candidates for R[i] in L
            if best_score > λ × P(L[j]) then
                break;
            //check score upper bound
            if score(L[j]) > best_score then
                best_score = score(L[j]);
                c = j;
            j++;
        R[i] = L[c]; //add the best candidate to R
        Swap L[i...c−1] and L[c];
        i++;
```

The worst case complexity of the Algorithm 4.1 is $O(l \times r)$, where l is the number of query interpretations in L and r is the number of query interpretations in the result list R. The maximal total number of similarity computations is $\frac{l^2-l}{2}$.

4.5 Evaluation Metrics

α-NDCG [CKC+08] and S-recall [CK06] are established evaluation metrics for document retrieval in presence of diversity and subtopics. As results of keyword search over structured data differ from conventional documents, these metrics require some adaptation.

A search result of DivQ is a ranked list of query interpretations. Therefore, a "document" in traditional IR corresponds to the union of tuples returned for one particular query interpretation in DivQ. Each tuple can be represented by its primary key in the database. Thus, a primary key corresponds to the notion of information nugget in α-NDCG and to subtopic in S-recall. However, the correspondence is loose: whereas α-NDCG and S-recall assume equal relevance of information nuggets and subtopics contained in a document, relevance of primary keys in a query result may vary a lot. Thus it is important to take into account their relevance for estimating gain and recall explicitly. In the following, we adapt α-NDCG and S-recall to this end.

4.5.1 Adapting Gain for alpha-NDCG-W

nDCG (normalized Discounted Cumulative Gain) has established itself as the standard evaluation measure when graded relevance values are available [CKC+08, JK02]. The first step in the nDCG computation is creation of a gain vector G. The gain $G[k]$ at rank k can be computed as the relevance of the result at this rank to the user's keyword query. The gain may be discounted with increasing rank, to penalize documents lower in the ranking, reflecting the additional user effort required to reach them. The discounted gain is accumulated over k to obtain the DCG (Discounted Cumulative Gain) value and normalized using the ideal gain at rank k to finally obtain the nDCG value.

To balance relevance of search results with their diversity, the authors of [CKC+08] proposed α-nDCG, where the computation of the gain $G[k]$ is extended with a parameter α, representing a tradeoff between relevance and novelty of a search result. To assess novelty of a document in the search result, α-nDCG views a document as the set of information nuggets. If a document at rank i contains an information nugget n, α-NDCG counts how many documents containing n were seen before and discounts the gain of this document accordingly. α has a value in the interval [0, 1]; α=0 means that α-NDCG is equivalent to the standard nDCG measure. With increasing α, novelty is rewarded with more credit. When α is close to 1, repeated results are regarded as completely redundant such that they do not offer any gain. In [WZ09], the authors fix α as 0.5 for a balance between relevance and novelty.

In the context of database keyword search, where an information nugget corresponds to a primary key, the relevance of nuggets with respect to the user query can vary a lot. To reflect the graded relevance assessment on the nuggets in the evalua-

tion metrics, α-NDCG-W measures the gain $G[k]$ of a search result at rank k as the relevance of the query interpretation at rank k, i.e., Q_k. We penalize the gain of an interpretation retrieving overlapping results using the following formula:

$$G[k] = relevance(Q_k) \times (1-\alpha)^r, \tag{4.5}$$

where r is the factor, which expresses overlap in the results of the query interpretation Q_k with results of the query interpretations at ranks $1 \dots k-1$.

To compute r, for each primary key pk_i in the result of Q_k we count how many query interpretations with pk_i were seen before (i.e. at ranks $1 \dots k-1$), and aggregate the counts:

$$r = \sum_{pk_i \in Q_k} \sum_{j \in [1,k-1]} |pk_i \in Q_j|. \tag{4.6}$$

Note that we consider primary keys in the result of one interpretation to be distinct (each primary key counts only once). As in document retrieval the presence of a particular information nugget in a document is uncertain, the gain computation in [CKC+08] focuses on the number of nuggets contained in a document and does not take into account graded relevance of information nuggets. In contrast, in the context of database keyword search an information nugget in α-nDCG-W corresponds to a primary key in the result of a query interpretation, such that the presence of an information nugget in the result is certain. At the same time, relevance of retrieved primary keys with respect to the user query can vary a lot. This graded relevance is captured by Equation 4.5. Agrawal et al. [AGHI09] suggest an alternative approach called NDCG-IA (for Intent Aware NDCG) to take into account graded relevance of information nuggets to queries. However, a drawback of NDCG-IA is that it may not lie between [0, 1]. Moreover, NDCG-IA does not take into account result overlap. In contrast, the value of relevance-aware α-nDCG takes into account result overlap and is always in the interval [0, 1], where 1 corresponds to ranking according to the user assessment of query interpretation relevance averaged over users.

4.5.2 Weighted S-Recall

Instance recall at rank k (S-recall) is an established recall measure which is applied when search results are related to several subtopics. S-recall is the number of unique subtopics covered by the first k results, divided by the total number of subtopics [CK06, WZ09].

In database keyword search, a single primary key in the search result corresponds to a subtopic in S-recall. However, other than in document retrieval, where all subtopics can be considered equally important, relevance of retrieved primary keys (tuples) can vary a lot with respect to the user query. To take the graded relevance of subtopics into account, we developed a WS-recall measure (weighted S-recall).

WS-Recall is computed as the aggregated relevance of the subtopics covered by the top-k results (in our case query interpretations) divided by the maximum possible aggregated relevance when all relevant subtopics are covered:

$$WS - recall@k = \frac{\sum_{pk \epsilon Q_{1...k}} relevance(pk)}{\sum_{pk \epsilon U} relevance(pk)}, \tag{4.7}$$

where U is the set of relevant subtopics (primary keys). In case only binary relevance assessments are available, WS-recall corresponds to S-recall. We average WS-recall at k and α-NDCG at k over a number of topics to get their means over the query set.

4.6 Experiments

To assess the quality of the disambiguation and diversification schemes we performed a user study and a set of experiments.

4.6.1 Dataset and Queries

In our experiments, we used two real-world datasets: a crawl of the Internet Movie Database (IMDB) [IMD] and a crawl of a lyrics database from the Web [LYMC06]. The IMDB dataset contains seven tables, such as movies, actors and directors, with more than 10,000,000 records. The Lyrics dataset contains five tables, such as artists, albums and songs, with around 400,000 records. As these datasets do not have any associated query log, we extracted the keyword queries from the query logs of MSN and AOL [PCT06] Web search engines. We pruned the queries based on their target URLs, and obtained thousands of queries for the IMDB and lyrics domains.

To obtain the most popular keyword queries, we first sorted the queries based on frequency of their usage in the log. For each domain, we selected 200 most frequent queries for which multiple interpretations with non-empty results exist in the database. These queries were mostly either single keyword or single concept queries, often referring to actor/artist names or movie/song titles. We refer to this part of the query set as *single-concept queries (sc)*. To obtain an additional set of more complex queries, we manually selected about 100 queries for each dataset from the query log, where we explicitly looked for queries containing more than one concept, e.g. a movie/song title and an actor/artist name. We refer to this set as *multi-concept queries (mc)*.

As diversification of results is potentially useful for ambiguous queries [CSA+09], we estimated ambiguity of the resulting keyword queries using an entropy-based measure. To this end, for each keyword query, we ranked interpretations of this query

available in the database using Equation 4.2 and computed the entropy in the top-10 ranks of the resulting list. Intuitively, given a keyword query, high entropy over the top ranked interpretations indicates potential ambiguity. Finally, we selected 25 single-concept and 25 multi-concept queries with the highest entropy for each dataset.

4.6.2 User Study

To assess relevance of possible query interpretations we performed a user study. We selected a mix of single-concept and multi-concept queries as described in Section 4.6.1, for which we generated all possible interpretations sorted by their probability. As the set of possible interpretations grows exponentially with the number of concepts involved, we took at most the top-25 interpretations. This should not rule out meaningful interpretations, as probabilities fall very quickly with their rank: Figure 4.1 gives the maximum and the average ratio of the probability of a query at rank i and the aggregated probabilities of queries at rank $j < i$: $PR_i = \frac{P(Q_i|K)}{\sum_{j<i} P(Q_j|K)}$.

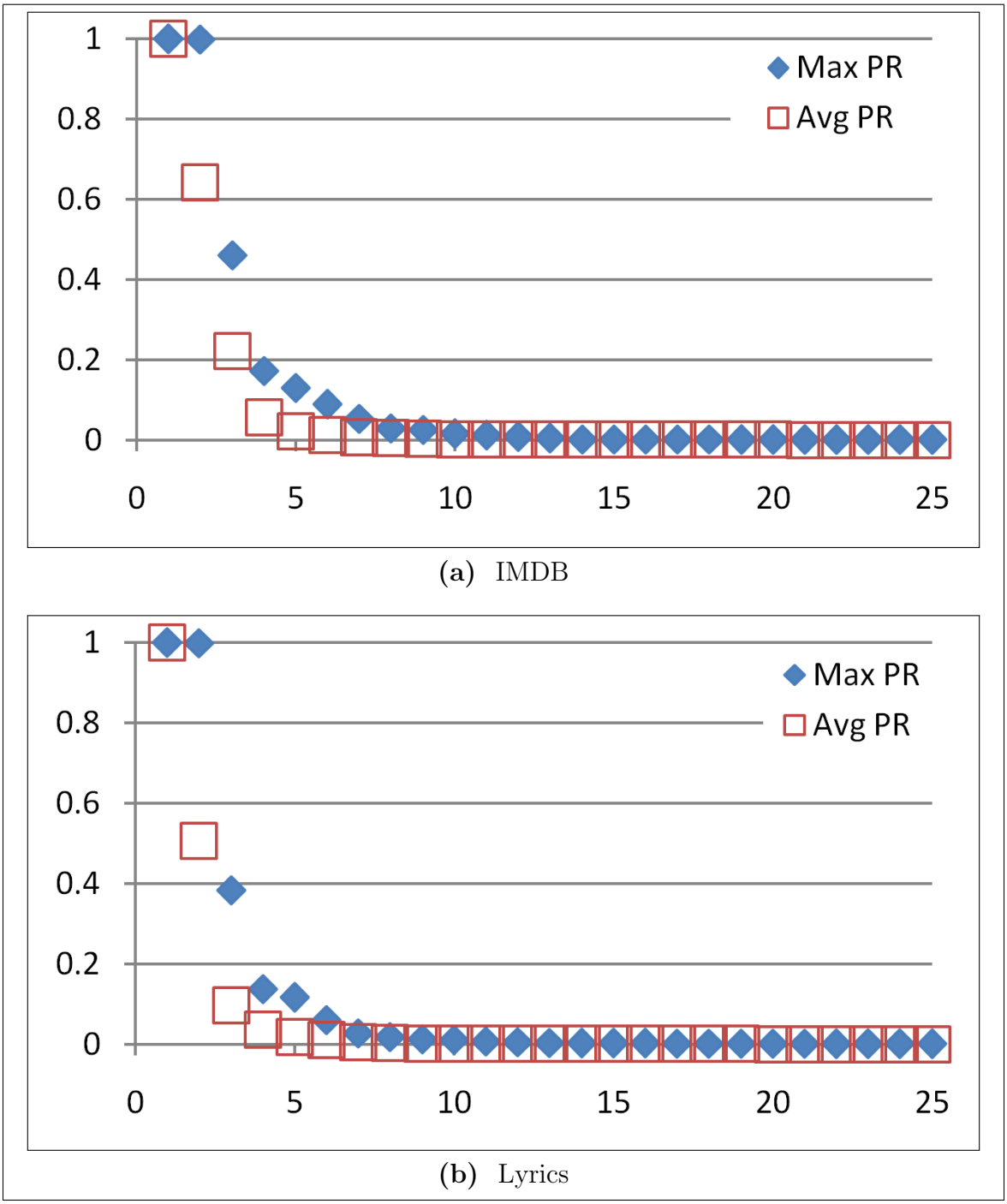
Each data point on the X-axis of Figures 4.1a and 4.1b presents the rank. The Y-axis presents the corresponding average and maximum PR_j value. As can be seen, queries at rank 10 already are only 0.01 as likely as queries at rank < 10, and queries at rank 25 are at most $2.95 \times E^{-04}$ as likely as queries at rank < 25.

For each query we pruned all query interpretations Q_i whose probability constituted less than 0.1% of the aggregated probability of all possible interpretations. Additionally, for each query we included at most five more interpretations with probability below this threshold and randomized the order in which the interpretations were presented for user assessment, in order to avoid a bias towards top ranked queries.

In total, each user had to evaluate 630 interpretations for IMDB and 517 interpretations for the Lyrics dataset. For each interpretation of a given keyword query, the participants were asked to indicate on a two-point Likert scale, if they think that this interpretation could reflect an informational need implied by the keyword query. Multiple interpretations of one query were possible and explicitly encouraged. In total, we had 16 participants, from whom 10 completed all evaluation tasks in both datasets and the rest completed 30% of tasks in IMDB and 9% of tasks in lyrics dataset on average. We computed agreement between the participants using kappa statistics [MRS08]. We observed average kappa values of 0.33 in IMDB and 0.28 in Lyrics. We consider this low agreement as an additional indication of ambiguity of the selected queries. Finally, we computed the relevance scores of each query interpretation by averaging scores over the participants.

4.6.3 alpha-nDCG-W

Given a keyword query, DivQ first creates a ranked list of query interpretations and then applies the diversification algorithm to this list to obtain the most relevant and



(a) IMDB

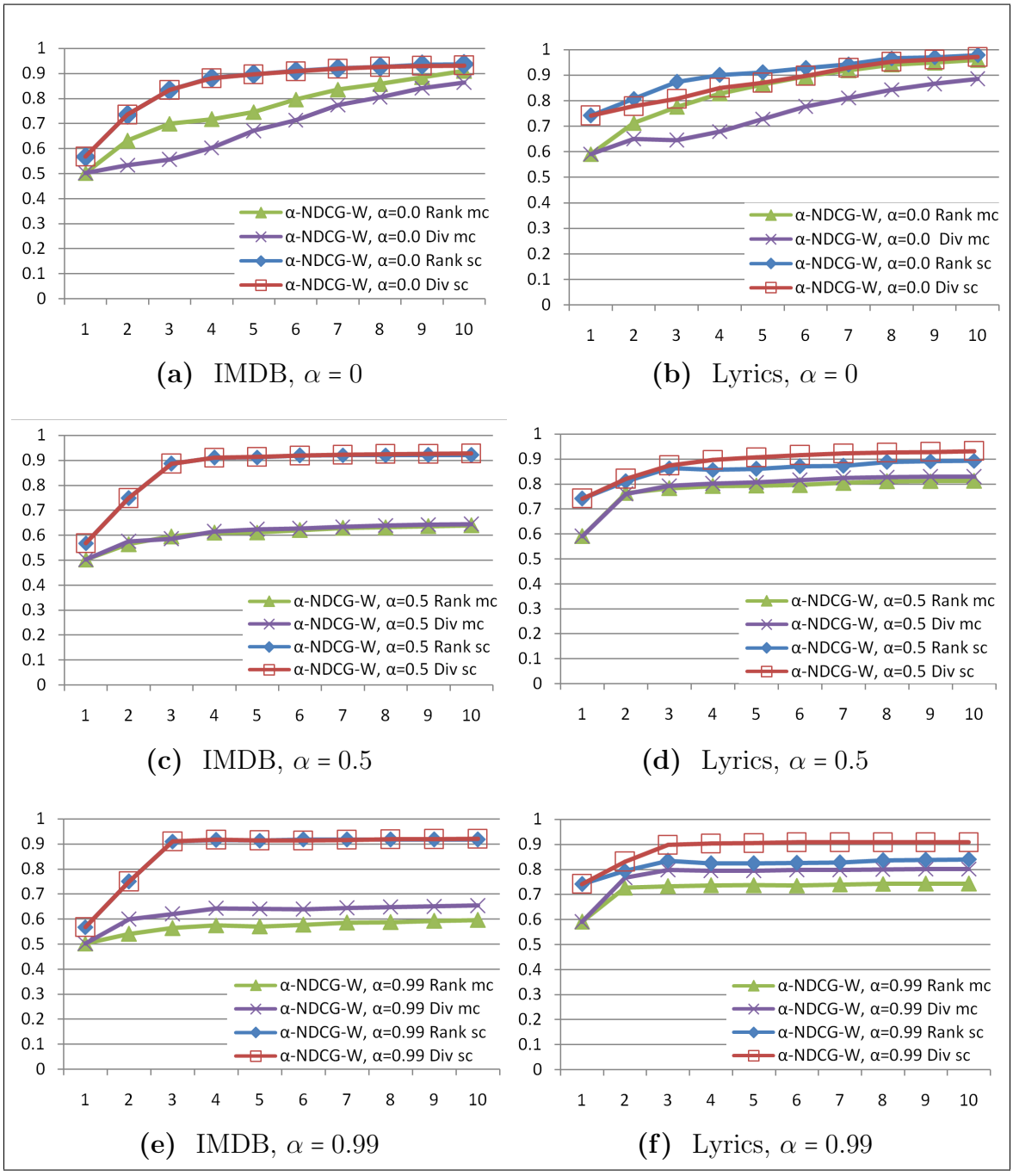
(b) Lyrics

Figure 4.1 Selecting Meaningful Query Interpretations: Maximum (Max PR) and Average (Avg PR) Probability Ratio.

novel results. To assess quality of query ranking and diversification, we measured α-NDCG-W by varying α parameter from 0, to 0.5 and to 0.99. In the case of α=0, novelty of results is completely ignored, and α-NDCG-W corresponds to the standard NDCG. With α=0.5, novelty is given a certain credit. With α=0.99, novelty becomes crucial, and results without novelty are regarded as completely redundant. As the optimal ranking for normalization of DCG we ranked query interpretations by their user score. To achieve better overview of the available results in this experiment we set $\lambda = 0.1$ (Equation 4.4); this enables the system to emphasize novelty of results in both datasets. We discuss the influence of λ value in Section 4.6.5. The results of the α-NDCG-W evaluation are presented in Figure 4.2.

Each diagram of Figure 4.2 corresponds to a different α value. Each data point of the X-axis of a diagram represents the k for top-k query interpretations. The Y-axis represents the corresponding α-NDCG-W value. We use the symbol "Rank" to denote the ranking algorithm without diversification, and "Div" to denote the ranking algorithm with diversification. The α-NDCG-W values in the diagrams are averaged on single-concept queries (sc) and multi-concept queries (mc) respectively. As we can see, in our experiments on the IMDB dataset (Figures 4.2a, 4.2c, and 4.2e), the average α-NDCG-W for top-1 result of both ranking and diversification on single concept queries was always 0.58, given any value of α. For top-5 results, the gain of single-concept queries increased to 0.9 in both datasets. For multi-concept queries, with α=0 the gain of ranking reaches 0.8 and 0.9 at top-6 in IMDB and Lyrics respectively. The relatively high α-NDCG-W values for α=0 confirm the quality of the ranking function.

As Figure 4.2 shows, for α=0 ranking dominates diversification in all the cases. This is expected, as for α-values below 0.5, relevance is rewarded over novelty. In this case, diversification does not show its benefit. In the Lyrics dataset, the first benefits of diversification for single-concept queries become visible already with α=0.5 at k=4, where α-NDCG-W improves by about 4%. This advantage increases with growing α, and achieves 8% at α=0.99. For single-concept queries on IMDB, we did not observe any difference between ranking and diversification (the lines Rank sc and Div sc almost overlap in all diagrams of the Figures Figures 4.2a, 4.2c, and 4.2e). This is because the top query interpretations returned by ranking already deliver distinct results. In this case diversification preserves the high gain values achieved by ranking. For multi-concept queries, the gain of diversification grows with increasing α. When α=0.99 and k¿3, diversification on mc queries outperforms ranking by about 7% in both datasets. The results of the paired ttest confirm statistical significance of this result for the confidence level of 95%. In summary, diversification performed on top of query ranking achieves significant reduction of result redundancy, while preserving retrieval quality in the majority of the cases.



(a) IMDB, $\alpha = 0$ **(b)** Lyrics, $\alpha = 0$

(c) IMDB, $\alpha = 0.5$ **(d)** Lyrics, $\alpha = 0.5$

(e) IMDB, $\alpha = 0.99$ **(f)** Lyrics, $\alpha = 0.99$

Figure 4.2 α-NDCG-W for Single-Concept (sc) and Multi-Concept (mc) Queries for Diversification (Div) and Ranking (Rank) for $\alpha = 0$, $\alpha = 0.5$, and $\alpha = 0.99$.

4.6.4 WS-Recall

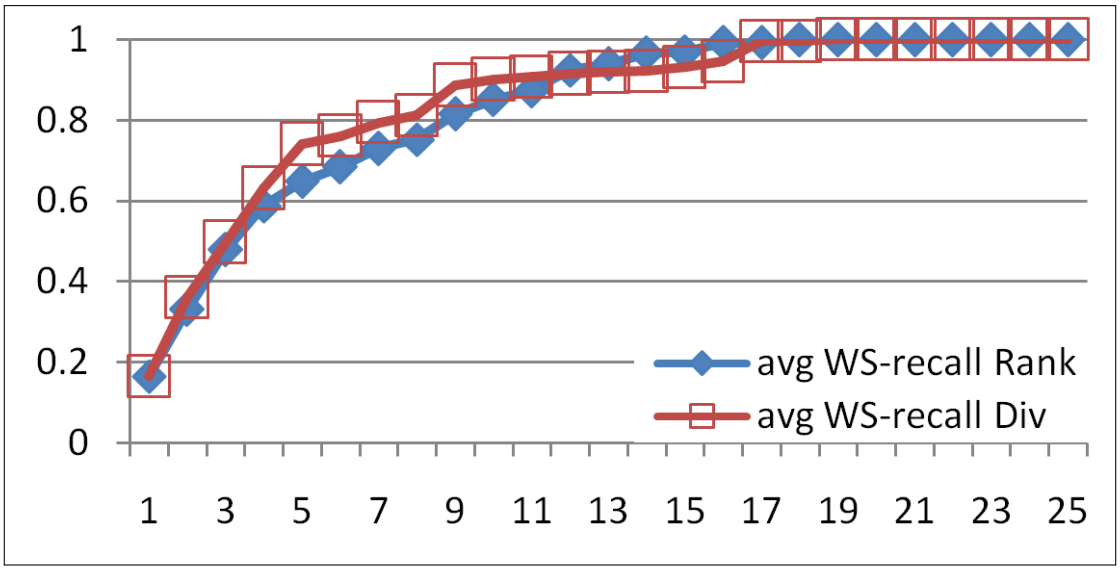
We evaluate recall quality of the system using the WS-recall measure presented in Section 4.5.2. WS-recall computation requires user assessments of subtopic relevance. As graded relevance assessments of top query interpretations were available to us as a result of the user study, we compute relevance of a subtopic (primary key) as the relevance of the interpretation which returns this primary key. As one and the same primary key can be returned by multiple distinct query interpretations, we take the maximal score. As user judgments were available only for a subset of the interpretation space, the absolute recall values obtained by this approach might be too optimistic. However, they enable a fair comparison of the algorithms. We present the results of the WS-recall evaluation in Figure 4.3.

Each data point of the X-axis of Figures 4.3a and 4.3b corresponds to k for top-k interpretations. The Y-axis represents the corresponding WS-recall value of ranking (Rank) and diversification (Div) averaged over a set of queries. For example, in the Lyrics dataset (Figure 4.3b) the WS-recall of ranking increased from 0.2 in top-1 to 0.8 in top-6. As Figures 4.3a, 4.3b show, on average, ranking and diversification perform similar with respect to recall. We observed a slight improvement by diversification for $k = 2 \dots 11$ in the IMDB dataset, whereas in Lyrics WS-recall at corresponding k values slightly decreased. Inspection of the actual query interpretations reveals that this is mainly due to the fact that ranking by relevance in Lyrics prefers complete query interpretations with large result sizes (i.e. large total number of returned tuples), whereas diversification pushes partial query interpretations with smaller result sizes. All other things equal, a larger result size increases WS-recall more. Normalizing result sizes for WS-recall is subject to future work. In total we did not observe any significant effect of diversification on WS-recall values.

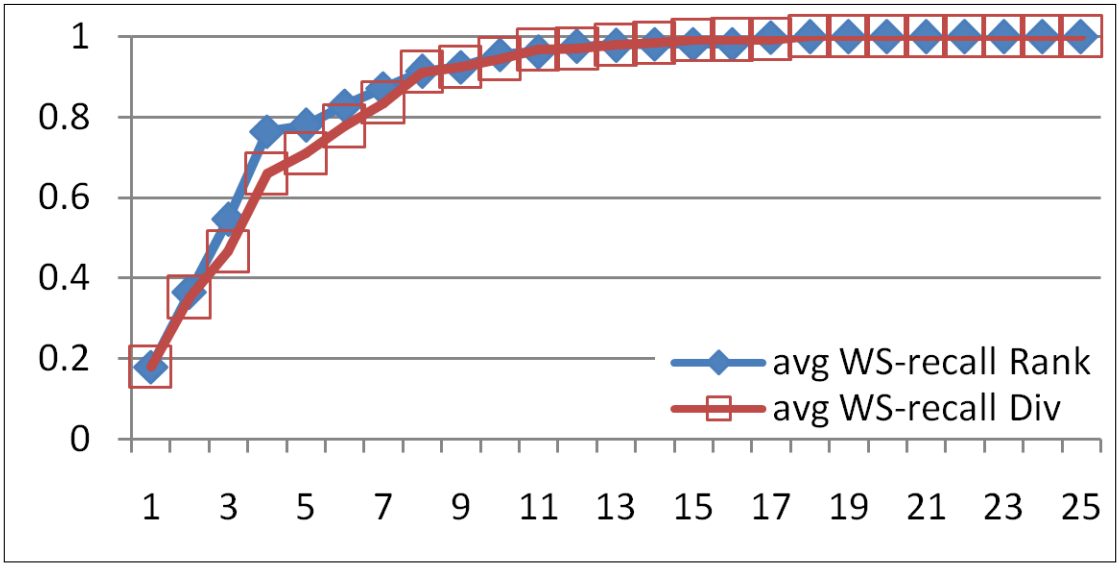
4.6.5 Balancing Relevance and Novelty

In Equation 4.4, λ is a parameter to balance query interpretation relevance against novelty. We evaluated influence of λ on α-NDCG-W at top-5 by α=0.99. The results on the lyrics dataset are presented on Figure 4.4.

The X-axis of Figure 4.4 presents the values of λ. The Y-axis represents the corresponding value of α-NDCG-W at top-5 by α=0.99. Each bar on Figure 4.4 presents α-NDCG-W for ranking and diversification of single-concept (sc) and multi-concept (mc) queries averaged over a set of queries. For example, the average α-NDCG-W of diversification for single concept queries increased from 0.82 by λ=1 to 0.91 by λ=0.01. As can be seen, high α-NDCG-W values achieved by diversification of both, single-concept and multi-concept queries decrease with increasing λ, until they meet α-NDCG-W of the original ranking in λ=1. The smaller the value of λ, the more visible is the impact of diversification and the more α-NDCG-W values of diversification outperform the original ranking. In contrast, with increasing λ, relevance of query



(a) IMDB



(b) Lyrics

Figure 4.3 WS-recall for Ranking (avg WS-recall Rank) and Diversification (avg WS-recall Div).

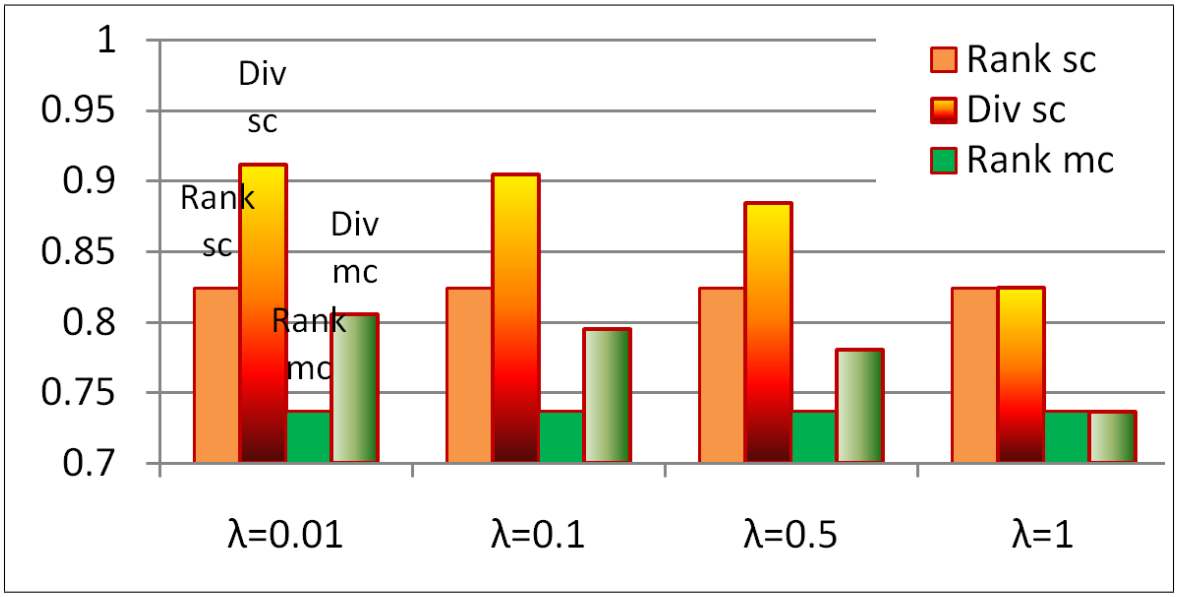


Figure 4.4 Relevance vs. Novelty: Selection of the Value for λ-Parameter. α-NDCG-W, k=5, α=0.99, Lyrics Dataset.

interpretations dominates over novelty and the amount of re-ranking achieved by diversification becomes smaller. For example, for λ= [0.01, 0.5] the Spearman's rank correlation coefficient between the ranks of the query interpretations in the initial ranking and their ranks after diversification ranges between [0.74, 0.82] for single-concept queries and [0.4, 0.67] for multi-concept queries in Lyrics. In the IMDB dataset multi-concept queries perform similar with rank correlation of [0.36, 0.69]. As different interpretations of single-concept IMDB queries already deliver distinct results, we did not observe any significant re-ranking by varying λ on this query set.

4.7 Discussion

In this chapter we presented DivQ - an approach to search result diversification over structured data to address Problem 2 presented in Chapter 1. In contrast to the state-of-the-art diversification approaches such as [AGHI09, CG98, CK06, CKC+08, GS09, WZ09, CL07, VSS+08] operating on search results directly, DivQ performs diversification at the query interpretation level before any search results are materialized. This enables DivQ to perform diversification of search results over structured data efficiently, as such materialization is computationally expensive. To the best of our knowledge, DivQ is the first approach that performs diversification of query interpretations over structured data.

To enable efficient search result diversification over structured data, in this chapter we first introduced a probabilistic query disambiguation model that enabled us to create relevant query interpretations. The query disambiguation model in this chap-

ter improves upon the initial model presented in Chapter 3 by taking into account keyword dependencies in database attributes. We evaluated the quality of the model in a user study. In the next step, we proposed a query similarity measure based on the Jaccard coefficient. This similarity measure operates on the syntactic level of query expressions and can thus be evaluated efficiently. Then, in order to obtain relevant and diverse query interpretations we presented a greedy algorithm. Following that, as results of keyword search over structured data differ from conventional documents considered in the existing work, we adapted state-of-the-art evaluation metrics for diversification in document retrieval to take into account graded relevance of subtopics and overlapping results. To this end we proposed α-NDCG-W and WS-Recall - an adaptation of α-NDCG [CKC+08] and S-recall [CK06] measures to assess quality of diversification in database keyword search.

Our evaluation results have demonstrated the quality of the proposed model and have shown that using our algorithms the novelty of keyword search results over structured data was substantially improved. Diversification performed on top of query ranking achieved significant reduction of result redundancy, while preserving retrieval quality in the majority of the cases. This way search results obtained using the proposed algorithms also better characterized possible answers available in the database than the results obtained by the initial relevance ranking.

5 Scaling Interactive Query Construction on a Very Large Database

5.1 Introduction

In this chapter we address Problem 3 from Chapter 1, namely enabling efficient incremental query construction over large scale databases. The amount of structured data available on the Web is constantly growing. With the prevalence of Web 2.0, a number of open databases have emerged on the Web, attempting to provide a platform for users to collaboratively create and maintain structured information. A typical example is Freebase[1], which currently contains more than 22 million entities and about 350 million facts from more than 100 domains, organized in 7,500 tables. Other examples include DBpedia [BLK+09], WikiTaxonomy [PS08], Probase [WLWZ12], and others, whose sizes have already reached the magnitude of several gigabytes. Databases of this kind are intended to accommodate heterogeneous information and knowledge. It is natural that each of these datasets contains a very large schema and a large volume of data. To a normal Web user, information seeking over such a heterogeneous database is a challenge.

The technology of interactive query construction introduced in Chapter 3 enables novice users to interactively create structured queries and retrieve desired information from a database. The interface of interactive query construction combines the usability of keyword queries with the expressiveness of structured queries. It enables a user to start with a keyword query and refine keywords into a structured query by interacting with the system. Through interaction, the user can provide additional information to disambiguate the semantics of the keyword query, and finally determine the structured expression reflecting user's informational need. Compared to keyword queries, structural interpretations created in such user interaction process offer enhanced expressiveness to retrieve the results with complex semantics, includ-

[1] www.freebase.com

ing collective results, e.g. *"All films starring Tom Hanks"*, or results involving more than one entity, e.g. *"The role of Tom Hanks in the film The Terminal"*. In this way, interactive query construction opens the world of structured queries to unskilled users, who are not familiar with structured query languages. It is also a useful tool for expert users who want to explore the data organized in an unfamiliar and a complex database schema.

Interactive query construction can be especially useful for information seeking on a large scale database, such as Freebase. On the one hand, to form structured queries, users needs to understand the database schema thoroughly, which is a time consuming process. On the other hand, keyword queries are normally highly ambiguous for such databases. For instance, in Freebase, the phrase *"Tom Hanks"* can be matched with the entities from the domains of *person*, *film*, *book*, *tv*, and *music*. As a result, there can be a large number of plausible answers to the keyword query *"Tom Hanks"*, such as the actor Tom Hanks, an American seismologist Thomas C. Hanks, and a book entitled "Tom Hanks". To find the desired information through a keyword search interface, a user may have to scan through a long list of search results. In contrast, a query construction interface suggests a few interaction options such as *"Tom Hanks is an actor"*, and *"Terminal is a location"* for the users to clarify their intents. By clicking the correct options, users help the system to form structured queries, which are able to retrieve the desired content more accurately.

As suggested in Chapter 3, approaches of interactive query construction work well for medium-sized databases of a particular domain, such as IMDB [IMD] and Lyrics [LYMC06], which contain around 20 tables. Our experiments in Chapter 3 have demonstrated that our initial approach to interactive query construction scales well on the databases containing up to 100 tables. However, initial approaches fail to scale on heterogeneous multi-domain databases composed of several thousands of tables, such as Freebase. First, when the database schema is very big, the interaction options generated by the existing schemes are usually not informative enough. As a result, a user may have to go through a laborious interaction procedure to construct the desired query. For example, the phrase *"Tom Hanks"* appears in more than thirty Freebase attributes, such that it can be interpreted into more than thirty meanings. Using the existing interaction schemes, the user may have to respond to each of the interpretations to finally clarify her intent. For a more complex keyword query, the procedure of interaction can become unacceptably long. Second, the interpretation space of a keyword query in a very big database is usually too big to materialize. State-of-the-art approaches to schema-based keyword search (e.g. [SA02, HP02, HGP03, LYMC06]) as well as our initial approach to incremental query construction that rely on the entirely materialized interpretation space, become infeasible in these settings.

The FreeQ system presented in this chapter builds upon the work presented in Chapers 3 and 4, and aims to scale interactive query construction over a very large database, addressing Problem 3 from Chapter 1. First, we propose to connect a hierarchical ontology with the database schema. Using the general concepts in the

ontology, we can form more informative interaction options that enable more efficient query construction. The hierarchical ontology can be constructed manually, such as the domain set in Freebase. It can also be a generic external ontology, such as YAGO [SKW07]. We conducted both theoretical and experimental studies to evaluate how a hierarchical ontology can enable efficient interactive query construction. Second, we design a scheme that explores interpretation spaces of keyword queries incrementally. This scheme can efficiently generate top-k structured queries and optimal interaction options. We conducted extensive experiments on Freebase. The results demonstrate the efficiency of our approach.

The rest of this chapter is organized as follows: Section 5.2 provides a summary of contributions contained in this chapter. Following that Section 5.3 discusses the specific background for the query construction over large scale databases. Then, Section 5.4 provides an overview of the basic concepts of query construction which were first introduced in Chapter 3 and discusses the advantages and limitations of the query construction approach introduced in Chapter 3. Then, Section 5.5 presents the query construction options of FreeQ that overcome some limitations in the previous work. Following that, Section 5.6 describes scalable algorithms for the query and option generation over a large scale dataset. Section 5.8 presents the evaluation results over Freebase. Finally, Section 5.8 discusses the results.

5.2 Summary of FreeQ Contributions

State-of-the-art approaches to schema-based database keyword search described in Section 2.2.3 such as [SA02, HP02, HGP03, LYMC06, AME07, KKR+06, TZC+06, LLWZ07, TCRS07, ZWX+07, CBC+09, QYC09] as well as the approach to incremental query construction proposed in Chapter 3 work well for databases with medium-sized schemas such as Internet Movie Database [IMD] and Lyrics dataset [LYMC06]. In contrast, to the best of our knowledge, neither state-of-the-art approaches to schema-based keyword search in databases nor incremental query construction we proposed in Chapter 3 can handle databases with very large schemas efficiently.

Scaling incremental query construction on a large scale dataset calls for two main adaptations: First, state-of-the-art approaches to schema-based database keyword search (e.g. [SA02, HP02, HGP03, LYMC06]) as well as approaches to incremental query construction presented in Chapter 3 rely on a completely materialized query interpretation space. In face of a large scale database, such enumeration of all query interpretations is not feasible. Therefore, in this chapter we develop scalable algorithms that retrieve the most relevant top-k query interpretations efficiently. Second, the approach to incremental query construction presented in Chapter 3 used only partial query interpretations as items for user interactions. As the number of such partial interpretations grows sharply with an increasing schema size, these interpretations alone cannot guarantee an efficient query construction process over large scale

datasets. To this extent, in this chapter we propose novel user interaction options based on ontologies.

In summary, in this chapter we take an important step to interpret keyword queries and perform iterative query construction over large scale datasets efficiently. To this extent, we further develop and generalize the approach to interactive query construction presented in Chapter 3 and make the following contributions:

- First, we proposed a new type of interaction options based on ontologies to enable scalable interactive query construction, and provide a theoretical justification about the effectiveness of these options.
- Then, we developed a scheme to enable incremental exploration of very large query interpretation spaces to generate top-k structured queries and interaction options efficiently, without the complete knowledge of the interpretation space.
- Following that, we performed an experimental study on Freebase to verify the effectiveness and efficiency of the proposed approach.
- To the best of our knowledge, this is the first attempt to enable efficient keyword-based query construction on such large scale database as Freebase, considering that most existing work on database keyword search uses only test sets of small schemas, such as DBLP, IMDB, etc.

Our experiments on Freebase, a large scale dataset containing more than 7,500 tables, have shown that the proposed FreeQ system is effective and efficient in interactive query construction over large scale data. Our results confirmed the effectiveness of the ontological layer created using the native taxonomy of Freebase. Furthermore, we have demonstrated that external ontologies, such as the YAGO ontology [SKW07], can be used to further increase the efficiency of incremental query construction.

5.3 Specific Background

In Chapter 3 we proposed a probabilistic incremental query construction model for an interactive user interface. In Chapter 4, we further developed the probabilistic model presented in Chapter 3 and developed methods to provide an overview of search results available within a database. While these methods are well-suited for medium-sized schemas, they do not provide a sufficient solution to large scale datasets with flat schemas, such as the schema of Freebase. This is because these methods relied only on the database internal statistics and partial query interpretations to generate query construction options. In large scale databases, such partial query interpretations are not informative enough to enable efficient reduction of the search space in the user interaction process. FreeQ presented in this chapter alleviates this problem by using ontologies, such as the domain hierarchy of Freebase and the YAGO

ontology [SKW07], in the option generation process. Furthermore, previous work on keyword query disambiguation and incremental query construction was performed with an assumption, that it is feasible to completely materialize the entire space of query interpretations. To this end, we relied on a set of query templates generated a-priori. In contrast, FreeQ presented in this chapter will relax this assumption and present the algorithms to incrementally materialize very large query interpretation spaces and generate the top-k structured queries and query construction options on the fly over a large scale database.

5.4 Preliminaries of Interactive Query Construction

A user interface for interactive query construction is presented in Fig. 5.1[2]. The interface is composed of four parts: (1) an input field for keyword queries, (2) a query construction panel for presenting the interaction options, (3) a query window for presenting structured queries, and (4) a result window for query results. Suppose a user, whose name is Alice, issues a keyword query to the system. The system first tries to guess Alice's intent and generates the top-k most likely structured queries in the query window (3). If one of the top-k structured queries matches Alice's intent, she can click on the query to obtain the results (4). If no query in the top-k list (3) interprets Alice's intent correctly, she can interact with the query construction panel (2) to construct the desired structured query. Whenever Alice clicks on an interaction option, the structured queries in the query window (3) are refined, such that only those queries complying with Alice's selection are preserved. Simultaneously, a new set of query construction options is presented in the query construction panel (2). The interaction continues until Alice obtains the desired structured query and results.

In this section, we introduce the basic model for enabling such interface for interactive query construction. We also elaborate on the challenges posed by large scale databases.

5.4.1 The Model

We model the schema of a database as a graph.

Definition 5.4.1. *A* **schema graph** *is a directed graph $G = (V, E)$, where each vertex $v \in V$ represents a relational table and each edge $e \in E$ represents a foreign key relationship. In the graph, each node v is associated with a set of attributes, denoted by $A(v)$, where the i^{th} attribute is represented by $v.a_i \in A(v)$.* □

[2] Fig. 5.1 includes the following images: http://de.fotolia.com/id/10056489 ©ioannis kounadeas - Fotolia.com; http://de.fotolia.com/id/9974098 ©XYZproject - Fotolia.com

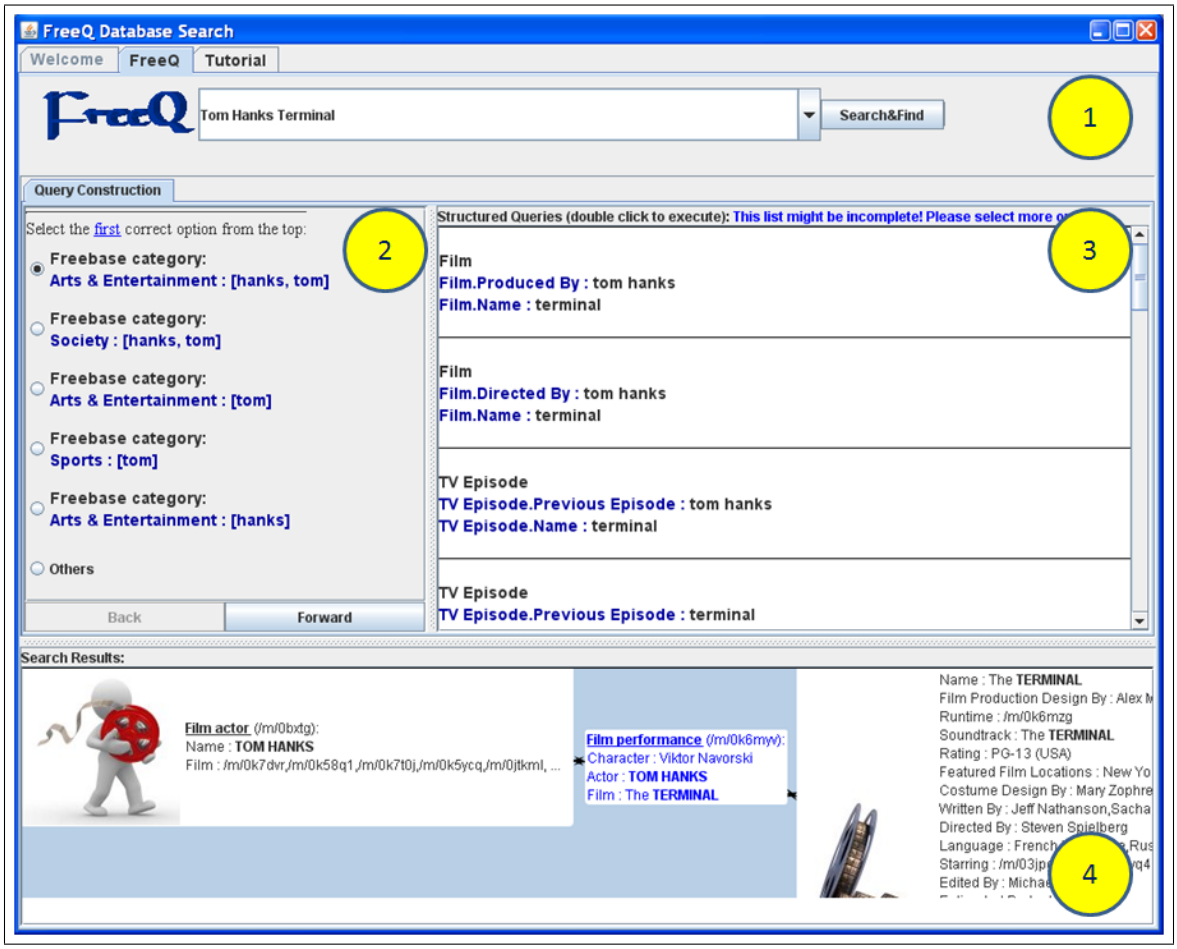


Figure 5.1 FreeQ User Interface. Its components include: (1) an input field for keyword queries, (2) a query construction panel, (3) top-k structured queries, and (4) query results.

We use the number of vertices to represent the size of a schema graph. Using the schema graph, we can create structured queries.

Definition 5.4.2. *Given a schema graph $G = (V, E)$, a* **structured query** *is an edge preserving map $G' = (V', E')$, such that there is a function $L : V' \to V$ which satisfies: for each vertex $v' \in V'$ in the structured query, there is a vertex $L(v') \in V$ in the schema graph such that v' and $L(v')$ represent the same relational table, and for each edge $\{v'_1, v'_2\} \in E'$ in the structured query, there is an edge $\{L(v'_1), L(v'_2)\} \in E$ in the schema graph.*

In addition, each vertex v' in the structured query can be associated with a number of predicates. Each predicate is in the form $v'.a_i$ op c_i, where $v'.a_i$ is an attribute of v', op is a comparison operator, and c_i is a constant. □

For instance, given a film database, a query looking for all the actors who have collaborated with Tom Hanks can be expressed as:

Q_1= {structure:$actor_1 \bowtie acts \bowtie film_1 \bowtie acts \bowtie actor_2$,
predicates:$actor_1.name$ = "$Tom\ \ Hanks$"}.
It is worth mentioning that each table in the database occurs only once in the schema graph. In contrast, each table can occur multiple times in a structured query.

In the process of interactive query construction, users express their informational needs as keyword queries.

Definition 5.4.3. *A* **keyword query** *is a bag of terms* $K = \{k_1, k_2, ..., k_n\}$*, where duplicates are allowed.* □

In Definition 5.4.3, a term is a normalized class of tokens that is included in the system's dictionary. For token normalization, state-of-the-art Information Retrieval techniques such as case folding and word segmentation can be applied [MRS08].

The main function of FreeQ is to translate a user's keyword query into the intended structured query. We call such translation a **query interpretation**. We say that a query interpretation is **complete** if this query interpretation contains all keywords from the initial user query. Otherwise we talk about **partial** query interpretation. We call the set of all complete query interpretations of a keyword query K an **interpretation space** of K.

For instance, keyword query *"Tom Hanks Film"* can be interpreted to:
Q_2={structure:$actor \bowtie acts \bowtie film$,
predicates:$actor.name$ = "$Tom\ \ Hanks$"}.
In this interpretation, keywords *"Tom Hanks"* are mapped to the constant of a predicate, and *"Film"* is mapped to a table name. The following query is a partial interpretation of *"Tom Hanks Film"*, where only *"Tom Hanks"* is interpreted:

O_1 ={structure: $actor$,
predicates: $actor.name$ = "$Tom\ \ Hanks$"}.

In the process of query construction we interpret user's keywords and generate query construction options (QCOs) to assess the meaning of the keywords intended by the user. We can do that in two ways: First, we can interpret keywords very specifically as a part of a structured query (as performed in Chapter 3). We refer to these QCOs as **query-based QCOs**. For instance, O_1 can be used as a QCO, which indicates that *"Tom Hanks"* should be interpreted as an actor's name. Second, we can also assess the general meaning of the keywords and interpret them as a generic concept representing a class of structured queries. For example, we can interpret *"Tom Hanks"* as a more generic class person, which is a superclass of actor:

O_2 ={structure: $person$,
predicates: $person.name$ = "$Tom\ \ Hanks$"}.

To enable efficient user interaction over large database schema, in this chapter we introduce ontology-based QCOs. In a generic object-relational database, a table can be regarded as an entity type, and an attribute of the table can be regarded as a property. Using a hierarchical ontology, a set of entity types can be abstracted into a superclass, and a set of properties can be abstracted into a super-property. For instance, entity types *painter* and *musician* can be abstracted into *artist*, and their properties *painting* and *music* can be abstracted into *work*. Using these superclasses and super-properties, we can create general QCOs that subsume larger proportions of a query interpretation space than the query-based QCOs.

Definition 5.4.4. *Given a schema graph $G = (V, E)$, we use $sv \vdash v$ to denote that sv is a superclass of $v \in V$ and $se \vdash e$ to denote that se is a super-property of $e \in E$. Superclass and super-properties are partial order relationships.* □

With the concepts of superclass and super-property, we define ontology-based query interpretations and ontology-based QCOs.

Definition 5.4.5. *Let $K = \{k_1, k_2, ..., k_n\}$ be a keyword query. Let $Q = (V, E)$ be a query interpretation of K. Let Q^o be isomorphic to Q, where the isomorphism function is $f(.)$ (that applies to the predicates too). Q^o is an* **ontology-based interpretation** *of K, iff $f(.)$ satisfies: (1) for all $v \in V$, $f(v) \vdash v$; (2) for all $e \in E$, $f(e) \vdash e$; (3) for all attributes $v.a_i \in A(v)$ in the predicates, $f(v.a_i) \vdash v.a_i$. We say that Q^o is a* **super-interpretation** *of Q.* □

When we use ontology-based interpretations as QCOs, we call them **ontology-based QCOs**. In summary, QCOs generated by our system can be either query-based or ontology-based QCOs.

Definition 5.4.6. *A* **Query Construction Option** *(QCO) is a mapping from a subset $K' \subseteq K$ of keyword query K to either:*

- *a structured query Q (a query-based QCO), or*
- *an ontology-based interpretation of K' (an ontology-based QCO).* □

In the interaction process the user is supposed to select the options that subsume her intended query interpretation.

Definition 5.4.7. *Given a QCO O and a QCO O', we say that O* **subsumes** *O', if either:*

- *O is a subgraph of O', or*
- *O is a super-interpretation of O'.*

Subsumption relationship is transitive, i.e. if O **subsumes** *O' and O'* **subsumes** *O'', then O* **subsumes** *O''.* □

Certainly, as O_1 is a subgraph of the interpretation Q_2, O_1 subsumes Q_2. O_2 is an ontological interpretation of *"Tom Hanks"*, and it is a super-interpretation of O_1. As subsumption relationship is transitive, both O_2 and O_1 subsume Q_2, which is a complete interpretation of the query *"Tom Hanks Film"*.

With the above concepts, the conceptual process of interactive query construction can be modeled as follows:

0. Given a database whose schema is $G = (V, E)$, a user issues a keyword query $K = \{k_1, k_2, ..., k_n\}$.
1. **Initialization**: Let ζ be an interpretation space of K based on G.
2. **Top-k Generation**: The system retrieves the top-k interpretations from ζ, and presents them to the user. If the user finds the intended interpretation in the top-k, the query construction process terminates. Otherwise, the process continues with Step 3.
3. **QCO Generation**: The system generates a QCO O and lets the user decide whether O subsumes the intended interpretation of K.
4. **Post Interaction**: If the user indicates that O is true, then the system removes all the interpretations that cannot be subsumed by O from ζ. Otherwise, the system removes all the interpretations subsumed by O from ζ. Go back to Step 2.

Each iteration of the process requires one round of interaction with the user. As the interaction goes on, the interpretation space ζ keep shrinking. Because ζ is finite, the process guarantees to terminate at a certain point. Nevertheless, we would like the process to be short, so that users can obtain desired information as early as possible. The efficiency of query construction can be measured naturally by the number of iterations of the process. We call this measure interaction cost.

Definition 5.4.8. *Given a process of interactive query construction, its* **interaction cost** *is the number of iterations it has been executed, which is equivalent to the number of QCOs evaluated by the user.* □

As shown in Figure 5.1, it is possible and probably more efficient to present multiple QCOs to a user simultaneously, so that the user can select the first correct QCO from the list. According to Chapter 3, such multi-option interface can be induced from a single-option interface, where the user decides on one option at a time. For simplicity of our analysis, we stick to the single-option interface for the rest of the chapter.

5.4.2 Limitations of the Existing Approaches

When a database, and especially its schema graph, becomes big, it is difficult for the existing approaches to incremental query construction to realize an efficient query construction process. This is due to the following two issues:

Problem 5.1. *Inefficient query-based QCOs:*

To minimize the interaction cost, the query construction process needs to shrink the query interpretation space quickly. In other words, the evaluation of each QCO should be able to remove a significant proportion of the interpretation space. Therefore, we desire the proportion of query interpretations subsumed by each QCO to fall in a certain range. This proportion should not be too small, as in this case the denial of a QCO could not reduce the interpretation space effectively. It should not be too big either, as in this case the acceptance of a QCO could not reduce the interpretation space effectively.

When the schema graph is big, a keyword can have a large number of occurrences spread across the database, resulting in a vast number of partial interpretations (query-based QCOs). The proportion of the interpretation space subsumed by each query-based QCO will be very small. As a result, query construction processes using only query-based QCOs, such as the one described in Chapter 3, cannot be efficient in face of a very large schema. Apart from query-based QCOs, we need more general QCOs to enable efficient query construction.

Problem 5.2. *Very large query interpretation spaces:*

When the schema graph becomes big, it is no longer feasible to materialize the interpretation space of a complex keyword query entirely. On the one hand, with an increasing size of a schema graph, the number of its subgraphs grows very sharply. On the other hand, the occurrences of keywords are more numerous in a larger database. As a result, the number of the possible interpretations of a keyword query can be very big for a large scale database.

The approach to interactive query construction presented in Chapter 3, and approaches to schema-based database keyword search [SA02], [HP02], [LYMC06], [LLWZ07] rely on an entirely materialized query interpretation space. To facilitate generation of query interpretations, in Chapter 3 we utilized pre-generated query templates. This approach can hardly work with a big schema, as it is infeasible to pre-generate all of the possible query templates. Therefore, we need a new mechanism which can enable efficient identification of good QCOs and top-k query interpretations, without the prior knowledge of the entire interpretation space.

5.5 Efficiency of QCOs

As pointed out by Problem 5.1 in Section 5.4.2, to minimize the interaction cost, the QCOs presented to the user need to shrink the query interpretation space quickly. In a large scale database, each single keyword can have numerous occurrences. As a result, the query-based QCOs utilized by the approach described in Chapter 3 become inefficient in reducing interpretation spaces. In this chapter, we introduce novel ontology-based QCOs. An ontology-based QCO can subsume a wider proportion of an interpretation space, such that it is usually more efficient than a query-based QCO. Intuitively, if the user provides feedback on an ontology-based QCO, we get an implicit user's feedback on multiple query-based QCOs subsumed by this QCO within a single user interaction. Thus a query construction process using ontology-based QCOs requires less steps. In this section, we consider the efficiency of ontology-based QCOs from the perspective of information theory.

5.5.1 Generation of Ontology-Based QCOs

Ontology-based QCOs can be created based on a hierarchical ontology or taxonomy. In order to create ontology-based QCOs, we need an ontology on top of the database schema, which defines superclasses and super-properties. This ontology can be a manually defined one, such as the domain hierarchy of Freebase. Alternatively, we can utilize external ontologies, such as e.g. WordNet [Fel98], and YAGO [SKW07], by mapping the elements of the database schema to the concepts in these ontologies. In this case state-of-the-art schema matching techniques (see [RB01]) can be used. We present our mapping of YAGO and Freebase later in Chapter 6.

With ontology-based QCOs, we can enable more efficient query construction, especially when confronted with a big database schema. To illustrate a query construction process using ontology-based QCOs, we consider the query *"Emperor Album"*, which intends to retrieve the albums of the artist Emperor from Freebase. To create the ontology-based QCOs, we make use of the domain hierarchy of Freebase. This hierarchy groups together Freebase tables such as *artist*, *album*, and *monarch* in the domains e.g. *music* and *royalty*, and further organizes these domains into the categories such as *Arts & Entertainment* and *Society*.

For this particular query, if we use only query-based QCOs (as performed in Chapter 3), our system requires a user to interact with 74 QCOs to identify the intended interpretation. Using ontology-based QCOs, the user only needs to interact with the 10 QCOs listed in Table 5.1. The keyword *"album"* is not very ambiguous, as it occurs mostly in the domain of *music*. To disambiguate this keyword, FreeQ does not utilize any ontology-based QCOs. In contrast, the keyword *"emperor"* is very ambiguous. *"Emperor"* occurs in 221 attributes of Freebase, which are spread across multiple categories and domains. To disambiguate *"emperor"*, it is much faster if we use ontology-based QCOs. With ontology-based QCOs, we manage to first restrict

QCOs (bold ones are ontology-based QCOs)	**User's feedback**
Table *artist* (domain *music*): "album"	×
Table *release* (domain *music*): "album"	×
Table *recording contribution* (domain *music*): "album"	×
Table *album* (domain *music*): "album"	✓
Domain *royalty* (Category *Society*): "emperor"	×
Category *Arts & Entertainment*: "emperor"	✓
Domain *fictional universe* (Arts & Ent.): "emperor"	×
Domain *opera* (Arts & Ent.): "emperor"	×
Domain *media common* (Arts & Ent.): "emperor"	×
Query: **album** ⋈ **artist** "emperor" ⊂ artist.name	✓

Table 5.1 A Query Construction Example for the Query *"Emperor Album"* using Ontology-based QCOs

the meaning of *"emperor"* to the category of *Arts & Entertainment*. Within this category, the exact meaning of *"emperor"* can be identified easily.

In what follows, we analyze how ontology-based QCOs achieve such efficiency.

5.5.2 A Measure of QCO Efficiency

As depicted in Section 5.4.2, in each round of interactive query construction, FreeQ needs to select one QCO to present to the user. For a keyword query, there is usually a large number of available QCOs. In principle, FreeQ should always select the most efficient QCO that can minimize the final interaction cost. The efficiency of a QCO can be quantified using the information theory.

Let ζ denote the interpretation space of a keyword query K. Then, the uncertainty of K's interpretation can be measured by the entropy $H(\zeta)$, which can be computed as:

$$H(\zeta) = -\sum_{I \in \zeta} P(I) \times log_2 P(I), \tag{5.1}$$

where $P(I)$ denotes the probability that the interpretation I is the interpretation intended by the user.

The process of query construction is the process of reducing the uncertainty of K's interpretation. After one round of interaction with the user, FreeQ obtains the knowledge of one QCO, say O. Then, the uncertainty is reduced to $H(\zeta|O)$, i.e., the conditional entropy of ζ given O. The difference between $H(\zeta)$ and $H(\zeta|O)$ is known as the expected *information gain* provided by O, denoted by:

$$IG(O) = H(\zeta) - H(\zeta|O). \tag{5.2}$$

To minimize the interaction cost, we need maximize the information gain of each QCO presented to the user. Obviously, the knowledge about ζ contains the knowledge of any O. In other words, the information gain provided by O is exactly the entropy of O. Therefore, we have:

$$IG(O) = H(\zeta) - H(\zeta|O) = H(O). \tag{5.3}$$

In turn, the entropy of O can be calculated as:

$$H(O) = -P(O)log_2P(O) - P(\neg O)log_2P(\neg O), \tag{5.4}$$

where $P(O)$ is the probability that O is accepted by the user. Let $\zeta(O)$ denote the complete set of query interpretations subsumed by O. Then, $P(O)$ can be computed as:

$$P(O) = \sum_{I \epsilon \zeta(O)} P(I). \tag{5.5}$$

To summarize, the efficiency of a QCO can be measured by its entropy. Therefore, FreeQ is supposed to select the QCO with the highest entropy in each round of user interaction.

5.5.3 Effects of Ontology-based QCOs

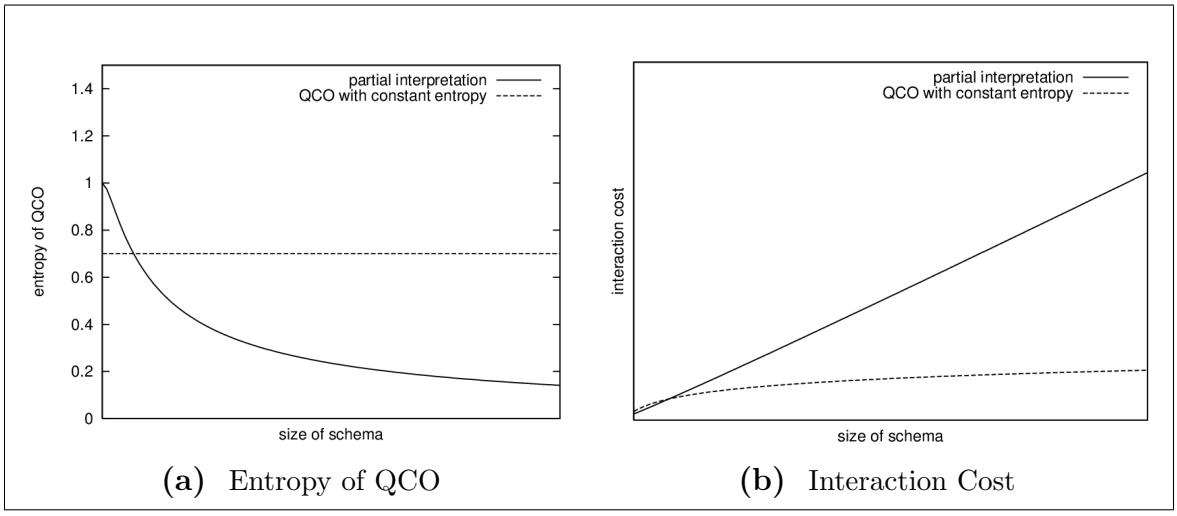
As discussed in Section 5.5.2, to achieve fast query construction, in each round of interaction, FreeQ is supposed to present the user with an efficient QCO, i.e., a QCO whose entropy is sufficiently high. The question is whether such QCOs would be available. To answer this question, we first propose the following measure to quantify the efficiency of an entire query construction process.

Definition 5.5.1. *During an interactive query construction process, if we can ensure with a high probability that the entropy of each QCO presented to the user is larger than ϵ ($0 < \epsilon \leq 1$), we say that the query construction process is* **ϵ-efficient**.

According to Definition 5.5.1, to minimize the interaction cost, we should maximize the lower bound (ϵ) of the efficiency of the QCOs presented to the user. We can show that query-based QCOs alone, i.e. the QCOs used in Chapter 3, cannot guarantee a lower bound. In contrast, if we have an ontology with a sufficient number of concepts of diverse generality, by using ontology-based QCOs, we can achieve ϵ-efficient query construction with a good lower bound.

To simplify our analysis, we assume: (1) the number of possible interpretations of a keyword grows linearly with the size of the schema graph; (2) the number of the possible interpretations of an entire keyword query increases polynomially; (3) all the complete query interpretations are equally probable.

Let the size of the schema be x. According to (1), the number of partial interpretations (i.e. query-based QCOs) for a keyword can be expressed as $\alpha \times x$, where α is a



(a) Entropy of QCO **(b)** Interaction Cost

Figure 5.2 Efficiency of QCO and Interaction Cost vs. Schema Size.

constant. Then the entropy of a query-based QCO for each keyword will be $H(\frac{1}{\alpha \times x})$, which can be plotted as the solid curve in Figure 5.2a. We can see that when the database schema grows, the efficiency of each query-based QCO will decrease. This efficiency can drop to a very small value when the database schema is big.

According to (2), the size of an interpretation space can be modeled as $\beta \times x^{\gamma}$, where β and γ are constants. Based on (3), the most efficient query-based QCO will be an interpretation for a single keyword, whose entropy can be modeled as: $H(\frac{1}{\alpha \times x})$. Therefore, the average interaction cost can be calculated as the entropy of the whole interpretation space divided by the maximum entropy of an QCO, i.e. $log_2(\beta \times x^{\gamma})/H(\frac{1}{\alpha \times x})$, which can be plotted as the solid curve in Figure 5.2b. As we can see, if we use only query-based QCOs, the interaction cost can increase quickly with the size of the database schema.

In contrast, if we can achieve a 0.7-efficient query construction process, that is, the entropy of each QCO be no less than 0.7 (as illustrated by the dashed line in Figure 5.2a), the growth of the interaction cost can be significantly reduced (as illustrated by the dashed line in Figure 5.2b). This is achievable, if we utilize ontology-based QCOs.

The concepts in an ontology normally have a variety of generality. There are very specific concepts that can subsume small sets of entity types, such as *artist* and *book*. There are also very general concepts that can subsume larger proportions of entity types, such as *person* and *artifact*. As a result, no matter how big the query interpretation space is, it is always possible to find suitable concepts to form QCOs that can subsume a certain proportion of the interpretation space that would yield big entropies. As a simple analysis, we assume that the probability of a random ontology-based QCO is a random value between 0 and 1. Then, within the set of N ontology-based QCOs, the probability that we can find a QCO whose entropy is large

than ϵ is:

$$P_{\exists o: H(o)>\epsilon} = 1 - \left(\frac{H^{-1}(\epsilon)}{0.5}\right)^N, \tag{5.6}$$

where $H^{-1}()$ is the inverse function of the binary entropy. In this function, no matter how big ϵ is, we can always find a big enough N, such that the resulting probability is close to 1. (As N is an exponent in the formula, it normally does not need to be very big.) In other words, as long as there is a rich ontology with a sufficient number of concepts of diverse generality, we can achieve ϵ-efficiency for interactive query construction.

5.6 Generation of Structured Queries

To perform query construction, FreeQ is required to quickly generate the most efficient QCOs and the most probable complete query interpretations. As mentioned in Problem 5.2, Section 5.4.2, the existing approaches to interactive query construction and schema-based database keyword search in general require a complete materialization of the query interpretation space [SA02], [HP02], [LYMC06], [LLWZ07], [DZNonb]. For a large scale dataset, the interpretation space of a keyword query is usually very big, such that it is no longer feasible to materialize this space entirely. To this end, we develop new algorithms that can generate top-k most probable query interpretations and QCOs without the knowledge of the entire interpretation space.

5.6.1 Query Hierarchy for the QCOs Generation

To enable efficient generation of QCOs and query interpretations, we organize the QCOs of a keyword query in a query hierarchy based on their subsumption relationships. Using the query hierarchy, we can materialize the interpretation space of a keyword query step-by-step by following the subsumption relationships of the QCOs. During the progress of the materialization, a lot of QCOs and interpretations can be eliminated based on the information provided by user interactions. Such progressive materialization is much less costly than the generation of an entire interpretation space.

Figure 5.3 illustrates a query hierarchy, in which the arrows represent reversed subsumption relationships. The more general the QCOs, the lower their positions in the query hierarchy. The most general QCOs are the single-node QCOs, i.e. the QCOs involving only one entity type such as O_2, O_1, and O_3. These QCOs are located at the bottom of the hierarchy. The top level of the query hierarchy consists of the complete query interpretations (e.g. Q_3, and Q_4), which constitute the interpretation space of a keyword query. The entire query hierarchy looks like an upside-down trapezoid, as there are much more complete query interpretations than single-node QCOs.

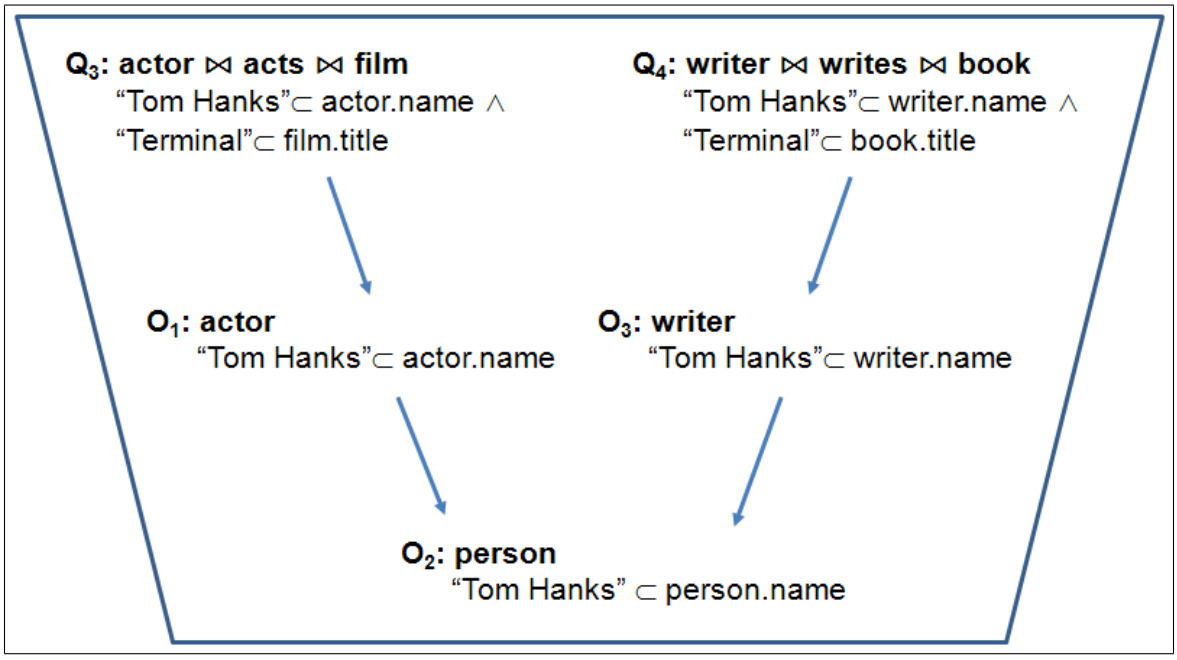


Figure 5.3 An Example of a Query Hierarchy using Query-based and Ontology-based QCOs for the Query "Tom Hanks Terminal" (the arrows represent reversed subsumption relationships, e.g. O_2 subsumes O_1).

As mentioned previously, it is infeasible to instantiate the complete query hierarchy at the query time. Given a big schema, the top levels of the query hierarchy can even be too big to be accommodated in the main memory. Therefore, FreeQ chooses to instantiate the query hierarchy incrementally throughout the process of query construction. The query construction process of FreeQ conforms to the generic process defined in Section 5.4.1, except that it does not materialize the entire query interpretation space. This process works as follows:

- **1. Initialization:** Upon receiving a keyword query, FreeQ identifies database attributes and schema elements containing keyword occurrences using an inverted index. Based on the keyword occurrences and an ontology, it generates the most general QCOs. As a result, the bottom of the query hierarchy is instantiated. As the bottom of the query hierarchy is small, its instantiation does not incur much cost.

- **2. Top-k Generation:** To generate the top-k complete query interpretations, FreeQ performs a **depth first traversal (DFS)** of the query hierarchy from the bottom up. All the QCOs and interpretations encountered during the DFS are instantiated. DFS enables FreeQ to reach the top of the query hierarchy by instantiating the minimal number of QCOs. The first k complete interpretations encountered by the DFS are presented to the user as the top-k interpretations.

- **3. QCO Generation:** FreeQ evaluates the QCOs in the instantiated part of the query hierarchy and selects the QCO with the highest entropy. This QCO is presented to the user.
- **4. Post Interaction:** After the user provides feedback on the QCO, FreeQ truncates the query hierarchy according to the user's selection. If the user has denied the QCO, all the QCOs or query interpretations subsumed by the denied QCO are removed from the query hierarchy. If the user has accepted a QCO, only the QCOs and interpretations subsumed by the accepted QCO are preserved. After the truncation, the query hierarchy becomes smaller. If the size of the instantiated part of the hierarchy falls below a threshold, FreeQ perform a **breadth first traversal (BFS)** of the query hierarchy (from the bottom up), during which more QCOs are instantiated. The BFS stops as the size of the instantiated part reaches a predefined upper bound.

As we can see, as the process of the interactive query construction goes on, the instantiated part of the query hierarchy remains stable in size. On the one hand, this part is truncated as the user reveals more information in the interaction process. On the other hand, the instantiated part is continuously expanded by the BFS and DFS to ensure that its size is sufficient to generate efficient QCOs and high quality top-k interpretations.

5.6.2 Efficient Hierarchy Traversal

Breadth first traversal (BFS) and depth first traversal (DFS) are the two basic approaches FreeQ employs to explore a query hierarchy. To make interactive query construction smooth to the user, it is important to ensure the efficiency of BFS and DFS. Both BFS and DFS traverse the query hierarchy by following the subsumption relationships. Each traversal step expands a QCO to one of the QCOs it subsumes, e.g. expanding O_2 to O_1, or O_1 to Q_3 in Figure 5.3. Therefore, it is crucial to make the QCO expansion efficient.

In principle, a QCO can be expanded in two different ways. **The first type of QCO expansion** is to replace an entity type or a property of the QCO with its subclass or sub-property. Such expansion does not add any additional keywords to the QCO. For example, the expansion of O_2 to O_1 in Figure 5.3 belongs to the first type. This type of expansion requires only the knowledge of the ontology structure. Once we have indexed the ontology, such that its subclasses or sub-properties can be retrieved quickly, this type of expansion will be very efficient. **The second type of QCO expansion** is to find the paths in the schema graph to connect a QCO and an additional keyword occurrence. For instance, to expand O_1 to Q_3 in Figure 5.3, we need to identify the path "$actor \bowtie acts \bowtie film$" to connect O_1 and the keyword occurrence $film.title : terminal$.

To identify the paths for connecting a QCO with a keyword occurrence efficiently, a common approach is bi-directional search. Basically, this search starts from both the QCO and the keyword occurrence, and conducts breadth first search on the schema graph for connecting paths. As such bi-directional search will be repeated frequently during the query construction process, a recurring traversal of the schema graph can be very inefficient. To this extent, FreeQ pre-indexes all the paths in the schema graph starting with any keyword occurrence. The index is constructed at the beginning of the query construction, as the user issues a keyword query. To ensure scalability of the indexing, we restrict the maximal size of the structured queries that can be constructed by FreeQ. The maximal path length can be induced from this restriction. As bi-directional search is used, the length of the paths being indexed does not need to exceed a half of the maximum path length.

To maximize the quality of the top-k query interpretations and QCOs generated by FreeQ, we always start the expansion with the QCOs that are most likely to be intended by the user. This applies to both DFS and BFS. As a result, the top-k interpretations generated by DFS will be more likely to meet the user's requirements, and the QCOs instantiated by BFS will be more likely to have high entropy. To ensure a short response time of FreeQ, we also enforce a time limit on both BFS and DFS. That is, we stop BFS and DFS once the time limit has been reached, even though the top-k interpretations may have not been completely generated, or the number of the instantiated QCOs has not yet reached a predefined upper bound. We demonstrate through the experiments that the resulting hierarchy traversal procedure is effective and efficient.

The algorithms of the BFS and DFS are presented in Algorithm 5.1 and Algorithm 5.2 respectively.

Algorithm 5.1: A BFS Algorithm for the Hierarchy Expansion

```
input  : topLevelQ: the upper frontier of the hierarchy
         timeLimit: time limit for option generation
         queue: the priority queue
         acceptedOptions: QCOs accepted by the user
         deniedOptions: QCOs denied by the user
output: resultQCO: a new set of QCOs

begin
    //add the QCOs to the priority queue
    queue.add(topLevelQ);
    //go through the queries in the queue
    priorityQueueLoop: while
    !queue.isEmpty() AND checkTime(timeLimit) do
        //take the next QCO from the queue
        QCO = queue.poll();
        pathList: while QCO.hasMorePaths() do
            // check the time limit
            if !checkTime(timeLimit) then
                //preserve the QCO for the next round
                queue.add(QCO);
                return resultQCO;
            path = QCO.getNextPath();
            // mark the path as explored path.markExplored();
            if !isValid(path, acceptedOptions, deniedOptions) then
                continue pathList;
            // create a new query construction option
            QCO_new = QCO.addPath(path);
            resultQCO.add(QCO_new);
    // check if the next level of the hierarchy is complete
    if queue.isEmpty() then
        levelGenerationComplete = true;
    return
```

Algorithm 5.2: A DFS Algorithm for top-k Query Generation

input : *topLevelQ*: the upper frontier of the hierarchy
timeLimit: time limit for option generation
acceptedOptions: QCOs accepted by the user
deniedOptions: QCOs denied by the user
k: number of queries to be generated

output: *resultTopK*: a new set of top-k queries

```
begin
    //rank the QCOs according to their probability
    rank(topLevelQ);
    //add QCOs to the DFS stack
    stack.addAll(topLevelQ);
    list: while !stack.isEmpty() AND resultTopK.size() < k do
        if !checkTime(timeLimit) then
            return resultTopK;
        // take the next option
        query = stack.pop();
        path = query.getNextPath();
        // mark the path as explored
        path.markExplored();
        if query.hasMorePaths() then
            stack.push(query);
        if !isValid(path, acceptedOptions, deniedOptions) then
            continue list;
        // create a new query
        query_new = query.addPath(path);
        if checkComplete(query_new) then
            // query already contains all user keywords
            resultTopK.add(query_new);
        else
            // push query to the stack for expansion
            stack.push(query_new);
        return resultTopK;
```

5.6.3 Probability Estimation

The QCOs in the query hierarchy are all the candidates to be presented to the user. According to Section 5.5.2, the most efficient QCO is the QCO with the maximum entropy. To identify this QCO, we need to estimate the probabilities of the QCOs. These probabilities cannot be computed by Equation 5.5, as we do not have the entire interpretation space materialized. Instead, we choose the frontier of the instantiated part of the query hierarchy (i.e., the most specific query-based QCOs that have been materialized so far), and treat all the QCOs in the frontier as samples. For each sample s, we compare this sample against each candidate QCO, say o. The comparison can end up with one of the following conclusions: (1) o subsumes s; (2) s conflicts with o (e.g. one keyword is being interpreted into two different meanings); (3) none of the above. If the conclusion is (3), s does not provide any helpful information. With (1) or (2), we can know that either $\zeta(s) \subset \zeta(o)$, or $\zeta(s) \cap \zeta(o) = \varnothing$, respectively. By applying the maximum likelihood principle, we can estimate the probability $P(o)$ using $P(s)$, that is,

$$P(o) = \frac{\sum_{\zeta(s)\subset\zeta(o)} P(s)}{\sum_{\zeta(s)\subset\zeta(o)} P(s) + \sum_{\zeta(s)\cap\zeta(o)=\varnothing} P(s)}. \tag{5.7}$$

With $P(o)$, we can obtain the entropy of o based on Equation 5.5. Therefore, the problem of finding the QCO with the maximum entropy is reduced to the estimation of the probabilities of (partial) query interpretations $P(s)$. FreeQ applies the probabilistic model introduced previously in Chapter 3, and further developed in Chapter 4 to estimate these probabilities.

5.7 Experimental Evaluation

We have implemented FreeQ, an interactive query construction system based on the mechanisms proposed in this chapter. To evaluate the performance of FreeQ, we conducted extensive experiments. Our experiments include two parts. First, we evaluated the impact of ontology-based QCOs on the efficiency of query construction processes. Then, we evaluated the time efficiency of the proposed algorithms in handling large scale databases.

5.7.1 Experiment Setup

Our experiments were performed on a Freebase dataset downloaded in June 2011 [Goo11]. This dataset contains approximately 7,500 tables with more than 20 million entities in about 100 domains. We imported the dataset into a MySQL database and indexed the data and the schema using Apache Solr[3]. FreeQ was implemented as a

[3]https://lucene.apache.org/solr

client-server Java application. For our experiments, we used two cores and 10GB of memory on a server, which was equipped with 8x Quad-Core AMD Opteron 2.7GHz processors and 256GB of memory.

Our query set was based on the user-defined views of Freebase. Freebase allows users to create views using a dedicated query language called MQL. Each of the views is given a descriptive title in a natural language. Some examples of the view titles are: *"Lionel Richie discography"*, *"Directing Award: U.S. Dramatic - Winning Films"*, and *"TV Celebrities on Twitter"*. For evaluation, we can regard the title of each view as a keyword query and the MQL definition of the view as the structural interpretation. Then we can study how FreeQ can assist users to construct the structural interpretation from the keyword query. The ground truth is a plausible mapping between the keyword query and the structural interpretation and can be automatically established through a projection program. For our experiments, we randomly selected a set of 615 keyword queries (views). The number of keywords in each query ranges from 1 to 8 keywords. The structural interpretations of the keyword queries are of different complexity. We measure the complexity of an interpretation by the number of keyword nodes it contains. (A keyword node is a table containing at least one keyword occurrence.) The complexity of the queries in our query log ranges from 1 to 3. Table 5.2 presents the average complexity of queries with different number of keywords. As we can observe, the relatively complex queries mostly contain 3-5 keywords.

#Keywords	Avg. #nodes	Examples of keyword queries
1	1.00	location, book, event, disease, election
2	1.43	emperor album, hockey team, alpine skier
3	1.92	artist lived vancouver, founding figure kagyu
4	2.40	olympic athletes table tennis
5	2.11	canada hockey 2010 winter olympics
6	1.48	2011 san francisco international film festival
7	1.29	fictional character created by edgar allan poe
8	1.13	school type university of puerto rico at ponce

Table 5.2 Complexity of Keyword Queries

5.7.2 Effectiveness of Ontology-based QCOs

Our first set of experiments was intended to evaluate the effectiveness of the ontology-based QCOs. For each query in our query set, its correct interpretation is already known. Thus, the correctness of each QCO generated by FreeQ can be determined without any user intervention. In our experiments, we let the computer simulate a user and interact with FreeQ automatically.

To evaluate the effectiveness of the ontology-based QCOs, we ran the experiments

in a number of scenarios, in which different ontologies were used. The ontologies are of different size and complexity, as summarized in Table 5.3. *NoOntology* represents the baseline approach presented in Chapter 3, where no ontology is used. *Freebase* represents the ontology based on the domains and categories given in the Freebase website. *YAGO* represents an external ontology known as YAGO. We associated each table of Freebase with a suitable YAGO category by applying instance-based schema matching techniques [RB01]. We describe the mapping in Chapter 6 in more detail.

Notation	Description	Size
NoOntology	no ontology is used, representing the baseline approach in Chapter 3	zero
Freebase	the ontology of Freebase, consisting of domains and categories	medium
YAGO	an external ontology YAGO	big

Table 5.3 Ontologies of Different Size

In the experiments, we measured the interaction cost of query construction for the queries with different complexity in different scenarios. According to Definition 5.4.8, interaction cost is the number of the QCOs a user needs to evaluate to construct a structured query. The results for the 615 test queries are presented in Figure 5.4a.

Figure 5.4a presents the mean and the standard deviation of the interaction cost, with respect to query complexity (the number of keyword nodes) and number of query terms. As we can see, compared to the baseline *NoOntology* approach, the interaction cost can be significantly reduced by using the ontology-based QCOs. With a larger ontology, such as YAGO, the interaction cost tends to be smaller. For example, a 2-node query *"ami suzuki album"* requires 38 QCOs with *NoOntology*, 19 QCOs with *Freebase*, and 10 QCOs with *YAGO*. The benefits of the ontology-based QCOs become stronger with an increasing number of keyword nodes in the structural interpretation too. As we can observe, the majority of the 1-node queries, such as *"university"* and *"football game"*, can still be efficiently answered in the baseline *NoOntology* scenario, where an average cost amounts to 3. With the more complex 2-node, and 3-node queries, such as *"don shirley album", "film performance tom everett", and "location leonardo da vinci lived"*, the interaction cost of *NoOntology* goes up quickly to about 30 and can occasionally exceed 70. By applying the ontology-based QCOs, this cost can be limited to a much smaller range. For example, the average cost for 2-node queries in the *YAGO* scenario is around 10. In addition, we can observe that the interaction cost tends to increase with the number of keywords, while this trend may not hold in all the cases. This is because a longer keyword query does not necessarily imply the more complex structural interpretation (see Table 5.2).

To evaluate the effectiveness of interactive query construction in general, we also compared the interaction cost of query construction against that of query ranking. To

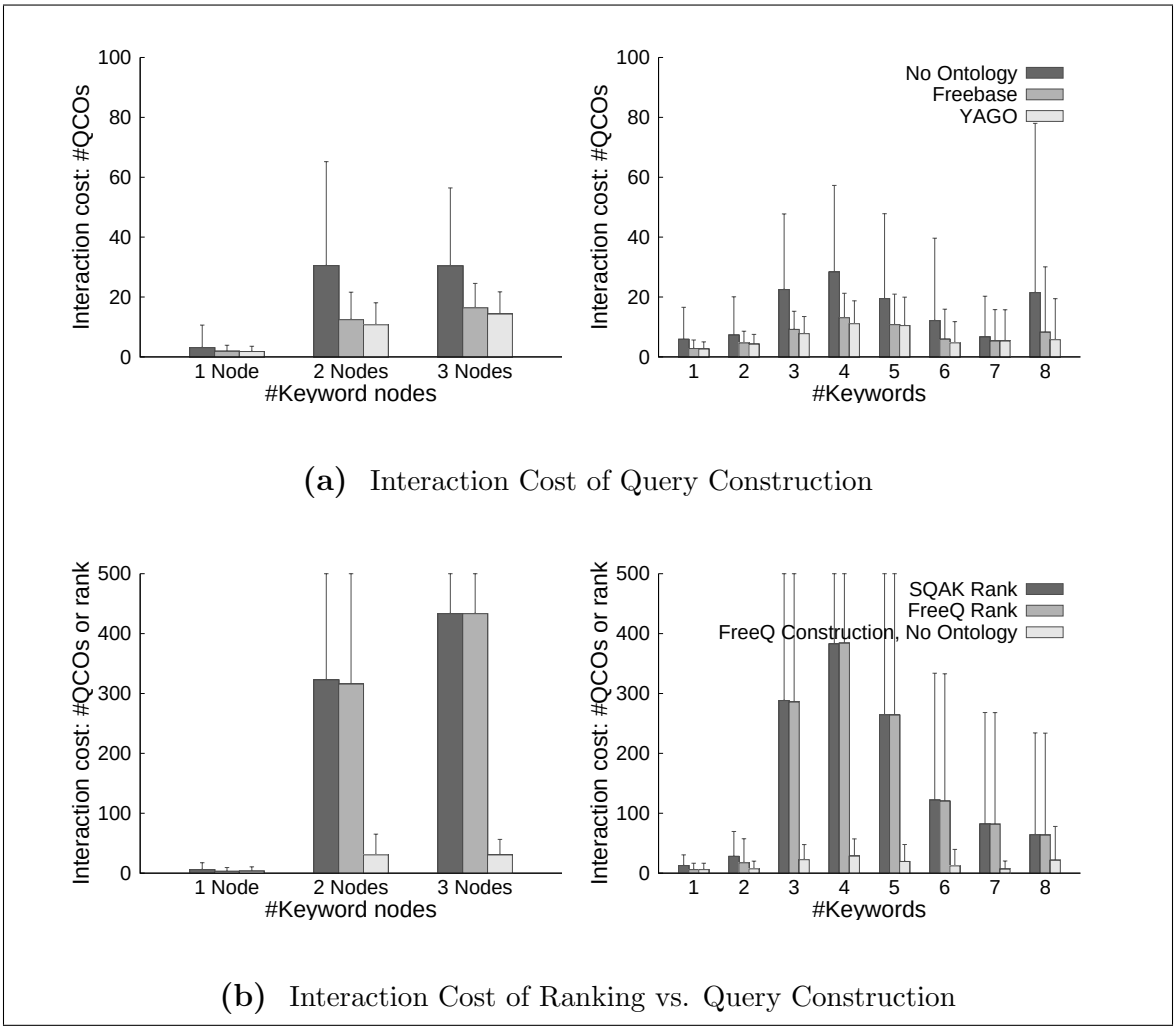
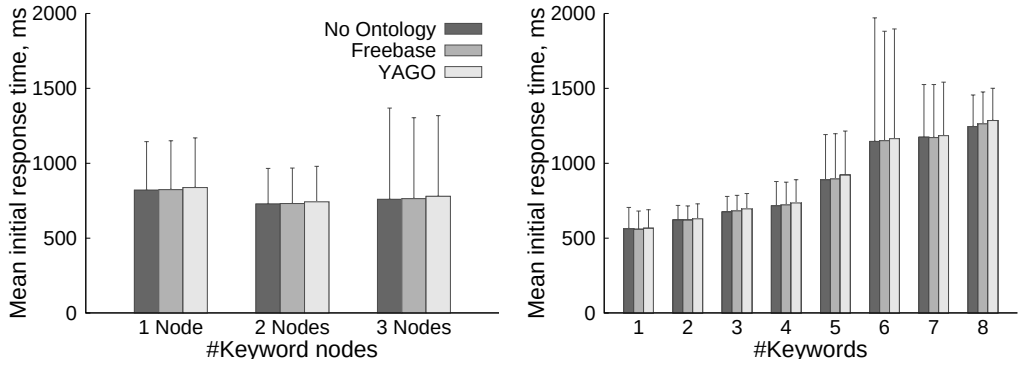


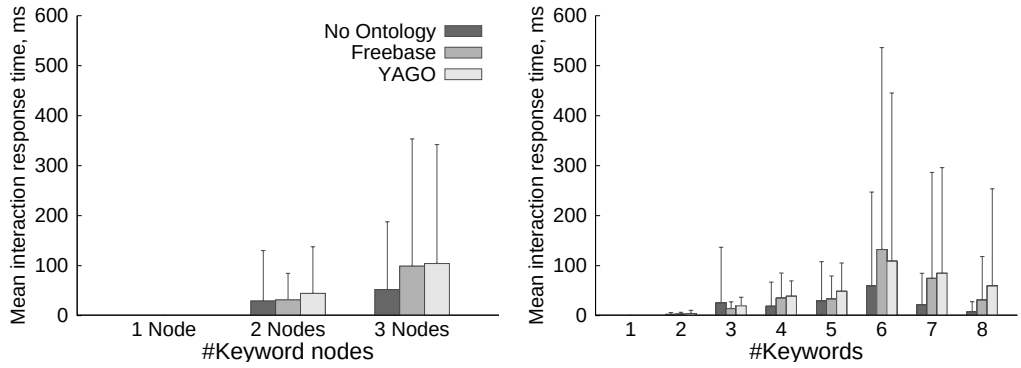
Figure 5.4 Interaction Cost of Query Construction over Freebase.

this end, we generated the top-500 interpretations using the DFS algorithm of FreeQ. We ranked these interpretations using two ranking functions proposed recently: the ranking function of FreeQ described in Chapter 3, and SQAK [TL08]. The interaction cost of ranking corresponds to the number of interpretations a user needs to evaluate before the intended interpretation is identified, which is exactly the rank of the intended interpretation. If the intended interpretation was not contained within the top-500, we set its rank to 500. Figure 5.4b presents the results.

As we can observe, the interaction cost of both ranking and construction for the simplest queries (with 1-2 keywords or 1 keyword node) is acceptable. For the keyword queries that are relatively complex, ranking become ineffective. This is because the query interpretation space on Freebase is too big, such that there can always be a large number of non-intended interpretations that receive good ranks.



(a) Mean Initial Response Time, ms



(b) Mean Interaction Response Time, ms

Figure 5.5 Response Time of Query Construction over Freebase.

5.7.3 Performance of the System

Existing schema-based approaches to keyword search and interactive query construction in databases typically rely on the completely materialized interpretation space of a keyword query [SA02, HP02, DZNonb]. We have tried to run a number of existing algorithms, including [HP02], and our initial algorithm presented in Chapter 3 on Freebase. However, as the schema of Freebase is much bigger than the schemas these algorithms were designed for, they could not finish in a reasonable time. In contrast, the BFS and DFS approach allows FreeQ to explore the query interpretation space smartly. Thus it can achieve reasonable response time on a large scale dataset.

To assess the time efficiency of FreeQ, we measured its response time in two phases: (1) The *initial response time* when a user submits a query, and (2) the *interaction response time* when a user interacts with the query construction panel.

When a user issues a keyword query, FreeQ will perform a series of pre-processing, including the identification of the keyword occurrences in an inverted index and the creation of a bi-directional index for performing BFS and DFS expansions. The mean and the standard deviation of the initial response time for our query set are shown in Figure 5.5a. As we can see, the mean initialization time stays around 1 second, and its maximum does not exceed 2 seconds. With an increasing number of keywords, the initial response time increases slowly. This is because the time required for the identification of keyword occurrences in an inverted index increases with the number of query terms. This index access time ranges from 70 to 700ms for our query set. We can also see that the initialization time does not increase with the increasing query complexity. As presented in Table 5.2, a more complex query does not necessarily contain more query terms. The time required for the creation of a bi-directional index for DFS and BFS ranges from 360 to 460ms for our query set. This time mainly depends on the size and the complexity of the schema graph rather than the complexity of queries. We can also observe that the use of different ontologies has limited impact on the initial response time.

We also measured the *interaction response time*, i.e. the time required by FreeQ to present a new set of QCOs and top-k structured queries after each user interaction. This time comprises the time consumed by the BFS and DFS algorithms and the QCO selection. Figure 5.5b presents the results. As we can observe, the interaction response time of FreeQ varies from 1 to 130ms, meaning that in most cases the user would feel that the system is reacting instantaneously [Nie93]. The interaction response time increases with an increasing query complexity. This is expected. As more keywords imply a larger query hierarchy and more QCOs to evaluate, the BFS and DFS algorithms and the process of QCO selection will all be more time consuming. Nevertheless, this increase does not appear steep in the results. Moreover, a larger ontology seems to incur higher interaction time too. This is because a larger ontology enables FreeQ to generate more QCOs to evaluate.

In summary, the interaction response time of our system is always below one second. Its initialization time stays within one second most of the time, except for some long queries, whose initialization time can take up to 2 seconds. While further optimization can be conducted, such performance can already ensure that the user's flow of thought stays uninterrupted [Nie93].

5.8 Discussion

While approaches to incremental query construction and search result diversification presented in the previous chapters of this thesis work well for databases with small and medium-sized schemas such as IMDB and Lyrics, these approaches require further adaptation to handle large scale datasets efficiently. In this chapter, we considered Problem 3 presented in Chapter 1 of scaling interactive query construction on a very

large multi-domain dataset containing thousands of tables. To achieve this goal, in this chapter we further developed the approach of incremental query construction presented in Chapter 3 taking into account scalability aspects.

Firstly, we analyzed the efficiency of the query construction options in face of a large scale dataset. We found that although the query-based QCOs proposed in Chapter 3 enable an efficient query construction over medium-sized datasets such as IMDB and Lyrics, they alone are not sufficient for a large scale dataset, such as Freebase. Therefore, in this chapter we extended the query hierarchy proposed in Chapter 3 with a novel ontological layer. The QCOs in the ontological layer summarize the database schema and group together keyword occurrences in different database tables. Using ontology, we can create highly informative QCOs and reduce the user interaction cost in a query construction process over large scale datasets significantly.

Secondly, in the query construction approach in Chapter 3, we assumed that it is feasible to materialize the entire query interpretation space efficiently. This assumption is typically applied in the state-of-the-art approaches to schema-based database keyword search (e.g. [SA02, HP02, HGP03, LYMC06]) that normally operate on small and medium-sized schemas. Therefore, in our approach in Chapter 3, we employed a set of pre-generated query templates to generate query interpretations. While this assumption holds for medium-sized datasets such as IMDB and Lyrics, it is not applicable to the datasets with large schemas. In this chapter we extended our approach to query construction with a set of algorithms that incrementally explore interpretation spaces of keyword queries over large scale data. Using these algorithms we can generate the top-k query interpretations and QCOs in the query hierarchy for a large scale database efficiently.

To the best of our knowledge, FreeQ is the first system that enables efficient schema-based keyword search and incremental query construction over large scale datasets. Our experiments have shown that the proposed FreeQ system, which builds upon the work in the previous chapters of this thesis, is effective and efficient for interactive query construction over large scale datasets. Our evaluation results have confirmed the effectiveness of the ontological layer, which we added to the query hierarchy first proposed in Chapter 3, for the complex queries over large scale data. In order to create this ontological layer we made use of different ontologies. Firstly, we created an ontological layer using the native taxonomy of Freebase. Secondly, in our experiments we have demonstrated that external ontologies, such as the YAGO ontology, can be used to further increase the effectiveness of incremental query construction. This is especially important in order to enable portability of the FreeQ system to the databases without predefined ontologies.

6 Combining a Large Scale Database with an Ontology

6.1 Introduction and Motivation

In Chapter 5, we considered the problem of efficient incremental query construction over a large scale dataset such as Freebase. At the time of writing, Freebase [BEP+08] contains about 22 million entities and more than 350 million facts in more than 100 domains. The users of Freebase can collaboratively create, structure and maintain database content over an open platform. In addition, automatic imports from the external data sources such as Wikipedia [Wik], MusicBrainz [Mus], and others enable further growth of the data size. Internally, Freebase data is organized as a relational database with more than 7,500 tables, mostly containing textual data.

Given the scale of Freebase, it becomes crucial to provide effective and efficient structures that give users a quick and informative overview of the data available and provide a backbone for the wide variety of applications such incremental query construction over large scale data discussed in Chapter 5, as well as Linked Data [BHBL09], database schema summarization [YJ06], question answering [ATS+11], and many others. Using these applications users who are not familiar with an internal database schema can efficiently narrow down the search space and retrieve the desired data quickly and accurately.

Ontologies are typically used for organizing large scale information and knowledge in a wide variety of domains. In Chapter 5, we proposed creation of an ontological layer at the top of a large scale database schema in order to organize the data in rich semantic categories and enable scalable and efficient incremental query construction. The YAGO ontology [SKW07] is a lexical resource that contains facts, their relations, and categories automatically extracted from Wikipedia. YAGO unifies the extracted Wikipedia categories with the concepts of the WordNet thesaurus [Fel98] and arranges these concepts into a taxonomic hierarchy. Among other ontologies, YAGO is a

natural choice for organizing Freebase data, as both YAGO and Freebase share a large number of instances originating from Wikipedia.

Figure 6.1 exemplifies the concepts of the Freebase and YAGO hierarchies. For example, in YAGO an instance *Stephen King* is associated with the leaf concepts *"American Novelist"*, *"Writers from Maine"*, and *"People from Country Dublin"* which are the sub-concepts of *"Writer"*, *"Communicator"*, *"Person"*, and *"Entity"*. In Freebase, the same instance *Stephen King* is found in the table *"Author"* located in the *"Books"* domain that is further categorized in the *"Arts & Entertainment"* top-level domain of Freebase.

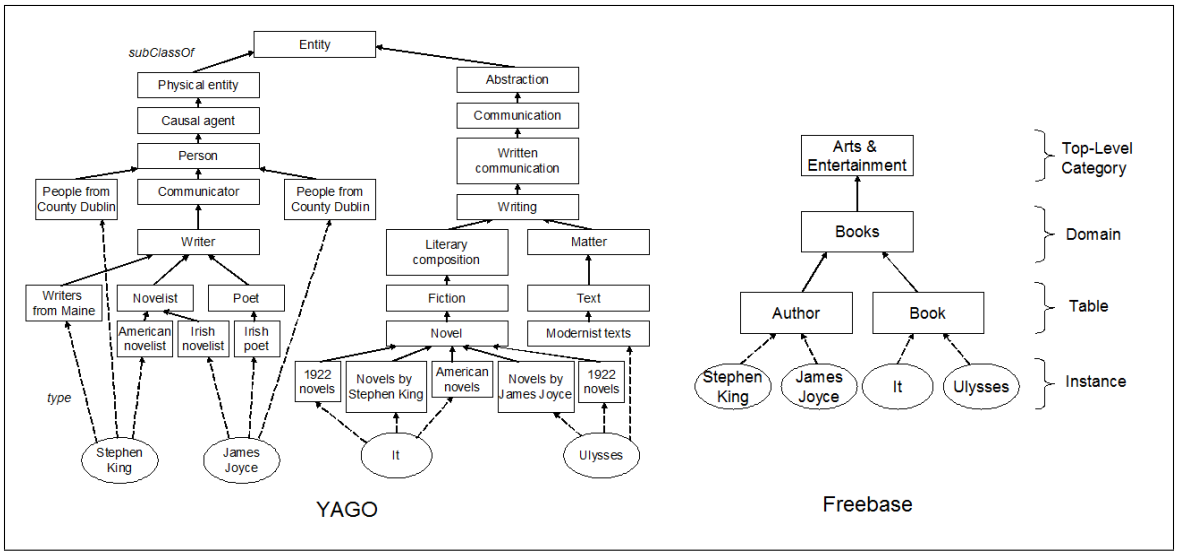


Figure 6.1 Examples of YAGO and Freebase Concepts.

In this chapter, we focus on the Problem 4 of enrichment of a large scale database with the semantic categories at the example of Freebase and YAGO. Our contributions are as follows: First, we analyze the initial structure of the large scale YAGO hierarchy which contains more than 360,000 categories. Second, we provide a matching algorithm, which identifies the most suitable YAGO category for every entity table of Freebase. Third, we compare the structure of the sub-hierarchy of YAGO that is relevant for the Freebase mapping, which we call YAGO+F, with the original YAGO hierarchy and show which part of the YAGO ontology is relevant to describe a large scale real-world dataset like Freebase. Finally, we evaluate and discuss matching quality and make the matching results available to the community[1].

In related work, YAGO and Freebase were brought together in the context of Linked Data [BHBL09]. Linked Data is a method of publishing on the Semantic Web that connects pieces of structured data, information, and knowledge to build the Web of Data. This way of publishing enables data from different sources to be connected and queried. Linked Data already includes Freebase and YAGO, loosely connecting

[1]The YAGO+F mapping is available at: http://iqp.l3s.uni-hannover.de

their shared instances through the DBpedia references. The systematic integration of YAGO and Freebase at the schema level described in this chapter takes an important step further towards tighter integration of the Linked Data, empowering discovery of new relations across the datasets contained in different knowledge bases and enlarging the scope of the possible applications that use the Web of Data.

Approaches to interactive query construction described in Chapters 3 and 5 enable naive users to pose expressive queries to databases starting from simple keywords and incrementally refining the keywords into the intended structured expressions. The efficiency of these approaches highly depends on their ability to pose informative questions to the users to quickly reduce the search space. In a large scale database such as Freebase, the keywords are highly ambiguous such that the questions based on the database schema are not informative enough. As pointed out in Chapter 5, a combination of the Freebase schema with the YAGO ontology can provide a way to create questions with high information gain to optimize the query construction process over a large scale dataset.

A further interesting application of the YAGO+F mapping is database schema summarization. In the context of database schema summarization [YJ06], YAGO ontology can provide a backbone for grouping Freebase tables in semantic classes to create effective schema summaries. The schema of Freebase is big and complex. Expert users who is not familiar with this schema need to perform a lot of work to understand the schema and find the relevant parts to be able to pose structured queries. A schema summary of Freebase that uses the YAGO ontology can provide an overview of the entire schema and enable users to quickly identify and understand the relevant schema parts.

Another kind of applications that can potentially profit from the YAGO+F integration are the question answering systems such as YAGO-QA [ATS+11]. YAGO-QA is a system that enables answering natural language questions using ontological knowledge of YAGO. Thereby the mapping of the user's keywords contained in the questions (SPARQL query expressions) is performed based on how good these keywords map to the YAGO instances. As the instances in YAGO mostly originate from Wikipedia, the scope of the possible questions in YAGO-QA is also limited to the Wikipedia instances. Combination of YAGO and Freebase can enrich YAGO concepts with the Freebase instances that come from many different sources, thus extending the scope of the YAGO-QA system significantly.

The rest of this chapter is organized as follows: We summarize the contributions of this chapter in Section 6.2. Then, in Section 6.3 we provide a brief overview of the specific background from the area of schema matching. Following that, in Section 6.4 we perform a detailed analysis of the concept and instance distribution in the YAGO ontology. Then, in Section 6.5, we present matching techniques used to align YAGO and Freebase. Section 6.6 analyzes the concept and instance distribution in the resulting YAGO+F structure and provides an evaluation of the matching quality. Finally, in Section 6.7 we discuss the results.

6.2 Summary of YAGO+F Contributions

In this chapter we consider the problem of enrichment of the Freebase dataset with the semantic categories of the YAGO ontology addressing Problem 4 presented in Chapter 1. Freebase and YAGO datasets build a part of recently emerged Linked Data [BHBL09] that loosely connects shared instances of the both datasets through the DBpedia references. To the best of our knowledge, in this thesis we perform the first attempt to interconnect these datasets systematically at the schema level.

First of all, in this chapter we perform a detailed analysis of the concept and instance distribution in the YAGO ontology. Then, we perform matching using state-of-the-art instance-based matching techniques [DMDH02, BN05]. Following that, we evaluate matching quality and describe and characterize the resulting YAGO+F hierarchy.

In summary, the YAGO+F contributions described in this chapter include:

- We perform a detailed analysis of the concept and instance distribution in the large scale YAGO ontology.
- We perform instance-based matching of two multi-domain large scale datasets such as YAGO and Freebase and evaluate matching quality.
- We describe and characterize the resulting YAGO+F hierarchy and make it available to the community.
- Our experiments in Chapter 5 have shown that using the YAGO+F dataset resulting from the matching, we increase efficiency of incremental query construction over large scale Freebase data.
- YAGO+F also provides a further important step towards interconnection of the Linked Data subcollections [BHBL09], and will hopefully enable many future applications that can profit from a wide variety of Freebase data clearly arranged into the semantic categories of YAGO.

The results of our experiments confirmed the good quality of the matching in cases where the YAGO and Freebase category structures are compatible, but also have shown some incompatibilities between the category schemas of YAGO and Freebase. In future work we can investigate how to improve matching by introducing new categories that incorporate both YAGO and Freebase schema information into a richer ontology, and potentially improve both the YAGO hierarchy as well as the Freebase schema information.

6.3 Specific Background

To perform the mapping between the Freebase dataset and the YAGO ontology we made use of the state-of-the-art techniques for schema matching.

Schema matching plays an important role in the context of relational databases (see e.g. survey of Rahm et.al. [RB01]), as well as in the context of XML (e.g. [DR02]) and ontologies on the Semantic Web [DMDH02]. The aim of the schema matching in relational databases is to identify the most similar matching element(s) in a flat relational schema. In addition, malleable schemas enable more flexibility in schema matching by relaxing definitions of attributes or relationships [ZGBN07]. In contrast, the systems for XML and ontology matching are required to identify the most-specific-parent or the most-general-child within the relevant branch of the hierarchy [DMDH02]. PARIS [SAS11] quantifies the probability of whether the classes of two ontologies are in a subclass relation. Schema matching techniques in all the domains mentioned above make use of similar features, such as similarities on the element names, instance overlap [DMDH02, BN05] and schema structures [DJMS02].

Instance-based matching techniques assess similarity of the concepts based on the instances these concepts contain [DMDH02], [BN05]. The instances like a *film*, a *location*, or a *person* often coincide across ontologies. Given a set of the corresponding instances, the similarity of the concepts can be measured as the instance overlap using e.g. the Jaccard coefficient.

Element-based matching techiques analyze the similarity of the names of the concepts. This analysis can include language-based techniques such as stop-word removal, and stemming of the concept names [DR02], as well as the computation of the string similarity between the concept names. To compute the string similarities, various measures such as Levenstein distance, n-gram-based similarity, cosine similarity, and others can be applied [CRF03, Kon05]. Element-based matching can also take into account synonyms that can be obtained from external linguistic resources (e.g. WordNet) [Pat06].

Structure-based matching techniques analyze relationships within the ontologies to find the correspondences between the ontology elements. Graph-based techniques (such as [EV04]) view ontologies as graphs and analyze the similarity in the neighborhood of the nodes to find the matches. Taxonomy-based techniques such as [LDKG04, ES04] rely on the is-a hierarchy to determine the matches. The similarity of the concepts is then established based on the similarity of the super-concepts, and sub-concepts.

Dependent on the scenario, the instance-based, element-based, and structure-based matching techniques described above can be used either in isolation, or they can be combined to achieve a higher matching precision. For instance, COMA [DR02] - a system for XML schema matching uses a combination of name similarities of the schema elements, path similarities within the XML tree, as well as the string similarity of the data contained in the leaf nodes. The GLUE system for the Semantic

Web [DMDH02] performs ontology matching by computing the joint probability distribution between the concepts, using the instances shared across these concepts.

6.4 Concepts and Instances in YAGO

To enable an effective matching between YAGO and Freebase, we perform several steps. In this section, we first analyze the distribution of the categories and the instances within the initial YAGO hierarchy. Then, we consider an overlap of the YAGO and Freebase instances and examine the distribution of the instances shared between YAGO and Freebase within the initial YAGO hierarchy.

6.4.1 Concept Structure of YAGO

The common elements of YAGO ontology are the concepts, e.g. *"Entity"* and *"Person"*. The concepts represent semantic categories and are hierarchically organized using the *"subClassOf"* relation. A concept can be associated with a set of instances, e.g. the concept *"Person"* is associated with *"Stephen King"* and *"James Joyce"*. The relation *"type"* links together a concept with its associated instances.

To facilitate our analysis, we assign each category within the YAGO hierarchy a *depth* value. The categories at the top level of the hierarchy (depth=0) do not possess any parent categories. Then, the depth of the category C is determined as the length of the path from C to the top level category that is associated with C using the *"subClassOf"* relation.

A YAGO instance can be associated with multiple YAGO categories connected using the *"subClassOf"* relation. In order to differentiate the most specific categories that have instances directly assigned to them from the categories that are only indirectly connected to the instances, we introduce the notion of *Leaf Category*. A *Leaf Category* L is a category that is associated with an instance I and has the highest depth value across all the categories associated with I and connected to L with the *"subClassOf"* relation. For example, an instance *"Alexander the Great"* is associated with the Wikipedia leaf categories *"4th-century BC Greek people"*, *"Macedonian monarchs"*, *"Ancient Macedonian generals"*, *"Monarchs of Persia"*, which are connected to the more general WordNet categories *"Person"*, *"Head of state"*, *and "General"*.

As of March 2011 [fI11], YAGO possesses 361,211 semantic categories that are organized in a hierarchical structure with 20 levels. The backbone of YAGO is build from the Wikipedia and WordNet categories. YAGO includes 292,070 Wikipedia categories located at the depth of 1-19 of the hierarchy. These categories can be very specific and include e.g. *"Burials at Kensico Cemetery"*, *"1729 essays"*, *"Multidirectional shooters"*, *"Burials at Montmartre Cemetery"*, *"Paris"*, *"Founders of utopian communities"*, *"59 crimes"*, *and "1st-century executions"*. Further, YAGO includes

68,446 more general WordNet categories such as *"Parent", "Life", "City university", "Clip art", "Logic diagram", "Routine", and "Call"*. The WordNet categories are spread over the depth 0-19 of the YAGO hierarchy. In addition, 642 Geo categories such as *"Water mill", "Copper mine", "Phosphate works", "Factory", and "Research institute"*, are located at the depth of 1-13. Finally, YAGO offers a set of 53 its own categories located at the depth of 0-5, e.g. *"Length", "Number", "Weight", "ISBN", and "Monetary value"*.

From the total number of 361,211 categories in the YAGO hierarchy, about 80% (288,569 categories) are the leaf categories that have instances directly assigned to them. For example, an instance *"San Francisco"* is directly assigned to the following Wikipedia categories: *"Populated places established in 1776", "County seats in California", "Populated coastal places in California", and "California counties"*. 7,394 categories do not have instances as direct children. Among them are: *"Accident", "Action", "Young person", "Legal actor", "Organism", "Motor", "College", and "Comedian"*. Finally, 65,248 categories do not possess any instances, neither direct nor indirect. These YAGO categories include: *"Length", "Number", "Weight", "ISBN", and "Monetary value"*. Table 6.1 presents the distribution of the categories at each level of the YAGO hierarchy. As Table 6.1 illustrates, 60% of all YAGO categories are assigned to the depth 6-8 of the YAGO hierarchy. 90% of the categories are located within the depth of 4-10.

The categories that do not possess any parent categories are located at the top level of the YAGO hierarchy (depth=0). The most important YAGO category at the top level is the WordNet *"Entity"* category. This category is a parent of the majority of the categories in YAGO. However, the YAGO hierarchy does not form a clean tree structure. This is mostly because some of the WordNet categories in YAGO are not related to the *"Entity"* category and do not possess any parent categories. These categories include: *"Administrative district", "Brahman", "Control character", "Epacris", "Evangelicalism", "Exorcist", "Faun", "Fen", "Fundamentalist"*, and others. Finally, the YAGO category *"Relation"* is an additional YAGO category that does not possess any parents. This explains an unusually big number of categories and instances at the depth one, as many of the categories at this depth are the child categories of the WordNet categories located at the top level of the hierarchy. For example, the top level WordNet category *"Administrative district"* has 141 subordinated categories, among them are the Wikipedia categories: *"Settlements in New Brunswick", "Neolithic settlements in Crete", and "Settlements established in 1312"*. This also leads to a big number of associated instances already at the second level of the YAGO hierarchy.

Table 6.1 Distribution of Categories and Leaf Categories in YAGO

Depth	Categories in YAGO						Leaf Categories in YAGO				
	Total	Total, %	WordNet	Wikipedia	YAGO	Geo	Total	WordNet	Wikipedia	YAGO	Geo
0	25	0.00007	24	0	1	0	2	1	0	1	0
1	221	0.00061	16	180	22	3	123	0	113	10	0
2	79	0.00022	35	10	6	28	1	0	1	0	0
3	3,726	0.01032	419	3,286	17	4	3,293	19	3,271	1	2
4	42,979	0.11899	2,465	40,390	5	119	40,262	60	40,134	0	68
5	27,635	0.07651	5,926	21,561	1	147	21,586	165	21,333	0	88
6	67,928	0.18806	9,819	57,950	0	159	57,732	298	57,352	0	82
7	91,455	0.25319	14,954	76,388	0	113	75,234	317	74,857	0	60
8	63,015	0.17445	11,171	51,802	0	42	51,460	154	51,283	0	23
9	28,901	0.08001	8,006	20,880	0	15	19,912	72	19,835	0	5
10	16,115	0.04461	6,257	9,849	0	9	9,498	33	9,461	0	4
11	8,520	0.02359	3,953	4,565	0	2	4,520	9	4,510	0	1
12	4,603	0.01274	2,349	2,253	0	1	2,192	4	2,188	0	0
13	3,294	0.00912	1,270	2,023	0	1	1,910	0	1,909	0	1
14	1,443	0.00399	761	682	0	0	620	0	620	0	0
15	571	0.00158	459	112	0	0	102	0	102	0	0
16	461	0.00128	375	86	0	0	72	0	72	0	0
17	191	0.00053	163	28	0	0	25	0	25	0	0
18	47	0.00013	23	24	0	0	24	0	24	0	0
19	2	0.00001	1	1	0	0	1	0	1	0	0
Total	361,211	1	68,446	292,070	52	643	288,569	1,132	287,091	12	334

6.4.2 Instance Distribution in YAGO

In total, 2,632,948 unique instances of YAGO are assigned to the 295,963 categories using the *"type"* relation. Majority of the YAGO instances (2,632,756) originate from Wikipedia. As YAGO allows an instance to be assigned to the multiple categories in the hierarchy, the total number of instances located at the leaves of the YAGO hierarchy is 2.76 times higher than the number of unique instances (7,255,584 instances in total). For example, an instance "Stephen King" is assigned to the multiple categories such as *"American school teachers"*, *"People from Portland, Maine"*, *"Film director"*, and *"American horror writers"*.

As presented in Table 6.2, the most populated level of the YAGO hierarchy with respect to the instance distribution is located at the depth 7 and contains 29% of the instances. The levels 6-8 together contain 64% of the instances. The majority of the instances (90%) are located within the depth 4-9 of the YAGO hierarchy.

In order to better understand how good the YAGO ontology structure fits to the Freebase dataset, we analyze how the instances shared by the both datasets are distributed across the different levels of the YAGO hierarchy. Table 6.2 presents the distribution of the shared instances of YAGO and Freebase in the YAGO hierarchy. In total, 99% of YAGO instances are shared with Freebase. In Table 6.2, "YAGO Leaf Cat." is the number of the leaf categories of YAGO that contain shared instances. The majority of the shared instances (94%) are located within the depth 3-10 of the YAGO hierarchy. In total these instances are assigned to the 260,488 leaf categories of YAGO.

In Table 6.2 "Freebase Cat." is the number of Freebase categories that are associated with the shared instances found at a certain depth of the YAGO hierarchy. At this point we do not yet perform the matching, such that each Freebase table can be associated with multiple levels of the YAGO hierarchy. For this reason, the total number of the Freebase categories presented in Table 6.2 (8,745) is much higher than the number of Freebase tables containing the shared instances (1,391). This is because the instances of one Freebase table are, on average, distributed across 6.3 YAGO categories.

Table 6.2 Distribution of Instances in YAGO

Depth	All Instances in YAGO						Shared Instances	
	Total	Total,%	WordNet	Wikipedia	YAGO	Geo	YAGO Leaf Cat.	Freebase Cat.
0	32	0.00001	3	0	29	0	0	0
1	31,140	0.00429	0	895	30,245	0	106	225
2	5	0.00001	0	5	0	0	1	4
3	124,679	0.01718	26,039	98,594	37	9	2,752	549
4	1,251,280	0.17246	4,450	1,234,315	0	12,515	37,735	833
5	402,104	0.05542	11,249	380,542	0	10,313	20,045	947
6	1,561,758	0.21525	26,181	1,529,220	0	6,357	51,969	979
7	2,152,886	0.29672	12,819	2,128,606	0	11,461	68,706	1,129
8	1,051,682	0.14495	4,985	940,515	0	106,182	45,404	1,033
9	305,063	0.04205	2,712	302,314	0	37	17,888	861
10	200,300	0.02761	85	200,173	0	42	8,555	769
11	85,298	0.01176	58	85,232	0	8	3,511	477
12	50,678	0.00698	92	50,586	0	0	1,852	336
13	27,193	0.00375	0	27,192	0	1	1,309	191
14	8,470	0.00117	0	8,470	0	0	447	156
15	1,451	0.00020	0	1,451	0	0	95	82
16	862	0.00012	0	862	0	0	67	82
17	536	0.00007	0	536	0	0	22	53
18	155	0.00002	0	155	0	0	23	28
19	12	0.00001	0	12	0	0	1	11
Total	7,255,584	1	88,673	6,989,675	30,311	146,925	260,488	8,745

6.4.3 Instance-Based Overlap between YAGO and Freebase

We view a database table of Freebase as a concept. Each Freebase concept is represented by the set of instances contained in this table. The aim of the matching function is to map each Freebase concept to the most similar concept of YAGO. To assess similarity of the concepts, matching techniques often make use of the instances these concepts share [DMDH02, BN05]. In order to create an effective matching function for YAGO and Freebase concepts, we first analyze the instance overlap.

Freebase data dump used in our experiments [Goo11] contains 1,578 entity tables. On the one hand, we observed that 88% (1,391) of the Freebase entity tables contain Wikipedia instances that are also found in the YAGO ontology. For example, *"Stephen I of Hungary"* from Freebase and *"King Stephen"* from YAGO denote one and the same person, as they share one and the same Wikipedia identifier. On the other hand, as majority of the YAGO instances come from Wikipedia, these instances are also contained in the Freebase dataset. The instances coming from Wikipedia are uniquely distinguished by their Wikipedia identifiers in the both datasets and can be directly used for the concept matching.

Each Freebase concept is associated with a set of instances that can be partly shared with YAGO. Freebase dataset contains 22,542,665 unique instances, from which about 16% (3,661,329 instances) originate from Wikipedia. As an instance can be associated with multiple Freebase categories, for instance *"Stephen King"* with *"Author", "Award winner", "Fictional character creator", and "Pet owner"*, the overall number of the Freebase instances is 2.47 times higher than the number of the unique instances (55,684,113 instances in total). In total, 2,632,756 unique instances that originate from Wikipedia are shared between the YAGO and Freebase datasets, which is about 12% of the Freebase instances. The pie chart presented in Figure 6.2 illustrates the number of the Freebase concepts associated with a given percentage of the shared instances. For example, we can see that for 72% (1150 out of 1578) Freebase concepts, more than 20% of the associated instances are within the shared set. The Freebase tables with the highest instance overlap (80-100%) include: *"Astronomy asteroid", "Baseball coach", "Chess player", "Film critic", "Geography mountain", "US president", "Olympic games", "Religion Monastery", "Royalty kingdom", "Sports boxer", and "Theater actor"*. For these concepts, given a compatible YAGO concept structure, an instance-based matching should already provide good results.

Another part of the Freebase concepts (about 16%) is associated with less than 20% shared instances. These are, for instance, *"Astronomy comet", "Business location", "Business industry", "Business job title", "Film editor", "Food drinking establishment", and "Public library"*. In these cases, an additional evidence can be necessary to perform an effective matching. Finally, 12% of the Freebase concepts are not associated with any shared instances. These are, for example, *"Luminous flux unit", "Astronomy galaxy classification code", "Book technical report", and "Law US patent type"*. In order to include these tables in the mapping, an extension of the

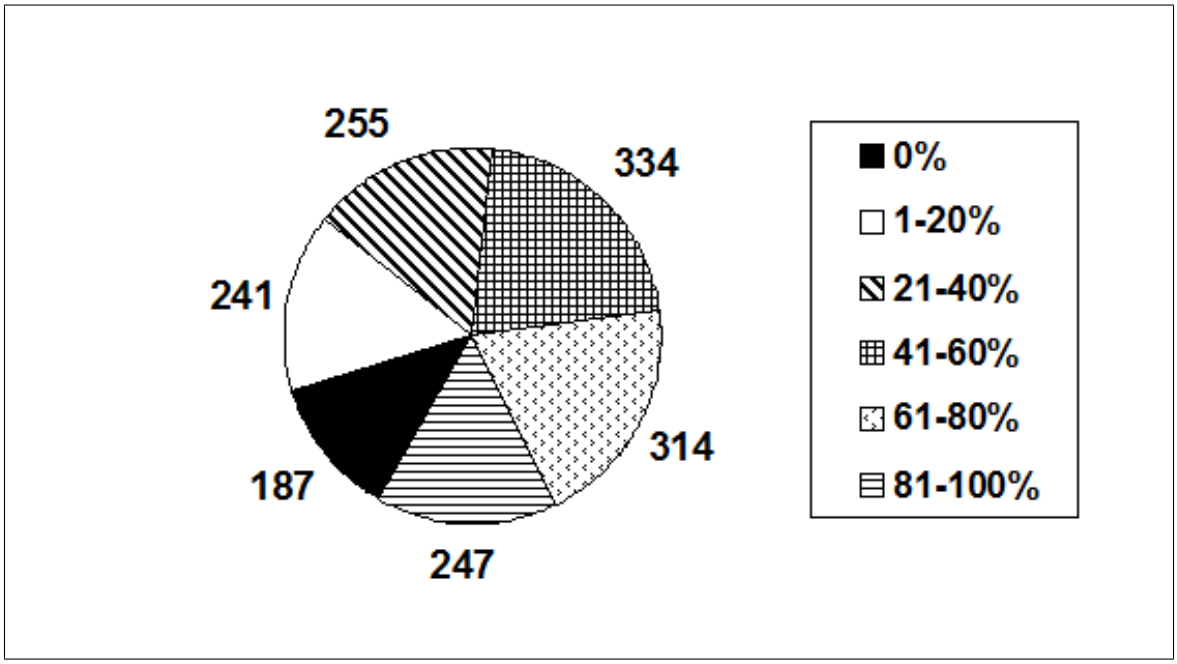


Figure 6.2 Distribution of Shared Instances in Freebase: Number of Freebase Concepts with a Given Percentage of Shared Instances.

YAGO concept structure might be required.

6.5 Matching YAGO and Freebase

As Freebase and YAGO share a significant number of instances coming from Wikipedia as discussed in Section 6.4.3, instance-based matching techniques appear to be the most suitable to align these ontologies. *Instance-based matching techniques* assess similarity of the concepts based on the instances these concepts share [DMDH02], [BN05]. The instances like a film, a car, or a person often coincide across ontologies. Given a set of the corresponding instances, the similarity of the concepts can be measured as the instance overlap using e.g. Jaccard coefficient.

Instance-based matching will enable us to match about 90% of the Freebase entity tables. In the future, we plan to investigate adding further matching techniques such as element-based matching, and structure similarity [ES07] to further increase the number of the Freebase tables that can be mapped to the YAGO ontology and to incrementally improve the quality of the matching.

Matching Design

Freebase dataset is a collection of entity tables, that describe the objects of the real world, like *"Person"*, *"Book"*, *"Location"*, *"Airline"*, and *"Award"*, as well as the

facts associated with these entities. For the matching between the YAGO categories and Freebase entity tables we treat an entity table of Freebase as a concept, and the data entries in this table as instances associated with this concept. For example, the table *"Author"* is a concept that is associated with the instances such as *"Stephen King"* and *"James Joyce"*.

In the matching process, we automatically assign each Freebase entity table to the most similar YAGO category. For example, we assign the Freebase table *"Author"* to the YAGO category *"Writer"*. An example of the matching between YAGO and Freebase concepts is illustrated in Figure 6.3. The output of the matching process is a

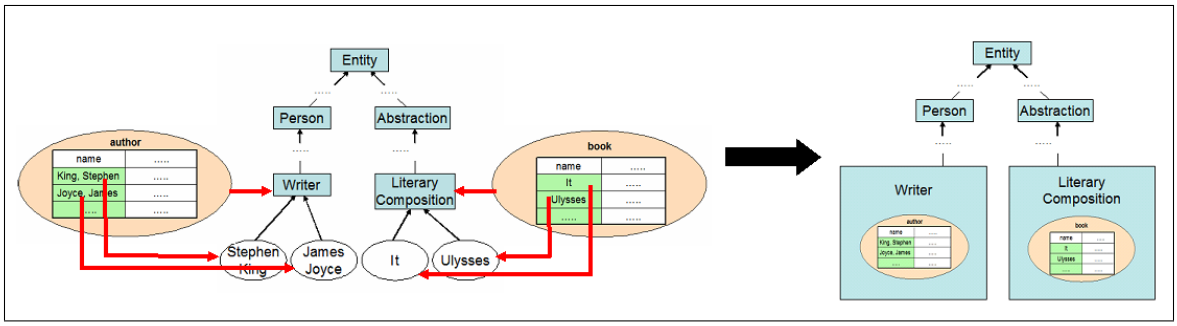


Figure 6.3 Matching YAGO and Freebase Concepts.

mapping of each Freebase table to the most likely semantic category of YAGO. Each Freebase concept (table) is mapped only to the most likely YAGO concept, whereas a YAGO concept can be associated with several Freebase concepts. For example, the YAGO concept *"Writer"* may unify the Freebase concepts *"Author"* and *"Comic book author"*.

Similarity Score Computation

The intuition behind this approach is that the concepts F and Y are same if F contains the same instances as Y. The more instances are shared by F and Y, the more similar the concepts are. For example, *"Writer"* and *"Author"* share the instances *"Stephen King"* and *"James Joyce"* and are considered to describe the same concept, whereas the concepts *"Writer"* and *"Literary composition"* do not have any instances in common and thus are not similar.

For the calculation of the similarity between the two sets of instances we use the similarity measure known as the Jaccard coefficient, which is based on the joint probability. We define the instance-based similarity of two concepts $\alpha_i(Y, F)$ as:

$$\alpha_i(F, Y) = \frac{P(F, Y)}{P(F, Y) + P(\bar{F}, Y) + P(F, \bar{Y})}, \tag{6.1}$$

where $P(F, Y)$ is the part of the shared instances that belongs to F and Y,

$P(\bar{F}, Y)$ is the fraction that belongs to Y but not to F, and $P(F, \bar{Y})$ - is the fraction that belongs to F but not to Y.

$\alpha_i(F, Y)$ has the lowest value 0 when the instance sets of F and Y are disjoint, e.g. *"Literary Composition"* and *"Author"*, and the highest value 1 when F and Y contain the same set of instances and thus represent one and the same concept, e.g. *"Writer"* and *"Author"*.

The overall matching function is then computed as:

$$Y_{map}(F) = argmax_{y \in Y_C} \ \alpha_i(F, y). \tag{6.2}$$

This function assigns each Freebase table F to the most likely YAGO concept Y_{map} that has the highest similarity score from all YAGO concepts Y_C according to the scoring function in Equation 6.1.

6.6 Describing and Characterizing the YAGO+F Hierarchy

Using the techniques described in the previous sections, we matched Freebase dataset from June 2011 [Goo11] with the YAGO2 ontology [fI11]. The Freebase dataset includes approximately 1,578 entity tables containing more than 20 million entities in about 100 domains. As described above, 88% (1,391) of the Freebase entity tables contain Wikipedia instances that are also found in the YAGO ontology. The hierarchy of YAGO2 possesses 361,211 categories, from which more than 80% have associated instances. In total, YAGO contains 2,632,948 unique instances, most of which are shared with Freebase. The results of our matching are available for download in the .tsv and .n3 formats from http://iqp.l3s.uni-hannover.de. In this section, we analyze the structure of the YAGO+F hierarchy that results from the matching and evaluate the matching quality.

6.6.1 The Concepts and the Instances in the YAGO+F Hierarchy

In this section we discuss the results of the matching between YAGO and Freebase. Specifically, we are interested in the part of the initial YAGO ontology, which is relevant to the real-world large scale dataset such as Freebase. To this end, we analyze the YAGO+F hierarchy which is obtained using the matching described in Section 6 and connects Freebase and YAGO concepts.

We used the matching technique described in Section 6.5 to assign each Freebase table to the corresponding YAGO category. We call the YAGO categories directly matched to the Freebase tables *YAGO+F leaf categories*. Then, we extracted the

sub-structure of YAGO required to describe the leaf categories matched to the Freebase dataset. To this end, we extracted all paths from the top level of the YAGO hierarchy to all the YAGO+F leaf categories. We call the resulting sub-structure of YAGO *YAGO+F*. The structure of YAGO+F is presented in Table 6.3.

In Table 6.3, "Total YAGO+F Categories" is the total number of categories that are relevant for the matching and "Leaf YAGO+F Categories" is the number of the leaf categories directly matched to the Freebase tables. The number of the leaf categories in presented in Table 6.3 is 1.12 times smaller than the total number of the Freebase categories. This is because a YAGO category can group several Freebase categories together.

We observed that the grouping of the Freebase tables to one YAGO category mostly happens if the Freebase category structure has a higher granularity than the YAGO category structure. In this case, the Freebase categories mapped to one YAGO category can be either sibling categories or sub-categories of each other. For example, the sibling categories *"Film actor"* and *"TV actor"* of Freebase are both mapped to the WordNet *"Actor"* category of YAGO. Also, *"Location statistical region"* and *"Location citytown"* are mapped to the WordNet *"Geographical area"*. In the latter case, 99,9% instances of the Freebase category *"Music song writer"* are also contained in the Freebase category *"Music lyricist"*, such that *"Music song writer"* is a sub-category of *"Music lyricist"*. The both categories are mapped to the WordNet *"Song writer"* category of YAGO. Also, *"Tennis player"* and *"Tennis tournament champion"* are both mapped to the WordNet *"Tennis player"*, as YAGO does not possess the more specific *"Tennis tournament champion"* category. In these cases, the more specific Freebase categories can represent a possible extension to the YAGO category structure.

Then, Table 6.3 presents the number of Freebase tables assigned to each level of the YAGO+F hierarchy. The majority of the tables is assigned to the levels 4-8, which corresponds to the distribution of the shared instances in the YAGO hierarchy. Finally, Table 6.3 presents the number of the shared instances and the total number of instances associated with the Freebase categories located at the specific level of YAGO+F. The majority of the Freebase instances (50%) is assigned to the levels 3-8. As an instance can be associated with multiple categories of Freebase, the total number of the instances presented in Table 6.3 includes duplicates.

Comparing the YAGO+F structure in Table 6.3 with the original YAGO hierarchy presented in Table 6.1, we can see that the resulting sub-structure of YAGO only contains 0.4% of the leaf categories compared to the original YAGO ontology that contained 288,569 leaf categories. This is expected, as each of the 1,391 Freebase categories was assigned to only one specific YAGO leaf category. The total number of categories in the YAGO+F hierarchy is 2,141, which is only 0.6% of the total number of the all YAGO categories (361,211). As we can see, only a small proportion of YAGO categories (less than 1%) is enough to describe a large scale database such as Freebase.

The total number of instances in the YAGO+F mapping is smaller than the total number of Freebase instances. This is because 187 tables containing 3,172,694 instances (5,7% of the overall number of the Freebase instances) are not assigned to any YAGO category as they do not share any instances with YAGO.

We can see that 80% of the Freebase tables and 50% of the instances are assigned to the levels 3-8 of the original YAGO hierarchy. This distribution roughly corresponds to the distribution of the categories in the original YAGO hierarchy, where the most populated levels were located at the depth 4-9 (see Table 6.2). We also observe a high instance concentration at the depth 0 of the YAGO+F hierarchy. This is because a general and the most populated Freebase table *"Common topic"*, that contains information about entities and their relations from the wide variety of Freebase domains is assigned to the top-level *"Entity"* category of YAGO.

The levels at the depth higher than 14 did not get any Freebase tables assigned. These is because the granularity of the YAGO categories at these levels is too high compared with the Freebase structure.

Table 6.3 Distribution of the Categories and Instances in YAGO+F after the Matching

Depth	**Categories YAGO+F**				**Freebase**		**Instances YAGO+F**			
	Total	Total, %	Leaf	Leaf, %	Cat.	Cat., %	Shared	Shared, %	Freebase	Freebase, %
0	1	0.0005	1	0.0008	1	0.0007	1,967,590	0.3075	22,577,777	0.4300
1	4	0.0019	1	0.0008	1	0.0007	463,039	0.0724	1,378,280	0.0262
2	19	0.0089	1	0.0008	2	0.0014	70,738	0.0111	100,409	0.0019
3	85	0.0397	23	0.0186	30	0.0216	1,113,000	0.1739	2,644,774	0.0504
4	235	0.1098	92	0.0746	106	0.0762	1,100,473	0.1720	2,748,704	0.0523
5	370	0.1728	170	0.1378	191	0.1373	472986	0.0739	4,155,620	0.0791
6	414	0.1934	232	0.1880	255	0.1833	698,365	0.1091	2,974,928	0.0567
7	469	0.2191	332	0.2690	379	0.2725	370,462	0.0579	13,606,038	0.2591
8	271	0.1266	182	0.1475	201	0.1445	74,510	0.0116	2,029,859	0.0387
9	137	0.0640	97	0.0786	110	0.0791	41,842	0.0065	203,555	0.0039
10	92	0.0430	72	0.0583	82	0.0590	20,597	0.0032	76,981	0.0015
11	30	0.0140	21	0.0170	22	0.0158	5,006	0.0008	12,085	0.0002
12	10	0.0047	7	0.0057	7	0.0050	549	0.0001	1,309	2.E-05
13	3	0.0014	2	0.0016	2	0.0014	93	1.E-05	1007	2.E-05
14	1	0.0005	1	0.0008	2	0.0014	10	2.E-06	93	2.E-06
Total	2,141	1	1,234	1	1,391	1	6,399,260	1	52,511,419	1

6.6.2 Matching Quality

In order to assess the matching quality, we compute the Rand Index (RI) [MRS08] and the Jaccard Index (JI), that are the standard measures to evaluate the quality of clustering algorithms. Both RI and JI are the measures of similarity between two data clusterings. In the context of our matching, we compute RI and JI for each YAGO+F leaf category to assess the similarity between the original classification of the instances in Freebase with the YAGO+F classification that we obtain in the matching process.

To this end, we view the mapping of Freebase and YAGO categories as a series of decisions, one for each of the $N(N-1)/2$ pairs of shared instances in the both datasets. We want to assign two instances to one and the same YAGO+F category if and only if these instances belong to the same Freebase table. A *true positive decision* $TP(y, f)$ assigns two instances from one Freebase table f to one YAGO+F category y. As in the praxis multiple Freebase tables can be mapped to one YAGO+F category, to compute the Rand Index $RI(y)$ of the YAGO+F leaf category y, we take into account all Freebase tables $f \in F_y$ mapped to y. The number of true positive decisions is $TP(y, F_y)$ where y is the YAGO+F leaf category and F_y is a set of the Freebase tables mapped to y in the matching process. Further, a *true negative decision* $TN(y, F_y)$ does not assign two instances from the different Freebase tables to one YAGO+F leaf category y.

In the matching, two types of errors can occur. A *false positive decision* $FP(y)$ assigns two instances from the different Freebase tables to one YAGO+F category y. This happens whenever more than one table is assigned to a YAGO+F category in the matching process. A *false negative decision* $FN(F_y)$ assigns two instances from a Freebase table $f \in F_y$ to the different YAGO+F categories. The Rand Index for a YAGO+F leaf category y measures the percentage of decisions that are correct for this category (i.e. accuracy), that is:

$$RI(y) = \frac{TP(y, F_y) + TN(y, F_y)}{TP(y, F_y) + FP(y) + FN(F_y) + TN(y, F_y)}. \tag{6.3}$$

In case a Freebase category contains only one instance, the value of RI may be not well-defined, as the number of instance pairs in such table is zero. To this end, we apply Laplace smoothing and add one to the number of shared instances in each Freebase table.

The RI values range from $[0, 1]$, where $RI = 1$ is the best possible value. This value can be achieved in a perfect situation, where one Freebase table is exclusively matched to one YAGO+F category. In the Equation 6.3, the TN-factor takes the size of the Freebase category into account, such that the influence of the categories that do not contain many instances is reduced. In order to get a better overview of the matching results for the categories independent of their size, we also compute the RI value without the TN-factor. This corresponds to the Jaccard Index JI that

measures the level of agreement over the pairs from the Freebase tables assigned to a YAGO+F category:

$$JI(y) = \frac{TP(y, F_y)}{TP(y, F_y) + FP(y) + FN(F_y)}. \tag{6.4}$$

In our experiments, we compare the RI and JI values of leaf categories obtained using the YAGO+F mapping described in this chapter with the values obtained using PARIS [SAS11] - a recent approach to align ontologies. PARIS computes the probability of subclass relationships among the classes of different ontologies. To facilitate our comparison, while using PARIS, we align each Freebase table with its most probable superclass in the YAGO ontology, computed using Equation 15 in [SAS11]. In contrast to Equation 6.4, the score computed by PARIS does not take into account false positive decisions $FP(y)$ that assign instances from different Freebase tables to one YAGO+F category y: $PARIS(y) = \frac{TP(y,F_y)}{TP(y,F_y)+FN(F_y)}$. Figure 6.4 presents the RI values for each leaf category obtained using YAGO+F and PARIS alignments. The X-axis of Figure 6.4 presents the percentage of leaf categories in the resulting mapping sorted by the RI and JI value correspondingly. The Y-axis presents the corresponding RI and JI values.

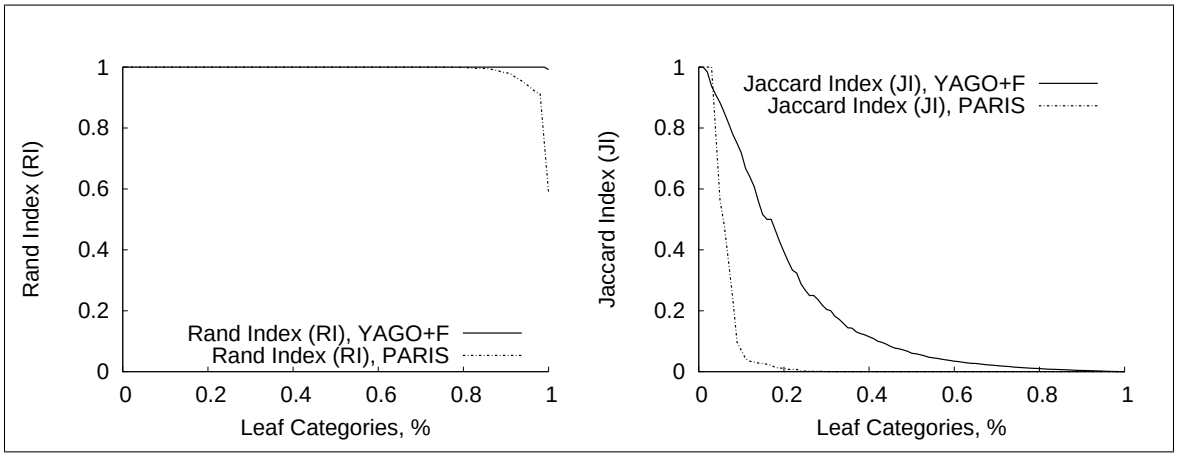


Figure 6.4 Matching Quality: Rand Index and Jaccard Index of the Leaf Categories using YAGO+F and PARIS.

As we can observe, the YAGO+F mapping performs better than the PARIS-based mapping with respect to both, RI and JI values. This is because with PARIS, the probability of an alignment is higher for more general classes, such that more Freebase tables are jointly assigned to one YAGO category. In total, using the PARIS-based alignment, 1,391 Freebase tables are assigned to only 67 YAGO categories, whereas YAGO+F assigns these tables to 1,234 YAGO categories. This way, the mapping obtained by YAGO+F is more specific than the PARIS-based mapping.

With respect to the YAGO+F mappping, on the one hand, the RI values are very high for all the YAGO+F categories, which reflects the fact that the matching of

the Freebase tables highly populated with shared instances is very accurate. On the other hand, the values of JI vary. 22% of the YAGO+F categories possess a value $JI \in [0.9-1]$, for the further 16% of the categories the JI is within [0.5-0.9). Finally, the rest of the YAGO+F categories possesses the JI values below 0.5.

The group of the YAGO+F categories with the highest JI values includes the cases where all the shared instances of a Freebase category are contained in one YAGO category. For example, *"Location tw provincial city"* is matched to the Wikipedia category *"Provincial cities of Taiwan"*, *"Medicine artery"* to the WordNet category *"Artery"*, *"Food bottled water"* to the Wikipedia category *"Bottled water brands"*, *"Luminance unit"* to the Wikipedia category *"Units of luminance"*, and *"Food culinary technique"* to the Wikipedia category *"Cooking techniques"*.

The YAGO+F categories with the JI values of (0.5-0.99] possess many shared instances, almost all of which are found in one YAGO category. In this case the YAGO category will most likely provide an adequate mapping for the Freebase table. For example, *"Location jp prefecture"* is matched to the Wikipedia category *"Prefectures of Japan"*, *"Medicine hospital"* to the WordNet category *"Medical building"*, *"Film"* to WordNet *"Movie"*, *"Book poem"* to WordNet *"Poem"*, *"Business company type"* to the Wikipedia category *"Types of companies"*, and *"Education academic"* to WordNet *"Scientist"*. This group includes 280 YAGO+F categories.

The lower JI values of the remaining YAGO+F categories are due to the incompatibilities in the category structures of Freebase in YAGO. First, the shared instances of a Freebase table can fall into a wide spectrum of the YAGO categories, i.e. there does not exist any clearly defined equivalent YAGO category. These Freebase categories include *"Visual art subject"*, *"Media common quotation subject"*, *"Book periodical subject"*, *"Film subject"*, and *"Amusement parks ride theme"*. Second, in comparison to Freebase, YAGO may lack a specific intermediate category, such that a Freebase table can either be assigned to the most specific parent, which is in fact too general, or to one of the children categories, which are too specific. For example, the Wikipedia categories *"Aviation accidents and incidents officially attributed to pilot error"*, and *"Airliner accidents and incidents caused by fuel exhaustion"* are direct sub-classes of the WordNet category *"Accident"*. Then, the Freebase table *"Aviation accident type"* can either be mapped to one of the more specific categories (with a high FN value), or to a too general category (with a high FP value).

In summary, given a compatible category structure, our mapping provides good results for a significant number of the YAGO+F categories. Nevertheless, some Freebase categories cannot be clearly mapped to YAGO due to incompatibilities in the YAGO and Freebase structure. In future work we will investigate how to improve matching by introducing new categories that incorporate both YAGO and Freebase schema information into a richer ontology.

6.7 Discussion

In this chapter we considered the problem of enrichment of the large scale Freebase dataset with the semantic categories of the YAGO ontology, to address Problem 4 presented in Chapter 1 and to connect these datasets at the schema level supporting efficient incremental query construction over the large scale Freebase dataset. Freebase and YAGO datasets build a part of recently emerged Linked Data [BHBL09] that loosely connects shared instances of the both datasets through the DBpedia references. To the best of our knowledge, the enrichment performed in this thesis is the first attempt to interconnect these datasets systematically at the schema level.

At the beginning of this chapter, we provided a brief overview of the specific background from the area of schema matching. Following that, we performed a detailed analysis of the concept and instance distribution in the YAGO ontology. In order to better understand how good the YAGO ontology structure fits to the Freebase dataset, we also analyzed how the instances shared by the both datasets are distributed across the different levels of the YAGO hierarchy. Then, we presented the matching techniques to align YAGO and Freebase. We analyzed the concept and instance distribution in the resulting YAGO+F structure and provided an evaluation of the matching quality.

The results of our experiments confirmed the good quality of the matching in cases where the YAGO and Freebase category structures are compatible, but also have shown incompatibilities between the category schemas of YAGO and Freebase. In future work we can investigate how to improve matching by introducing new categories that incorporate both YAGO and Freebase schema information into a richer ontology which will provide even richer semantic information suitable for Freebase data, avoid the difficulties caused by incompatible YAGO and Freebase hierarchy structures and incompatible class instance assignment, and potentially improve both the YAGO hierarchy as well as Freebase schema information.

The results of YAGO and Freebase alignment described in this chapter improve efficiency of incremental query construction over large scale data as discussed in Chapter 5. They also provide a further important step towards interconnection of the Linked Data subcollections. Furthermore, we made the resulting mapping available to the community, such that this mapping can possibly also be used in many different areas that can profit from a wide variety of Freebase data clearly arranged into the semantic categories of YAGO. This areas can include database schema summarization [YJ06, YPS09] and ontology-based question answering [ATS+11], just to give some examples.

7 Conclusions and Future Work

The amount of structured data available to end users on the Web, in organizations and enterprises grows rapidly. At the same time, access to this data for end users become increasingly difficult due to heterogeneity of schemas and complexity of structured query languages. On the one hand, keyword search in databases possesses a high usability and enables end users to query a database without knowing the schema or learning the syntax of a query language. On the other hand, keyword search lacks expressiveness, such that users can not precisely specify their informational needs with keywords only. As a result, they can obtain imprecise or incomplete results. Query disambiguation techniques enable bridging the gap between usability of keyword search and expressiveness of the database queries by translating keyword queries into their structural interpretations that describe informational needs of the user more precisely. However, majority of the existing database search applications only looks for the most likely interpretations, living interaction and diversification aspects aside. While these techniques are sufficient for the most simple and straightforward keyword queries, intended interpretations of ambiguous queries may not be found within the top ranked results. Moreover, many useful parameters for effective query disambiguation are not sufficiently explored. This observations motivated us to develop new approaches to enable end users going beyond the most likely interpretations.

7.1 Summary of Contributions

In this thesis we considered several important usability aspects of database keyword search such as *incremental query construction*, *search result diversification*, *efficient incremental query construction for large scale databases*, and *enrichment of a large scale database with the semantic categories of an ontology at the example of Freebase and YAGO* that have not been sufficiently addressed by the existing approaches.

In Chapter 3 we analyzed the problem of probabilistic incremental query construction and presented IQP - a novel system, which enables construction of structured

queries from keywords. Given an ambiguous keyword query, IQP asks a minimal number of questions and enables a user to efficiently construct the desired structural query interpretation in an interactive way. We presented a conceptual framework for the incremental query construction as well as a probabilistic model, which enables consistent assessment of the probability of a query interpretation. We presented two algorithms for generating optimal query construction plans, which enable users to obtain intended structured queries with a minimal number of interactions.

Our experimental results on two real-world datasets and a user study have shown that IQP was highly helpful when user intended structured queries cannot be found within the top ranked results. We analyzed the effectiveness of the proposed probability estimates for a set of real-world keyword queries. Our experiments have shown that the proposed probability estimates enabled us to significantly reduce the interaction cost of query construction. We also have shown that query ranking model of IQP outperformed that of the other recent approach to query ranking in related work. In our user study we have observed that a simple ranking interface was sufficient only for the most simple queries. For the queries where the ranks of the intended query interpretations ranged between 40 to 80, the IQP interface started to outperform the query ranking interface. For the queries where the ranks of the intended query interpretations were above 120, the advantage of the IQP interface became very obvious. Finally, our simulations confirmed the scalability of the proposed algorithms with respect to schema and keyword query size for the datasets with up to 100 tables. Our experiments also have shown that the quality of the query construction plans which can be generated by a base line brute force algorithm was only slightly better than those of the efficient greedy algorithm we proposed.

Following that, in Chapter 4 we addressed Problem 2 presented in Chapter 1, namely how to provide database search results with an increasing level of novelty. To this end we presented an approach to search result diversification over structured data. We further enhanced the probabilistic query disambiguation model first presented in Chapter 3 to create relevant query interpretations over structured data taking into account dependencies between keywords in database attributes. Then, we evaluated the quality of the model in a user study. Furthermore, we proposed a query similarity measure and a greedy algorithm to efficiently obtain relevant and diverse query interpretations. As results of keyword search over structured data differ from conventional documents, evaluation metrics existing for document retrieval in presence of diversity and subtopics required some adaptation for the database keyword search. To this end we proposed an adaptation of the established evaluation metrics such as α-NDCG [CKC+08] and S-recall [CK06] to measure quality of diversification in database keyword search.

Our evaluation results have demonstrated the quality of the proposed model and have shown that using our algorithms the novelty of keyword search results over structured data was substantially improved. Diversification performed on top of query ranking achieved significant reduction of result redundancy, while preserving retrieval

quality in the majority of the cases. This way search results obtained using the proposed algorithms also better characterized possible answers available in the database than the results obtained by the initial relevance ranking.

Then, in Chapter 5 we studied the problem of scaling interactive construction presented in Chapter 3 on a very large dataset to address Problem 3 presented in Chapter 1. To the best of our knowledge, FreeQ is the first system that enables efficient schema-based keyword search and incremental query construction over large scale datasets, considering that most existing work on database keyword search uses only test sets of small schemas, such as DBLP, IMDB [IMD], etc. At the beginning of this chapter, we reviewed the query construction model that had been introduced in Chapter 3 and discussed its advantages and limitations. Then we analyzed the efficiency of query construction options and extended the query hierarchy of IQP first presented in Chapter 3 with an ontological layer to reduce the user interaction cost in face of a large database schema. We introduced novel ontology-based query construction options that could summarize the database schema and improved the efficiency of interactive query construction in face of a large scale dataset significantly. Furthermore, we developed algorithms that enable efficient exploration of query interpretation spaces over large scale data. Finally, we experimentally evaluated the efficiency of the proposed solution.

Our experiments have shown that the proposed FreeQ system was effective and efficient in interactive query construction over large scale data. Our results confirmed the effectiveness of the ontological layer created using the native taxonomy of Freebase. Furthermore, we have demonstrated that external ontologies, such as the YAGO ontology, could be used to further increase the efficiency of incremental query construction.

Finally, in Chapter 6 we considered the problem of enrichment of the Freebase dataset with the semantic categories of the YAGO ontology to support efficient incremental query construction over large scale Freebase data addressing Problem 4 presented in Chapter 1. To the best of our knowledge, the enrichment of Freebase dataset with YAGO categories performed in this thesis is the first attempt to interconnect these datasets systematically at the schema level. At the beginning of this chapter, we provided a brief overview of the specific background from the area of schema matching. Following that, we performed a detailed analysis of the concept and instance distribution in the YAGO ontology. Then, we presented matching techniques used to align YAGO and Freebase. We analyzed the concept and instance distribution in the resulting YAGO+F structure and provided an evaluation of the matching quality.

The results of our experiments have confirmed the good quality of the matching in cases where YAGO and Freebase category structures were compatible. The resulting merged data set called YAGO+F not only supportes an efficient query construction over large scale data as we have demonstrated in the experiments presented in Chapter 5, but also provides a further important step towards interconnection of

the sub-collections of Linked Data [BHBL09], and will hopefully enable many future applications that can profit from a wide variety of Freebase data clearly arranged into the semantic categories of YAGO.

7.2 Open Research Directions

In this thesis we presented novel approaches to incremental query construction and search result diversification over structured data. Furthermore, we applied the proposed incremental query construction to a large scale database and developed new ontology-based solution to increase scalability of the system in the large scale application scenario. Moreover, to enable our solution for large scale data, we provided a detailed analysis of the mapping of two large-scale datasets such as Freebase and YAGO ontology. Nevertheless, there is still a room for improvements and further research directions.

Some of interesting future research questions refer for example to a detailed investigations regarding enabling interactive keyword-based operations other than search. This can include keyword-based "insert" and "update" functionalities for databases. In this scenario, the user could use an interactive approach similar to that of IQP and FreeQ described in Chapters 3 and 5 to either add missing information to the database, or update existing information about the entities in the database as well as database schemas starting from simple keywords.

With respect to diversification of search results over structured data presented in Chapter 4, our current approach that takes into account only structural information can be incrementally enhanced by incorporating evidences at the result level, such as e.g. term frequencies, in addition to the syntax-based diversification at the interpretation level performed currently.

Regarding the methods of scaling interactive query construction on a very large dataset discussed in Chapter 5, we believe that further improvement can be achieved through a detailed investigation of usability of the specific ontology used in the ontological layer. Given a large scale database, this study could help us to select the most suitable ontology to effectively summarize the database schema and to generate highly intuitive and informative query construction options.

Finally, the quality of enrichment of large scale data with semantic categories discussed in Chapter 6 can be incrementally enhanced in cases where ontology and database category schemas indicate incompatibilities. In these cases we could introduce new categories that incorporate schema information of the database and an ontology and into a richer ontology. This new ontology could provide even richer semantic information suitable for the specific database, and potentially improve both the ontology as well as the database schema information.

A Curriculum Vitae

Elena Demidova, born on 29 June 1974, in St. Petersburg, Russia.

Studies

06/2006 - 01/2013	Ph.D. studies. Gottfried Wilhelm Leibniz Universität Hannover, Germany
10/2003 - 03/2006	M.Sc. in Information Engineering. University of Osnabrück, Germany and University of Twente, the Netherlands
03/2000 - 06/2003	Computer Science Engineer (Dipl.-Inf. (FH)). University of Applied Sciences, Osnabrück, Germany
09/1991 - 06/1996	Teacher for General Technical Disciplines. A.I. Herzen State Pedagogical University, St. Petersburg, Russia

Professional Experience

06/2006 - 01/2013	Junior researcher. L3S Research Center and Distributed Systems Institute, Gottfried Wilhelm Leibniz Universität Hannover, Germany Research projects: ARCOMEM, LivingKnowledge, OKKAM, TENCOMPETENCE, iSearch.
WS 08/09 - WS 12/13	Lecture "Web Technologies I / Foundations of Information Retrieval", teaching assistant.
12/2003 - 12/2005	Student/research assistant in the CASHMERE-int project. University of Osnabrück, Germany
09/2002 - 08/2003	Internship in software development. MPDV Mikrolab GmbH, Mosbach, Germany
12/2000 - 08/2002	Student assistant. University of Applied Sciences, Osnabrück, Germany

Bibliography

[ABC+02] B. Aditya, Gaurav Bhalotia, Soumen Chakrabarti, Arvind Hulgeri, Charuta Nakhe, Parag Parag, and S. Sudarshan. Banks: browsing and keyword searching in relational databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 1083–1086. VLDB Endowment, 2002.

[AGHI09] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14, New York, NY, USA, 2009. ACM.

[AME07] Muhammed Al-Muhammed and David W. Embley. Ontology-based constraint recognition for free-form service requests. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, The Marmara Hotel, Istanbul, Turkey*, pages 366–375, 2007.

[And95] L. Androutsopoulos. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

[ATS+11] Peter Adolphs, Martin Theobald, Ulrich Schäfer, Hans Uszkoreit, and Gerhard Weikum. Yago-qa: Answering questions by structured knowledge queries. In *Proceedings of the 5th IEEE International Conference on Semantic Computing (ICSC 2011), Palo Alto, CA, USA, September 18-21, 2011*, pages 158–161, 2011.

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

[AYS05] Sihem Amer-Yahia and Jayavel Shanmugasundaram. Xml full-text search: challenges and opportunities. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 1368–1368. VLDB Endowment, 2005.

[BCC05] Daniele Braga, Alessandro Campi, and Stefano Ceri. Xqbe (xquery by example): A visual interface to the standard xml query language. *ACM Trans. Database Syst.*, 30(2):398–443, 2005.

[BCSW07] Holger Bast, Alexandru Chitea, Fabian Suchanek, and Ingmar Weber. Ester: efficient search on text, entities, and relations. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 671–678, New York, NY, USA, 2007. ACM.

[BEP+08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. of the SIGMOD 2008*, pages 1247–1250, New York, NY, USA, 2008. ACM.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 431, Washington, DC, USA, 2002. IEEE Computer Society.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 564–575. VLDB Endowment, 2004.

[BLCL09] Zhifeng Bao, Tok Wang Ling, Bo Chen, and Jiaheng Lu. Effective xml keyword search with relevance oriented ranking. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 517–528, Washington, DC, USA, 2009. IEEE Computer Society.

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7(3):154–165, September 2009.

[Blu99] Adam Blum. Microsoft english query 7.5: Automatic extraction of semantics from relational databases and olap cubes. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 247–248, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[BMS+06] Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. Io-top-k: index-access optimized top-k query processing. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 475–486. VLDB Endowment, 2006.

[BN05] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 69–80, Washington, DC, USA, 2005. IEEE Computer Society.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[BRWD+08] Senjuti Basu Roy, Haidong Wang, Gautam Das, Ullas Nambiar, and Mukesh Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 13–22, New York, NY, USA, 2008. ACM.

[CBC+09] Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, and Jeffrey Naughton. Combining keyword search and forms for ad hoc querying of databases. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 349–360, New York, NY, USA, 2009. ACM.

[CD09] Surajit Chaudhuri and Gautam Das. Keyword querying and ranking in databases. *Proc. VLDB Endow.*, 2(2):1658–1659, August 2009.

[CG98] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, New York, NY, USA, 1998. ACM.

[CGRM06] Venkatesan T. Chakaravarthy, Himanshu Gupta, Prasan Roy, and Mukesh Mohania. Efficiently linking text documents with relevant

structured information. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 667–678. VLDB Endowment, 2006.

[CH02] Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 346–357, New York, NY, USA, 2002. ACM.

[CK06] Harr Chen and David R. Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 429–436, New York, NY, USA, 2006. ACM.

[CK09] Surajit Chaudhuri and Raghav Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 707–718, New York, NY, USA, 2009. ACM.

[CKC+08] Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666, New York, NY, USA, 2008. ACM.

[CL07] Zhiyuan Chen and Tao Li. Addressing diverse user preferences in sql-query-result navigation. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 641–652, New York, NY, USA, 2007. ACM.

[clu] Clusty. http://clusty.com/.

[CMKS03] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. Xsearch: a semantic search engine for xml. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 45–56. VLDB Endowment, 2003.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In Craig Knoblock and Subbarao Kambhampati, editors, *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, Acapulco, Mexico, August 2003.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *CIDR*, pages 1–12, 2005.

[CSA+09] Paul Clough, Mark Sanderson, Murad Abouammoh, Sergio Navarro, and Monica Paramita. Multiple approaches to analysing query diversity. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 734–735, New York, NY, USA, 2009. ACM.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[CWLL09] Yi Chen, Wei Wang, Ziyang Liu, and Xuemin Lin. Keyword search on structured and semi-structured data. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 1005–1010, New York, NY, USA, 2009. ACM.

[DFZN10] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. Divq: Diversification for keyword search over structured databases. In *Proceedings of the 33rd Annual ACM SIGIR Conference, 19-23 July 2010, Geneva, Switzerland.*, 2010.

[DGKT06] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. Answering top-k queries using views. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 451–462. VLDB Endowment, 2006.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[DJMS02] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 240–251, New York, NY, USA, 2002. ACM.

[DKO+07] Elena Demidova, Philipp Kärger, Daniel Olmedilla, Stefaan Ternier, Erik Duval, Michele Dicerto, Carlos Mendez, and Krassen Stefanov. Services for knowledge resource sharing & management in an open source infrastructure for lifelong competence development. In *Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies, ICALT 2007, July 18-20 2007, Niigata, Japan*, 2007.

[DKS08] Bhavana Bharat Dalvi, Meghana Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *Proc. VLDB Endow.*, 1(1):1189–1204, August 2008.

[DMD+12] Stefan Dietze, Diana Maynard, Elena Demidova, Thomas Risse, Wim Peters, Katerina Doka, and Yannis Stavrakas. Preservation of social web content based on entity extraction and consolidation. In *2nd International Workshop on Semantic Digital Archives (SDA) in conjunction with the 16th International Conference on Theory and Practice of Digital Libraries (TPDL), Pafos, Cyprus, September 2012*, 2012.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, pages 662–673, 2002.

[DN06] Elena Demidova and Wolfgang Nejdl. Integrating rdf querying capabilities into a distributed search infrastructure. In *Proceedings of the Proceedings of the "Web Search Technology - from Search to Semantic Search" Workshop, in conjunction with the 1st Asian Semantic Web Conference (ASWC 2006), September 3-7, 2006, Beijing, China.*, 2006.

[DN09] Elena Demidova and Wolfgang Nejdl. Usability and expressiveness in database keyword search: Bridging the gap. In *Proceedings of the VLDB 2009 PhD Workshop. Co-located with the 35th International Conference on Very Large Data Bases (VLDB 2009). Lyon, France, August 24, 2009*, 2009.

[DOF09] Elena Demidova, Irina Oelze, and Peter Fankhauser. Do we mean the same? disambiguation of extracted keyword queries for database search. In *Proceedings of the First International Workshop on Keyword Search on Structured Data, KEYS 2009, Providence, Rhode Island, USA, June 28, 2009*, 2009.

[DR02] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proc. of the 28th Int'l. Conf. on Very Large Data Bases*, pages 610–621, 2002.

[DTO+07] Elena Demidova, Stefaan Ternier, Daniel Olmedilla, Erik Duval, Michele Dicerto, Krassen Stefanov, and Naiara Sacristán. Integration of heterogeneous information sources into a knowledge resource management system for lifelong learning. In *Proceedings of the 2nd TenCompetence Workshop, January 11-12, 2007, Manchester, United Kingdom.*, 2007.

[DXYW+07] Bolin Ding, J. Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 836 –845, april 2007.

[DZN12a] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. Freeq: An interactive query interface for freebase. In *Proceedings of the WWW 2012, April 16–20, 2012, Lyon, France.*, 2012.

[DZN12b] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. Scaling interactive query construction on a very large database. In *submission*, 2012.

[DZNona] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. Iq^P: Incremental query construction, a probabilistic approach. In *Proceedings of the 26th IEEE International Conference on Data Engineering, ICDE 2010, Long Beach, California, USA, March 1-6, 2010*, 2010. ©2010 IEEE. Reprinted with permission.

[DZNonb] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. A probabilistic scheme for keyword-based incremental query construction. *IEEE Trans. on Knowl. and Data Eng.*, 24(3):426–439, March 2012. ©2012 IEEE. Reprinted with permission.

[DZON10] Elena Demidova, Xuan Zhou, Irina Oelze, and Wolfgang Nejdl. Evaluating evidences for keyword query disambiguation in entity centric database search. In *Proceedings of the 21st International Conference on Database and Expert Systems Applications (DEXA 2010), 30 August - 3 September 2010, Bilbao, Spain.*, 2010.

[DZZN09] Elena Demidova, Xuan Zhou, Gideon Zenz, and Wolfgang Nejdl. Suits: Faceted user interface for constructing structured queries from keywords. In *Proceedings of the Database Systems for Advanced Applications, 14th International Conference, DASFAA 2009, Brisbane, Australia, April 21-23, 2009*, 2009.

[ES04] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In Christoph Bussler, John Davis, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the First European Semantic Web Symposium*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91, Heraklion, Greece, may 2004. Springer Verlag.

[ES07] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

[EV04] J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In R. López de Mántaras and L. Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, pages 333–337. IOS Press, 2004.

[Fag99] Ronald Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, February 1999.

[Fag02] Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, June 2002.

[Fel98] editor Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, 1998.

[fI11] Max-Planck Institute for Informatics. Yago2 ontology. http://www.mpi-inf.mpg.de/yago-naga/yago/downloads.html, 2011.

[FLW11] Jianhua Feng, Guoliang Li, and Jianyong Wang. Finding top-k answers in keyword search over relational databases using tuple units. *Knowledge and Data Engineering, IEEE Transactions on*, 23(12):1781 –1794, dec. 2011.

[GKS08] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 927–940, New York, NY, USA, 2008. ACM.

[GKST11] 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.

[Goo11] Google. Freebase data dumps. http://download.freebase.com/datadumps/, 2011.

[GS09] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 381–390, New York, NY, USA, 2009. ACM.

[GSBS03] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 16–27, New York, NY, USA, 2003. ACM.

[GSVGM98] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 26–37, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Hea06] Marti A. Hearst. Clustering versus faceted categories for information exploration. *Commun. ACM*, 49(4):59–61, 2006.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 850–861. VLDB Endowment, 2003.

[HHP08] Vagelis Hristidis, Heasoo Hwang, and Yannis Papakonstantinou. Authority-based keyword search in databases. *ACM Trans. Database Syst.*, 33(1):1:1–1:40, March 2008.

[HKPS06] V. Hristidis, N. Koudas, Y. Papakonstantinou, and Divesh Srivastava. Keyword proximity search in xml trees. *Knowledge and Data Engineering, IEEE Transactions on*, 18(4):525 – 539, april 2006.

[HLC08a] Yu Huang, Ziyang Liu, and Yi Chen. extract: a snippet generation system for xml search. *Proc. VLDB Endow.*, 1(2):1392–1395, August 2008.

[HLC08b] Yu Huang, Ziyang Liu, and Yi Chen. Query biased snippet generation in xml search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 315–326, New York, NY, USA, 2008. ACM.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.

[HPB03] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword proximity search on xml graphs. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 367–378, 2003.

[HRZ98] C. S. Helvig, G. Robins, and A. Zelikovsky. Improved approximation bounds for the group steiner problem. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '98, pages 406–413, Washington, DC, USA, 1998. IEEE Computer Society.

[HWYY07] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 305–316, New York, NY, USA, 2007. ACM.

[IAE04] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, September 2004.

[IMD] The internet movie database. www.imdb.com.

[J. 83] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Analysis*. Chapman and Hall, New York, 1983.

[JCE+07] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 2007. ACM.

[JJ08] Magesh Jayapandian and H. V. Jagadish. Expressive query specification through form customization. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 416–427, New York, NY, USA, 2008. ACM.

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[Kö5] Mika Käki. Findex: search result categories help users when document ranking fails. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 131–140, New York, NY, USA, 2005. ACM.

[KGM09] Lingbo Kong, Rémi Gilleron, and Aurélien Lemay Mostrare. Retrieving meaningful relaxed tightest fragments for xml keyword search. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 815–826, New York, NY, USA, 2009. ACM.

[KKD11] Parthasarathy K., Sreenivasa P. Kumar, and Dominic Damien. Ranked answer graph construction for keyword queries on rdf graphs without distance neighbourhood restriction. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 361–366, New York, NY, USA, 2011. ACM.

[KKR+06] Eser Kandogan, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and Huaiyu Zhu. Avatar semantic search: a database approach to information retrieval. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 790–792, New York, NY, USA, 2006. ACM.

[Kon05] Grzegorz Kondrak. N-gram similarity and distance. In Mariano P. Consens and Gonzalo Navarro, editors, *12th International Conference String Processing and Information Retrieval (SPIRE)*, volume 3772 of *Lecture Notes in Computer Science, Berlin, Germany*, pages 115–126, Buenos Aires, Argentina, 2005. Springer.

[KPC+05] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 505–516. VLDB Endowment, 2005.

[KS06] Benny Kimelfeld and Yehoshua Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 173–182, New York, NY, USA, 2006. ACM.

[KSI06] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Précis: The essence of a query answer. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, pages 69–78, 2006.

[KSI10] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Explaining structured queries in natural language. In *roceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 333–344, 2010.

[KZGM09] Georgia Koutrika, Zahra Mohammadi Zadeh, and Hector Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 391–402, 2009.

[LC07] Ziyang Liu and Yi Chen. Identifying meaningful return information for xml keyword search. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 329–340, New York, NY, USA, 2007. ACM.

[LC08] Ziyang Liu and Yi Cher. Reasoning and identifying relevant matches for xml keyword search. *Proc. VLDB Endow.*, 1(1):921–932, August 2008.

[LCVA02] Wen-Syan Li, K. Selçuk Candan, Quoc Vu, and Divyakant Agrawal. Query relaxation by structure and semantics for retrieval of logical web documents. *IEEE Trans. on Knowl. and Data Eng.*, 14(4):768–791, July 2002.

[LCY+07] Yunyao Li, Ishan Chaudhuri, Huahai Yang, Satinder Singh, and H. V. Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1165–1168, New York, NY, USA, 2007. ACM.

[LDKG04] Bach Thanh Le, Rose Dieng-Kuntz, and Fabien Gandon. On ontology matching problems - for building a corporate semantic web in a multi-communities organization. In *6th International Conference on Enterprise Information Systems (ICEIS 2004)*, pages 236–243, 2004.

[Ley09] Michael Ley. Dblp: some lessons learned. *Proc. VLDB Endow.*, 2(2):1493–1500, August 2009.

[LFWZ08] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. An effective and versatile keyword search engine on heterogenous data sources. *Proc. VLDB Endow.*, 1(2):1452–1455, August 2008.

[LJ09] Bin Liu and H. V. Jagadish. Using trees to depict a forest. *Proc. VLDB Endow.*, 2(1):133–144, 2009.

[LLWZ07] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 115–126, New York, NY, USA, 2007. ACM.

[LLYM04] Shuang Liu, Fang Liu, Clement Yu, and Weiyi Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 266–272, New York, NY, USA, 2004. ACM.

[LOF+08] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 903–914, New York, NY, USA, 2008. ACM.

[LT11] Günter Ladwig and Thanh Tran. Index structures and top-k join algorithms for native keyword search databases. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 1505–1514, New York, NY, USA, 2011. ACM.

[LWC07] Ziyang Liu, Jeffrey Walker, and Yi Chen. Xseek: A semantic xml search engine using keywords. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1330–1333, 2007.

[LWL08] Yi Luo, Wei Wang, and Xuemin Lin. Spark: A keyword search engine on relational databases. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1552 –1555, april 2008.

[LWL+11] Yi Luo, Wei Wang, Xuemin Lin, Xiaofang Zhou, Jianmin Wang, and Kequi Li. Spark2: Top-k keyword query in relational databases. *IEEE Trans. on Knowl. and Data Eng.*, 23(12):1763–1780, December 2011.

[LYJ04] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-free xquery. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 72–83. VLDB Endowment, 2004.

[LYJ06] Yunyao Li, Huahai Yang, and H. V. Jagadish. Constructing a generic natural language interface for an xml database. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 737–754, 2006.

[LYMC06] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2006. ACM.

[MCYC06] N. Mamoulis, Kit Hung Cheng, Man Lung Yiu, and D.W. Cheung. Efficient aggregation of ranked inputs. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, page 72, april 2006.

[MDN08a] Ivana Marenzi, Elena Demidova, and Wolfgang Nejdl. Learnweb 2.0: Integrating social software for lifelong learning. In *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications, Vienna, Austria, 2008*, 2008.

[MDN+08b] Ivana Marenzi, Elena Demidova, Wolfgang Nejdl, Daniel Olmedilla, and Sergej Zerr. Social software for lifelong competence development:

Challenges and infrastructure. *International Journal of Emerging Technologies in Learning (iJET)*, 3(S1):18–23, 2008.

[MDNO08] Ivana Marenzi, Elena Demidova, Wolfgang Nejdl, and Daniel Olmedilla. Social software for lifelong competence development: Scenario and challenges. In *Empowering Learners for Lifelong Competence Development: pedagogical, organisational and technological issues. Proceedings of the 4th TENCompetence Open Workshop Madrid, Spain, April 2008. ISBN 978-90-6813-8474.*, 2008.

[MdSdM+07] Filipe Mesquita, Altigran S. da Silva, Edleno S. de Moura, Pável Calado, and Alberto H. F. Laender. Labrador: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inf. Process. Manage.*, 43(4):983–1004, 2007.

[MHG09] Michael McCandless, Erik Hatcher, and Otis Gospodnetić. *Lucene in Action, Second Edition.* Manning Publications Co., 2009.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[Mus] Musicbrainz - the open music encyclopedia. http://musicbrainz.org.

[MV00a] U. Masermann and G. Vossen. Sisql: schema-independent database querying (on and off the web). In *Database Engineering and Applications Symposium, 2000 International*, pages 55 –64, 2000.

[MV00b] Ute Masermann and Gottfried Vossen. Design and implementation of a novel approach to keyword searching in relational databases. In *Proceedings of the East-European Conference on Advances in Databases and Information Systems Held Jointly with International Conference on Database Systems for Advanced Applications: Current Issues in Databases and Information Systems*, ADBIS-DASFAA '00, pages 171–184, London, UK, UK, 2000. Springer-Verlag.

[NCS+01] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 281–290, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Nie93] Jakob Nielsen. *Usability engineering.* Morgan Kaufmann Publishers, San Francisco, Calif., 1993.

[NJ07] Arnab Nandi and H. V. Jagadish. Assisted querying using instant-response interfaces. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1156–1158, New York, NY, USA, 2007. ACM.

[ODN12] Irina Oelze, Elena Demidova, and Wolfgang Nejdl. Yago meets freebase: Combining a large-scale database with an ontology. In *submission*, 2012.

[Pat06] Siddharth Patwardhan. Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In *In: Proceedings of the EACL*, pages 1–8, 2006.

[PCT06] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 1, New York, NY, USA, 2006. ACM.

[PS08] Simone Paolo Ponzetto and Michael Strube. Wikitaxonomy: A large scale knowledge resource. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 751–752, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[PY08] Ken Q. Pu and Xiaohui Yu. Keyword query cleaning. *Proc. VLDB Endow.*, 1(1):909–920, August 2008.

[Qui86] J. R Quinlan. Induction of decision trees. In *Machine Learning*, pages 81–106, 1986.

[QYC09] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Keyword search in databases: the power of rdbms. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 681–694, New York, NY, USA, 2009. ACM.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, December 2001.

[SA02] G. Das S. Agrawal, S. Chaudhuri. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 5–, Washington, DC, USA, 2002. IEEE Computer Society.

[SAS11] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[SCG07] Chong Sun, Chee-Yong Chan, and Amit K. Goenka. Multiway slca-based keyword search in xml data. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1043–1052, New York, NY, USA, 2007. ACM.

[SGB+07] Feng Shao, Lin Guo, Chavdar Botev, Anand Bhaskar, Muthiah Chettiar, Fan Yang, and Jayavel Shanmugasundaram. Efficient keyword search over virtual xml views. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1057–1068. VLDB Endowment, 2007.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[SLDG07] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. Efficient keyword search across heterogeneous relational databases. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 346–355, 2007.

[TCRS07] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-based interpretation of keywords for semantic search. In *ISWC'07/ASWC'07: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, pages 523–536, Berlin, Heidelberg, 2007. Springer-Verlag.

[TL08] Sandeep Tata and Guy M. Lohman. Sqak: doing more with keywords. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 889–902, New York, NY, USA, 2008. ACM.

[TWC11] Arash Termehchy, Marianne Winslett, and Yodsawalai Chodpathumwan. How schema independent are schema free query interfaces? In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 649–660, 2011.

[TWRC09] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 405–416, Washington, DC, USA, 2009. IEEE Computer Society.

[TY09] Yufei Tao and Jeffrey Xu Yu. Finding frequent co-occurring terms in relational keyword search. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 839–850, New York, NY, USA, 2009. ACM.

[TZC+06] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, New York, NY, USA, 2006. ACM.

[vLGOvZ09] Reinier H. van Leuken, Lluis Garcia, Ximena Olivares, and Roelof van Zwol. Visual diversification of image search results. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 341–350, New York, NY, USA, 2009. ACM.

[VOPT08] Quang Hieu Vu, Beng Chin Ooi, Dimitris Papadias, and Anthony K. H. Tung. A graph method for keyword-based selection of the top-k databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 915–926, New York, NY, USA, 2008. ACM.

[VSS+08] Erik Vee, Utkarsh Srivastava, Jayavel Shanmugasundaram, Prashant Bhat, and Sihem Amer Yahia. Efficient computation of diverse query results. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 228–236, Washington, DC, USA, 2008. IEEE Computer Society.

[Wei07] Gerhard Weikum. Db&ir: both sides now. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 25–30, New York, NY, USA, 2007. ACM.

[Wik] Wikipedia. www.wikipedia.com.

[WLWZ12] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, SIGMOD, 2012.

[WPZ+06] Shan Wang, Zhaohui Peng, Jun Zhang, Lu Qin, Sheng Wang, Jeffrey Xu Yu, and Bolin Ding. Nuits: a novel user interface for efficient keyword search over databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1143–1146. VLDB Endowment, 2006.

[WSR07] Ping Wu, Yannis Sismanis, and Berthold Reinwald. Towards keyword-driven analytical processing. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 617–628, New York, NY, USA, 2007. ACM.

[WZ09] Jun Wang and Jianhan Zhu. Portfolio theory of information retrieval. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122, New York, NY, USA, 2009. ACM.

[XCH06] Dong Xin, Chen Chen, and Jiawei Han. Towards robust indexing for ranked queries. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 235–246. VLDB Endowment, 2006.

[XIG09] Yanwei Xu, Yoshiharu Ishikawa, and Jihong Guan. Advances in web and network technologies, and information management. chapter Effective Top-k Keyword Search in Relational Databases Considering Query Semantics, pages 172–184. Springer-Verlag, Berlin, Heidelberg, 2009.

[XP05] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 537–538, 2005.

[XP08] Yu Xu and Yannis Papakonstantinou. Efficient lca based keyword search in xml data. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 535–546, New York, NY, USA, 2008. ACM.

[YJ06] Cong Yu and H. V. Jagadish. Schema summarization. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 319–330. VLDB Endowment, 2006.

[YLST07] Bei Yu, Guoliang Li, Karen Sollins, and Anthony K. H. Tung. Effective keyword-based selection of relational databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 139–150, New York, NY, USA, 2007. ACM.

[YPS09] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Summarizing relational databases. *Proc. VLDB Endow.*, 2:634–645, August 2009.

[YQC10a] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. *Keyword Search in Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[YQC10b] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. Keyword search in relational databases: A survey. *IEEE Data Eng. Bull.*, 33(1):67–78, 2010.

[YS09] Xiaohui Yu and Huxia Shi. Query segmentation using conditional random fields. In *Proceedings of the First International Workshop on Keyword Search on Structured Data*, KEYS '09, pages 21–26, New York, NY, USA, 2009. ACM.

[ZDC10] Sergej Zerr, Elena Demidova, and Sergey Chernov. Ddeskweb2.0: Combining desktop and social search. In *Proceedings of the Desktop Search Workshop, in conjunction with SIGIR 2010, 23 July 2010, Geneva, Switzerland.*, 2010.

[ZDO+08] Sergej Zerr, Elena Demidova, Daniel Olmedilla, Wolfgang Nejdl, Marianne Winslett, and Soumyadeb Mitra. Zerber: r-confidential indexing for distributed documents. In *Proceedings of the EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008*, 2008.

[ZGBN07] Xuan Zhou, Julien Gaugaz, Wolf-Tilo Balke, and Wolfgang Nejdl. Query relaxation using malleable schemas. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 545–556, New York, NY, USA, 2007. ACM.

[Zlo75] Moshé M. Zloof. Query by example. In *AFIPS '75: Proceedings of the May 19-22, 1975, national computer conference and exposition*, pages 431–438, New York, NY, USA, 1975. ACM.

[ZMKL05] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 22–32, New York, NY, USA, 2005. ACM.

[ZSHD12] Sergej Zerr, Stefan Siersdorfer, Johnathon Hare, and Elena Demidova. Privacy-aware image classification and search. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval, August 2012, Portland, Oregon*, 2012.

[ZWX+07] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. Spark: adapting keyword query to semantic search. In *ISWC'07/ASWC'07: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, pages 694–707, Berlin, Heidelberg, 2007. Springer-Verlag.

[ZZDN08] Xuan Zhou, Gideon Zenz, Elena Demidova, and Wolfgang Nejdl. Suits: Constructing structured queries from keywords. In *Technical Report at the L3S Research Center*, 2008.

[ZZM+09] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From keywords to semantic queries-incremental query construction on the semantic web. *Web Semant.*, 7(3):166–176, September 2009.