A FLEXIBLE QUERY TRANSFORMATION FRAMEWORK FOR STRUCTURED RETRIEVAL

GAN KENG HOON

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2013

A FLEXIBLE QUERY TRANSFORMATION FRAMEWORK FOR STRUCTURED RETRIEVAL

GAN KENG HOON

THESIS SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2013

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Gan Keng Hoon (I.C./Passport No.:760711-07-5538)

Registration/Matrix No.: WHA050004

Name of Degree: Doctor of Philosophy

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"): A Flexible Query Transformation Framework For Structured Retrieval

Field of Study: Information Retrieval

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:
Designation:

ABSTRACT

Recent years, there exist meaningful structured collections that can be exploited in search task. When searching for these structured collections, the expressiveness of structured queries allows structures to be specified at the query layer in order to obtain a more focused and precise search results. However, constructing such queries in an adhoc search environment is difficult as users need to be familiar with the syntax of the query languages. Heterogeneities of structure usages across different collections also hinder users from selecting appropriate structure or concept when writing queries.

In this thesis, we are motivated to automate the construction of these queries from keywords query which are more familiar to any user. The work of query transformation results in two main challenges. First, to propose a generic framework such as it can be easily adapted to changes in structured retrieval environment such as retrieval systems, collections, scoring models. Second, to propose a query interpretation within the framework that will handle structure complexities in collection. Since the usage of markups and structures in current structured collections can be loosely defined, these collections are now richer and more complex in their information structures, especially for text centric collection. Current works have yet to explore into these newly emerging complex structures when capturing knowledge for query interpretation.

In order to address these challenges, a flexible query transformation framework (FQT) is proposed. The flexibility feature is desired such that the framework can cater for various settings of structured retrieval environment e.g. different types of structured collections and structured query interfaces. This framework consists of a novel intermediate query representation that will be the central of the transformation process, i.e. a structure that captures the information needs of query and the syntax of query separately. Its main strength is to allow the transformation to be generic to cater for more than single type

of structure query. Supporting this intermediate query representation are the query interpretation and query construction algorithms. The former uses context-based probabilistic approach for interpreting source query, whereas the latter constructs the interpreted query into an intermediate query. Once a source query is interpreted and represented as intermediate query, it can be easily mapped to a structured query language using a set of predefined query templates in knowledge base.

Lastly, experiments are carried out at the algorithm, application and representation levels on both synthetic and real world data sets to demonstrate the feasibility and scalability of the query transformation framework. The experimental results confirm that our framework is more effective in terms of query interpretation especially dealing with collection with complex structures. The framework is also able to represent various kinds of information needs and structured query languages with its proposed intermediate query representation. Better performance in terms of precision has also been achieved when structured query generated by the framework is applied in structured retrieval task.

ABSTRAK

Kebelakangan ini, terdapat banyak koleksi data yang lebih kaya dari segi makna serta strukturnya. Koleksi ini amat berguna untuk tugas seperti carian di internet. Untuk pencarian koleksi jenis ini, sekiranya topik pencarian dapat dinyatakan dalam bentuk yang berstruktur, iaitu *bahasa pencarian berstruktur*, maka hasil carian akan menjadi lebih fokus and tepat. Walau bagaimanapun, pembinaan topik pencarian adalah sukar atas sebab pengguna perlu arif dalam membentuk sintaks bahasa pencarian berstruktur tersebut. Tambahan pula, kepelbagaian jenis struktur-struktur yang digunakan dalam koleksi data berstruktur turut menjadi halangan kepada pengguna untuk memilih atau menggunakan struktur yang betul semasa membentuk topik pencarian.

Dalam tesis ini, kami mengalihkan tugas untuk membentuk topik pencarian dalam bahasa pencarian berstruktur kepada sistem transformasi pencarian, dimana pengguna hanya perlu membentuk topik pencarian dalam bentuk kata-kata kunci sahaja. Namun, terdapat beberapa masalah yang kita perlu selesaikan dalam sistem transformasi pencarian ini. Pertama, suatu sistem yang lebih umum adalah diperlukan supaya ia mudah disesuaikan dengan perubahan pada persekitaran pencarian berstruktur seperti jenis sistem pencarian, koleksi dan model pemarkahan. Kedua, satu kaedah tafsiran pencarian dicadangkan untuk mengendalikan struktur yang rumit dalam koleksi. In disebabkan kemudahan serta kelonggaran penggunaan struktur dalam koleksi jenis ini telah mengakibatkan kehadiran struktur-struktur yang lebih kaya namun kompleks untuk diekploitasikan oleh pengguna. Pendekatan sekarang masih belum meneroka untuk pengguna struktur yang kompleks ini secara efektif.

Dengan ini, tesis ini mencadangkan satu *rangka kerja untuk transformasi pencarian* yang lebih fleksibel, FQT (Flexible Query Transformation). Ciri fleksibiliti FQT adalah diperlukan supaya ia dapat memenuhi keadaan pencarian berstruktur yang melibatkan pelbagai jenis koleksi berstruktur and bahasa pencarian berstruktur. Idea utama FQT terletak pada *perwakilan pencarian perantaran*, merupakan satu struktur yang mengasingkan maklumat percarian dan sintaks bahasa pencarian. Pembentukan struktur perantaraan ini disokong oleh modul pentafsiran pencarian dan modul pembinaan pencarian. Modul pentafsiran pencarian menggunakan *model kebarangkalian berpandukan konteks* untuk membuat pentafsiran, manakala modul pembinaan pencarian bertujuan untuk membentuk topik yang ditafsir sebagai struktur perantaraan. Struktur perantaraan ini kemudiannya akan dipetakan kepada bahasa pencarian berstruktur dengan menggunakan template yang terpilih dari pangkalan pengetahuan.

Penilaian prestasi atas FQT dijalankan pada tiga peringkat, iaitu algoritma, applikasi and perwakilan struktur pencarian dengan menggunakan dua jenis data, data sintetik dan data sebenar. Keputusan eksperimen mengesahkan bahawa FQT adalah lebih berkesan dari segi pentafsiran konteks pencarian and pembentukan pencarian dalam bahasa berstruktur terutamanya bila digunakan untuk koleksi dengan struktur kompleks. Prestasi yang lebih baik juga dicapai bila topik pencarian berbentuk bahasa struktur yang dihasilkan oleh FQT diaplikasikan dalam sistem pencarian berstruktur.

ACKNOWLEDGEMENTS

I am indebted to my supervisor, Dr. Phang Keat Keong, for the guidance he afforded me during my entire studies. Dr. Phang has given me plenty of freedom to explore the directions I was most interested in. At the same time, he constantly reminding me on focusing at the right directions. Moreover, he also provided me all the support at the time when I encountered problems with my thesis direction. I am very grateful for his patience and his confidence in me.

Also, I would like to thank Universiti Sains Malaysia and the School of Computer Science for their financial support of my doctoral studies.

Besides, I am thankful to Dr. Rosni Abdullah, Dr. Chan Huah Yong, Dr. Tang Enya Kong, Dr. Tan Tien Ping, and many more mentors, colleugues and peers that i have not mentioned here, for their words of encouragements and advices.

I am grateful to the SIGIR 2008 Doctoral Consortium Program Committee for giving me the opportunity to present my research and get their feedback. In particular, I want to thank my doctoral consortium advisors Jamie Callan and Charles Clarke for the fruitful one-on-one discussions.

I am grateful to Sara Tan, my longtime friend. Sara has offered her helps on things too many to mention, from programming to research ideas. She has given me great assistance in my development works and endless discussions when i am working on my thesis. Also, I want to thank Lian Tze for being the greatest Latex guru of thesis formatting.

I am deeply beholden to my parents for their immeasurable love and support.

And lastly, I am especially thankful to my husband for his unconditional love, for putting up with the countless hours I spent on my thesis work, and for being there when I needed it. Finally, to my two lovely daughters, Yuan and Mei, this thesis is dedicated to them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ACRONYMS

BEP	Best Entry Point.
CAS	Content-and-Structure.
CO	Content-Only.
DBLP	DBLP Computer Science Bibliography.
DTD	Document Type Definition.
HTML	Hyper Text Markup Language.
INEX	Initiative for the Evaluation of XML retrieval.
IR	Information Retrieval.
LM	Language Model.
MRR	Mean Reciprocal Rank.
NEXI	Narrowed Extended XPath I.
NLP	Natural Language Processing.
NLQ	Natural Language Query.
POS	Part-of-Speech.
RDF	Resource Description Framework.
RR	Reciprocal Rank.
SIGIR	Special Interest Group of Information Retrieval.
SLCA	Smallest Lowest Common Ancestor.
SQL	Structured Query Language.
TFIDF	Term Frequency Inverse Document Frequency.
TFIEF	Term Frequency Inverse Element Frequency.
XML	Extensible Markup Language.
XPath	XML Path Language.
XQuery	XML Query Language.

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

The number of public accessible structured resources over the web like Extensible Markup Language (XML) is growing rapidly. The flexibility of XML enables it to be used to express meaningful contents. Several popular sites like SIGMOD, DBLP publish structured resources on their sites for the purpose of information exchanged and retrieval. In addition, there are also collections of structured resources that have been produced for research evaluation from well-known real world data like Wikipedia, IEEE journal, IMDB etc. Moreover, many works (Graupmann et al., 2004) (Schenkel, Suchanek, & Kasneci, 2007) (Ley, 2009) have proven that these resources can be easily created from web contents like semi-structured hypertext documents or contents stored in database.

The primary intention of marking up and structuring these contents is for software agents to easily access them for various purposes, however, since these structured contents are openly and publicly accessible over the web, this expands the usage of structured resources to not only exchanging of information among pre-agreed machines, but enabling retrieval task such as information search. Moreover, meaningful markups used in these structured resources promote wider exploitation of such resources over the web, rather than limited to usages among agreed parties only.

Hence, these resources have become an important subset of the information published and shared on the web. And, it is obvious that much of the potentials of this subset of web remain untapped. It would be an advantage to current retrieval systems if they can utilize the structures or markups of documents for answering query needs. This leads to the active development of XML retrieval systems in recent years, which can be seen from the collaborative effort of Initiative for the Evaluation of XML retrieval (INEX) (Fuhr,

Gövert, Kazai, & Lalmas, 2002b).

Structural retrieval system exploits the structural information available in documents to implement a more focused retrieval strategy. The system returns document components or more precisely XML elements instead of complete documents in response to a user query (Pal & Mitra, 2007). The emergence of research in structured retrieval system will nevertheless benefit the field of information searching. By integrating structured retrieval (or also known as XML retrieval) methods in contemporary search systems, users will be able to directly lookup information from structured resources on the web. In such scenario, in order for users to benefit from structures or markups available in resources, query can be formulated in structured forms using methods like query languages (e.g. XML Query Language (XQuery) (Chamberlin, 2002), Narrowed Extended XPath I (NEXI) (Trotman & Sigurbjörnsson, 2004a)), forms (e.g. advance search (Barranco, Campaña, & Medina, 2005; Zwol, Baas, Oostendorp, & Wiering, 2006)) etc, whereby users can explicitly specify structures or markups in the query.

1.1 Motivation

1.1.1 On Exploiting Structural Information in Search

Structural information (i.e. markups and structures) are very useful if they are specified correctly as search constraints in a search process, whereby it can directly reflect the scope or context of a query information needs. The ability to utilize the structural information highly depends on factor like how these information can be included in the querying process (Kamps, Marx, Rijke, & Sigurbjörnsson, 2005). There are several methods of querying in structured retrieval that enable users to specify structural information. These methods can be classified into path-based, fragment-based, concept-based, form-based and keywords-based querying as follows.

1.1.1 (a) Path-based Querying

In path-based querying, user queries for desired information using expressions and paths. The most popular path-based languages for querying structured resources would be XML Path Language (XPath) (Boag et al., 2007) and XQuery (Chamberlin, 2002). XQuery is similar to Structured Query Language (SQL) for querying records in database system, whereby it allows user to specify keywords and structural constraints in a query. And, the query returns all matched answer to user without performing any ranking. Following the emergence of XML retrieval systems, needs arise in order to allow user to express precise information needs but in a simpler manner. Hence, query language like NEXI (Trotman & Sigurbjörnsson, 2004a; Trotman, 2009) is introduced to provide a more convenient querying. Unlike XQuery which is more suitable for expert user like XML application developers, NEXI uses simplified syntax. Nevertheless, these languages still require a great effort of syntax formulation and validation, which is less appropriate to be in real time search needs. The complexities of this querying method also hinder users from using the structural information efficiently.

1.1.1 (b) Fragment-based Querying

Compare to path-based querying, an effective and simpler querying method would be XML fragment query (Carmel, Maarek, Mandelbrod, Mass, & Soffer, 2003). This work avoids complex querying by allowing users to pose their query using xml fragment, e.g. <chapter><title>XML tutorials</title></chapter>. Since xml fragment is a direct adaptation of XML format, this avoids the needs to learn or remember another query language. And, different from language-based query that requires users to write a syntactically correct query path, this method gives users higher flexibility when composing the target, constraint and structure paths for a query. Responsibility of handling users' needs is passed to the system ranking mechanism.

1.1.1 (c) Concept-based Querying

An even simpler yet expressive way of querying that utilizes structural information is concept-based querying. As featured in the work by Graupmann et al. (2004) and Graupmann (2004), structural information of keywords can be expressed as concept-value condition in the form of concept=value, e.g. title="War and Peace", author="Tolstoy". Very similar to web query, this querying method can be easily exploited by general users for specifying more precise information needs. An example of an online search service which deploys similar querying format is the DBLP categorical refinement search. Concepts such as venue and author are used to refine scope of information look up. With similar intention, Cohen, Mamou, Kanza, and Sagiv (2003) also uses this querying format, i.e. label keyword in its semantic search engine for XML.

1.1.1 (d) Form-based Querying

No matter how simple a query is to be written, requiring a general user to manually specify the structural information or concept of keywords is not as straight forward as it seems. If the underlying structure of search collection is homogeneous one, i.e. based on simple, fixed and straightforward concepts like author, venue and year in DBLP Search (see Figure 1.1), then remembering and selecting the correct structural information is not a problem. However, for heterogeneous collection with rich annotated concepts like Wikipedia (Graupmann et al., 2004), it is impose such feature in the look up process. Hence, Bricks (Zwol et al., 2006) introduces a graphical approach, using an advanced form-based query builder, to help user in selecting structural information or concepts. Although selecting structural information for richly markups collection (e.g. Wikipedia) or across different collections could be possible by using the approach proposed in this work, there are issues like too many unique concepts, confusion on usage of same naming for different concepts etc. that need to be looked into.

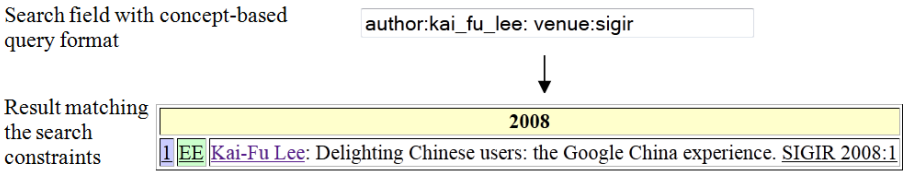


Figure 1.1: Search query with *concept and value* format in DBLP faceted search feature.

1.1.1 (e) Keywords-based Querying

The usage of keywords only queries (also known as content only or CO topic) in structured retrieval systems can be seen in INEX (Fuhr, Gövert, Kazai, & Lalmas, 2002a). Although structured resources were initially designed to be queried using structured query, in order to allow more users to harness information from openly available structured resources, keyword-based querying has become an important way of retrieving that is much more familiar and easy for users. If we looked at the user survey carried out by (Kazai & Trotman, 2007), comparing usage preference between keywords and advance search form on the Web, most still prefer the former. When only keywords are available, current works either ignore the usage of structural information in their retrieval process, or automatically add the structural information to the query.

It is obvious that without some kind of structural hints (i.e. markups or structures) in a query, it is hard to even determine the granularity of elements to retrieve, which is crucial in structured retrieval. Hence, there are recent works that automate this process (Petkova, Croft, & Diao, 2009; Kim, Xue, & Croft, 2009; Hsu, Lee, & Wu, 2004) to improve the effectiveness of keyword-based querying.

1.1.2 On Automated Construction of Structured Query

From the querying methods mentioned above, we can see that there are basically two ways of including structural information in a query, either manually specified by users or automatically included by systems in their retrieval processes.

Comparing both, a more straight forward way for exploiting structural information is to let users decide what they want manually, and directly include those information in the query by either formulating the information as syntax or select them through friendly interfaces. However, when we pose this as a search problem over the web whereby the environment is heterogeneous and information needs is defined in an adhoc manner, issues like complex syntax, naming variations and structures heterogeneity arise during query formulation process.

When users need to explicitly specify structural information, a primary obstacle is that they need to be familiar with the syntax of querying languages in order to be able to include the information in the query. Hence, we can see that many works trying to simplify methods of querying as discussed in previous section. For example, work by Zwol et al. (2006) presents that users have problem expressing the structural information needs if they need to deal with syntactical features of such languages. Similarly, another work by Carmel et al. (2003) also tries to assist users by simplifying the querying syntax.

Further, if we assume that syntax formulation issue can be addressed by using some visual aids or tools, users still need to be familiar with the underlying document structure in both explicit (e.g. naming, path, schema) and implicit manners (e.g. domain, application, context). For instance (refer Table 1.1), users need know the structural path to be able to define the correct target (e.g. inproceedings) or constraints (e.g. author). Same goes for the issue of naming. It is not practical to expect a user to remember constraint names like "booktitle" or "mtitle". Graphical approaches may solve the issues related to utilization of explicit features of document structure but not the implicit one. For example, if there is a drop down list to let users refine the search for "SIGIR", users must be aware that "booktitle" in DBLP collection refers conference proceedings, while "journal" refers to newsletter issue. This makes prerequisite knowledge necessary in order to utilize the structural information effectively.

Table 1.1: Some examples of NEXI query

Collection Type	Source	Example NEXI Query
Conference Article	DBLP	//inproceedings[about(.//author, Kai-Fu Lee) and about(.//booktitle, SIGIR)]
Forum Newsletter	DBLP	//article[about(.//author, Bruce Croft) and about(.//journal, SIGIR Forum)]
Conference Abstract	SIGIR Abstract	//proceedings[about(.//author, Croft) and about(.//mtitle, SIGIR)]

If we look at the results of INEX 2005 adhoc search track for the Wikipedia collection, queries with added structural constraints appear to perform similarly to those that do not specify one (Trotman & Lalmas, 2006). These results contrasted the theory of structured query, where structural constraints improve the precision of structured retrieval systems. The main reason mentioned is that users are bad at specifying structural hints. A later work (Trotman, Rocio Gomez Crisostomo, & Lalmas, 2009) then reinforces this claim through an analysis of the queries in INEX 2008 collection. The work shows that the usages of structures are merely for targeting the size of results only, similar to the observation made by Lehtonen, 2006.

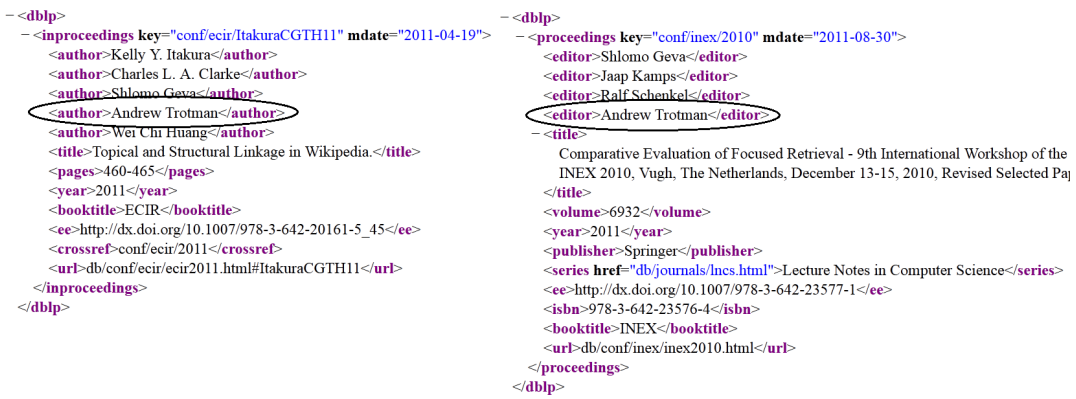


Figure 1.2: Example of semantic markups in XML documents from DBLP.

Here, we have noted that this problem is highly related to the type of structures used in resources. As mentioned in Zwol et al., 2006, three types of markups may be used in a structured document, i.e. semantical, logical and presentation markups. And, collections like IEEE, SIGMOD XML, INEX Wikipedia (up to 2008) fall into the logical

category. When resource structures are logical type, users cannot exploit much of the markups to further refine or reflect their information needs conceptually. For instance, when meaningful structural information like <author> or <editor> is used (see Figure 1.2), it will narrow the search scope to a specific concept, which will significantly improve answers relevancy. Whereas for logical markups such as <article>,<section>, <figure> etc., they are mostly used for defining the size of result, leading to little or no improvement in precision.

Therefore, most of the time users find it difficult to use the correct markups as structural constraints in their queries, not to mention structuring the queries manually. This has motivated solution that will automatically infer structural information (both markups and their structures), switching the burden of users to retrieval system.

1.2 State of the Art

Current works on query transformation from unstructured to structured form for structured collections on the web can be seen from those from information retrieval field or databases field. Although both communities may refer to a similar structured representation, i.e. XML, the former works on XML documents (Petkova et al., 2009; Tannier, 2005) while the latter works on XML database repositories (Calado, Silva, Vieira, Laender, & Ribeiro-Neto, 2002; Barranco et al., 2005). Since contents from XML documents are publicly available, while contents from XML database are remained for internal usage, as such, it is more likely for the current web search solution to acquire contents from the former collections rather than the latter. Therefore, in this section, we present the state of the art concerning approaches used by the information retrieval rather than databases. The state of the art is discussed from two aspects of the query transformation process, i.e. the inferring of structural information and the construction of structured queries.

1.2.1 Inferring Structural Information

The key source of structural information that can be used for query structuring is in fact the markups and structures of the collection itself. Unless the structures are logical one, where corpus knowledge would not be relevant in inferring query's intention, otherwise collections schema or annotations are useful sources for query context analysis. The simplest way to obtain the relationship between a term and a particular structure is by capturing all the relationships between term and its markup/structure/structure path in the collection, and then use them for marking up query during retrieval time. Probabilistic methods are used to estimate the association between a term and its structure (Petkova et al., 2009; Kim et al., 2009; Hsu et al., 2004; Bao, Lu, Ling, & Chen, 2010). As it is one-to-many relationship, this estimation applies well when a collection has simple or homogeneous structure, with little or no ambiguities in its structural concept for a term. For example, an estimation that "andrew" is an "actor", followed by "title" at a lower probability, is probably still satisfactory under a domain with few structural types like movie domain.

However, as we extend our problem to scenario such as searching a more general collection like web site, which consists of many different page types, or even a collection where there are many possible schematic views; further analysis on the markups and structures usage are required to disambiguate differences of structural concepts a term may have. For example, for collection with different schematic views like bibliography domain, "andrew" can be an "author" of a "journal article", or "proceedings article", or "book chapter" etc. Or, when we look at "andrew" from different sites, he can be a "chairman", "senior pc", "lecturer", etc. Hence, current works have limitations in handling this kind of ambiguous situations.

1.2.2 Constructing Structured Queries

There are two main approaches used by the works on the construction of structured queries, i.e. templates or operations. In the first approach, Woodley & Geva, 2004 and Woodley & Geva, 2006 create a set of grammar templates based on structured queries samples collected from INEX forum. Each grammar template corresponds to an individual information request. Similarly, Tannier, 2005 uses XSL Transformation to generate NEXI structured query from its generic query representation known as DRS.

As template approach may suffer from its coverage of structured query formats, there may be difficulties when new templates need to be added. Hence, in the second approach, a set of transformation operators are used to construct the contents of a structured query, which is mainly used to identify target term and content term in the query (Petkova et al., 2009; J. Li, Liu, Zhou, & Ning, 2009). For example, in Petkova et al., 2009, the operations are used to formulate target and constraint terms identified from keywords query into NEXI query language. However, rules needs to be crafted for every possible operation of the structured query language. As we are trying to look into the possibility of a generic query transformation process, ability to accommodate new structured query has become our concern.

With respect to the motivation and current state of the art of query transformation, the next section presents the problems that make this research challenging.

1.3 Query Transformation as a Structured Retrieval Problem

A practical application of query transformation from unstructured query to structured from is to enable integration of structured retrieval features into web search solution. Consider situations where these retrieval systems are used over the web. Here, different retrieval systems mean that we are dealing with different collections (see Figure 1.3). And, this signifies that different structured collections need to be addressed accord-

ingly based on their own concepts and structures in its query construction. This is due to heterogeneities in terms of information structures, document nature and lexical ambiguity among these collections.

Next, dealing with multiple retrieval systems also means that we are dealing with different retrieval methods, so as the structured queries employed. As these queries could fall into categories of either concept-based, fragment-based, or path-based, therefore they have different levels of complexity. For example, system for a text-centric structured resource collection like Wikipedia may deploy a less strict matching option by using the concept-based query in its retrieval method. Whereas a record-centric one like DBLP may deploy a straight forward matching by using a path-based query.

Therefore, transformation between an unstructured query and a structured one does not only involves a set of transformation rules, but many sets of rules if we want to enable the transformation to more possible structured queries forms. And, it is tedious to create different rules for different pairs of queries. Moreover, there are certainly needs of accommodating new querying interfaces, or variants of the existing one. Thus, the approach of redefining rules each time a new interface is introduced is less flexible in applicability and do not generalize well across new structured forms. E.g. a separate transformation process, *SQT1*, *SQT2* and *SQT3* (see Figure 1.3) are required for each interface. Instead, we attempt to reduce many pairs of rules that link to different structures into a more generic form by generalizing the structuring process.

1.4 Goals of This Thesis

The current query transformation solution for structured retrieval is designed specifically for a single type of query, scoring model as well as collection. However, this solution lacks flexibility to cater for evolving structured retrieval environment, especially multiple query types, collection complexities and query interpretation models. The objective of

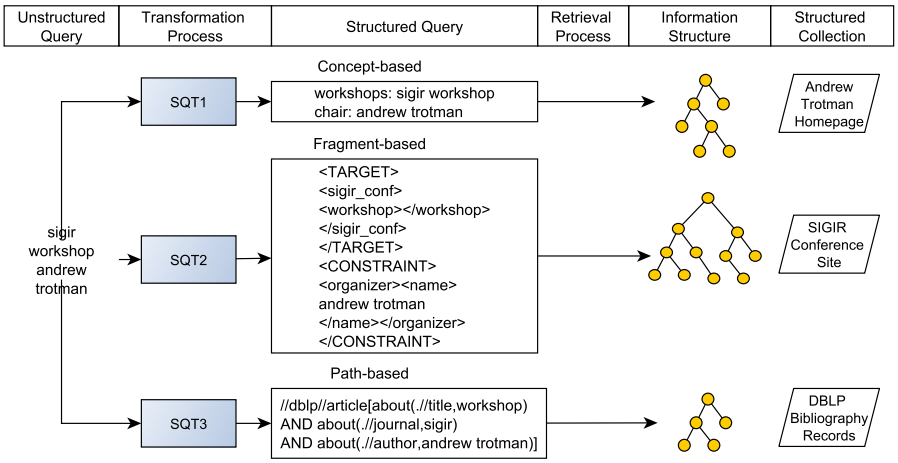


Figure 1.3: A search scenario using structured retrieval systems on the web.

this thesis is to design a flexible framework that can handle the variations or evolutions of these components, with friendlier user interface and better retrieval performance. With respect to this objective, we present a series of goals as follows.

- Improve template-based or rule based method for incorporating new structured queries instead of using fix templates or transformation rules.
- Optimize information utilization from collection side for better query interpretation by
  - extending the probabilistic method to include context factor for query interpretation of complex collection with heterogeneous structures.
  - allowing the probabilistic method to incorporate various type of basic term scoring methods to suit its collection.
- Optimize information utilization from query side for better interpretation using structural keywords in query, in which these keywords are used in indirect manner.

Research Questions In order to justify the feasibility of our research goals, there are several questions that need to be answered.

Q1 Can the proposed flexible query transformation framework scales to differnet structured collections and structured queries types?

Q2 Can the proposed flexible query transformation framework generates a structured query that gives a better retrieval performance compare to the original query?

Q3 Can the extension of probabilistic method with context factor helps in improving the accuracy of query transformation for collection with higher structural complexities?

Q4 Can the proposed extension of probabilistic method be used with existing term weighting models?

Q5 Can query with certain features of information needs such as *"more specific"*, *"longer in size"* and *"inclusion of structural keywords"* give better accuracy for its translated query?

From the stated research goals and questions, we proceed with the following methodologies:

1. Review and identify literatures and their limitations related to various aspects of query transformation in structured retrieval environment (Chapter 2).

2. Develop a formal framework for flexible query transformation (Chapter 3).

3. Instantiate the framework on real information needs and structured resources. A set of algorithms for query interpretation, representation and ranking are designed based on the theory of formal framework (Chapter 4).

4. Evaluate the performance of flexible query transformation framework based on the raised research questions. Evaluation is carried out at from multiple aspects such as query interpretation algorithms, query representation and retrieval outcome (Chapter 5).

1.5 Proposed Framework

The principle underlying our proposed query transformation framework is to have a generic process that could be easily adapted to changes in structured retrieval environment. As such, we propose an intermediate query representation, that can represent the interpreted query in a generic query structure form. This query structure is independent of a specific query language. To convert this structure to a query language, a structure to syntax mapping is defined. The mappings of query structure to syntax are defined using an example-based knowledge base method. This method does not refine the creation of mappings for a single query language, but it is flexible to be applied to more than one query types.

In addition, the proposed solution also handles current limitations of query term interpretation that may occur in complex collection. In complex collection, there exists deeper structure path for a term, hence, it is insufficient if a term is only associated to its immediate parent for term interpretation. Our solution handles this by considering additional good ancestor structures besides its parent. As complex collection also contain heterogeneous elements instead of fewer types as in homogeneous collection, the ambiguity of term is higher. To handle this, our solution introduces context within a collection, where a term will be associated to these contexts during interpretation. This makes the selection of structure/concept context-specific. To capture the probability of a term with its structural under a particular context, a context-based probabilistic model that combines statistical information of contents occurrences and hierarchical information of schema/structures is used.

We present an illustration of the proposed query transformation framework in Figure 1.4. There are three major parts in the proposed framework.

- *the query interpretation process* that consists of two sub modules, i.e. query inter-

pretation and query construction,

- *the query representation model* that represents interpreted queries in generic form and a set of mappings for conversion to structured queries,

- *the knowledge modules* that consists of a context-based term weighting model for query interpretation, a query template knowledge base for storing created templates and mappings for structured queries conversion,

In a query transformation process, an unstructured query will go through the query interpretation module for analysis of information needs. The interpretation and optimization of the query will be carried out using the collection knowledge generated by context-based probabilistic model. Then the interpreted contents will be constructed and represented in a generic query form, known as intermediate query representation. This intermediate form is represented using two separate query structures, that capture the semantic and syntax of queries separately. The interpreted contents in this structure are transformed to query syntax via pre defined mappings. The generation of these mappings are obtained from a set of structured query examples.

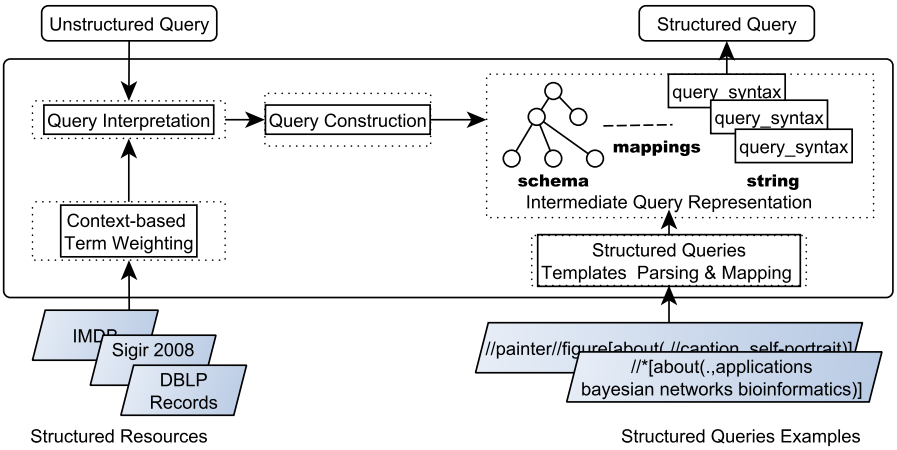


Figure 1.4: The Proposed Query Transformation Framework

1.6 Thesis Contributions

The contributions of the thesis are as followed.

The main contribution of this thesis is its flexible framework that makes query transformation process generic and adaptable for different settings of structured retrieval tasks, such as the collections type, structured querying interface, and query interpretation scoring method. Compare to existing query transformation solutions which have been developed for single transformation, our framework meant to be easily extensible and customized to any domains and retrieval systems. Although existing solutions may have stated that the model they employed are extensible, e.g. from simple hierarchy to complex hierarchy for its query interpretation scoring model, however, it has only been stated briefly without a detailed discussion. In contrast, our framework defines in detailed these aspects and proves them in both theoretical and practical manners. This differentiates the framework from current solutions as the framework targets to be a generic solution to query transformation rather than a one time solution.

Within this framework, we have made three sub contributions.

1. A context-based term weighting approach for query interpretation. This approach focuses on capturing a more precise concept for query term interpretation based on its usage under different contexts in collections. This approach overcomes the limitation of current concept weighting approaches as they still weigh concepts based on the entire collection view, where a term may only have one best concept per collection. This works fine for homogeneous collection but not heterogeneous collection. Whereas, our approach addresses this problem by introducing one best concept per context in collection. In this approach also, intermediate concepts factor is introduced to extend the current immediate concept binding for term. This is to address the issue of non-meaningful immediate concept node.

2. An interchangeable term scoring model for query interpretation. This flexibility allows incorporation of external term scoring model in its concept weighting based on

user preferences. Instead of using only one scoring model as in current works, this flexibility allows usage of simple to complex scoring models based on the nature of collections.

3. A novel intermediate query representation structure is used to represent interpreted query. Using this uniform structure, only single query construction operations set is required for generating the interpreted query. This structure is used to overcome the needs of individualized operations when constructing structured queries. The framework will only require the same operations set rather than individualized one for each structured query type.

4. A structure-syntax template is used to reconstruct a structured query string. For this, we propose a simple annotation and parsing method to generate the template. This method enables the incorporate new or modified structured query syntax with predefined query examples.

In addition, there are additional contributions.

1. We have formalized the query transformation framework. Such formalization is necessary to ensure its applicability and reusability.

2. The practicality of the framework is shown via the instantiation of the framework using both homogeneous and heterogeneous collections, various complexities of information needs and term scoring models.

3. The evaluation of the framework from algorithm aspect and application aspect. The result of the evaluation shows that context-based concept weighting approach is able to suggest better constraint concept and target concept for query interpretation. For application in retrieval task, the structured query generated by the framework out performs its original query in unstructured form. Overall, this thesis changes the

way of designing query transformation solutions from rigid manners to a customizable way under a flexible framework. Although the framework may not always give the best combinations of its components, it is always adjustable without affecting other parts. This characteristic is very desirable when as we are dealing with an evolving environment. Up to date, we are not aware of any work that considers this characteristic under a flexible framework.

1.7 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2 we provide background on structured resources and querying methods. We also include a detailed analysis of related works with respect to the problems of this thesis. Chapter 3 describes our query transformation framework that uses intermediate query representation to overcome limitations of the inflexibility of conventional transformation approach. This chapter also describes the knowledge modules, i.e. a probabilistic model for context-based query interpretation and a structure query template knowledge base for the mapping of interpreted query to query language. Chapter 4 presents the algorithms used in query transformation. It shows how a query is interpreted and constructed into structured query form. In Chapter 5 we present the evaluation of the framework based on its algorithm, application and representation. Chapter 6 concludes the thesis with discussions and outlines directions for future research.

CHAPTER 2

RELATED WORKS

In this chapter, we present the related works of this thesis. The chapter will first present the background of structured resources, querying methods and query transformation. Then, we proceed to present the related works related to the problem of this thesis, whereby we first focus on the methods used for query interpretation and then the approaches used for constructing structured query. From there, we discuss the issues and limitations of current works. And lastly, we conclude with a summary of related works.

2.1 Background

2.1.1 Structured Resources

Structured resources refer to contents that are well represented with markups and structures for purposes like data exchange, data sharing, contents organization or even contents enrichment. These markups or structures may carry some meanings (i.e. concept, type, category, role etc.) that describe the contents. A number like "2008" is associated with the concept of "year"; an entity like "Gerard Salton" is associated to the concept of "creator", "scientist", etc. Some examples of semantically rich structured resources (see Figure 2.1) include data-centric ones like DBLP records , SIGMOD records, conference CFP. Data centric resources often has a *nicer* structures as they are normally created based on some controlled schema. Whereas, in text-centric resource like Wikipedia collection from INEX, the markups are not controlled, created based on various needs such as for meaning annotations, contents presentation etc. More efforts are required to process these markups from text-centric resources compared to data centric.

Figure 2.1: A more meaningful form of Wikipedia contents (left) and DBLP records (right).

2.1.1 (a) Markups

As there are different kinds of meaningful markups which have been created from different intentions, we classify them into three types, i.e. sharing-based markups (marking up and structuring contents for data exchange purpose), presentational-based markups (marking up and structuring contents for presentational or publishing purpose), and annotation-based markups (marking up contents for the purpose of enriching the meaning of an information unit).

Resources with Sharing-based Markups Sharing-based markups are often used in data-centric contents representation. The markups usually refer to Document Type Definition (DTD) or schema for meaning standardization. Thus, the meaning in markups is usually well-defined, catering to the needs of the application, and the semantics are familiar among the pre agreed users. Schema-based markups can be seen in the XML version of DBLP records as in Figure 2.1. The structure is often clear and straightforward, representing unit like a record, an object, a transaction etc. It can be simple or complex,

depending on the type of contents.

Resources with Presentational-based Markups Presentational-based markups are often used in text-centric contents representation. The markups are created following the needs of structuring contents for publishing purpose. They can have a corresponding DTD or schema as references. The design of the schema focuses more for publishing rather than sharing. An example of these markups can be found in the XML version of SIGMOD Record (Sigmod, 2007) (Figure 2.2).

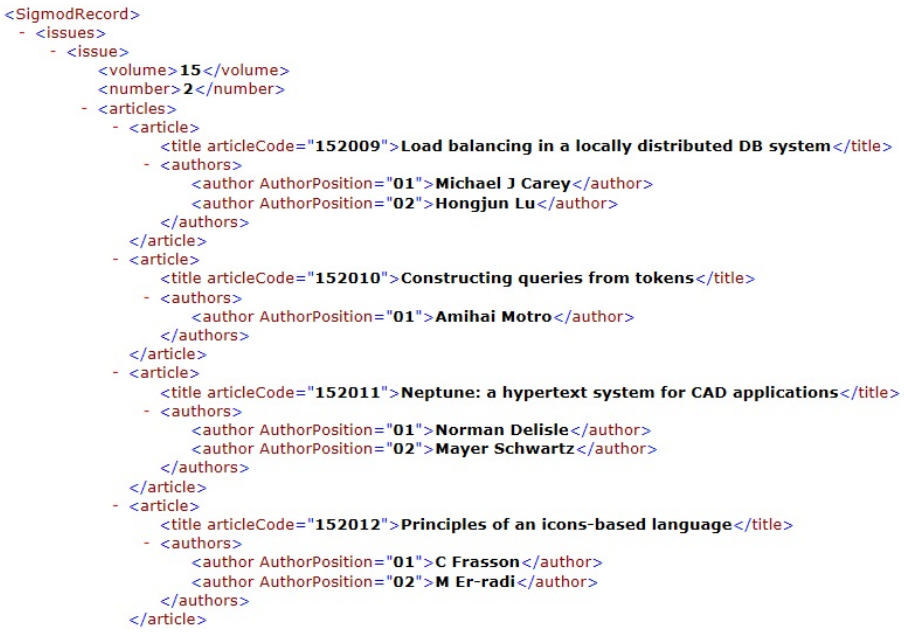


Figure 2.2: Presentational Markups

Resources with Annotation-based Markups Another type of markups found in structured resources are those used for defining the meaning of contents. These markups are usually based on some standards like dictionary or hand crafted knowledge base like ontology or taxonomy. The semantics of markups tend to reflect generic concepts for a term or entity, such as named entity type (e.g. person, organization, location), or categories (e.g. Google Employee, Computer Scientist, Conferences) or descriptive concept from title or headers. Some collections with such markups are YAWN (Schenkel et al., 2007) and The New York Times Annotated Corpus (Sandhaus, 2008).

Being able to utilizing the meaningful markups in these structured resources is an advantage to the current information retrieval processes.

2.1.1 (b) Meaningful Markups and Their Document Structures

Another advantage of current structured resources are their document structures (i.e. taxonomy). In addition to showing how information is organized, such taxonomy enables a better understanding of a meaning (i.e. markup) used to describe a term or an entity. This is due to the many possible meanings for an entity or term that occurs under different scenarios and circumstances, leading us to more specific interpretation about its function, role etc.

For example, consider DBLP collection, the entity, "Andrew Trotman" may appear in different document structures, as illustrated in Figure 2.3. These document structures allow us to capture application-oriented semantics based on real contents usages, which provide us with richer meaning that explains whether "Andrew Trotman" is a proceedings paper author, workshop proceedings editor etc.

As such, as long as contents are represented as some forms of document structure, they carry hidden semantics as a result of the process of creation of the document structure itself. Often, in this process, new semantics associations are formed. Such associations lead to better understanding of meanings, which can be achieved with approaches like probabilistic estimation.

2.1.1 (c) Elements

A fundamental feature of structured resources is that the retrieval of its document can be carried out at various granularities, i.e. elements, based on the document structure. Therefore, instead of returning the entire document in a retrieval task, *meaningful* elements are preferred instead. Some *meaningful* elements presented in a straight forward manner, e.g. for a data centric collection, where each document may corresponds to a

Figure 2.3: Different document structures used to describe a “proceedings", and “article of journal", “article in proceedings", in DBLP.

well-defined retrieval unit. Whereas for a text centric collection, its meaningful elements are less obvious and they may vary per query. According to study carried out by Dopichaj,

2007, smaller elements have been proven to be useful in structured retrieval.

2.1.1 (d) XML

Up to date, XML (Bray, Paoli, & Sperberg-McQueen, 1997) is the most widely adopted standard used to represent structured resources or documents. With its well-defined standard, it is adopted for representing contents that requires both meanings and structures. As it has been well received by both research and commercial communities, development of methods like query languages (Chamberlin, 2002; Boag et al., 2007; Trotman, 2009; Carmel et al., 2003), query optimization (Petkova et al., 2009; J. Li et al., 2009), retrieval models (Itakura & Clarke, 2010; R. Li & Weide, 2009), search engines (Taha & Elmasri, 2010; Theobald, Bast, Majumdar, Schenkel, & Weikum, 2008; Liu, Walker, & Chen, 2007; Graupmann et al., 2004; Cohen et al., 2003), evaluations (Piwowarski, Trotman, & Lalmas, 2008; Lalmas & Tombros, 2007; Voorhees, 1998; Kazai, Lalmas, & Vries, 2004; Pehcevski & Thom, 2005), schema definitions (Fallside & Walmsley, 2004) can be seen in many recent works.

2.1.2 Querying Structured Resources

The potential of semantically rich resources on the web is obvious. With contents represented in a conceptual and structural rich form, these resources have more to offer to solutions in the information seeking domain. When resources are incorporated with concepts like role, category, topic, class, attributes, etc. (Huffman & Baudin, 1997), a query would be able to utilize it for a better definition of information needs.

2.1.2 (a) Structured Queries

The most practical way to take advantage of structured resources is to use it in a query. There are a number of query languages proposed so far that can utilize this advantage. Some are proper standards and widely adopted by various parties (e.g. XQuery

(Chamberlin, 2002), NEXI (Trotman, 2009; Trotman & Sigurbjörnsson, 2004b)) while some remained as research proposal (e.g. XML fragment (Carmel et al., 2003)). Follows, we present some structured queries which are more popular among the community of XML.

XQuery XQuery (Chamberlin, 2002) is the most widely used query language for XML. It can be used to specify both value and structure of parts of document to be returned. Its structure is in the form of path and its expressions have similar functions as SQL to database. For example, it can solve an information need like "Select all journal papers where author is Andrew Trotman in the XML document called dblp.xml".

XQuery:
for $x in doc("dblp.xml")/dblp/article
where $x/author="Andrew Trotman"
return $x/article

However, this query language requires exact expressions for it to retrieve desired results correctly, which differentiate it from a *search* query. Its main limitation is that it is lack of full text search feature that makes it not suitable to text-centric XML collection.

NEXI NEXI was introduced as an extension of XPath since INEX 2004 (Trotman & Sigurbjörnsson, 2004b, 2004a). Thus, it was designed to be simpler than XPath, where information needs can be specified in IR similar form. In XPath, the semantics of query are stated, whereas for NEXI, the interpretation of semantics are handled by retrieval engine. The main reason of using a simpler query is due the the high error rate of queries written by IR experts when when XPath was as the query language for INEX. NEXI successfully reduces the error rate from 63% to 12% (Trotman, 2009). Refer to the same topic from the previous section, its query can be written as follow.

NEXI:
//article[about(.//author, Andrew Trotman)]

XML Fragment XML fragment (Carmel et al., 2003) was introduced with the intention to avoid another complex XML query language. It allows information needs to be specified in a user familiar way, which is similar to the XML documents. Using this query, the structural keywords can easily expressed as level of tags. Approximate matching is used to retrieve the most similar elements that match the query fragment (Carmel, Efraty, L, Maarek, & Mass, 2002). Compared to two previous queries, the main advantage of this query is that it is friendlier in terms of its syntax. Its limitation is that it could not express query that requires a join operation.

XML Fragment:
<article>
<author>Andrew Trotman</author>
</article>

Other Queries Some other works that proposed queries with structures include XML template of INEX 2002 (Kazai, Gövert, Lalmas, & Fuhr, 2003), keyword and label query of XSEarch (Cohen et al., 2003), COMPASS query language (Graupmann et al., 2004).

For XML template query, a query (<Title>) may consist of different components: target elements (<te>), a set of search concepts (<cw>), and a set of context elements (<ce>).

INEX 2002 XML Template:
<Title>
<te>article</te >
<cw>Andrew Trotman</cw><ce>author</ce>
</Title>

The keyword-label query is an extension of the normal list of keywords of a standard search. Each keyword, k, can have a additional label, l to indicate its structure. The query can be in the form of $l : k$, $l :$ or $: k$.

Keyword-label query:
paper: author:Andrew Trotman

Another SQL-like query is COMPASS query language. Given that a list of journal papers is listed on a page, A, a concept-value condition, $concept = value$, can be applied in the query.

COMPASS:
SELECT A FROM INDEX
WHERE A.author="Andrew Trotman"

2.1.2 (b) Structured Queries for Information Seeking and Question Answering

The main advantage of structured queries is that more controls are given to user in query formulation to determine what to be retrieved. When more information is specified at query layer, this will be useful to the retrieval models in getting desired results. Here, we show two kinds of result elements of a structured query that could be beneficial to retrieval application.

- An information/broad element. An information element is an element that satisfies the information needs via methods like exact or approximate matching. The purpose of this result type is to help user focus on a smaller and relevant part of document, instead of browsing through the entire document. In structured retrieval evaluation, a relevant information element can even be a document (root element).

- An answer/focused element. An answer element is an exact element that fulfills the information needs, which is similar to Q&A answer. Different from information element that has a looser relevancy measurement, answer element requires that a result to be accurate and exact.

Here, if we analyzed the information needs of a query statement, "Find the tel of river view in singapore", searching a semantically enhanced web site, one of the information element that is relevant to this query is the "<hotel><name>River View Hotel </name>...</hotel>" element. Hence, in an information seeking process, it is already

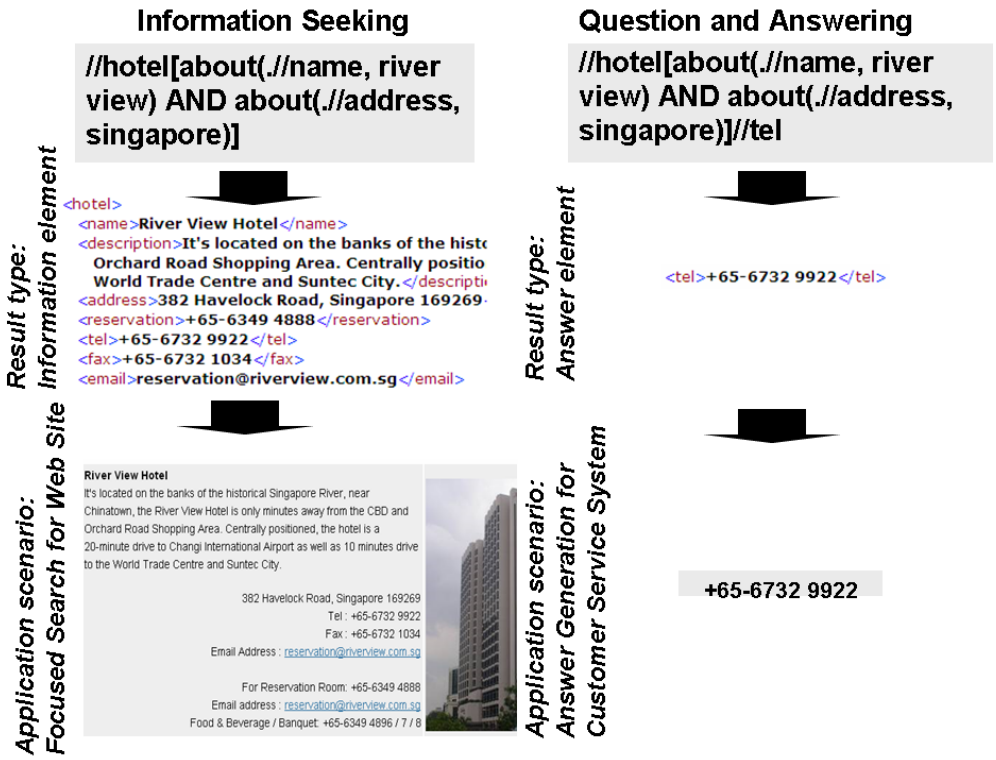


Figure 2.4: An illustration of structured queries on information seeking scenario and question and answering scenario for a query statement, "Find tel of river view in singapore".

useful to focus at this entry point in a large article. However, in a question and answering process, it is obvious that the answer element "<tel>+65-6732 992</tel>" will be the relevant one. Figure 2.4 shows that a more detailed structured query is required to obtain an answer element. It is obvious that when an accurate (i.e. syntactically and semantically correct) structured query is formed, the accuracy (i.e. precision) of results will also be leveraged.

2.1.2 (c) Unstructured Form of Querying

The possibility of making structured contents on the web a success (Pereira et al., 2009) has become the reason why a simpler form querying method, in an unstructured manner, is necessary. Unstructured querying on structured resources is similar to querying in information retrieval. There are mainly two forms of unstructured queries in area of structured retrieval, i.e. keywords or natural language queries.

```
<inex_topic topic_id="358" ct_no="125">
  <title>ontologies information retrieval semantic indexing</title>
  <castitle>//article//section[about(.,ontologies information retrieval
    semantic indexing)]</castitle>
  <description>Retrieve sections of articles about the use of ontologies in
    information retrieval such as semantic indexing.</description>

 <topic id="2009015" ct_no="200">
   <title>Voice over IP</title>
   <castitle>//*[about(.//section, voice over ip)]</castitle>
   <phrasetitle />
   <description>voice over internet protocol technology</description>
```

Figure 2.5: Example topics used in INEX

There are mainly two forms of unstructured queries in area of structured retrieval, i.e. keywords or natural language queries. A keywords query normally contains a number of terms (usually up to three terms as reported in Spink & Jansen, 2004 or two as reported in Arampatzis & Kamps, 2008), describing the information needs. Example of keywords query of a structured retrieval task is shown in <title> of Figure 2.5. Compare to keywords query, a natural language query is longer and contain more details. It normally appears as a description or narration. From the same figure, example of natural language queries is shown in <description>. These queries are often used by INEX NLP track (Woodley & Geva, 2006; Tannier, 2005) in its retrieval task using natural language interface.

2.1.2 (d) Comparing Unstructured and Structured Queries

Unstructured queries represent information needs created by users, usually in an ad hoc manner to achieve their search requests. Although query appears in unstructured form, most of the time, it consists combination of information such as content keywords, concept keywords, language structures (e.g. conjunctions, articles). On the other hand, structured queries allow these needs to be expressed explicitly, through the usage of markups and, structures, increasing the ability of expressibility and interpretable of the query's intention. Some main differences between the unstructured and structured ones are presented in Table 2.1.

Table 2.1: Comparison of features between unstructured and structured queries.

	Query Type	
	Unstructured	Structured
Query complexity	Natural language, keywords	Formal language
Query composition	Adhoc, general user	Crafted, experience user
Information Needs Expressiveness	Implicit	Explicit
Information Needs Interpretability	Loose, as in information retrieval	Strict, as in database retrieval
Query applicability	Interfacing, user layer	Internal, system layer

Understanding the features differences between unstructured and structured queries has shown the advantage of keeping both queries type in a structured retrieval process. The unstructured query is retained for user side so that they can easily specify the query without the need to know a structured query language, while the structured one is retained for system side to carry out some optimization and formulation automatically. This can be done with a query transformation process that will convert the query in unstructured to structured form. The next section presents related works of query transformation.

2.2 Works in Query Transformation

In general, the research of query transformation is carried out due to two main reasons, i.e. to automate the writing of complex query languages, and to optimize the query with additional knowledge. We can see that this research has been going on for the fields like database (Calado et al., 2002; Gonçalves et al., 2004) and semantic web (Zenz, Zhou, Minack, Siberski, & Nejdl, 2009; Bobed, Trillo, Mena, & Ilarri, 2010), where their query languages are complex. Although the research of structured retrieval has been relatively new compared to these two fields, there are quite a number of works on query transformation that have been carried out in the field of structured retrieval. Petkova et al., 2009; J. Li et al., 2009; Tannier, 2005; Woodley & Geva, 2004 auto construct structured queries from keywords queries. Others like Bao et al., 2010; Kim et al., 2009 interprets and find structures from keywords queries (used directly for retrieval without writing as structured

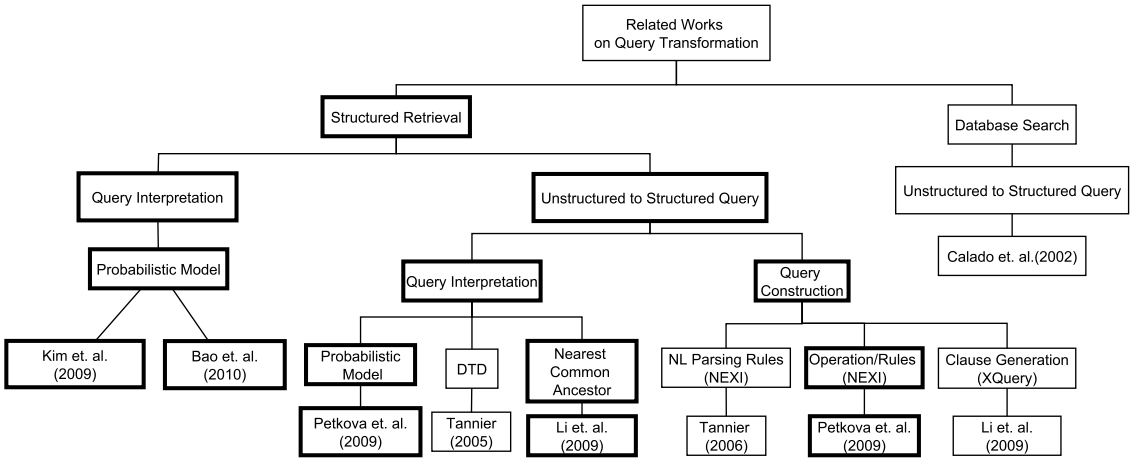


Figure 2.6: Related Works of Query Transformation in Structured Retrieval Environment

queries) (see Figure 2.6).

Follow, we will discuss two main features involved in query transformation, highlighting the methods used by current approaches.

2.2.1 Interpreting Unstructured Query

In query transformation, there are two types of structure/concept to be interpreted, one is the structure that reflects the concept of a term. Second is the structure that defines the type of information to be returned to user. Therefore, if we can select a correct concept or structure to differentiate the meaning or role of a term, it will greatly improve the relevancy of result. Such selection requires the ability to differentiate the structures when ambiguous situations occur. Also, if we can find the correct target concept or structure, it will improve the relevancy of result as well as the interface of the result, with a better information size. To achieve this, a term is often pre associated with a structure or a structure path based on usage in a collection. The structures priorities are ranked based on methods like probabilistic model (Petkova et al., 2009; Kim et al., 2009; Bao et al., 2010), or defined using nlp rules (Tannier, 2005). There are also works that look up potential structures during query time, and decide relevant structures based on similarity measures (J. Li et al., 2009; Hsu et al., 2004).

2.2.1 (a) Term-Structures Association

The first step of obtaining a structure for a term is by associating them before query time. The common associations created are between a term and a single structure, which is its immediate parent node (Kim et al., 2009). Association can also be performed during query time. According to Petkova et al., 2009, a term will be associated to all its parent structure, but only the one best structure will be considered during query interpretation. Similarly, in Hsu et al., 2004, a term is associated to the structure path known as context path. For J. Li et al., 2009, a term is associated to a subtree of structures. Compare to a single structure, path and subtree are more detailed, and provide a better context during query reasoning. After term and structures associations are formed, they can be further weighted before query time (see next section), or ranked during query time.

2.2.1 (b) Term-Concept Weighting

A prediction of what a term means (or its association with a structure) can be inferred using some term scoring methods such as term frequency probabilistic model (e.g. TFIDF, BM25), language model etc. These models are commonly used for element/document term scoring in xml or information retrieval. For this purpose, extension is made on element/document term scoring algorithm to enable the weighting of term based on unique structures or better known as concept. This results in a set of possible ranked concepts (structures) based on the collection's statistics. The prediction works reasonably within collection using the same schema/dtd, or in the case where there is none or little ambiguity of a term or different roles of an entity. Although the prediction may give us a ranked prediction of a term, such as "singapore" is an organization, address, country and so forth, in general, this set of structures resembles a list of meanings in a dictionary without senses.

When it comes to interpretation of query term, having access to the set of structures

is insufficient as we are still lacking hints of which structure to pick. Therefore, it is necessary to include details that decide whether a structure is relevant to the term contextually in addition to its overall occurrences. Including context analysis is beneficial in text centric collections, where existences of structures with large context variables (complex taxonomy) are largely seen.

Here, we show how concepts are generated for a term in general (see algorithm 2.1). Assume a collection with a set of terms and structures, where each unique structure is known as concept. A term in the collection is associated to a number of concepts, $c_i \subset C = \{c_1 ... c_n\}$. Let us denote by D_i the collection of elements associated with concept c_i, $D_i = \cup_{k=0...mi} e_{i,k}$.

The scoring of concepts are carried out in two levels, first the element level, and second, the concept level. For element level, concepts for term are first scored based on individual elements related to the concepts. Various scoring models (Wang, Li, & Wang, 2007) (denoted as *Scoremodel* in algorithm 2.1) can be used in term weighting for elements, ranging from basic Term Frequency Inverse Document Frequency (TFIDF) (Cohen et al., 2003), unigram language model to more complex one with length normalization like BM25 (Robertson & Zaragoza, 2009), and hierarchial weighting like hierarchical language model (Ogilvie & Callan, 2004). In query transformation works that perform concept weighting, Petkova et al., 2009 adopts the unigram language model for its term element weighting while Kim et al., 2009 proposes the use the hierarchical language model.

At the second level, scores for elements of the same concept are combined (denoted as *Conceptmap* in algorithm 2.1) to form a single term weighting for the concept. Mapping elements score to concept can be carried out by extending term weighting method for individual elements to its type, i.e. generalizing elements by their types (Petkova et al., 2009; Kim et al., 2009).

Algorithm 2.1 CONCEPT GENERATOR

 # get element level scoring
for $c_i = 1 \rightarrow n$ **do**
    # get elements subsets for each concept
    $Element(c_i) \leftarrow \cup_{k=0...mi} e_{i,k}$
    # generate term weight for elements
    $Element(c_i) \leftarrow termweighting(Scoremodel, Element(c_i))$
end for
 # get concept level scoring
for $c_i = 1 \rightarrow n$ **do**
    # merge elements score for each concept
    $Score(c_i) \leftarrow conceptweighting(Conceptmap, Element(c_i))$
end for
 # rank concepts
$ConceptList \leftarrow conceptrank(Score, Rankmethod)$ **return** $ConceptList$

From the generated concept list (denoted as *Conceptlist* in algorithm 2.1), concept selection for the term can be carried out based on first of the ranked concepts, or a subset cut off by a threshold. In Bao et al., 2010, author also suggests to let user intervene to select a concept for query.

The weighted concepts for term in a collection are then used in the interpretation of terms given in a query. These concepts are used extensively in two manners, i.e. in finding an overall target for a query and finding concepts of terms used in the query.

2.2.1 (c) Query's Target Concept

During query analysis, a concept can be used to define the overall query's scope or focus. For example, in fragment query by Carmel et al., 2003, a target concept signifies the component type expected as target result, e.g. <target>book</target>. In NEXI Content and Structure (CAS) query by Trotman, 2009, a target concept defines what to be returned to user.

Thus, in query transformation, finding target is part of the process to form a complete structured query. In Petkova et al., 2009, target for a query is obtained from concepts of query terms via a set of operations (i.e. expand, aggregate and order). J. Li et al. utilizes the root node of subtree (known as master entity) associated to its query terms

to obtain the target. Using a different approach, Hsu et al. selects concept nodes using a context analysis method to form its structured query. Nodes selection is carried out by exploring structure paths of query's terms based on semantic distance of query terms on the document structure. Bao et al. also proposes that an effective keyword search in xml search should be able to identify the correct type of the target node(s). This work uses two factors to select the target node, i.e. the frequency of target node and the depth of the target node in a document.

2.2.1 (d) Query's Term Concept

Besides identifying target concept for a query, a concept is used to constraint the meaning of terms in a query. For example, when a term is used in various kind of elements, indicating a concept will restrict the query to a specific type of elements. Collection-based probabilistic methods are often used in the selection of the most relevant concept for a term based on collection statistics, e.g. Petkova et al., 2009 and Kim et al., 2009 uses unigram language model to determine the most relevant concept for a term. When collection-based frequency is insufficient, Bao et al., 2010 incorporates node type (equivalent to our concept) frequency ($C_{via}(T,q)$), with an additional factor known as In Query Distance (IQD), utilizing keywords distance within a query in its concept selection.

Other Query Interpretation Methods Although there are additional methods that assist in query interpretation using linguistic methods like linguistic parsing (Bilotti, Ogilvie, Callan, & Nyberg, 2007), semantic role labelling (Zhao & Callan, 2009), etc. we focus on works that carry out query interpretation using structures provided in documents.

2.2.2 Getting Information Needs Across

An important issue in structuring a query is to get the contents across from the unstructured form to structured form, so that both queries are as similar as possible in its information needs. As such, it is important that information needs specified in the source query are understood well, before an equivalent target query can be formed. One way is by getting the most out of the keywords used, via their combination, implicit hint, explicit hint. For example, looking at its combination will assist us in understanding the user's intention, or context of query. In query interpretation, query context is important hint leading to concepts selection. Query context has been a popular approach used in improving IR works (Bai, Nie, Cao, & Bouchard, 2007, Chi, Ding, & Lam, 2002).

Besides, source query also contain implicit hints, that can be used in differentiating the type of keywords used. If we look at a topic *2011104* in INEX data-centric track (Wang, Ramírez, Marx, Theobald, & Kamps, 2011), a query "movie Ellen Page thriller" implicitly contains both content and concept keywords, i.e. Ellen Page (content keyword), thriller (content keyword) and movie (concept keyword). To determine the type, Petkova et al. uses a thesaurus to determine whether a keyword is content or concept. Being able to identify keyword's type gives us better view of possible concepts a user is looking for.

In addition, for a source query that describe its needs in a explicit manner, like topic *219* in INEX Natural Language Query (NLQ) Track, "Find sections that discuss the granularity of learning objects". Natural Language Processing (NLP) parsing method is used to differentiate the type of keywords, i.e. sec (concept keyword), learning objects (content keyword) and granularity (content keyword) in Tannier, 2005.

Query Ranking Sometimes, an unstructured query may be transformed to multiple structured queries. This happens especially when probabilistic methods are used to suggest possible structures for query. Depending on the diversities of structures available

from collections, a query may be related to more than one structures, hence results in multiple structured queries. When this occurs, queries are ranked by its structures relevance scores, such that the most popular structures w.r.t. a query gets the highest rank and so forth. J. Li et al., 2009 proposes a dynamic plan to obtain top-k results by evaluating each generated structured queries by its rank until no further relevant results are obtained.

2.2.3 Forming Structured Queries

Out of the many related works, three shows full construction of structured queries, i.e. Petkova et al., 2009; J. Li et al., 2009; Tannier, 2005. Petkova et al., 2009 shows a full transformation from keywords query to NEXI query language. In this work, the inferred structures and keywords from source query are constructed into NEXI query using a set of operations, i.e. expansion, aggregation and ordering. For example, the aggregation function is to form a single NEXI target by combining two targets with the same structure.

Similarly, Tannier, 2005 also attempt to construct a NEXI query, but using a natural language query approach. Rules are defined for certain linguistic patterns of query description, such as "c2 discusses c4" is mapped into about(c2, c4). For this transformation, it is important that the natural language query follows the predefined rules.

Two types of inputs have been proposed in this paper, i.e.:

Rule 1. **Simple noun phrase [NP $\rightarrow$ (ADJ—NOUN)+ NOUN]** This input consists of combination of adjectives (or noun) followed by a primary noun. E.g. "semantic networks" (ADJ NOUN).

Rule 2. **Complex noun phrase [NP $\rightarrow$ NP (PREP NP)+]** This input consists of noun phrases linked by prepositions. E.g. "history of Artificial Intelligence".

These queries are then represented in a semantic representation that will automatically convert to NEXI query.

J. Li et al., 2009 constructs XQuery from keywords query. This work requires keywords in a query to be specified in the form of *label:term* pair, e.g. "year:2006 author:Philip title:xml". Subtrees are identified based on schema and query labels, e.g. "$\{year, \{title, author\}_{book}\}_{bib}$", where *book* and *bib* are nodes of subtrees.

Two types of clauses of XQuery are generated, i.e.:

FOR Clause A set of FOR clauses according to the subtrees. E.g. "For $b in bibliography/bib" for $\{\}_{bib}$ and "For $b2 in $b/book" for $\{\}_{book}$.

WHERE Clause A set of WHERE clauses for the FOR clause. E.g. "Where $b/year='2006' and contains($b2/title, 'xml') and contains($b2/author, 'Philip')".

2.3 Issues in Query Transformation

In the previous section, we have gone through works related to some important aspects of query transformation. Now, we discuss the issues of the methods used in current works w.r.t. the goals of this thesis.

2.3.1 Information Utilization from Query Side

In an unstructured query, whether it is keywords form or short description, it may consist of some structural keywords. Our first issue discusses whether current works are able to exploit these structural keywords effectively. When structural keywords are used in a query, they can be tricky as the keyword can be meant as what it tries to look for, such as "article", or it can meant for constraining a keyword, such as "author". The issue is that we need to be able to identify those keywords if they are used. However, there are some limitations with the current solutions.

Petkova et al., 2009 proposes the usage of structure thesaurus to identify structural keywords used in a query. However, it has limitation of identifying a keyword used as constraint. It only applies when a specified structure is meant to be a query target. If a structural keyword is used for constraining a term to "author" instead of "editor" in "dblp

author andrew trotman", it could be mistakenly interpreted as target (see result i. from Step 3 below).

1. Query splitting and target bindings

dblp $\rightarrow$ structure keyword $\rightarrow \{//dblp\}$

author $\rightarrow$ structure keyword $\rightarrow \{//author\}$

andrew trotman $\rightarrow$ content keyword $\rightarrow \{//author[$ "andrew trotman$''] \}$

2. Expansion operation.

$\{//dblp\} \rightarrow \{//dblp\}$ (note: no expansion)

$\{//author\} \rightarrow \{//dblp//article//author\}$

$\{//author[$ "andrew trotman"$]\} \rightarrow \{//dblp//article//author[$ "andrew trotman$''] \}$

3. Aggregation operation.

$\{//dblp\} + \{//dblp//article//author\} + \{dblp//article//author[$ "andrew trotman$''] \}$

$\rightarrow$

i. $\{//dblp//article[.//author[$ "andrew trotman$'']]//author\}$

ii. $\{//dblp//article[.//author][.//author[$ "andrew trotman$'']]\}$ (Invalid NEXI query)

Compare to Petkova et al., 2009, Bao et al., 2010 takes a step further in identifying both target (known as *search for node*) and constraint (known as *search via node*) structures used in a query. It identifies a constraint structure by using in query distance (*IQD*) method to find pairs of structure and keyword. And, it utilizes structure specified as target in finding the correct subtree. However, it has a limitation if the target structure happens to be contained within a subtree. For example, a query looking for url of articles where title contains xml can be written as "url article title xml". Based on the contents of tree structure in Figure 2.7, structures identified are shown below.

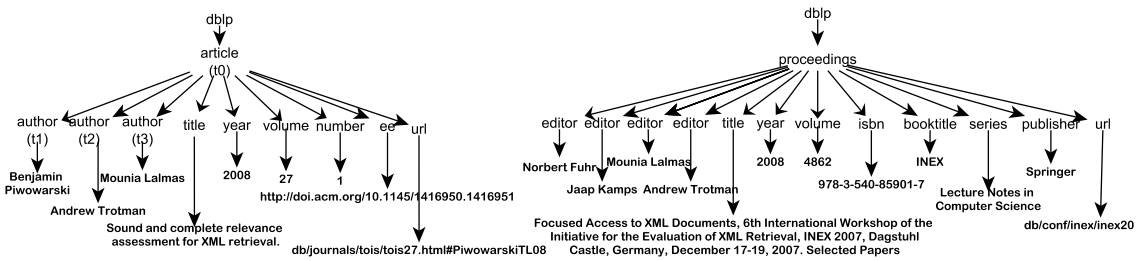


Figure 2.7: Examples of Document Tree of DBLP XML

1. Infer *Search for* Node

$$\begin{matrix} \text{“}url\text{”} \\ \text{“}article\text{”} \\ \text{“}title\text{”} \\ \text{“}xml\text{”} \end{matrix} \rightarrow \text{“}article\text{”}$$

2. Infer *Search via* Node

“xml” $\rightarrow$ “$title$”

Using Bao et al., 2010's method, we are not able to get “url" as what the query has targeted. From these two works, we can see that the current approaches still have limitation in exploiting structural keywords effectively.

In J. Li et al., 2009, we see that the step of structure keyword identification can be omitted by letting user specifies them in structure:keyword form, such as “author:David title:XML". Similar to J. Li et al., 2009, Tannier, 2005 also requires the structure to be specified in certain way so that it can be parsed into a corresponding structure and keyword template. Although these two works proposed a simpler way structural keywords identification, the main drawback is their input queries have become more rigid. In this thesis, we are interested to explore into a looser form of query instead.

2.3.2 Information Utilization from Collection

Besides being able to utilize keywords in query effectively, the knowledge from document structures in collection is also very important in suggesting good structures for constructing structured query. However, it is easier to find a good structure when the

document structure of a collection is simpler, e.g. the tree structure is shallow, its contents are data centric, and its elements are homogeneous. When a collection is complex, issues like element granularities and term ambiguities will arise. We present two main problems that occur when structures of a collection are nested and heterogeneous.

Nested and Heterogeneous Structures In query interpretation, one way to locate relevant structures for constructing structured query is by finding common subtree of the query. Smallest Lowest Common Ancestor (SLCA) is one of most widely used approaches to find a common subtree given a set of keywords in a query. For example, the root of the subtree is used as the target of a query, e.g. "workshop", "tutorial" etc. The main idea of SLCA is to find a smallest subtree that contain all the keywords used in the query. Smallest subtree means that there is no other subtree (that also contain all the keywords) within SLCA (Xu & Papakonstantinou, 2008; J. Li et al., 2009). However, this approach has a limitation when the desired query's target is not the ancestor.

Consider a query, "andrew trotman jaap kamps", that is looking for any cooperation between these two persons that can be a paper, a workshop, a tutorial. When objects are nested, SLCA select the lowest common node, i.e. organizers, which is too small as a answer of an exact element. In this case, the node with concept "workshop" is preferred.

Different from SLCA that looks for subtree, Petkova et al., 2009 uses structure expansion, aggregation and ordering operators to find a common concept based on schema. Refer to the same query, Petkova et al., 2009 only manage to obtain "organizer", which is slightly poorer than SLCA. This is because common concept on a schema can be linked to two different nodes on a physical document tree.

1. Structure expansion operation.

andrew trotman $\rightarrow \{//name[$ "andrew trotman$''] \} \rightarrow$

$\{//organizer//name[$ "andrew trotman$''] \}$

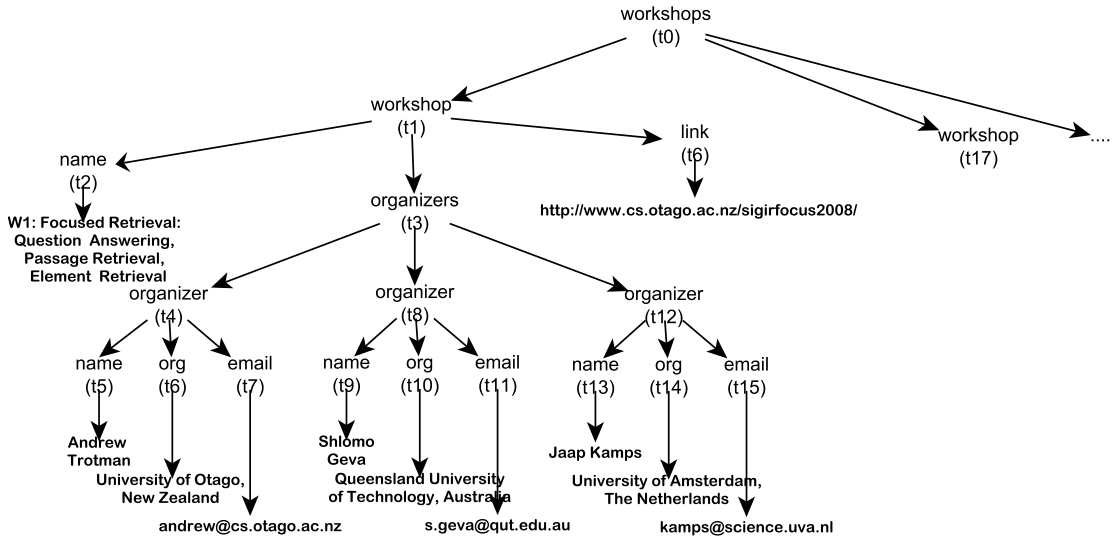


Figure 2.8: Part of Document Tree of a Nested Structure XML

jaap kamps $\rightarrow \{//name[\text{ “jaap kamps}'']\} \rightarrow \{//organizer//name[\text{ “jaap kamps}'']\}$

2. Structure aggregation operation.

$\{//organizer//name[\text{ “andrew trotman}'']\} + \{//organizer//name[\text{ “jaap kamps}'']\}$

$\rightarrow \{//organizer//name[\text{ “andrew trotman}''][\text{ “jaap kamps}'']\}$

We can see that both scenarios are suggesting a concept which is too small or too low. This is partly because current methods are used for collection with simple document structure. When a complex collection is used, they would not be able to consider higher level structures in the hierarchy.

2.3.3 Query Construction Methods and Outputs

Current works transform a source query to a query language form using two different approaches, by using query construction algorithm (Petkova et al., 2009; J. Li et al., 2009) or query parsing & templates mapping (Tannier, 2005; Woodley & Geva, 2006).

In Petkova et al., 2009, its query construction algorithm finds and transforms structures (known as targets) for a source query to form a formal XML query. The construction rules are designed for transformation into a specific query type, i.e. NEXI. Similarly, J. Li et al., 2009 uses a structured queries construction algorithm to generate clauses of

XQuery. Structures (known as master entities) are generated into XQuery clause through steps like,

...
3: for all each master entity $v_m \in V_m$ do
4: Generate FOR clause with v_m, i.e. "For $x in r/.../$v_m$";
...

Tannier, 2005 performs linguistic analysis to infer structure for NLP query. Analysis steps like, Part-of-Speech (POS) parsing, specific rules reduction and structure parsing are used to identify patterns of keywords used in a query (Tannier, Girardot, & Mathieu, 2005). Specific rules per collection were also proposed to allow recognition of more precise query expressions for particular domain. This approach requires information needs to be specified with correct linguistic structure in order to be mapped to the form of query language structure.

Woodley & Geva, 2006 carries out the transformation by using a set of predefined query templates obtained from previous NEXI topics. Tagged query is then matched with the predefined templates. For a better tagging of source query, special connotations are used to find types of words used in a query like structures requirements (e.g. section, abstract), boundaries separating structural and content requirements (e.g. contain, about) and instructions that indicate target to be return (e.g. find, retrieve). As NLP queries are very diverse in nature, this approach requires consecutive extension of special connotations.

Currently, most query transformation framework is designed for its intended structured query language. For example, algorithm proposed by J. Li et al., 2009 construct a XQuery language. Since current solution remains individualized, we are interested in a generic framework instead, which can be achieved by separating the query interpretation outcome from the query construction of a particular query language. A generic framework is desirable as a single and unified solution, that could easily scales to accommodate

interpretation methods or even structure query language types.

2.3.4 Issues Summary

To give an overall picture of the issues of literatures discussed, the features of methods/approaches used are summarized in Table 2.2. From the table, we can see that current solutions have not addressed query transformation as a complete solution. This limitation will be addressed in our proposed solution with the aim to provide a unified solution within single framework.

Table 2.2: Features of Existing Query Transformation Works

Features of Query Transformation	J. Li et al., 2009	Petkova et al., 2009	Kim et al., 2009	Bao et al., 2010	Tannier, 2005	Woodley & Geva, 2006
Information utilization from query side						
Identify target structure (Thesaurus)	no	yes	no	no	yes	yes
Identify target structure (Template)	no	no	no	no	yes	yes
Identify constraint structure (IQD)	no	no	no	yes	no	no
Identify constraint structure (Template)	yes	no	no	no	no	no
Information utilization from collection (method used in bracket)						
Suggest new constraint structure (Parent Node Binding)	no	yes	yes	yes	no	no
Suggest new constraint structure (Ancestor Node Binding)	no	no	no	no	no	no
Suggest new target structure (SLCA)	yes	yes	no	yes	no	no
Suggest new target structure (Subtree Frequency)	no	no	no	yes	no	no
Query construction methods/outputs						
Use operators/algorithms to construct query	yes	yes	-	-	no	no
Use linguistics rule/templates to construct query	no	no	-	-	yes	yes
Construct NEXI query	no	yes	-	-	yes	yes
Construct XQuery query	yes	no	-	-	no	no

2.4 Summary

In this chapter, we have presented some backgrounds and related works of query transformation in a structured retrieval environment. We have highlighted three issues where current solutions still lack a comprehensive approach to address the various features required in query transformation. These issues will be addressed in Chapter 3 and Chapter 4 respectively. The methods discussed in the issues will be used as the baselines of our evaluation in Chapter 5.

CHAPTER 3

A FLEXIBLE QUERY TRANSFORMATION FRAMEWORK

In this chapter, we introduce our framework for the study of query transformation, and present it in a formal manner. Such formalization is important for the purposes of further reusability, development and comparison of the framework. This chapter is divided into four sections. We start off by specifying the basic requirements of query transformation framework. Then, we define the framework for query interpretation and construction in structured retrieval environment. Following this, we describe a probabilistic approach called context-based term weighting to capture collection-based knowledge for query interpretation. And last, we define a novel query representation that is used to capture the interpreted query as well as the mappings required to form the final structured query.

3.1 Requirements for Query Transformation

Unstructured Query, Q_U An unstructured query is a query written by user, describing his information needs for the purpose of searching or finding some information from a specific domain. An unstructured query, Q_U can be of multiple types, e.g. keywords, phrase, incomplete questions, etc. We classify them into two broad categories, either a keyword query or a descriptive query (i.e. for phrases, partial sentence, question etc.), $Q_U = \{Q_{U_{keyword}}, Q_{U_{descriptive}}\}$. Each Q_U consists of a set of term, qt, given as $Q_U = \{qt_i : 1 \leq i \leq n_{qt}\}$, where n_{qt} is the total number of terms. The terms used in unstructured query of keywords type do not required to be ordered, whereas terms used in descriptive type can be ordered using natural language. A term is a lexical unit containing a single or multiple words, conveying a single meaning such as river view hotel, address, gabriella kazai etc.

Structured Query, Q_S A structured query is a query language written by expert user to retrieve structured resources such as XML or databases. In this context, we refer structured query as those queries meant for structured retrieval only, i.e. XML documents or data. Although these queries are more expressive for searching resources in structured domain, they are complex, thus not meant to be used by end user. A structured query, Q_S, ranges from path-based, concept-based and fragment-based, $Q_S = \{Q_{S_{path}}, Q_{S_{concept}}, Q_{S_{fragment}}\}$. Each Q_S is a string.

Domain, D and Element, E A domain, D, is a set of structured resources or documents. A domain can be a collection of XML in the form of web sites (e.g. conference site), records (e.g. bibliography), objects (e.g. actors, directors), articles (e.g. Wikipedia page) etc. Consider a domain of structured resources, D, the main objects of interest are elements, e, featuring different granularities of contents of these resources, denoted as $D = \{e_i : 1 \leq i \leq N_e\}$, where N_e is the total number of elements.

There are two kinds of information in an element, i.e. its structure and its content. The structure (also referred as markup in XML and Hyper Text Markup Language (HTML) or annotation in Resource Description Framework (RDF)) of an element is a descriptive term describing its contents. We denote it as E_S. Whereas, the content of an element usually comes in the form of an informative text, which ranges from paragraphs and sections of description, to entities and names like person's name, news title, web site url etc.

There are two types of elements, i.e. a simple element or a nested element. A simple element is obtained from the leaf node of a structured resource tree. It is represented with one structure and one piece of content, e.g. <tel>+65 6733 0880</tel>, <add>392 Havelock Road, Singapore 169663</add>, <url> db/conf/aaai/aaai2008.html </url>. A leaf element is the smallest element in D. A nested element is obtained from ancestor

of leaf node. It is represented with one structure and concatenation of its descendant elements, such as a <book><name>An Introduction to Information Retrieval</name> <author>Christopher Manning</author></book>.

3.2 Formal Semantics of Query Transformation

The main purpose for query transformation in a structured retrieval environment is the ability to transform a source query to a target query which has been improvised to suit the problem of the environment. Given a domain, D, a target query, Q_S is a structured query, that will be submitted to a retrieval system of the domain to retrieve resources as specified in the query. Examples of structured query are NEXI (Trotman & Sigurbjörnsson, 2004a), XSearch query (Cohen et al., 2003) and XML fragment query (Carmel et al., 2003). A source query, Q_U, is an unstructured query issued by users to look up information within the domain. An example of unstructured query is keywords query. A query transformation, F is a process that converts Q_U to Q_S, using domain knowledge of D.

Flexible Query Transformation, F Traditionally, query transformation is a one to one process where transformation rules are meant for transformation from one source query type to one target query type. This kind of transformation lacks flexibility in accommodating new structured query. We propose a flexible query transformation model, F where a single Q_U can be mapped to one or more Q_Ss. Our goal is to enable easier adaptation of a new Q_S by generalizing the transformation between Q_U and Q_S pairs using a more generic framework that separates the formulation of queries (i.e. syntax) from their information needs. In this flexible query transformation framework, we introduce an intermediate query structure, I, which is a representation for expressing the query's information needs (in terms of both contents and structures) so that such needs can be converted between queries without loss of meaning.

Now, let us define the query transformation framework in a formal manner. Let D be the collection where the retrieval is to be conducted. The source query of transformation process is unstructured query, Q_U, while the target query is structured query, Q_S. The transformation process requires the knowledge of D for query interpretation. First, we describe the specification of the input query.

3.2.1 Unstructured Query Specification

In query specification, we assume that the user who formulates Q_U understand the domain of D.

Assumption User must understand the domain in order to avoid semantic discrepancy of the seek information.

Definition 3.1. *(Query Term) An unstructured query can be specified as a list of terms, qt, where each term may consist more than one keywords, kw, where each keyword is a string.* $\forall qt \in Q_U.\ qt = \{kw | kw > 0, kw \text{ is string}\}$.

Two types of term can be used in the query, i.e. content term and structure term.

Definition 3.2. *(Query Content Term) Query content term,* $qt_{content}$*, is a term in query that indicates the information or data the query wants to retrieve. A query content term corresponds to the content or data of an element,* E_C *in structured collection.*

Definition 3.3. *(Query Structure Term) Query structure term,* $qt_{structure}$*, is a term in unstructured query that indicates the structure or tag for the information the query wants to retrieve. A query structure term corresponds to the structure or tag of an element,* E_S *in structured collection.*

For example, for a collection written in XML, a content term in query corresponds to the content or data of XML element, while a structure term in query corresponds to the tag or structure of XML element.

Axiom 3.1. *An unstructured query may contain both content term and structure term. Or it may contain content term only. However, it cannot contain structure term only.* $Q_U = \{qt_{type} | qt \geq 1, type = structure \cup content, qt_{content} \geq 1\}$

Where structural keywords are used in query, user may refer to schema or DTD of collection for specification of structures if there is any. Additional thesaurus that expands the vocabulary of the structures will improve the structural keywords specification. Otherwise, user may omit the usage of structure keywords in query.

In the case where structural keywords are used in a query, there are two ways where the query can be optimized. First, user may give a structure keyword indicating intended element, such as “article”, “paper”, “hotel”. Second, user may give a structure keyword indicating the meaning of the content keyword, such as “author” for content “andrew trotman”, “hotel” for content “river view”.

By default, an unstructured query is considered as a single intention query, where its intention is determined by the conjunction of terms used in the query. However, there is case where there are multiple sub intentions that occur within a main intention, such as “tutorial or workshop by andrew trotman”. Therefore, logical operator can be used to express such intentions, which are disjunctive. However, this specification is optional as user can leave the task of deciding the intentions to the framework.

3.2.2 Query Interpretation Requirements

Given an unstructured query in the above form, its interpretations, is determined by the semantics of the query itself as well as the semantics of the collection. We refer these semantics as query context and collection context. The reason behind why an interpretation requires both query context and collection context is that the former reflects what user requires while the latter reflects what collection could offer. For instance, given a query that looks for paper written by andrew trotman (query context), it can be inter-

preted as workshop paper or a journal paper or a conference paper (collection context). We observe that each interpretation refers to different part of structures used in collection. We formalize the requirements of query interpretation as follows.

Before a query can be interpreted, knowledge for interpretation is required. The source of knowledge is the structured document from collection, D. Given collection, D, first, we define the source of knowledge, and then we define the knowledge required for interpretation, called context interpretation.

Structured Document In this framework, structured documents can either be created based on a schema or without one. Although most structured resources contain both logical tags and descriptive tags in its contents representation, we focus on the latter as these tags reflects concept or meaning that can be used for query interpretation.

Assumption There exist descriptive structures (i.e. markups, tags or annotations) in a structured document, i.e. for the purpose of meaning enrichment, classification, content representation, rather than logical, which is meant for typesetting or presentation. The set of descriptive structures forms an information structure that resembles a knowledge representation like taxonomy that can be used for domain specific reasoning.

Definition 3.4. *(Structured Document) A structured document is a rooted, acyclic graph defined as* $G_{doc} = (V, ED)$*, where* V *is a set of nodes which can be either a structure or a content,* $V = \{v : v \in v_{content} \cup v \in v_{struc}\}$ *. In* G_{doc}*, its root node and all intermediate nodes are structures,* v_{struc}*, while its leaf nodes,* $v_{content}$ *are contents or data.* ED *is a set of directed edges representing relationship between two nodes. Likewise, there are two types of edges, one represents the relation (instance of) between a content node and a structure node, and one represents the relation (subclass of) between two structure nodes. We denote the former as* $v_{content} \rightarrow v_{struc} \in ED$ *where* $v_{content}$ *is a child and* v_{struc} *is a*

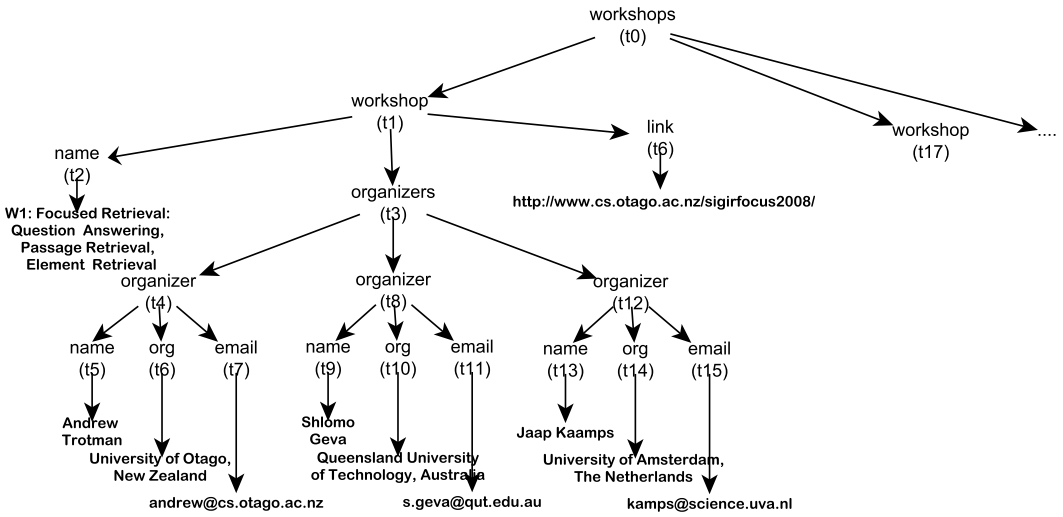


Figure 3.1: Partial Document Structure of A Structured Document from Conference Collection

parent node of $v_{content}$*. On the other hand, the latter is denoted as* $v_{struc_i} \rightarrow v_{struc_{i+1}} \in ED$*, where* $v_{struc_{i+1}}$ *is a parent node of* v_{struc_i}*.*

Example 3.1. *An excerpt of structured document about conference workshops from a conference site is shown in Figure 3.1.*

Having defined the structured document, now we can proceed to look at the useful items, i.e. term, concept and context, that can be extracted from the content node, $v_{content}$ and structure node, v_{struc} within the document as our knowledge source.

Term A term is a meaningful unit of string obtained from either a content node or a structure node of a structured document, G_{doc}.

Definition 3.5. *(Term) Given a structured document,* G_{doc}*, a content term, ct, is a term obtained from content node,* $v_{content}$*. A structure term, st , is a term obtained from structure node,* v_{struc}*. ct can be a single word or a phrase obtained from term parsing method, whereas st is required to refer to the exact string of* v_{struc}*.*

Concept In a structured document, the meaning of a term can be observed through the relation between a term (content node) and its structures (structure node). As such, we

can obtain the prediction of what a term means, by capturing the relationships between the term and its structures. Different from thesaurus, the meaning of a term are reflected through the usage of structures (including tags, markups, annotations) in the tree. We call these structures as concepts. A concept is structure (when perceived in a *meaningful* manner) giving an idea of what a term is about. We use the word *concept* to refer to the type (or class) of a structure, e.g. name, hotel, article etc. and the word *structure* to refer a unique physical unit of a structure term, or structure node.

Definition 3.6. *(Concept) Given a structured document, G_{doc}, a concept, cpt, for a content term, ct, is a structure obtained from structure node, v_{struc} of G_{doc}, where v_{struc} is an ancestor of content node, $v_{content}$ containing ct.*

Context A context defines a specific condition of where a concept is used. Context may not be significant in collection where its documents have homogeneous structures, due to the simplicity and the size of the information. However, in collection where documents contain heterogeneous structures, there may be different parts in a document that presents information of different kinds. Hence, when a document contains many different parts of information, it has become not meaningful if these parts are treated as the same type under the same document. Dividing document into contexts overcomes this by classifying parts of the same type under the same context.

Definition 3.7. *(Context) Given, a structured document, G_{doc}. A context, ctx, for a content term ct and its concept cpt, is a structure obtained from structure node, v_{struc} of G_{doc}, where v_{struc} is an ancestor of structure node, v_{struc} containing cpt.*

The number of contexts is highly depended on the heterogeneity of structures in collection, D.

Proposition 3.1. *A collection, D, may contain one or more contexts. Contexts in hetereogeneous collection, D_{hetero} is higher than contexts in homogeneous collection, D_{homo}.*

$\forall D.CTX_D = \{ctx|ctx \geq 1\}$.

Let x be the unique structure node of collection, total unique structures in heterogeneous collection is higher than homogeneous collection,

$|X_{D_{hetero}}| > |X_{D_{homo}}|$

Since the number of contexts in any collection is propotional to the number of unique structures in the collection,

$|CTX_D|\ \alpha\ |X_D|$

Hence, $|CTX_{D_{hetero}}| > |CTX_{D_{homo}}|$

Knowledge for Interpretation Follow, we define the knowledge required for query interpretation. The basic unit required for interpreting a query is to interpret its term. Here, the knowledge refers to a term and its associated concepts, known as term interpretation. Normally, term interpretation is captured based on entire collection. In this framework, a term interpretation is captured based on contextual view of collection. There are two types of term interpretation, one for content term and one for structure term.

Definition 3.8. *(Term Interpretation for Content Term) A term interpretation for content term, $I_{content}$ for collection, D, is a set of triple (ct, cpt, ctx), where ct is content term of element in D, cpt is concept describing ct, and ctx is the context where ct and cpt is in.*

$I_{content} = \{x|1 \leq x \leq |I|, x = (ct, cpt, ctx)\}$

Each $I_{content}$ has a set of properties, $I_{prop} = \{score_{CTXPROX}, score_{CW}, id\}$, where id is the ref (e.g. ref to an original document or element), $score_{CW}$ is the score to measure importance of ct in cpt and $score_{CTXPROX}$ is the score to measure whether the usage of cpt for ct is popular under ctx.

Definition 3.9. *(Term Interpretation for Structure Term) A term interpretation for structure term, $I_{structure}$ for collection, D, is a set of tuple (st, ctx), where st is structure term (i.e. concept) of element in D, and ctx is the context where st is in. In this case, since st is a concept itself, st and cpt can be used interchangeably.*

$I_{structure} = \{x | 1 \leq x \leq |I|, x = (st, ctx)\}$

Each $I_{structure}$ has a set of properties, $I_{prop} = \{score_{CTXPROX}, id\}$, where id is the ref (e.g. ref to an original document or element) and $score_{CTXPROX}$ is the score to measure whether the usage of cpt is popular under ctx.

In addition, we also define the knowledge that will assist in term identification from an unstructured query. Given a collection, two kinds of thesaurus are created, i.e. content term thesaurus, $THE_{content}$, and structure term thesaurus, $THE_{structure}$.

Definition 3.10. *(Content Term Thesaurus) A content term thesaurus, $THE_{content}$, is a set of lexical items obtained from content node, $v_{content}$ of document, G_{doc}, in a collection, D. Each item is a meaningful string, e.g. word, phrase, numbers, etc. It also contain name entity such as country, name, paper title etc.*

Definition 3.11. *(Structure Term Thesaurus) A structure term thesaurus, $THE_{structure}$, is a set of lexical items obtained from structure node, $v_{structure}$ of document, G_{doc} in a collection, D. Each item is a tag.*

Using thesaurus as knowledge source helps to identify meaningful consequtive keywords in a query, rather than treating them as individual keywords for query interpretation.

Information Needs in Query Interpretation A requirement for the query transformation is to maintain the information needs specified during the transition from unstructured form to its structured form. In order to minimize the information loss during the transi-

Table 3.1: Usage of content and concept keywords in information needs.

Query	Structured Form	Info Needs Keywords	
		Content	Concept
Path-based: NEXI (Trotman, 2009)	//TARGET_PATH [about(FILTER_PATH, FILTER_TERM) (e.g. *//movie[about(.//title, Avatar) AND about(.//director, James Francis Cameron)]*	FILTER_TERM (e.g. *Avatar, James Francis Cameron*)	TARGET_PATH, FILTER_PATH (e.g. *movie, title, director*)
Concept-based: COMPASS (Graupmann et al., 2004), XSEarch (Cohen et al., 2003)	CONCEPT=VALUE (e.g. *author=Tolstoy*), LABEL: KEYWORD, LABEL: or : KEYWORD (e.g. *authors: Kempster : Stirling*)	VALUE (e.g. *Tolstoy*, KEYWORD (e.g. *Tolstoy*)	CONCEPT (e.g. *Kempster, Stirling*), LABEL (e.g. *authors*)
Fragment-based: XML Fragment (Carmel et al., 2003)	<CONTEXT>TERM </CONTEXT> <TARGET>CONTEXT </TARGET> (e.g. *<book><year>1973</year> <title>Search</title> </book> <TARGET>book</TARGET>*)	TERM (e.g. *1973, Search*)	CONTEXT (e.g. *book, year, title*)

tion, it is necessary to find the common information needs features that could bridge both queries type. Hence, before we start interpreting a query, we must know what kind of contents that we need to achieve from the interpretation.

Considering various structured queries form as in Table 3.1, we see that information needs can be specified as content needs and concept needs. The content needs are keywords indicating the information user would like to seek. Concept needs are keywords containing the content keywords to a narrower subset of results based on categories, types, kinds, roles, topics etc. The concept needs in a query can be further classified into target concept and constraint concept. A target concept is used to focus the query to a certain concepts only. For example, setting "workshop" as a target concept results in elements of type "workshop" only. A constraint concept is used to refine a term, instead of a query. For example, "organizer" in *//workshop[about(.//organizer, andrew trotman)]* and "presenter" in *//workshop[about(.//presenter, andrew trotman)]* will return different results due to the constraint settings. Having understand this, we have concluded that three types of information needs need to be interpreted during query transformation, as stated below.

Definition 3.12. *(Query Target) A query target, , $target_{cpt}$, of an interpreted query is a structure that defines the type of element to be returned as a result.*

Definition 3.13. *(Query Constraint) A query constraint of an interpreted query is a con-*

tent and structure pair that constraints the elements to be returned. It consists of two parts, i.e. the content term, $constraint_{ct}$ and the structure that constraints the content term, $constraint_{cpt}$. $constraint_{ct}$ defines the content terms contained in elements to be returned as a result. $constraint_{cpt}$ defines the type of structure for content terms in the query.

Context-based Query Interpretation Having stated the knowledge for interpretation, and the types of contents that need to be interpreted, we proceed to define the output of query interpretation. First, we define the context sub graph which is the initial form of query interpretation. It consists all possible interpretations for every terms in a query. Second, we describe how information needs of an interpreted query are captured via unit called *query interpretation*.

Definition 3.14. *(Context Sub Graph) Consider a set of interpretations, I, for all the terms in a query, Q_U. An interpretation sub set with unique context is denoted as I_{CTX}, where $I_{CTX} \in I$. A context sub graph for I_{CTX}, is given as SG_{CTX}. SG_{CTX} is a rooted directed acyclic graph, $SG_{CTX}(V_{SG}, E_{SG})$, such that:*

- the root node, $V_{SG_{root}}$ is the ctx of I_{CTX}

- the leaf node, $V_{SG_{leaf}}$ is either a content node, ct, or a concept node, st, of I_{CTX}

- the intermediate node, V_{SG} is cpt of ct from I_{CTX}

- E_{SG} is a subset of E interconnecting the nodes in V_{SG}

Example 3.2. *Consider a query, "andrew trotman jaap kamps" that looks for any outcomes by these two person on a conference collection. The relevant context sub graphs wrt. this query includes "workshop" sub graph, "paper" sub graph, "poster committee" sub graph. If we looked another query, "andrew trotman focused retrieval", the relevant context sub graph w.r.t. this query may also have similar root "workshop" but with a dif-*

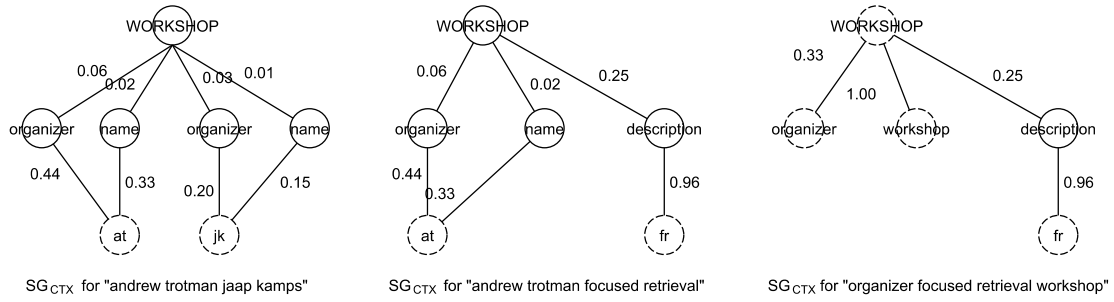


Figure 3.2: Context Sub Graph Examples

ferent set of constraint concepts. If structural term is used in the query, e.g. "organizer" and "workshop" in "organizer focused retrieval workshop", they can be reflected in the context sub graph as well.

Constraint Concepts for Query Interpretation Constraint concepts for query interpretation can be obtained from context sub graph, SG_{CTX} of the query. Here, we define the possible candidates of concepts that can be selected as constraint concepts.

Definition 3.15. *(Constraint Concept Candidates) Consider a context sub graph, SG_{CTX}, of an unstructured query, Q_U. A constraint concept candidate, $constraintCand_{cpt}$, for a content term in Q_U is a concept node for the content term in SG_{CTX}*

$\forall ct \in Q_U, constraintCand_{cpt} = \{V_{SG} \in SG_{CTX}\}$

where V_{SG} is cpt and $V_{SG_{leaf}}$ is qtct and $E(V_{SG}, V_{SG_{leaf}})$ is the edge connecting both nodes.

Constraint Concept Weighting The weight for a constraint concept of a term,$score_{CW}$ is a real number in the range of $[0,1]$ obtained from the term weighting function for concept, $score_{CW}(cpt, ct)$.

$$\forall ct, score_{CW}(cpt, ct) : constraintCand_{cpt} \rightarrow score_{CW}$$

Target Concepts for Query Interpretation Target concepts for query interpretation can be obtained from context sub graph, SG_{CTX} of the query. Here, we define the possible

candidates that can be selected as target concepts.

Definition 3.16. *(Target Concept Candidates) Consider a context sub graph, SG_{CTX}, of an unstructured query, Q_U. A target concept candidate, $targetCand_{cpt}$, for the query can either be the root node of SG_{CTX} or concept leaf node of SG_{CTX}*

$\forall Q_U, targetCand_{cpt} = \{V_{SG_{root}} \in SG_{CTX} \cup V_{SG_{leaf}} \in SG_{CTX}\}$

where $V_{SG_{leaf}}$ is st.

Target Concept Weighting The weight for a target concept of a term, $score_{CTXPROX}$ is a real number in the range of $[0,1]$ obtained from the context proximity function for context, $score_{CTXPROX}(cpt_{ct}, ctx)$.

$$\forall Q_U, score_{CTXPROX}(cpt_{ct}, ctx) : targetCand_{cpt} \rightarrow score_{CTXPROX}$$

Aligned with Axiom 3.1, an interpreted query contains both contents and concepts, but cannot contain concepts only. An interpreted query can have multiple target concepts as well as multiple constraint concepts. Each constraint concept needs to bind to a content. We define a query interpretation as follows.

Definition 3.17. *(Query Interpretation) A query interpretation, QI, is a tuple $QI = (TARGET, CONSTRAINT)$, where $TARGET = \{target_{cpt_i} | 1 \leq i \leq n\}$ and $CONSTRAINT = \{(constraint_{cpt_j} : constraint_{ct_j} | 1 \leq j \leq n)\}$.*

Given the domain, D, an unstructured query can have more than one interpretations from its query interpretation process.

Axiom 3.2. *Let QI be interpretations derived from domain, D. Each unstructured query, Q_U is interpreted to more than one interpretations, QI. The set can be null in the case where Q_U is not within the domain, D.*

$\forall Q_U . \exists QI = \{x || x| \geq 0\}$

3.2.3 Query Representation Requirements

In this section, we describe how an interpreted query can be represented in a generic structured form, rather than as a structured query language at this stage. We represent a query interpretation as an intermediate structure, that separates the semantic (i.e. contents and structures) of a query and the syntax of the query. Both queries are represented as different structures and can be mapped to one another using a schema matching function *SM*. Matching is used to find best matched semantic query structure and syntax query structure to enable the construction of a structured query.

Definition 3.18. *(Intermediate Query Representation) A query interpretation, QI can be represented using an intermediate query representation in the form of triple,* $I = (Q_{sem}, Q_{syn}, SM_{Q_{sem}Q_{syn}})$*, where* Q_{sem} *is a semantic query structure,* Q_{syn} *is a syntax query structure, and* $SM_{Q_{sem}Q_{syn}}$ *is a matching function that maps* Q_{sem} *to* Q_{syn}*.*

Both semantic query structure and syntax query structure are created based on the Intermediate Query Schema.

Definition 3.19. *(Intermediate Query Schema) Intermediate Query Schema,* I_{schema} *is a description of the intermediate query structure. Two main components featured in the schema are the structures (i.e. targets and constraints) and contents of a query. This schema is required for the construction of the semantic query structure and syntax query structure.*

Definition 3.20. *(Semantic Query Structure) A semantic query structure,* Q_{sem} *is a structure representing the structures and contents of a query. It is constructed from an interpreted query, QI, based on the intermediate query schema.*

The representation of semantic query structure is discussed in detail in section 3.4.2 (a). The construction of semantic query structure is discussed in detail in chapter 4 section 4.2.

Different from semantic query structure, a syntax query structure is a template that captures the syntax of a target structured query. It is generated based on a set of training structured queries, Q_{eg} (see section). A syntax query structure is introduced such as it is independent from the semantic query structure. Each syntax query structure is associated to syntax string to enable the construction of the structured query in string form.

Definition 3.21. *(Syntax Query Structure) A syntax query structure, Q_{syn} is a triple, consisting of a structure, a string and the mapping between the structure and string. It is given as $Q_{syn} = (syn_{struc}, syn_{str}, M(syn_{struc}, syn_{str}, X))$, where syn_{struc} is the structure constructed based on the intermediate query schema, syn_{str} is the query template in string form, M is a set of mappings that correspond structure to string and X is the query language.*

It represents the template of a query, without its content. For each query language type, there is a set of syntax query. Each syntax query is constructed from an example structured query, q_{eg}, for the language.

The representation of syntax query structure is discussed in detail in section 3.4.2 (b). Methods to create its knowledge base will be discussed in detail in section 3.4.2 (c).

Having defined the two query structures, now we proceed to describe the matching function for these two structures. The matching between semantic query structure and syntax query structure is obtained based on the structural similarity of the two structures.

Definition 3.22. *(Schema Matching between Semantic Query Structure and Syntax Query Structure) A schema matching, $SM_{Q_{sem}Q_{syn}}$ is a triple, $SM_{Q_{sem}Q_{syn}} = (Q_{sem}, Q_{syn}, \theta)$, where*

θ is a decision from the function, $F_{strucmatch}$ that matches the structural similarity between Q_{sem} and Q_{syn}. A matching decision is binary, $\theta = \{0,1\}$. $F_{strucmatch} : Q_{sem} \rightarrow \{0,1\}$

Once a semantic query structure are matched with a syntax query structure, the information needs (or contents) can be mapped to a structured query language string.

3.2.4 Structured Query Expectation

In the query transformation process, since there are more than one interpretations per unstructured query, hence similarly there can be more than one structured queries per unstructured query. The reason of having more than one structured queries is due to the possibilities of many interpretations in which a query may be relevant to.

Axiom 3.3. *Let F be a transformation from Q_U to Q_S. For each unstructured query, there exists a set of structured queries. Similar to the previous proposition, the set of structured queries can be null.*

$$\forall Q_U . \exists Q_S = \{y||y| \geq 0\}.F(Q_U, Q_S)$$

Axiom 3.4. *Given that X is a set of query language type, $X = \{NEXI, XMLFragment, ConceptValue\}$. Each structured query can be mapped to the type of X.*

$$Q_{S_X} = \{Q_{S_{NEXI}}, Q_{S_{XMLFragment}}, Q_{S_{ConceptValue}}\}$$

Ultimately, the transformation has to achieve a better or at least an equivalent query. A transformation process, F, transforms Q_U into an *equal*, *optimized* Q_S. Here, we say that both queries are equivalent if they can achieve similar outcome. If the latter can give a better outcome compare to its former, we say that it is optimized. Not forgetting, it is necessary to consider a poor transformed case.

Proposition 3.2. *Let us denote I as an interpretation on Q_U, where $I \in \{I_{equal}, I_{optimized}, I_{poor}\}$. Let us denote $P(Q_U)$ as the precision outcome of Q_U, while $P(Q_S)$ as the precision outcome of Q_S.*

$$\forall q \in Q_U, I_{equal} \models P(Q_S) \approx P(Q_U)$$

$$\forall q \in Q_U, I_{optimized} \models P(Q_S) > P(Q_U)$$

$$\forall q \in Q_U, I_{poor} \models P(Q_S) < P(Q_U)$$

3.3 A Probabilistic Approach for Query Interpretation

In this section, we discuss the probabilistic model used for query interpretation in our query transformation framework. From section 3.2.2, we have defined the requirements for query interpretation. In this section, we explain how to obtain the knowledge required for query interpretation. Our approach, i.e. probabilistic approach, is to exploit the structural information available from the collection for interpretation. This approach has been used widely in related works to capture the usage of term and structures so that they can be used to expand or optimize a source query to improve its effectiveness. However, current related works that use probabilistic methods have some limitations especially in, i) determining a query's target (Kim et al., 2009), ii) finding concepts for term in a nested structure (Petkova et al., 2009), iii) suggesting concepts without structural hints in query (Bao et al., 2010).

As such, our proposed context-based probabilistic model shall focus on how to address the limitations. We will also focus on how the probabilistic model works with complex document structure. To begin with, we present the characteristics of complex document structure. From these characteristics, we first summarize some features of complex document structure of XML, and propose a model that addresses the limitations faced by current works in handling these features. Then, we propose a context-based term weighting model, that extends the current model used for simple document structure. The model incorporates factors of local context of document subtree and hierarchical distance in our probabilistic term weighting for concept selection.

3.3.1 Complex Document Structure

A complex document structure has the characteristic as presented in Table 3.2. Compare to simple document structure, term weighting per element type (i.e. structure/concept) is more diversified in document with complex structure, due to the several reason below.

1. Since the type of structures can be freely defined, there are many different types of new structures which is relevant to a term in complex document structure compare to simple document structure.

2. Each document is a combination of several kind of objects, rather than a single type of object for document with simple structure. Hence, a term may not only be relevant to one kind of object, but it can be related to different objects. E.g. in a collection with complex document structure, a term, "information retrieval" can be related to a conference name, paper title, contents in keynote summary, track name, etc. Whereas in a collection with simple document structure, e.g. a bibliography record, the same term is related to less structure type, e.g. paper title and name of proceedings only.

3. Since the hierarchy of complex document structure is deeper due to nested elements, each term is related to a longer structure path, that indicates a series of structures. Hence, since it is not common to carry out term weighting per structure path, it has become an issue to find a good structure along the structure path. E.g. within a structure path, a structure can be meant for grouping, categorizing, annotating. In document with simple structure, that has been created based on a predefined schema, this is not an issue as when a structure is used for a term, it is meants as the data type of the term.

Table 3.2: Characteristic of complex document structure.

Factor	Simple Document Structure	Complex Document Structure
Collection	Homogeneous	Heterogeneous
Structure Source	Controlled, requires DTD, schema	Free
Structure Number	Little	Many
Content Creation	Data centric	Text centric
Hierarchy	Shallow	Deep
Information	Single type of information, e.g. an abstract, a journal, an actor	Combination of multiple types of information

3.3.2 Incorporating Context for Query Interpretation

In general, there are two purposes of incorporating contexts in our probabilistic approach, i.e. to improve the query target and constraint concepts selection during query interpretation.

First, to improve the capturing of concepts per term (i.e. *constraint concept*) from the information structures (e.g. content hierarchy, taxonomy, schema) of documents. Context is added to improve the better selection of concepts. It is used to cater for situation where a document has complex structures, where its consists of many small parts presenting different types of information. Under the different contexts, e.g. [context: high school] "Kai-Fu Lee" $\rightarrow$ "notable alumni", [context: google china] "Kai-Fu Lee" $\rightarrow$ "founder". With this feature, we are able to select a more accurate concept which is context oriented.

Second, to improve the accuracy of retrieval units (i.e. *target concept*). A common method used to determine a retrieval unit is based on the SLCA of all the terms in the query. The main difference between a SLCA node and a context node is that SLCA node is a *physical node* while a context node reflects a *class*. When we are dealing with a *class*, we can measure the importance of this class (e.g. [context: workshop], [context: conference]) based on its popularity in the collection to improve the selection of a retrieval unit.

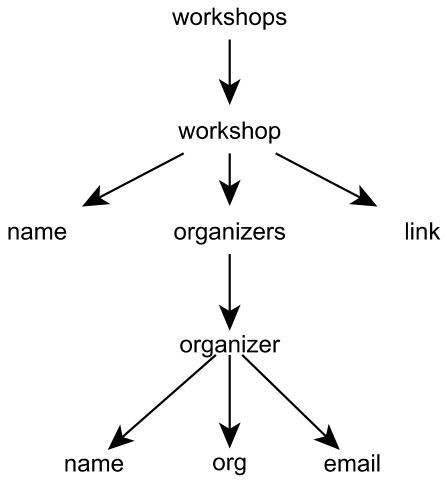


Figure 3.3: Concept Structure for Structured Document

3.3.3 Capturing Concept for Term Interpretation

Concept Structure A concept structure (equivalent to hierarchy or taxonomy) of a structured document is a set of concepts (i.e. descriptive structures) abstracted from a structured, G_{doc}. Its main difference from document structure is that it only consists of structure nodes of unique tree path. It is also different from schema or DTD as the concept structure is obtained from a structured document, i.e. showing how structures relate in real usage. Its purpose is to capture a summary of structures that appear in this document, rather than the entire structures for all instances in the document tree. The concept structure for structured document (featured in Figure 3.1) is shown in Figure 3.3.

Definition 3.23. *(Concept Structure) A concept structure is defined as* $G_{concept} = (V_{concept}, ED_{concept})$*, where* $V_{concept} = \{v : v \in v_{struc}\}$ *is a set of concept nodes and* $ED_{concept} = \{e : e \in ED\}$ *is a set of edges connecting these concepts.*

In G_{doc}, concepts are structures created based on application needs (defined based on predefined schema, or without one). Since the creation of concepts are flexible, relationships between these concepts can naturally reflect a unique context. In this thesis, we assume that a higher concept is a generalization (has broader meaning) than the lower one, therefore forming a taxonomy of concepts. A term's meaning can be obtained by

observing how concepts are used in this taxonomy.

Term Concept Association To obtain all possible structure interpretations for a term, we associate a term and its relevant concepts as featured in the structured document. There are two ways a concept can be formed, i.e. an individual concept (single structure node) or a concept set (multiple structure nodes obtained from a path). A concept set for a term consists of a sequence of structure nodes along the ancestor's path from the content node containing the term. The nodes can either be taken from a full path or a sub path as follow.

$cpt_{path_{full}} = \{v_{struc_0}, v_{struc_1}, v_{struc_2} ... v_{struc_{k-1}}, v_{struc_k}\}$, where v_{struc_0} is the parent node of a content node for the given term, and v_{struc_1} is the parent node of v_{struc_0} and so forth. k is the number of edges from v_{struc_0} to root node.

Although full concept path is often regarded to have better differentiation power than a shorter one, but its length and specificity also hinder it from being utilized practically. Hence, a shorter path is often more useful in identifying commonly used concepts for a term. We call this shorter path as concept subpath. A concept subpath is written as, $cpt_{path_{sub}} = \{v_{struc_0}, v_{struc_1}, v_{struc_2} ... v_{struc_{n-1}}, v_{struc_n}\}$, where $1 \leq n \leq k$. Finding a good range for n is important to avoid creation of too many unuseful concept nodes.

While path defines a series of concepts as a single meaningful unit, it may be too detailed to be used for query's term interpretation. Hence, associating individual concept node (e.g. name, hotel, accommodation in Figure 3.1) of the concept path with a term is also important to avoid an overly strict concept interpretation for a term. An individual concept node is given as $cpt_{single} = \{v : v \in v_{struc_j}\}$, where $0 \leq j \leq k$.

3.3.4 Context-based Term Weighting

The context-based term weighting is divided into two parts. Generally, term weighting for structured collection measures the importance of term in an element within the

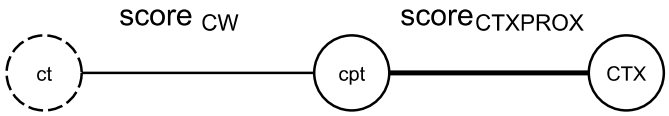


Figure 3.4: Weighted Edge in Term Interpretation Representation

collection. Since the term weighting is carried out based on *contexts* (see section 3.2.2) within collection instead on one single collection, two levels of weighting are required. First, we need to measure the importance of term wrt. to a structure (or concept) within a context, second, we need to measure the importance of term wrt. to a structure among multiple contexts. The application of these measures in a term interpretation representation is shown in Figure 3.4.

Within Context ($score_{CW}$) Under a context, we obtain the association between the term and its concept (i.e. within each *term* $\rightarrow$ *concept*). Here, we will extend the basic term weighting approaches, like TFIDF, Okapi BM25 to measure the weight of a term in a concept. Since the term weighting method is primarily designed to measure weight of a term in a document or element, we will generalize the element based on its type (i.e. concept).

Among Contexts ($score_{CTXPROX}$) For each term in a collection, we first obtain the association between *term* $\rightarrow$ *concept* and its context, where we use the taxonomy analysis method to measure the proximity between *term* $\rightarrow$ *concept* pair and its associated context. We propose a contextual proximity measure that combines two semantic similarity metrics, i.e. distance-based similarity and information-based similarity mentioned in (McHale, 1998), (Resnik, 1995).

The query interpretation will be carried out based on these two measures, where the former decides the selection of constraint concept for a term, while the latter decides the selection of target concept for a query.

3.3.4 (a) Term Weighting Within Context

Within a context, a term may be associated to multiple concepts. First, term is weighted against individual elements in the collection, followed by aggregation of weights according to the element type (concept). In our term weighting measure, we take into consideration distanced concepts in the structure hierarchy.

Term-Element Weighting Various term weighting models have been actively used in document retrieval, such as TFIDF (Salton & Buckley, 1988), OKAPI BM25 (Robertson & Zaragoza, 2009) and Language Model (Ogilvie & Callan, 2002). Since term weighting in document has been a mature field in information retrieval, its scoring models are extended to cater for term weighting in element (Wang et al., 2007).

In our concept weighting measure, we shall adopt these basic term weighting models. We show the basic formulae of a popular term weighting model that have been extended for element weighting below (refer to Appendix A for more details about these models). The weight of a content term, ct in an element, e, is denoted as $score_{TW}(e,ct)$ below.

TFIEF (Term Frequency Inversed Element Frequency)

$$score_{TW}(e,ct) = tf_{ct,e} * log\frac{N_e}{ef_{ct}}$$

, where e is element, ct is content term, $tf(ct,e)$ is frequency of tc in e, N_e is frequency of e in collection, and ef_{ct} is frequency of e in collection that contains ct.

Term-Concept Weighting Concepts are generalizations of elements based on the type of structure of the elements. The weight of a term wrt. to a concept is estimated based on the relatedness between a term and a type of structure (referred as concept in this thesis) instead of an element. For weighting concept on structure hierarchy, the distance of concept from term is taken as $D_{e,ct}$. Let us denote E_{cpt} as the set of elements of type

cpt, where cpt is a concept. The weight of a content term, ct in a concept, cpt, is denoted as $score_{CW}(cpt, ct)$ below.

$$score_{CW}(cpt, ct) = \frac{\sum_{e \in E_{cpt}} [score_{TW}(e, ct) * \frac{1}{D(e,ct)}]}{|E_{cpt}|}$$

, where cpt is concept, ct is content term, E_{cpt} is set of elements of type cpt.

This scoring factor, $score_{CW}(cpt, ct)$, captures the intuition that, when ranking a constraint concept, a direct relationship of concept and term is favoured. A direct relationship means that the lesser additional terms contained by the concept is better, e.g. <author>Andrew Trotman</author>, compared to the one contained together in a paragraph with other terms, e.g. <keynote_abstract>... Andrew Trotman began his career at ... </keynote_abstract>.

3.3.4 (b) Term Weighting Among Contexts

Besides weighting a term wrt. a concept within a context, it is necessary to measure the importance of the term if it appears in multiple contexts. This is to show the importance of a term and concept under a particular context compare to another. For example, "author: andrew troman" may be more important under the context "conference" compare to "tutorial". This is measures using the proximity between "concept:term" and its context.

Contextual proximity between *concept:term* and context is obtained by combining two proximity factors, i.e. Distance-based Closeness, and Content-based Closeness. Here we denote *concept:term* unit as cpt_{i,ct_j}, a context unit as ctx_k.

Distance-based Closeness For distance-based closeness, we measure the distance between cpt_{i,ct_j} and ctx_k using the approach of semantic space in a taxonomical tree. Edges are used as distance. Distance between concept nodes within the space is taken as mea-

surement of semantic closeness. To measure the semantic closeness, we measure distance similarity, *DISim*, between a concept and its context for each term unit, ct_j, is given as

$$DISim(cpt_{i,ct_j} : ctx_k) = \frac{1}{edge(cpt_{i,ct_j} : ctx_k)}$$

, where $edge$ is the number of nodes interval between cpt_{i,ct_j} and ctx_k.

Content-based Closeness For content-based closeness, we measure the occurrences of cpt_{i,ct_j} and ctx_k. The semantic closeness between two concepts node in a taxonomy can also be measured based on contents frequency that subsume concepts. Here, for a term unit, ct_j , the density, *DEN*, of a concept and context pair, is given as,

$$DEN(cpt_{i,ct_j} : ctx_k) = \frac{pf(cpt_{i,ct_j}, ctx_k)}{\sum_{i=1}^{N} pf(cpt_{i,ct_j}, ctx_k)}$$

, where pf is the pairs frequency of cpt_{i,ct_j} and ctx_k.

Contextual Proximity Score Contextual proximity is taken as the product of both scoring of distance-based closeness and content-based closeness,

$$score_{CTXPROX}(cpt_{i,ct_j} : ctx_k)$$
$$= DISim(cpt_{i,ct_j} : ctx_k) \times DEN(cpt_{i,ct_j} : ctx_k)$$

Example 3.3. *Consider a collection of conference domain, some examples of weighted terms with both* $score_{CW}$ *and* $score_{CTXPROX}$ *are shown in Figure 3.5. Weight on the thin edge,* $score_{CW}$*, of each term interpretation,* $I_{content}$*, captures the importance of a term wrt. a concept. Weight on the thick edge,* $score_{CTXPROX}$*, of each term interpretation,* $I_{content}$*, captures the importance of a* concept:term *pair wrt. a context. For instance, for term "andrew trotman" (see "at" in Figure 3.5), within the "WORKSHOP" context, this term*

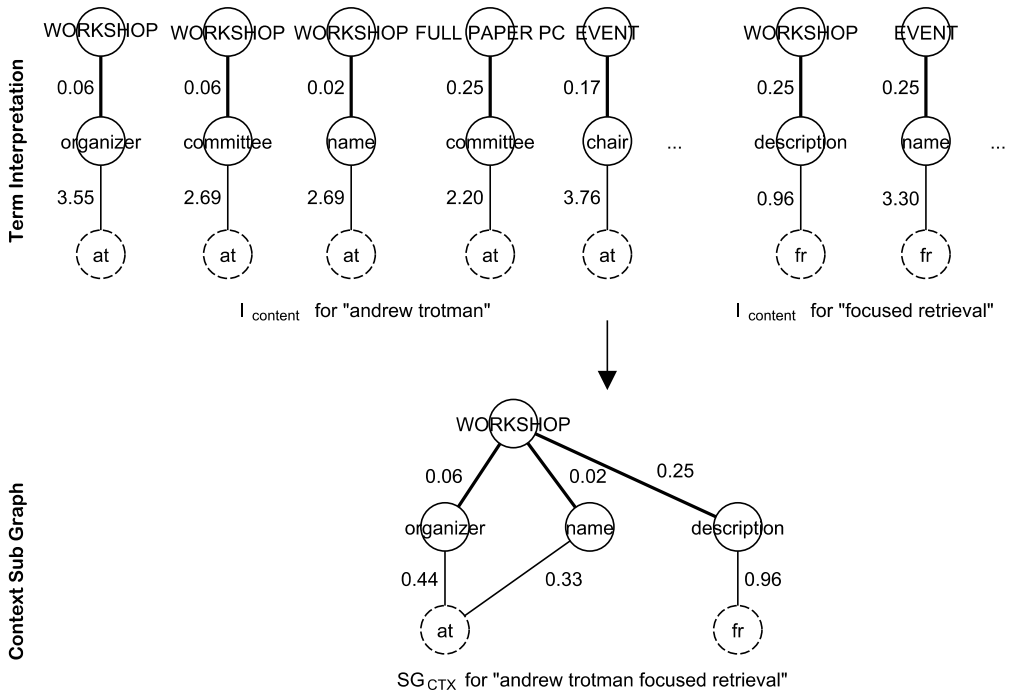


Figure 3.5: Examples of Weighted Edge in Term Interpretation and Context Sub Graph

has stronger relationship with the concept, "organizer" and "committee", based on the concept score, $score_{CW}$. For this example, term weighting model BM25 is used to calculate $score_{TW}$. Another information that is captured is the importance of context. For instance, when "committee:andrew trotman" appears in "WORKSHOP" context with a weight 0.06 and "FULL PAPER PC" contexts with a weight 0.25, this shows committee:andrew trotman *is more relevant to the context, "FULL PAPER PC".*

3.3.5 Context-based Term Weighting for Structure Term

As structure terms can be used in query, these terms are required to be weighted to differentiate their importance. For structure term weighting, the weight of a structure in a context is measured wrt. the taxonomical characteristic, which is based on how structures are related to each other in a collection. In this weighting, the main factor taken into consideration for calculating the importance of structures is the different ways how a structure is used in taxonomy. As repetitive structures such as a list of items of the same structure like <book>, merely reflect the usage of the same structure type, hence it

does not affect the importance of a structure. As such its frequency will not be taken for structure term weighting.

The scoring of structure term in a context is similar to distance-based closeness measure used for measuring content term. The main difference is the source of semantic space. For content term, its semantic space is taken from the taxonomical tree of document structure, whereas for structure term, its semantic space is taken from the taxonomical tree of concept structure. The former includes both concepts and contents in its tree structure, but the latter only includes concepts in its tree structure.

Distance-based Closeness for Structure Term For distance-based closeness of structure term, we measure the distance between st_i and ctx_j using the approach of semantic space in a taxonomical tree of concept structure, $G_{concept}$. Edges are used as distance. Distance between concept nodes within the space is taken as measurement of semantic closeness. To measure the semantic closeness, we measure distance similarity, *DISim*, between the corresponding concept for each structure term, and its context.

$$DISim(st_i : ctx_j) = \frac{1}{edge(st_i : ctx_j)}$$

, where $edge$ is the number of nodes interval between st_i and ctx_j.

Contextual Proximity Score for Structure Term Contextual proximity for structure term is taken as the average distance-based closeness for all non-repetitive term interpretations for structure term, $I_{structure}$ in collection,

$$score_{CTXPROX}(st_i : ctx_j) = \frac{\sum_{i=1}^{n} DISim(st_i : ctx_j)}{n}$$

, where n is the total of unique term interpretations for structure term, st_i.

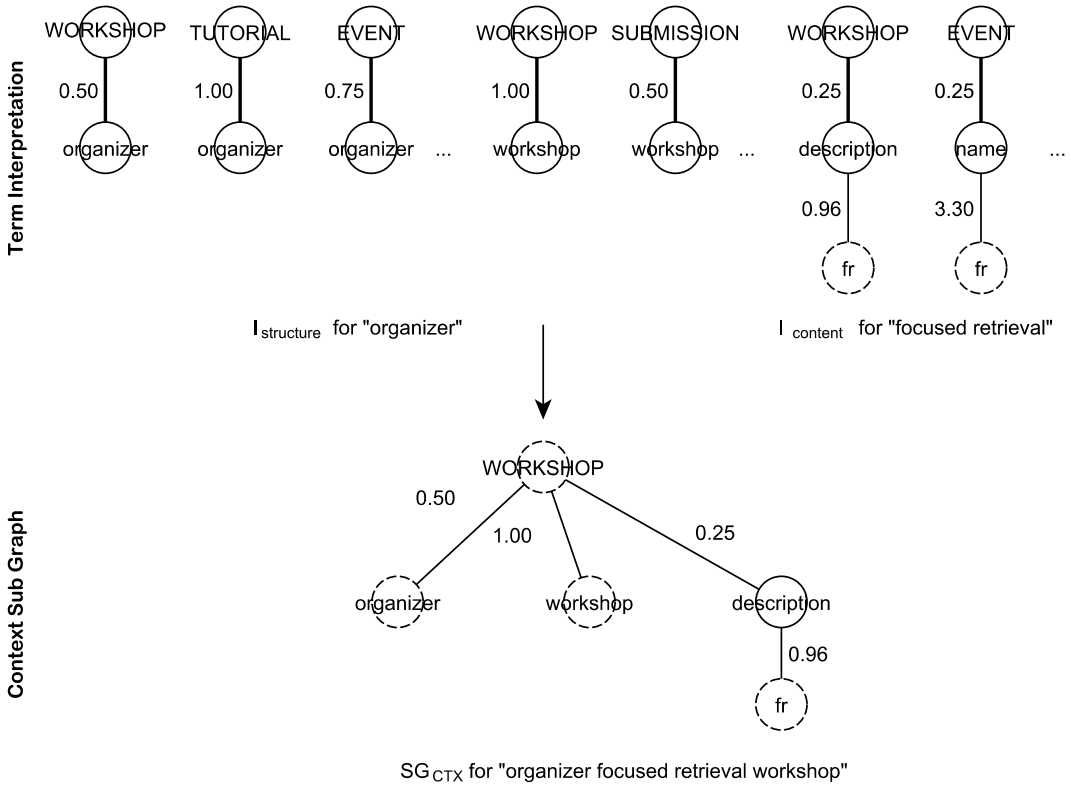


Figure 3.6: Examples of Weighted Edge in Structure Term Interpretation and Context Sub Graph

Example 3.4. *Consider the same collection of conference domain, the weighted structure terms with* $score_{CTXPROX}$ *are shown in Figure 3.6. Different from content term, structure term only has one weighted edge. The weight on the edge of each structure term,* $I_{structure}$*, captures the importance of the structure/concept wrt. context.*

In this example, we show how the weighted context sub graph for query "organizer focused retrieval workshop" from Figure 3.2 is obtained from the weighted term interpretations.

3.4 A Representation for Query Construction

In the query transformation framework, an intermediate query is introduced to represent the interpreted structured query in a generic form. Such form is useful to separate the structured query syntax from the contents of the query. It is designed so that it could retain the information needs for both unstructured or structured query. Its main purpose is to ensure genericness of the process while minimizing information loss during the trans-

formation.

This section first presents the intermediate query schema that serves as the base for the creation of intermediate query. Then, we present the structure of our proposed intermediate query. Lastly, we feature a parsing method that assists in the generation of knowledge of multiple structured query types.

3.4.1 Intermediate Query Schema

The intermediate query schema specifies how an intermediate query can be constructed. It defines a set of elements and their properties that may be used to represent an intermediate query in this transformation framework. Before we start defining the schema, we first present the information required to be represented in an intermediate query.

Representing Query Contents There are two types of content to be represented in an intermediate query, i.e. a query's target (as in Definition 3.12) and a query's constraints (as in Definition 3.13).

Representing Query Structure As the contents of query may contain hierarchical characteristics such as nested constraints or nested targets, these characteristics are required to be represented in the intermediate query.

Let us start by looking at an example on how query contents are represented in an intermediate query.

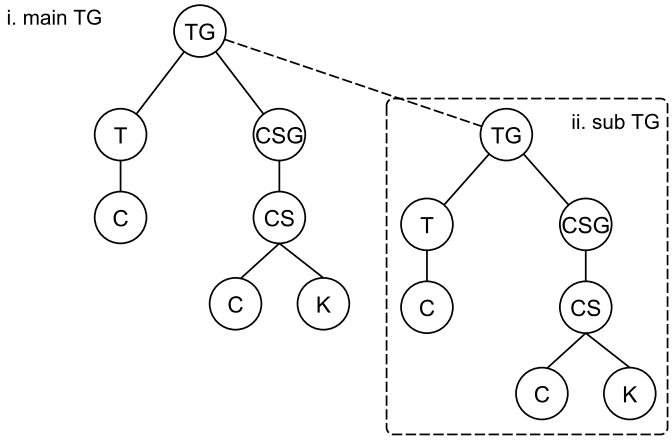
Example 3.5. *An intermediate query structure for query "workshop by andrew trotman" written in xml schema language.*

```
<targetGroup>
  <target>
    <concept>workshop</concept>
  </target>
```

```
  <constraintGroup op="AND">
    <constraint>
      <concept>organizer</concept>
      <keyword>andrew trotman</keyword>
    </constraint>
  </constraintGroup>
</targetGroup>
```

The above example shows a simple intermediate query structure. It consists of a main element, *targetGroup*. Within the *targetGroup* element, there are two types of subelements, *target* and *constraintGroup*. A *target* has *concept* subelement. The *concept* element contains a value, **workshop**. Under a *constraintGroup* element is *constraint* element. A *constraintGroup* has an attribute *op*. A *constraint* has two subelements, *concept* and *keyword*. The *concept* element contains a value, **organizer**, whereas the *keyword* element contains a value, **andrew trotman**.

From this example, we proceed to present the components of intermediate query schema, such as the elements in an intermediate query, the attributes of elements in an intermediate query, the child elements, order and number of the child elements etc. Its schema definition and properties are described as follow.



TG: target group CSG: constraint group
T: target CS: constraint C: concept K: keyword

Figure 3.7: Intermediate Query Schema

Schema Definition The definition of our proposed intermediate query schema is specified using XML schema definition language (Fallside & Walmsley, 2004). Refer to Figure 3.7 for the illustration of the schema.

```
<xsd:element name="targetGroup" type="TargetGroupType"/>

<xsd:complexType name="TargetGroupType">
  <xsd:sequence>
  <xsd:element name="target" type="TargetType" maxOccurs="unbounded"/>
  <xsd:element name="constraintGroup" type="ConstraintGroupType"/>
  <xsd:element name="targetGroup" type="TargetGroupType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ConstraintGroupType">
  <xsd:sequence>
  <xsd:element name="constraint" type="ConstraintType" maxOccurs="unbounded"/>
  <xsd:element name="constraintGroup" type="ConstraintGroupType"/>
  </xsd:sequence>
  <xsd:attribute name="op" value="AND,OR"/>
</xsd:complexType>

<xsd:simpleType name="TargetType">
  <xsd:element name="concept" type="Concept"/>
</xsd:simpleType>

<xsd:complexType name="ConstraintType">
  <xsd:sequence>
  <xsd:element name="concept" type="Concept"/>
  <xsd:element name="keyword" type="Keyword"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="Concept">
  <xsd:element name="concept" type="xsd:string" minOccurs="1"
  maxOccurs="unbounded"/>
</xsd:simpleType>

<xsd:simpleType name="Keyword">
  <xsd:element name="keyword" type="xsd:string"/>
</xsd:simpleType>
```

Target Group The purpose of target group is to capture the target of a query and its corresponding constraint. An intermediate query consists of one main target group element, TG. A target group element, TG, can have one or more target elements, T. Each target, T has one or more concept elements, C. More than one concept elements occurs if a concept is a set of structures (path). Each structure of the path is represented as separate concept element under the same target. The concept of target is optional. In the case when a concept is not specified, its value is null.

A target group can have one or more constraint groups, CSG. A target group can have another target group as its sub element (see ii. in Figure 3.7). A sub target group defines refined target elements. In general, common queries only have one target group. However there are cases where a specific target is required. In this case, additional target group can be attached to existing target group as subtarget group.

Constraint Group The purpose of constraint group is to group the constraints that belong to the same target. A constraint group element, CSG, has one or more constraint elements, CS. Each constraint, CS has a concept element, C, and a keyword element, K. The concept of constraint is optional. In the case when the concept is not specified, its value is null. A constraint group element has a logical operator as its attribute. The logical operator attribute can either have "OR" or "AND" as its value, to indicate where its constraint elements are disjunctive or conjunctive. A constraint group element has a logical operator as its attribute. The logical operator attribute can either have "OR" or "AND" as its value, to indicate whether its children are disjunctive or conjunctive.

3.4.2 Intermediate Query Representation

From the previous section, we have defined the schema of building intermediate query. In this section, we describe how the schema is used in the intermediate query representation of our query transformation framework. An intermediate query representation

consists of two query types, i.e. semantic query structure and syntax query structure. A schema matching function is used for corresponding these structures.

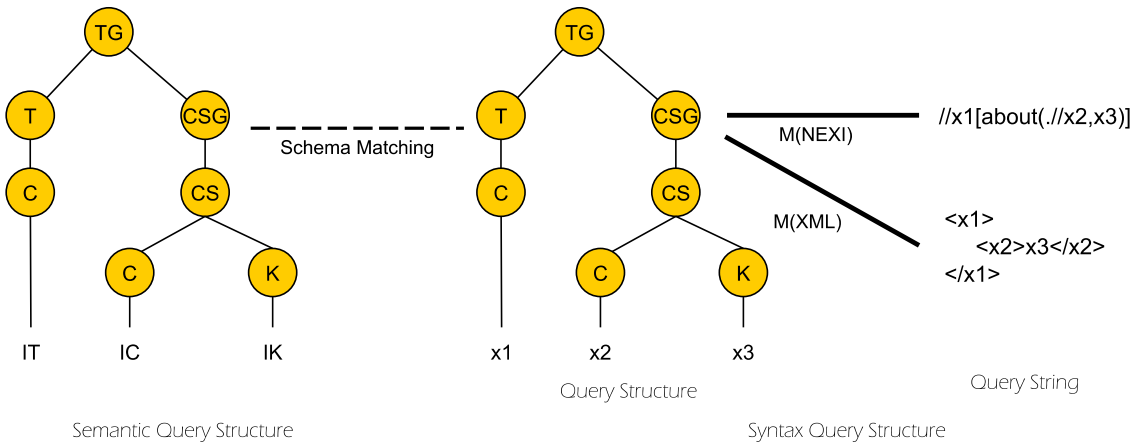


Figure 3.8: Intermediate Query Representation

3.4.2 (a) Semantic Query Structure, Q_{sem}

A semantic query structure represents the contents of an interpreted query. It is used to capture the interpreted query in a generic manner, such that it can be constructed into more than one target structured query later. Thus, it does not contain syntax of any structured query, but only contents (and logics) that will be used to construct the query.

It captures three types of contents in its leaves nodes, i.e. interpreted target concept, IT, interpreted constraint concept, IC, and interpreted constraint keyword, IK. An example of semantic query structure is shown in Figure 3.8. The contents captured at the leaves of semantic query structure can be mapped to its syntax query structure counterpart to generate a structured query string.

A major requirement of the semantic query structure is its ability to bridge information needs from unstructured to structured forms. Hence, this query structure should be able to support essential information needs so that they can maximize the purpose of structured queries. Follow, we show how contents and structural details of query are represented using the intermediate query schema. We start by discussing cases of repre-

sentation for simple information needs, then proceed to a more complex ones that involve nested targets and constraints.

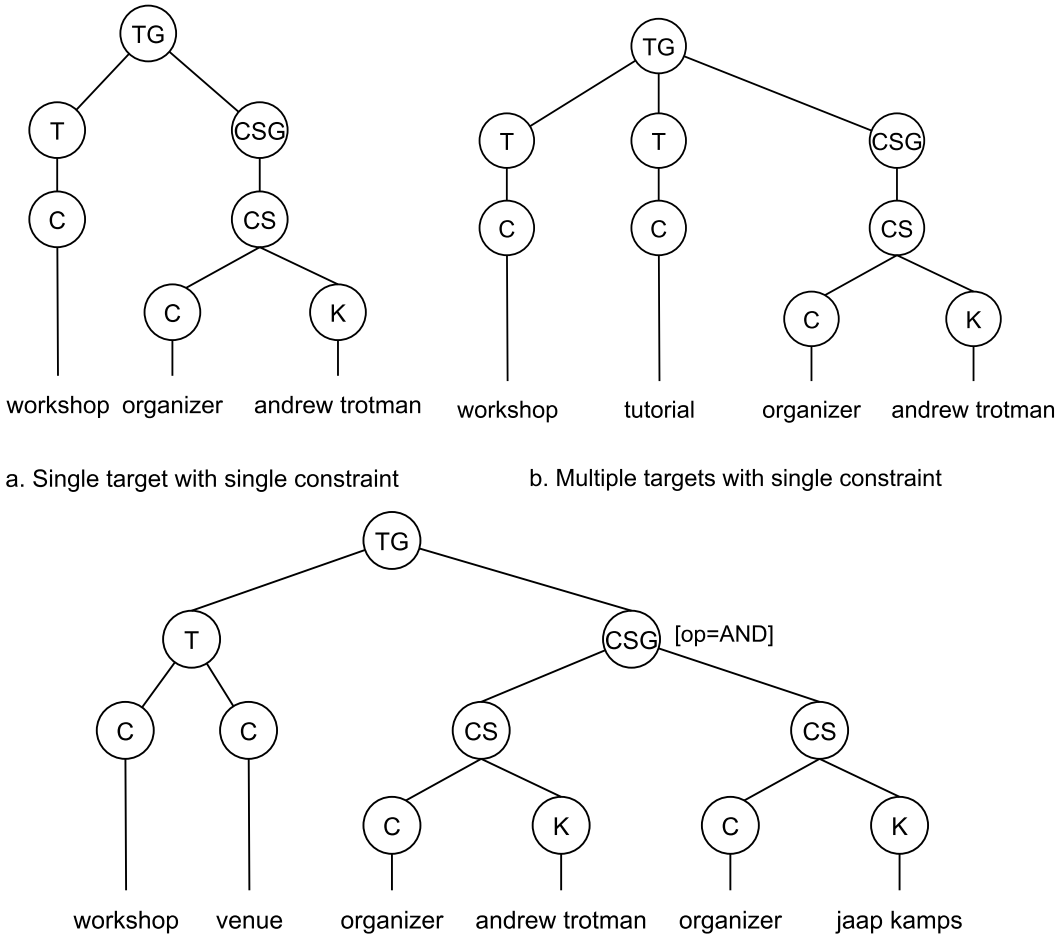


Figure 3.9: Representing Simple Information Needs

Representing Single Content The simplest information needs for structured query comprises of one target and one constraint. Minimally, an intermediate query should be able to represent these two contents as they are the basic requirements of structured queries. For example, an information needs "I am searching for the workshop organized by andrew trotman" is interpreted into a target, "workshop" and constraint pair, with "organizer" as concept and "andrew trotman" as keyword. Both target and constraint are represented using a target group element as in Figure 3.9 a.

Representing Multiple Contents A simple information needs may not be limited to a single target or a single constraint; it can include few of them each, but they should be indicating a single intention. An example of information needs with multiple targets is shown in Figure 3.9 b. The needs, "I am searching for workshop or tutorial run by andrew trotman" specifies two targets. Although this query looks for two different targets, we understand that the intention is the same where this person is looking for information about events organized by andrew trotman.

A query can have multiple constraints. For example in Figure 3.9 c., the query is looking for the workshop venue organized by andrew trotman and jaap kamps. It has two constraints, "andrew trotman" and "jaap kamps". Both constraints are represented as a set of conjunctive constraints under a constraint group.

Representing Concept Path To represent a target with a path of multiple concepts, such as "/workshop/venue" in Figure 3.9 c., the concept path is split and represented as sibling concept nodes under the same target. Please note the difference of target representation between a. and c. in Figure 3.9. Although rarely concepts of target and constraint are specified as path, our schema allow such representation as it is supported by path-based structured query like NEXI.

Now, let us see how an intermediate query can be used to represent a more complex information needs. Complex information needs occur when the interpreted query has multiple target groups (see a. in Figure 3.10) and constraint groups (see b. in Figure 3.10).

Representing Refined Targets When a user states a needs like, "I am looking for organizer from otago who organizes the focused retrieval workshop in 2010". This query is interpreted such that it contains a main target, indicating it is looking for information about "workshop", and a descendant target, indicating that it is looking for information

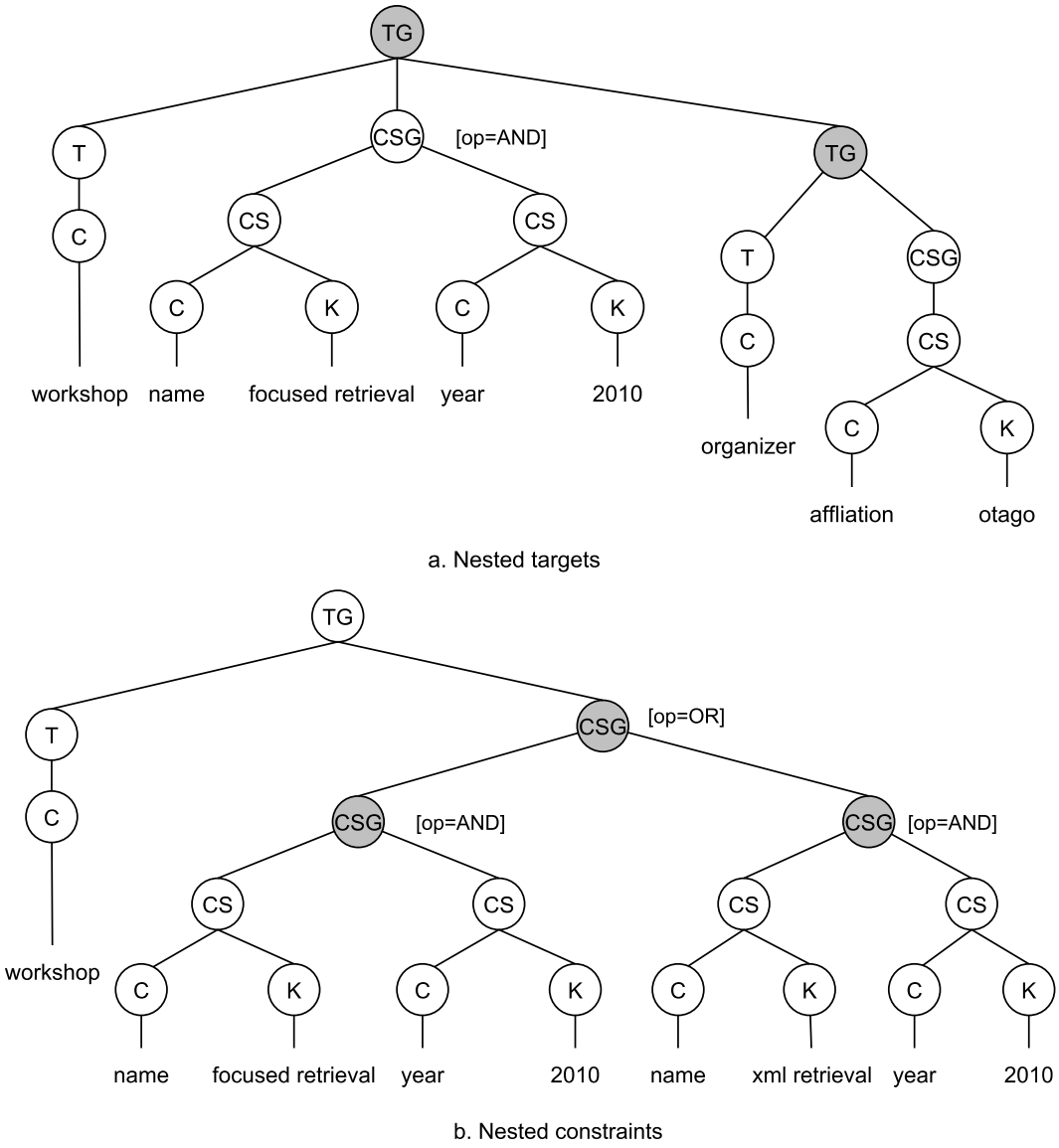


Figure 3.10: Representing Complex Information Needs

about "organizer" of the "workshop". In this case, the target group of the descendant target is represented as a sub target group element of the main target.

Representing Complex Constraints In some query, constraints are required to be operated based on some combinations of operators like "workshop about (focused retrieval AND 2010) OR (xml retrieval AND 2010)". However, each constraint group only support one type of logical operator. As such, in this case, constraint groups are used to represent the set of complex constraints based on their operators (see b. in Figure 3.10).

3.4.2 (b) Syntax Query Structure, Q_{syn}

A syntax query structure is a template that is used for generating structured query. Different from semantic query structure, a syntax query structure does not contain the contents of query. What it represents is the template of structured query, consisting the structured query in query string form and its corresponding query structure (see Syntax Query Structure in Figure 3.8). The contents of query is represented as variables for both query structure and query string. This common set of variables between structure and string enable transfer of contents from the former to the latter.

A syntax query structure is a triple, composes of a query structure, a query string and mapping between the structure and the string. For the same query structure, there can be multiple mappings, with each mapping denotes a different type of query languages, e.g. *M(NEXI)* and *M(XML)* in Figure 3.8. For each query language, X, there is a knowledge base of Q_{syn}. Although it is possible to obtain the same result by using a dedicated algorithm, our proposal of a syntax query structure is enable scalability of query transformation.

3.4.2 (c) Knowledge Base for Query Language, X

A knowledge base, KB_X is used to store the query templates for query language, X. As it is always hard to fill up an empty knowledge base from scratch, an example-based method is used to build a reasonable knowledge base with sufficient templates. Example-based method is commonly used in machine translation to build pairs of languages (Sumita & Iida, 1991; Somers, 1999). For language translation, the knowledge base consists of pairs of source language and target language of a content. Similarly, for our query language knowledge base, it consists of pairs of source format and target format. The source query is the structure form of the query example and the target query is the string form of the query example. As there may be repetitive examples, only unique

pairs are stored. In Figure 3.11, we show an example on how a query template is generated from an example query. The template can be defined manually or can be automated by using a intermediate query structure parser.

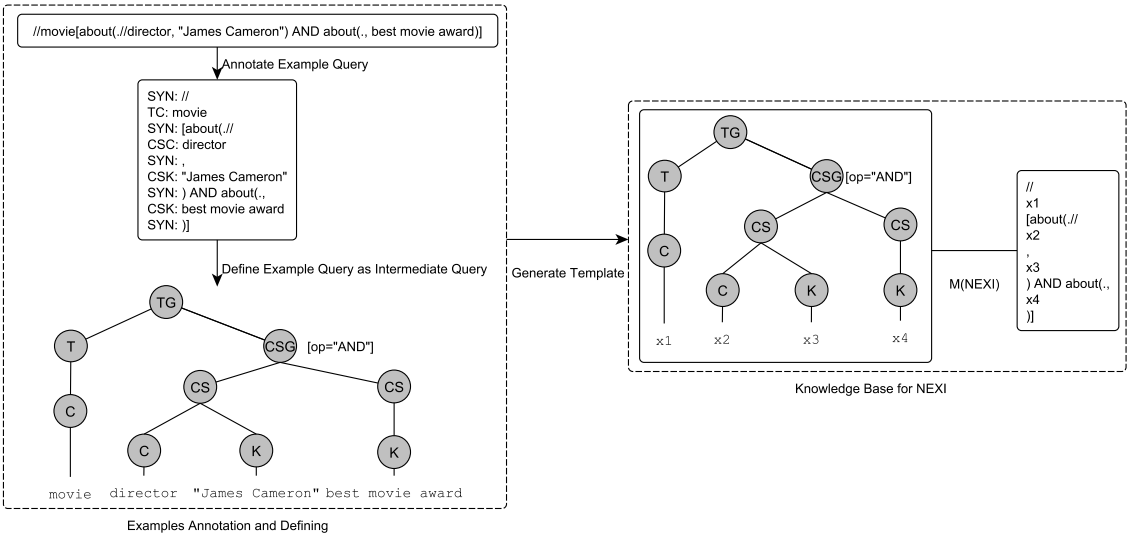


Figure 3.11: Knowledge Base Generation with Example Query

3.4.2 (d) Schema Matching, SM

Schema matching is used to match the schema of semantic query structure with the schema of syntax query structure (note that matching does not involve contents/leaves of query structures). Matching is determined based on their schema similarity. Such matching needs to be carried out in ad hoc manner since the semantic query structure is created during query time. Given a semantic query structure, the exact (or closest) match of its counterpart is looked up. Once a matched syntax query structure is obtained, the contents of the semantic query structure can be mapped the corresponding query string form.

3.5 Summary

In this chapter, we have defined the framework of query transformation. This framework serves as the basis of the query interpretation and construction in the subsequent

chapter later. In particular, this chapter focuses on the requirements for query transformation, such as the queries, resources and domains. We then provide a formal description of the overall framework to enable a better understanding of all the components involved in query transformation. Also, within this framework, we have also presented an improvised probabilistic approach for query interpretation and a novel intermediate representation for query construction. In the following chapter, we will show how query interpretation and construction will be carried out using this framework.

CHAPTER 4

QUERY INTERPRETATION AND CONSTRUCTION

In this chapter, we elaborate how query interpretation and representation is carried out using the flexible query transformation framework. Given a source query, Q_U, over a collection, D, with interpretation triple, I and a structured language mapping knowledge base, KB_X, this chapter describes how the query is interpreted, represented and finally mapped into the language form of a target structured query Q_S of language X.

The chapter is divided into four main sections. The first is formulated as the problem of finding the most significant interpretations. Three types of interpretations will be carried out, i.e. the context of the query, the concept where a query targets to find, and the concept to refine keywords used in the query. The second construct the best representation structure of the query, that maximizes the intention of the query. The third finds the best match query templates for the mapping of query contents to a structured query language. The last section presents ways of query ranking and selection.

4.1 Query Interpretation

Consider a query with a set of term, first, we will interpret the context of this query, based on the terms used in the query. Second, we are interested to interpret the query in terms of its desired retrieval unit, i.e. what the query target for (*target concept*). Third, we want to interpret the meaning of term used in the query, i.e. what does a term in a query means (*constraint concept*).

Before we present how an interpretation is carried out, we discuss the type of keywords that can be used in specifying a query. First type is content keyword that will be used to look up the contents part of documents, and the second type is structural keyword

that will be used to look up the structures part of documents. During query interpretation process, it will detect the usage of both keyword types automatically. For users side, they can either be aware or not aware of the keywords type. Users can write a query in normal way. However, for some applications, users may have access to some structural information that they can use in the query. In these situations, they can take advantage of the structural information more extensively.

4.1.1 Preliminary

A query, Q_U consists of a set of terms, QT. Each term, qt in QT, has one or more consecutive words. When they are composed as a query, the set of terms are infact a continuous set of words. Hence, an initial step is to identify the terms used in the query from this set of words.

Term Identification The main problem in term identification is that there are many possibilities of consecutive words combinations. As we cannot decide which term is the right one until at a later stage that involve query context analysis, therefore, we need to consider all the possibilities at this initial step. The identification of term is carried out based on two rules.

1. All possibilities of consecutive words must be considered.

2. For term variants, longer term is preferred over shorter.

Based on rule one, we will obtain a set of terms. However, these terms have overlapping words between them. The second rule helps to segmentize the words in query into non overlapping terms by giving priority to a term with longer keywords. In this rule, we assume when consecutive words appear in a query, it is not a coincidence, but reflects something that is required by the user. For example, a query written in the order of, “address river view hotel”, we get a set of inital terms like “address”, “river”, “view”, “hotel”,

"address river", "river view", and so forth (see no. 1 in algo 4.1). However, when we want to select query terms, both $QT_1 = \{$address, river view, hotel$\}$ and $QT_2 = \{$address, river view hotel$\}$ are possible combinations, if we are just based on checking on thesaurus. In this case, the second rule will prioritize the longer term first, and return QT_2 as the segmented query terms.

Algorithm 4.1 Identify Query Term

Input: unstructured query Q_U, structure thesaurus $THE_{structure}$ of D, content thesaurus $THE_{content}$ of D

Output: query terms $QT_{structure}$, $QT_{content}$

```
Let QT_initial = null
                                  ▷ 1. Generate initial set of terms using ngram phrase generator
Q_U ← tokenizetoword(Q_U)
for all n = {1 ≤ n ≤ |Q_U|} do
    QT_initial ← generateNgram(Q_U, n)
end for
QT_initial ← sortTermByLength(QT_initial)
                                  ▷ 2. Select query term based on length and collection's thesaurus
while QT_initial ! = {∅} do
    for all qt ∈ QT_initial do
        if qt exists in THE_content ∪ THE_structure then
            QT ← qt
            break
        end if
    end for
    QT_initial ← removeTermSubset(qt, QT_initial)
end while
                                                          ▷ 3. Classify query term
for all qt ∈ QT do
    if qt exists in THE_structure then
        QT_structure ← qt
    else if qt exists in THE_content then
        QT_content ← qt
    end if

end for
```

Keyword Classification Since there are two types of keywords which can be found in the query, it is necessary to differentiate them so that they can be optimized during query interpretation (see no. 3 in algo 4.1). This is carried out by matching them to two separate thesaurus, i.e. content thesaurus and structure thesaurus. A content thesaurus is

generated from the content of documents in collection. It consists of lexical items which are meaningful, such as “T. Andrew”, “River View Hotel”, “reservation form”, “Havelock Road” etc. For structure thesaurus, it can be obtained from a schema (if there is one) or generated from the structures of documents in collection. Example of items in structure thesaurus are “address”, “country”, “affiliation”, “workshop_program_committee” etc. In the case where a query term can either be a content keyword or structural keyword, then we will classify it as structural keyword. For richer thesaurus, the vocabularies can be expanded using abbreviation (e.g. “call for paper” and “cfp”), class (“paper” and “article”) and synonym (“keynote address” and “keynote speech”).

4.1.2 Query Context

Once we have identified keywords type of a query, Q_U, we can proceed to interpret the query. The first step is to identify the query context. Based on the keywords identified, we find all their sub tree candidates. A sub tree candidate, ST_{CTX} is obtained based on all the content and structure term interpretations for Q_U. Here, we used the notation, ST_{CTX}, instead of context sub graph, SG_{CTX} (as in previous chapter) as we will gradually refine the graph to tree data structure.

Since there can be multiple context sub trees for a given query, ranking of the sub trees is necessary, to differentiate the most relevant ones from the rest. As a context sub tree would serve as the central of a finding a good retrieval unit later, the scoring emphasizes on context sub trees which are neither too small nor too broad with its context proximity measure. Detailed discussion on the ranking is presented in section 4.4.1.

4.1.3 Query Target and Constraint

Given a context sub tree of a query, we can proceed to find the target and constraint concepts for the query. Depending on the source query type, the process of finding the target and constraint concepts is different. If an unstructured query only contain con-

Algorithm 4.2 Find Context Sub Tree

Input: unstructured query Q_U, a set of interpretation $I_{Q_U} = \{I | I = \bigcup_{qt \in Q_U} I_{qt}\}$
Output: a set context sub tree ST_{CTX}

▷ 1. Get Context Candidates For Each Keyword

for all $qt \in Q_U$ with $I_{Q_U} \neq \emptyset$ **do**
  $U_{CTX} \leftarrow getUniqueContext(I_{Q_U})$
end for

▷ 2. Generate Contexts Sub Tree For Each Unique Context

for all $u_{CTX} \in U_{CTX}$ **do**
  $ST_{CTX_{initial}} = generateContextSubTree(u_{CTX})$
  for all $st_{CTX} \in ST_{CTX_{initial}}$ **do**
    if st_{CTX} contains at least one $I_{Q_U} \forall qt \in Q_U$ **then**
      $ST_{CTX_{cand}} \leftarrow st_{CTX}$
    end if
  end for
end for

▷ 3. Rank Context Sub Tree

$ST_{CTX} = rankContexSubTree(ST_{CTX_{cand}})$

tent keywords, both concepts need to be identified from context sub tree. Otherwise, if structural keywords are used in a query, they can be used as hints of selecting target or constraint concepts.

4.1.3 (a) Unstructured Query with Content Keywords (UQC)

First, we look at the case where unstructured query contains only content keywords. The selection of the target concept and constraint concept for this query type is described in algorithm 4.4 *Select Target and Constraint (UQC)*. This algorithm has two inputs, the query, Q_U and its context sub tree, ST_{CTX}.

The first step is to find the target of the query. As a context sub tree can be comparable to a abstracted retrieval unit. Hence, taking the root of retrieval unit as a target concept is reasonable. A target, $TARGET$ is the root of ST_{CTX} (see no. 1 in algorithm 4.4).

Now, for same context sub tree, ST_{CTX}, we can proceed to find the constraint concept for terms in the query. We first obtain a subset of concept candidates for all the terms in Q_U (see no. 2 in algorithm 4.4) using algorithm 4.3 *Select Concept in Context Sub Tree*. This subset refines our concept candidates to a context sub tree rather than the entire

collection. Since there may be more than one concept candidates, the term weighting measure, $score_{CW}$ of context-based probabilistic model is used to rank the concepts. Next, for each term, the best constraint concept is selected (see no. 3 in algorithm 4.4). The constraint pair consisting of a concept and content term will be inserted the the constraint list, $CONSTRAINT$.

Lastly, the target and constraint is returned as query interpretation,

$QI = (TARGET, CONSTRAINT)$.

Algorithm 4.3 Select Concept in Context Sub Tree

Input: unstructured query Q_U, context sub tree ST_{CTX}
Output: a concept array CPT_{Q_U} (consisting ranked concept CPT_{rank} for each content term $qt_{content}$)

▷ 1. Get Concept Candidates for Each Content Term in Query
for all $qt_{content} \in Q_U$ **do**
  $CPT_{cand} \leftarrow getDistinctConcept(ST_{CTX}, qt_{content})$
  ▷ 2. Get Concept Score
  for all $cpt_{cand} \in CPT_{cand}$ **do**
    $cpt_{cand_{score}} \leftarrow score_{CW}(ST_{CTX}, cpt_{cand}, qt_{content})$
  end for
  ▷ 3. Rank Concept
  $CPT_{rank} \leftarrow rankConceptByScore(CPT_{cand_{score}})$
end for
$CPT_{Q_{U_{qt_{content}}}} \leftarrow CPT_{rank}$

Algorithm 4.4 Select Target and Constraint (UQC)

Input: unstructured query Q_U, context sub tree ST_{CTX}
Output: query interpretation, QI, consisting a set of target, $TARGET$ and a set of constraint, $CONSTRAINT$

▷ 1. Find Target Concept
$TARGET \leftarrow ST_{CTX_{root}}$
▷ 2. Find Concept Candidates of Query
$CPT_{Q_U} = selectConceptInContextSubTree(Q_{U,ST_{CTX}})$
▷ 3. Select Best Concept as Contraint Concept
for all $qt_{content} \in Q_U$ **do**
  $CONSTRAINT_{qt_{content}} \leftarrow getBestConcept(qt_{content}, CPT_{Q_U})$
end for
$QI \leftarrow TARGET$
$QI \leftarrow CONSTRAINT$

Example 4.1. *Consider a query, "river view hotel singapore", searching a conference collection. Its content term is* $QT_{content} = \{river\ view\ hotel,\ singapore\}$ *and its con-*

text sub tree, $ST_{CTX} = \{hotel, I\}$,where $I = \{$(river view hotel,name,hotel)$_1$, (river view hotel,description,hotel)$_1$, (singapore,address,hotel)$_1\}$. We first select the root as target, $TARGET = \{hotel\}$.

Then, we select the constraint for the first content term, "river view hotel". In this example, there are two concept candidates for this term, "name" and "description" with concept score "0.64" and "0.15". The best concept, with higher score is selected, giving us the first constraint, $CONSTRAINT = \{$name:river view hotel$\}$.

Next, we proceed to select the constraint concept for the second content term in a similar manner. This gives us the constraints set, $CONSTRAINT = \{$name:river view hotel, address:singapore$\}$.

Both target and constraint concepts are returned as query interpretation,

$QI = \{hotel\}\{$name:river view hotel, address:singapore$\}$

4.1.3 (b) Unstructured Query with Content and Structural Keywords (UQCAS)

For the case where query contains both content and structural keywords, the structural keywords can serve two purposes, either as a target concept for the query or as a constraint concept describing a term. Hence, when these keywords are used, they need to be identified. Algorithm 4.5 *Selecting Target and Constraint (UQCAS)* is an extension of algorithm 4.4 *Selecting Target and Constraint (UQC)* to address this need.

This algorithm has two inputs, the query, Q_U and its context sub tree, ST_{CTX}. First, we obtain a target concept based on the root of the context sub tree. However, for this algorithm, we will identify whether the target concept is given by the query (see no. 1a in algorithm 4.5). If it is specified by user in a query, it is given a source status, USER. Otherwise, it is given a source status, SYS, to indicate that target is selected by system. The status enables us to do prioritization later such as $score(USER) > score(SYS)$.

Next, we can further identify the constraint concept for term in query (see 2. in

algo 4.5). In a similar manner, the selection of constraint concept is obtained as in the previous algorithm. However, this time we do not pick the best concept first, but based on the priority of the concept that has been stated as structural keywords in query (see 2a. in algo 4.5). If only the best concept is considered at this level, we may miss other possible concepts which may be interested by user.

If a structural keyword used specified a query matches any of the concept of a content term, it is selected as the constraint concept for the term (see 2b(i). in algo 4.5). In this case, the structural keyword is also selected as a target (see 2b(ii). in algo 4.5). The reason of selecting it as a target is because when only a structural keyword is used in a query, it is likely to be a target as well.

In the case where there is no match between structural keyword and any concept of a content term, the selection is based on the best concept (see 2c. of algorithm 4.5).

The remaining structural keywords from query are treated as possible targets of a query (see 3. in algo 4.5).

Example 4.2. *Consider a query, "address river view hotel singapore", searching a conference collection. Its content term is* $QT_{content} = \{$*river view hotel, singapore*$\}$*, structure term is* $QT_{structure} = \{$*address*$\}$ *and its context sub tree,* $ST_{CTX} = \{$*hotel, I*$\}$ *,where* $I = \{$*(river view hotel,name,hotel)*$_1$*, (river view hotel,description,hotel)*$_1$*, (singapore,address,hotel)*$_1$*, (address,hotel)*$\}$.

We first select the root as target, $TARGET = \{$*hotel*$\}$*. Since "hotel" is specified in query, it is set as source from SYS, giving us* $TARGET = \{hotel_{SYS}\}$*. (case 1c. of algorithm 4.5)*

Similar to the previous example, we select the constraint of the first content term, "river view hotel". However, this time, we need to check the concept candidates against the structural keyword first. Since neither of the candidates are specified in the query, we

proceed to use the getBestConcept *function (case 2c. of algorithm 4.5). There are two concept candidates for this term, "name" and "description" with concept score "0.64" and "0.15". The best concept, with higher score is selected, giving us the constraint,* $CONSTRAINT = \{name_{SYS}$*:river view hotel*$\}$.

Next, we proceed to select the constraint for the second content term, "singapore" in a similar manner. In this case, the concept candidate "address" matches the structural keyword specified in the query. This gives us the constraint set, $CONSTRAINT = \{name_{SYS}$*:river view hotel,* $address_{USER}$*:singapore*$\}$. *(case 2b(i). of algorithm 4.5).*

As "address" can also be meant as target of query, it is insert to the target list with a source stated USER,

$TARGET = \{hotel_{SYS}, address_{USER}\}$. *(case 2b(ii). of algorithm 4.5).*

Both target and constraint are returned as query interpretation,

$QI = \{hotel_{SYS}, address_{USER}\}\{name_{SYS}$*:river view hotel,* $address_{USER}$*:singapore*$\}$.

4.1.4 Multiple Interpretations

When interpretation is carried out for a query, we often look for an interpretation that is best fitted for the query's information needs. However, there may be situations where more than one interpretations are relevant. This often happens to a general type of query, where it is meant for collecting information, instead of wanting to find out an exact piece of information. It may also occur for query that contains only content keywords. This is because normally when structural keywords are used in a query, it will define what the query wants exactly.

There are two situations where multiple interpretations may occur. First there are more than one relevant contexts per query. Second, there is one relevant context, but this context contains more than one relevant concepts.

Algorithm 4.5 Selecting Target and Constraint (UQCAS)

Input: unstructured query Q_U, context sub tree ST_{CTX}

Output: query interpretation, QI, consisting a set of target, $TARGET$ and a set of constraint, $CONSTRAINT$

▷ 1. Find Target Concept

$TARGET_i \leftarrow ST_{CTX_{root}}$
$TARGET_{i_{status}} \leftarrow ROOT$

▷ 1a. Verify Structural Keyword in Query
for all $qt_{structure} \in Q_U$ **do** ▷ 1b. Case Structural Keyword is Target
  if match$(qt_{structure}, TARGET_i)$ **then**
    $TARGET_{i_{source}} \leftarrow USER$
    $SELECTED \leftarrow qt_{structure}$
  else ▷ 1c. Case Structural Keyword is Not Target
    $TARGET_{i_{source}} \leftarrow SYS$
  end if
end for
$i++$;

▷ 2. Find Constraint Concept
$CPT_{Q_U} = selectConceptInContextSubTree(Q_U, ST_{CTX})$
for all $qt_{content} \in Q_U$ **do**
  $CPT_{Q_{U_{qt_{content}}}} \leftarrow getConceptPerTerm(CPT_{Q_U}, qt_{content})$
  ▷ 2a. Verify Structural Keyword in Query
  for all $qt_{structure} \in Q_U$ **do**
    ▷ 2b. Case Structural Keyword is Constraint
    if match$(qt_{structure}, CPT_{Q_{U_{qt_{content}}}})$ **then** ▷ 2b(i). Select Keyword as Constraint
      $CONSTRAINT_{j_{qt_{content}}} \leftarrow qt_{structure}$
      $CONSTRAINT_{j_{qt_{content_{source}}}} \leftarrow USER$ ▷ 2b(ii). Select Keyword as Target
      $TARGET_i \leftarrow qt_{structure}$
      $TARGET_{i_{source}} \leftarrow USER$
      $SELECTED \leftarrow qt_{structure}$
      $i++$;
    else
      ▷ 2c. Case Structural Keyword is Not Constraint
      $CONSTRAINT_{j_{qt_{content}}} \leftarrow getBestConcept(qt_{content}, CPT_{Q_U})$
      $CONSTRAINT_{j_{qt_{content_{source}}}} \leftarrow SYS$
    end if
  end for
  $j++$;
end for

▷ 3. Check Remain Structural Keywords
for all $qt_{structure} \in Q_U$ **do**
  if $not(SELETED)$ **then**
    $TARGET_i \leftarrow qt_{structure}$
    $TARGET_{i_{source}} \leftarrow USER$
    $i++$;
  end if
end for
$QI \leftarrow TARGET$
$QI \leftarrow CONSTRAINT$

More Than One Contexts Multiple interpretations normally occur in general query. The current algorithm 4.2 *Find Context Sub Tree* can address this problem by preserving every possible context which are relevant to the query.

Example 4.3. *Consider a query with no specific intention, "andrew trotman" searching the conference collection, some relevant context sub trees are as follow.*

$ST_{CTX_1} = (sigir\_conf, I_1)$

$ST_{CTX_2} = (article, I_2)$

$ST_{CTX_3} = (workshop, I_3)$

$ST_{CTX_4} = (paper, I_4) \ldots$

However, though the coverage of contexts are many but there will be issue of differentiating the relevant ones from irrelevant ones. Especially, in single keyword case, it results in very high number of contexts because there is no hint for context refinement.

Example 4.4. *Consider a content term, "andrew trotman", it has 20 relevant contexts in the conference collection. Now, we consider another combination, "andrew trotman" and "wei che huang", they contain 4 relevant contexts. Whereas, "andrew trotman" and "focused retrieval", contain 3 relevant contexts. These show that our intuition above is reasonable. Nevertheless, there may be situation where two content terms have much alike roles within the same collection, they may have high contexts number too, such as for "andrew trotman" and "jaap kamps" with 18 relevant contexts.*

More Than One Concepts Multiple interpretations can also happen in the case where there are more than one concept within the same context. In this case, an extension algorithm 4.6, *Generate Context Sub Tree Variants* is used to split a context sub tree into a set of unique variants based on a finer resource id such as document id or element id.

Thus, instead of having one context sub tree with one best concept, there will be variants with different concepts.

Consider a query with terms, "andrew trotman" and a suggested context "sigir_conf". We obtain concepts like "doctoral_consortium" and "workshop" etc. It is obvious that these two concept belong to different classes, and not selecting one of them will cause loss of information. To address this issue, we allow creation of variants of the context based on both concepts.

Example 4.5. *Revisiting the above example, a context sub tree "sigir_conf" for query, "andrew trotman" composes a set of interpretation triples,* $I = (qt, cpt, ctx)_{id}$*. Each interpretation has an resource id allocated to differentiate it in terms of document, element, path etc. To construct a context sub tree variants, we classify Is based on their id.*

$I = \{$*(andrew trotman,doctoral_consortium,sigir_conf)*$_{\{1\}}$,
(andrew trotman,workshop,sigir_conf)$_{\{2\}}$, *(andrew trotman,name,sigir_conf)*$_{\{1,2\}}$,
(andrew trotman,organizer,sigir_conf)$_{\{2\}}\}$

After classifying, we obtain

$I_1 = \{$*(andrew trotman,doctoral_consortium,sigir_conf)*$_{\{1\}}$,
(andrew trotman,name,sigir_conf)$_{\{1\}}\}$

$I_2 = \{$*(andrew trotman,workshop,sigir_conf)*$_{\{2\}}$, *(andrew trotman,name,sigir_conf)*$_{\{2\}}$,
(andrew trotman,organizer,sigir_conf)$_{\{2\}}\}$

And, the context sub tree variants are

$ST_{CTX_1} = (sigir\_conf, I_1)$

$ST_{CTX_2} = (sigir\_conf, I_2)$

Hence, we obtain two query interpretations, QI, using algorithm 4.4

$QI_1 = STC(Q_U, ST_{CTX_1})$

$= \{sigir\_conf_{ROOT:SYS}\}\{doctoral\_consortium_{SYS}$*:andrew trotman*$\}$

$QI_2 = STC(Q_U, ST_{CTX_2})$

$= \{sigir\_conf_{ROOT:SYS}\}\{workshop_{SYS}:andrew\ trotman\}$

Algorithm 4.6 Generate Context Sub Tree Variants

Input: context sub tree ST_{CTX} for unstructured query, Q_U
Output: context sub tree variants $ST_{CTX_{id}}$

▷ 1. Get Resource Id For Each Interpretation Triple
for all $I \in ST_{CTX}$ **do**
  $R_{id} \leftarrow getResourceId(I)$ ▷ 2. Create New Context Sub Tree Per Resource Id
  $ST_{CTX_{id}} \leftarrow classifyContextSubTree(I, R_{id})$
end for

4.1.5 Interpretation for Query with Logical Operator

A more complex type query consists of one or more sub queries in it, connected with disjunctive (and conjunctive) logical operators. This query will be decomposed based on the usage of the operators. Such as "paper or poster by andrew trotman", into two sub queries, i.e. (a)"paper by andrew trotman" and (b)"poster by andrew trotman". Each query will be processed separately. First, we obtain query interpretations for each sub query. For example,

$QI_{a_1} = \{\text{article}_{ROOT:SYS}, \text{paper}_{USER}\}\{\text{author}_{SYS}\text{:andrew trotman}\}$.

$QI_{a_2} = \{\text{sigir\_conf}_{ROOT:SYS}, \text{paper}_{USER}\}\{\text{paper}_{USER}\text{:andrew trotman}\}$.

$QI_{a_3} = \{\text{paper}_{ROOT:USER}\}\{\text{author}_{SYS}\text{:andrew trotman}\}$.

$QI_{b_1} = \{\text{sigir\_conf}_{ROOT:SYS}, \text{poster}_{USER}\}\{\text{poster}_{USER}\text{:andrew trotman}\}$.

$QI_{b_2} = \{\text{poster}_{ROOT:USER}\}\{\text{author}_{SYS}\text{:andrew trotman}\}$.

$QI_{b_3} = \{\text{poster}_{ROOT:USER}\}\{\text{reviewer}_{SYS}\text{:andrew trotman}\}$.

Both sub queries are combined to form a single query based on their common constraint. From the above combinations, combine QI_{a_3} and QI_{b_2}, we get

$QI = \{\text{paper}_{ROOT:USER}, \text{poster}_{ROOT:USER}\}\{\text{author}_{SYS}\text{:andrew trotman}\}$.

However, if two sub queries have different constraints, combining them would result in some combinations that are incorrect. E.g. if we combine QI_{a_3} and QI_{b_3}, we get

$QI = \{\text{paper}_{ROOT:USER}, \text{poster}_{ROOT:USER}\}\{\text{author}_{SYS}$:andrew trotman, reviewer_{SYS}:andrew trotman$\}$. [bad combination]

Hence, we do not combine the sub queries for this case.

4.2 Query Representation

This section shows how the interpreted query will be represented using the intermediate query representation propoposed in the query transformation framework. The main goal in this section is to structure the query so that it can reflect the intention of the query.

4.2.1 Basic Query Construction

We begin by looking at a basic query construction for a simple query. Let us assume that the query consists of single target and a set of constraints after query interpretation from section 4.1. Now, we want to construct the interpretation into a generic query structure form. For the construction, we will be referring to the intermediate query schema (IQS) defined in Chapter 3.4.1.

Algorithm 4.7 *Construct Basic Query* describes the query construction process. This algorithm requires the query interpretation, *QI* as input. It consists of 3 main processes. First, it creates an instance of semantic query by using the *IQSNew* function. Then, it will call the *IQSNewTarget* function to create the target group as the root of semantic query. Target candidate from *QI* is added to the target group using *IQSAddTarget* function. Next, the algorithm creates a new constraint group through the *IQSNewConstraint* function. A constraint group is bound to a target group based on the $refTarget$. Constraints candidate from *QI* are added to the constraint group using *IQSAddConstraint* function. The algorithm returns the constructed query as semantic query, Q_{sem}.

Algorithm 4.7 Construct Basic Query (Single Target Multi Constraint from QI)

Input: query interpretation QI of unstructured query Q_U, IQS functions
Output: semantic query Q_{sem}

Let $relTarget = SIB$

▷ 1. Create New Semantic Query Instance

$Q_{sem} \leftarrow IQSNew()$

▷ 2. Create Main Query Target

$Q_{sem} \leftarrow IQSNewTarget(Q_{sem})$

▷ 2a. Add Query Target

$TARGET_{cand} \leftarrow TARGET \in QI$
for all $cpt \in TARGET_{cand}$ **do**
  if $status(cpt) \equiv ROOT$ **then**
    $Q_{sem} \leftarrow IQSAddTarget(cpt, Q_{sem}, refTarget, relTarget)$
    $refTarget_{value} \leftarrow cpt$
  end if
end for

▷ 3. Create Main Query Constraint

$Q_{sem} \leftarrow IQSNewConstraint(Q_{sem}, refTarget)$

▷ 3a. Add Query Constraint

$CONSTRAINT_{cand} = CONSTRAINT \in QI$
for all $cons \in CONSTRAINT_{cand}$ **do**
  $Q_{sem} \leftarrow IQSAddConstraint(cons, Q_{sem}, refTarget)$
  $refTarget_{value} \leftarrow cons$
end for

Example 4.6. *Consider a query, "river view hotel singapore", and its query interpretation, $QI = \{hotel\}\{name:river\ view\ hotel,\ address:singapore\}$. The steps of query constructions are as follow. The example is shown in Figure 4.1.*

i. *First, a new semantic query structure is instantiated, which is an empty structure, e.g. QS.*

ii. *Then, a new target is created for the structure QS. This creates a node called "TG" (target group) as the root of QS.*

iii. *The target, "hotel" is added to the "TG" node. This target, "hotel" is also set as the reference, refTarget, for binding purpose later.*

iv. *Next, a new constraint is created, binding to the the "TG" node of "hotel". This creates a node called "CSG" (constraint group).*

v. *There are two constraints given by QI. The first constraint, "name:river view" is added to constraint group, binding under the "CSG" node. This constraint, "name:river view" is then set as the reference, refTarget, for binding of the next constraint.*

vi. *Lastly, the second constraint, "address:singapore" is added to the same constraint group, binding under the "CSG" node as the sibling of previous constraint.*

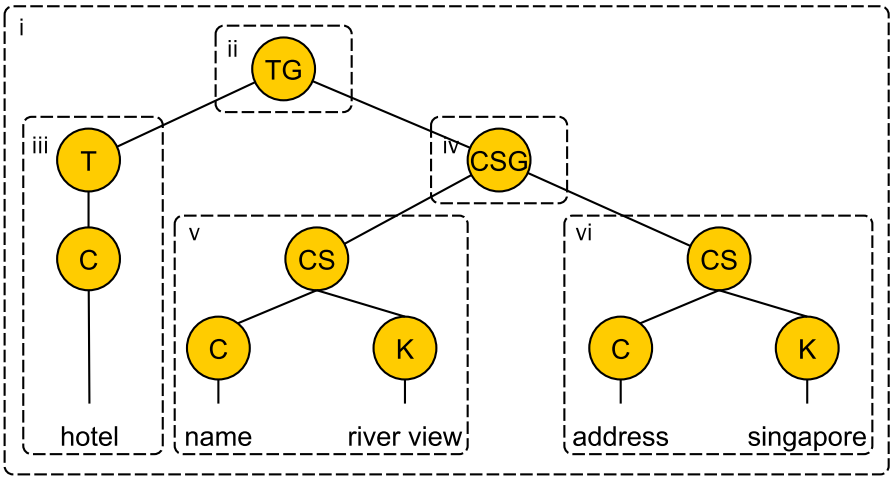


Figure 4.1: Example of Basic Query Construction

Reference of Target, *refTarget* Reference of target is used to identify which target group (or constraint group) a new target (or constraint) is to be bound to. For example, in order to add the second constraint to the same constraint group as the first constraint (see step vi. in Example 4.6), *refTarget* with the value of first constraint is used as reference to identify the correct constraint group.

Relationship with Target, *relTarget* Relationship with target is used to state whether a new target will be bound as a sibling, SIB, or a child, CHI, to the current target in target group. We show the difference between sibling binding and child binding in figure 4.2. Sibling binding is used to indicate two different retrieval concepts. For example, between i. and ii. in figure 4.2. This structure means that either "paper" or "poster" where its "title" contains "xml" will be retrieved. Whereas child binding is used to narrow down the scope

of the main target. For example, between iii. and iv. in the same figure. As the target "paper" is added to the target "conference" as a child, it is used to refines the scope of "conference" to "conference\paper". This structure means that "conference\paper" where "title" contains "xml" will be retrieved.

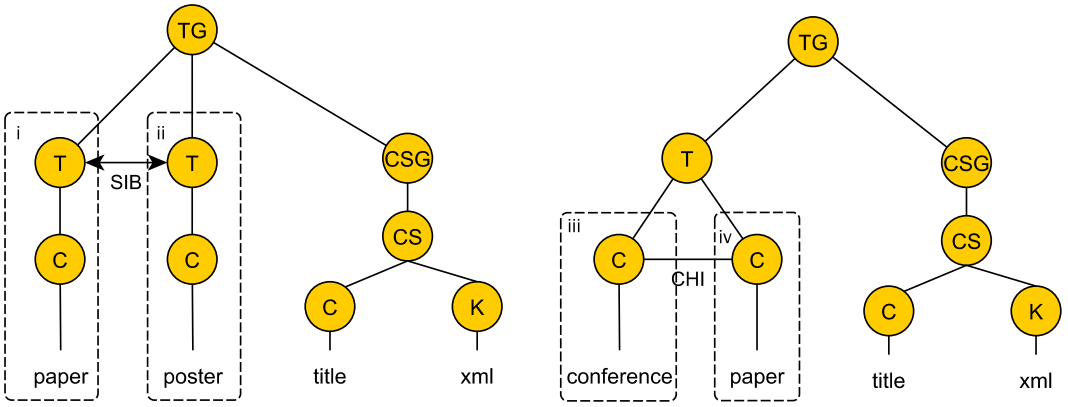


Figure 4.2: SIBling Binding vs. CHIld Binding for Multiple Targets

4.2.2 Query Construction for Multiple Targets

Multiple targets occur when there are more than one retrieval unit types which are relevant to the query. This situation can occur directly (user specifies multiple targets in query) or indirectly (system suggested multiple targets from its interpretation). In both cases, when multiple targets are suggested from query interpretation, they can be interpreted as follows.

a. there are multiple specific targets in additional to a main target.

b. there are multiple main targets.

Now, we proceed to see how these targets are structured. We shall discuss the case based on how the targets are interpreted in query interpretation, on whether they are contained in the same query interpretation (a.), i.e. single *QI*, or the targets are contained in different query interpretations (b.), i.e. variants of *QI*s.

Algorithm 4.8 *Construct Complex Query (Multiple Targets from Single QI)* shows how additional targets are constructed in a semantic query. This algorithm requires a query interpretation, *QI* as input. This main difference of this algorithm from algorithm 4.7 is the handling of its additional targets suggested from *QI*. We call these targets as refining targets, as they are meant for refining the main target. There are two types of refinements, i.e. as additional targets of the main target or as the sub target of the main target (see figure 4.3).

After the main target has been created (2a in algorithm 4.8) using the *IQSAddTarget* function, we proceed to identify additional target from *QI* by referring to its target status. Every remaining targets with ROOT status will be added to the same target as its siblings. See (figure 4.3a.) for example, target "paper" is the first main target created for the Q_{sem}, then target "poster" is added to the target group of "paper" as sibling.

Sub Targets Then, we proceed to process the remaining target candidates. Remaining targets are used to state a finer retrieval unit within the scope of main target, by placing them as sub targets of the main target. When placed as a sub target, its purpose is to return portion of the retrieval unit that matches the main target. To do this, algorithm 4.8 2c. creates a new target group under the existing one, indicating that these targets are refinements for the higher level target. Every remaining targets will be added to this new group. See (figure 4.3b.) for example, target "hotel" is the main target created for the Q_{sem}, then target "address" is added to the target group nested under the main target "hotel" as sub target.

Special Sub Target In a special case where a fine target is actually the retrieval unit itself but the suggested main target is a broader target. This occur for a query like "paper about xml", where the structural keyword "paper" is meant to be the query target. Based

on our intuition, when only one structural keyword is used, it has high possibility of being a target of the query. However, when interpreted, we may get some broader concepts as main target like “demo_papers”, “accepted_posters”, instead of “paper”. Since “paper” also a target, it will be used as a fine target. The issue is, it is semantically incorrect to be placed as sub target, as it will return all the papers of “accepted_posters”. In this case, “paper” needs to be placed as the sub path of the main target “accepted_posters”.

The identification of special sub target is carried out by checking against the content term. Here, we look at the content term “xml” and interpretation, QI for this query. It is given as,

$QI = \{\text{accepted\_posters}_{ROOT:SYS}, \text{paper}_{USER}\}\{\text{paper}_{USER}:\text{xml}\}$

When the fine target, “paper” is also suggested as the constraint of the content term, this shows that structurally, the fine target cannot be placed lower than or same level as the constraint. As Algorithm 4.8 2b. addresses this case by placing the fine target as a child of the main target (see (figure 4.3c.).

4.2.3 Query Construction for Multiple Query Interpretations

As discussed in section 4.1.4, there may be more than one interpretations. Revisiting this example, there are two query interpretations, QI, obtained for the query “andrew trotman”.

$QI_1 = \{\text{sigir\_conf}_{ROOT:SYS}\}\{\text{doctoral\_consortium}_{SYS}:\text{andrew trotman}\}$

$QI_2 = \{\text{sigir\_conf}_{ROOT:SYS}\}\{\text{workshop}_{SYS}:\text{andrew trotman}\}$

The construction queries for both interpretations will be carried out separately using algorithm 4.7 *Construct Basic Query*, leading to two semantic queries, Q_{sem}s.

Combining Queries The queries are combined using the IQS aggregation rule for constraint, CS.

$Q_{sem_1} = \{$

Algorithm 4.8 Construct Complex Query (Multiple Targets from Single QI)

Input: query interpretation QI of unstructured query Q_U, IQS functions
Output: semantic query Q_{sem}

Let $refTarget = null, relTarget = SIB$

▷ 1. Create New Query Instance

$Q_{sem} \leftarrow IQSNew()$

▷ 2. Create Main Query Target

$Q_{sem} \leftarrow IQSNewTarget(Q_{sem}, refTarget)$

▷ 2a. Add Main Query Targets

$TARGET_{cand} \leftarrow TARGET \in QI$
for all $cand \in TARGET_{cand}$ **do**
  if $status(cand) \equiv ROOT$ **then**
    $Q_{sem} \leftarrow IQSAddTarget(cand, Q_{sem}, refTarget, relTarget)$
    $refTarget \leftarrow cand$
    remove $cand$ from $TARGET_{cand}$
  end if
end for

▷ 2b. Remain Target Candidates (Add as Child of Main Target)

for all $cand \in TARGET_{cand}$ **do**
  if $source(cand) \equiv USER$ AND $matchConstraint(cand, CONSTRAINT)$ **then**
    $refTarget = null$
    $relTarget = CHI$
    $Q_{sem} \leftarrow IQSAddTarget(cand, Q_{sem}, refTarget, relTarget)$
  end if
end for

▷ 2c. Remain Target Candidates (Add as Sub Target of Main Target)

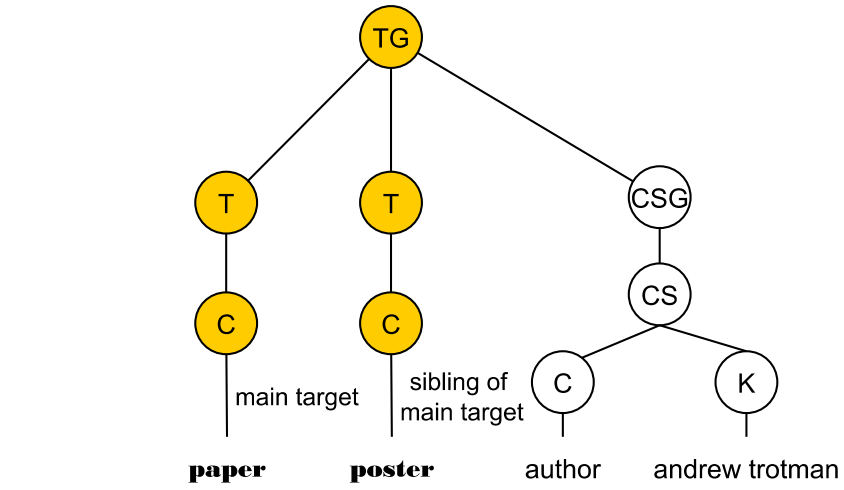
$Q_{sem} \leftarrow IQSNewTarget(Q_{sem}, refTarget)$
$relTarget = SIB$
for all $cand \in TARGET_{cand}$ **do**
  $Q_{sem} \leftarrow IQSAddTarget(cand, Q_{sem}, refTarget, relTarget)$
end for

▷ 3. Create Main Query Constraint

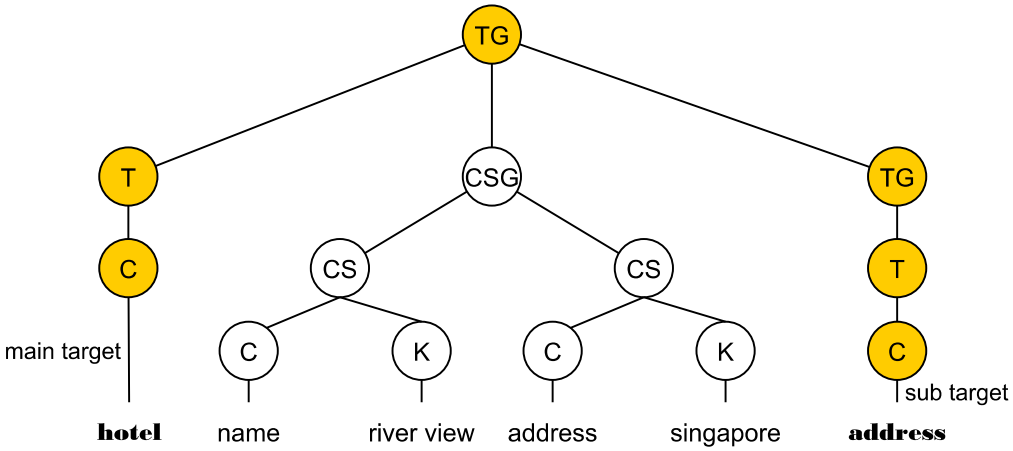
$Q_{sem} \leftarrow IQSNewConstraint(Q_{sem}, refTarget)$

▷ 3a. Add Query Constraint

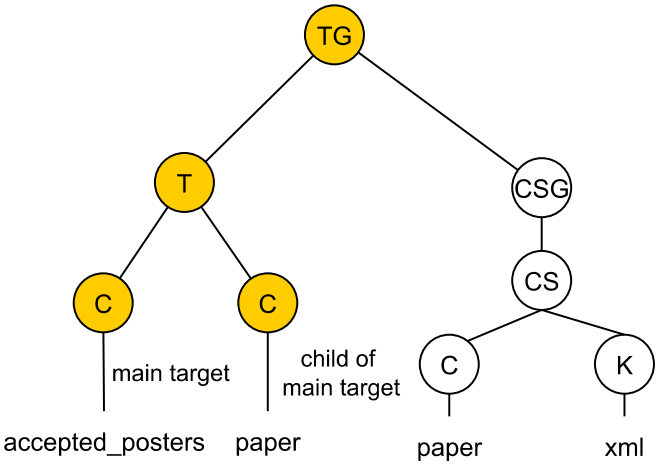
$CONSTRAINT_{cand} \leftarrow CONSTRAINT \in QI$
for all $cand \in CONSTRAINT_{cand}$ **do**
  $Q_{sem} \leftarrow IQSAddConstraint(cand, Q_{sem}, refTarget)$
end for



a. Multiple main targets



b. Main target and fine target (as sub targets)



c. Main target and fine target (as main target child)

Figure 4.3: Query Construction for Multiple Targets

$TG_1/T_1/C_1/$sigir_conf,

$TG_1/CGS_1/CS_1/C_1/$doctoral_consortium,

$TG_1/CGS_1/CS_1/K_1/$andrew trotman $\}$

$Q_{sem_2} = \{$

$TG_1/T_1/C_1/$sigir_conf,

$TG_1/CGS_1/CS_1/C_1/$workshop,

$TG_1/CGS_1/CS_1/K_1/$andrew trotman $\}$

$CONS\_AGG(Q_{sem_1}, Q_{sem_2}) = \{$

$TG_1/T_1/C_1/$sigir_conf,

$TG_1/CGS_{[OR]1}/CS_1/C_1/$doctoral_consortium,

$TG_1/CGS_{[OR]1}/CS_1/K_1/$andrew trotman,

$TG_1/CGS_{[OR]1}/CS_2/C_1/$workshop,

$TG_1/CGS_{[OR]1}/CS_2/K_1/$andrew trotman $\}$

Nested Constraints In the previous, we have discussed the aggregation of query interpretations with single constraint in each of them. Now, we shall look at a more complex case when there are multiple constraints per interpretations. These constraints will be combined as nested constraint group, instead of individual constraint.

Let us look at a more complex example in Figure 4.5, there are also two query interpretations, *QI*, obtained for the query "paper andrew trotman(xml OR inex)". The construction queries for both interpretations will be carried out separately using algorithm 4.7 *Construct Basic Query*, leading to two semantic queries, Q_{sem}s. The queries are combined using the IQS aggregation rule for constraint group, CSG.

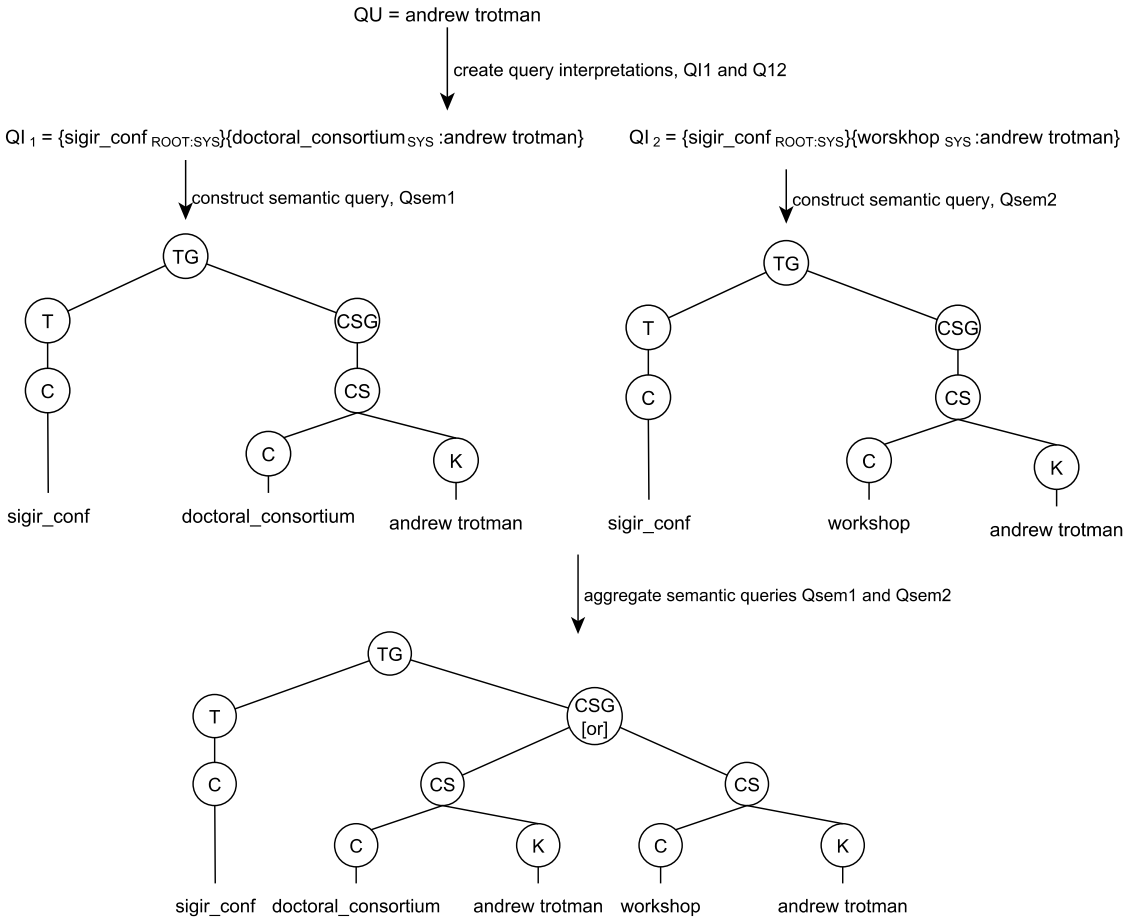


Figure 4.4: Query Construction for Multiple Query Interpretations

4.3 Query Mapping

Having interpreted and represented the query, this section describes how the query are transformed to a query language form, so that the query can be processed by retrieval system. The mapping of semantic query to a language type is carried out matching it with the syntax query structure of that language. Given a semantic query, we will look for the best match syntax query structure, and then convert the contents from semantic query to language via the query structure.

4.3.1 Schema Matching

We begin by discussing the matching between semantic query structure and syntax query structure. Consider a semantic query structure, Q_{sem} obtained from the query interpretation and representation, and a set of syntax query structure, Q_{syn} from the resource

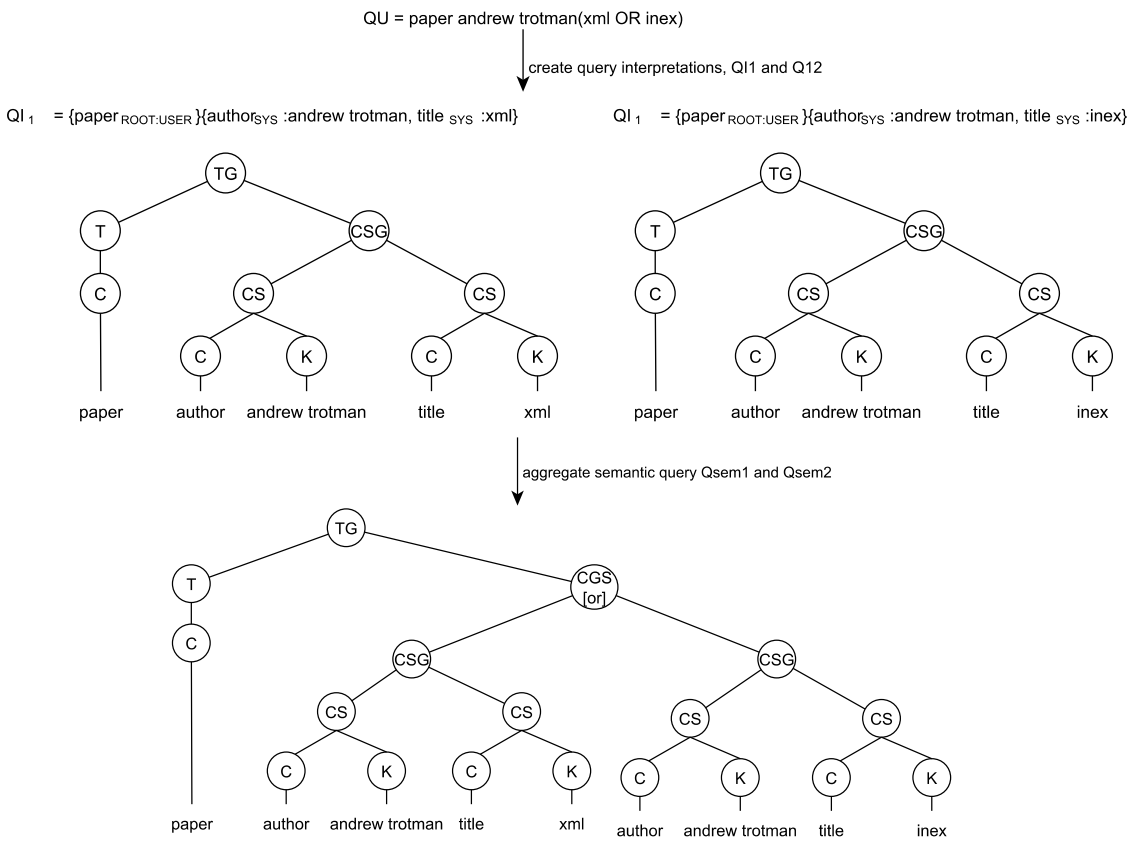


Figure 4.5: Query Construction for Multiple Query Interpretations (Nested Constraints)

of mappings for a language type, X. The problem of schema matching is to find the best match structure between Q_{sem} and Q_{syn} so that no information lost will occur at this stage of conversion to language string (see Figure 4.6).

Exact Match Algorithm 4.9 describes the matching of both semantic and syntax queries. Finding an exact match is quite straight forward. Comparison between paths of schema can be used to check whether both schemas match. However, if an exact match cannot be found, i.e. the syntax query is not available in the resource, then we need to look for a nearest partial match.

Partial or Overwhelm Match Algorithm 4.10 describes partial or overwhelm matching of both semantic and syntax queries. When an exact match is not found, a detailed matching is employed based on the overlapping paths of both query schemas. The selection of syntax query is carried out based on the highest score of schema with respect to

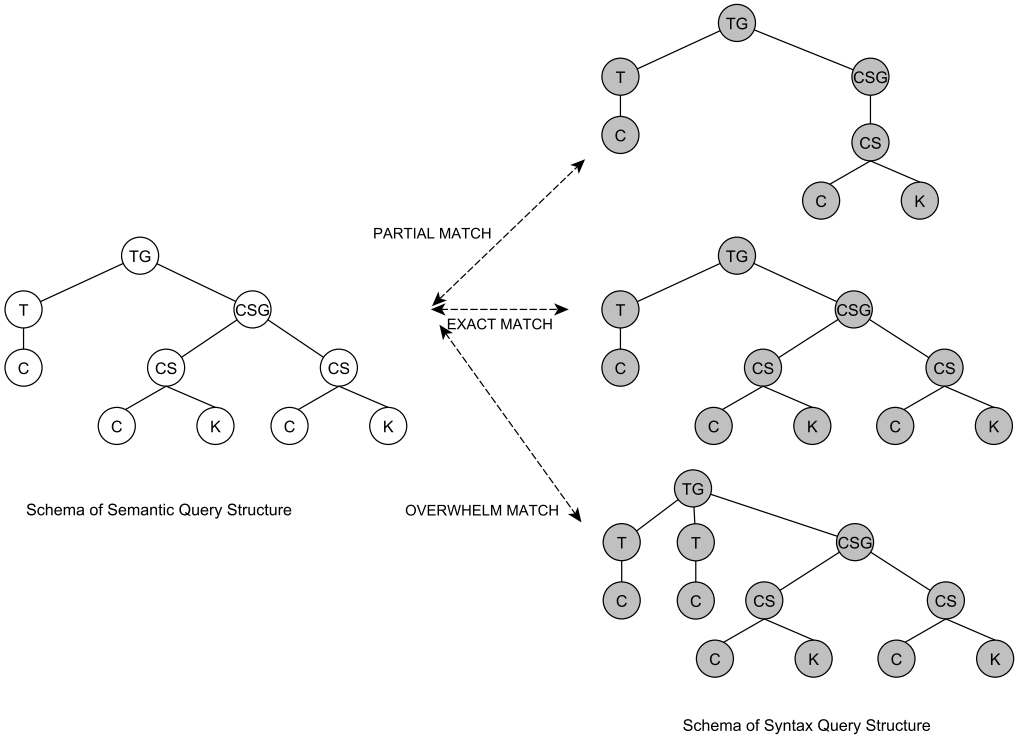


Figure 4.6: Finding Best Match Structure

Algorithm 4.9 Match Schema of Semantic Query and Syntax Query

Input: semantic query, Q_{sem} of unstructured query Q_U, a set of syntax query Q_{syn} from knowledge base, KB_X of a language type, X

Output: syntax query $Q_{syn_{exact}}$ of a language type

$schema_{Q_{sem}} \leftarrow getSchema(Q_{sem})$
$schema_{Q_{syn}} \leftarrow getSchema(Q_{syn})$
for $schema \in schema_{Q_{syn}}$ **do**
  if $match(schema_{Q_{sem}}, schema)$ **then**
    $Q_{syn_{exact}} \leftarrow schema$
    break
  end if
end for

the semantic query schema.

$$schemaSimilarity(Q_{sem}, Q_{syn}) = \frac{\frac{totalMatchPath_{Q_{sem},Q_{syn}}}{totalPath_{Q_{sem}}} + \frac{totalMatchPath_{Q_{sem},Q_{syn}}}{totalPath_{Q_{syn}}}}{2}$$

For the situation of non exact match, there can be two cases of a selected schema, i.e. overwhelm match and partial match. An overwhelm match happens when a syntax query structure is able to represent more than the required contents of the semantic query

(see Figure 4.7). Hence, it consists of structures which may not be utilized. When there are two syntax query structures which are both overwhelm match, the one with the higher score is selected as it has less under utilized structures.

Algorithm 4.10 Partial and Overwhelm Match Schema of Semantic Query and Syntax Query

Input: semantic query, Q_{sem} of unstructured query Q_U, a set of syntax query Q_{syn} from resource, R_X of a language type, X

Output: syntax query $Q_{syn_{sele}}$ of a language type

Let $PARTIAL_{score} = 0$
Let $OVER_{score} = 0$

$schema_{Q_{sem}} \leftarrow getSchema(Q_{sem})$ ▷ 1. Split semantic query schema into paths
$Q_{sem_{path}} \leftarrow getPath(schema_{Q_{sem}})$
▷ 2. For each syntax query schema, split schema into paths
for all $Q_{syn} \in R_X$ **do**
  $schema_{Q_{syn}} \leftarrow getSchema(Q_{syn})$
  $Q_{syn_{path}} \leftarrow getPath(schema_{Q_{syn}})$
  ▷ 3. Similarity Score Calculation
  $score_{Q_{syn}} \leftarrow schemaSimilarity(schema_{Q_{sem}}, schema_{Q_{syn}})$
  ▷ 4. Overwhelm Match Case
  if $totalMatchPath_{Q_{sem},Q_{syn}} == totalPath_{Q_{sem}}$ **then**
    if $score_{Q_{syn}} > OVER_{score}$ **then**
      $Q_{syn_{over}} \leftarrow Q_{syn}$
      $OVER_{score} \leftarrow score_{Q_{syn}}$
    end if
  ▷ 5. Partial Match Case
  else
    if $score_{Q_{syn}} > PARTIAL_{score}$ **then**
      $Q_{syn_{partial}} \leftarrow Q_{syn}$
      $PARTIAL_{score} \leftarrow score_{Q_{syn}}$
    end if

  end if
end for

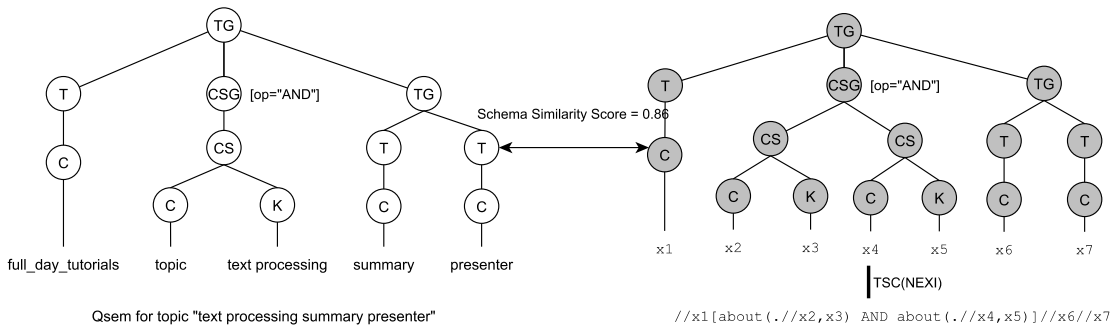


Figure 4.7: Case of Overwhelm Match for Query Schema Matching

Example 4.7. *Consider a query, "text processing summary presenter" and its interpreted semantic query structure, Q_{sem} as shown in Figure 4.7. When an exact match of syntax query structure is not available, the most similar one will be selected. Here, we show how the scoring of schema similarity is obtained.*

Paths for Q_{sem}, "text processing summary presenter",
$= \{TG_1/T_1/C_1,$
$TG_1/CSG_{1,op="AND"}/CS_1/C_1,$
$TG_1/CSG_{1,op="AND"}/CS_1/K_1,$
$TG_1/TG_1,T_1/C_1,$
$TG_1/TG_1,T_2/C_1\}$

$totalPath_{Q_{sem}} = 5$

Paths for Q_{syn} from NEXI knowledge base,
$= \{TG_1/T_1/C_1,$
$TG_1/CSG_{1,op="AND"}/CS_1/C_1,$
$TG_1/CSG_{1,op="AND"}/CS_1/K_1,$
$TG_1/CSG_{1,op="AND"}/CS_2/C_1,$
$TG_1/CSG_{1,op="AND"}/CS_2/K_1,$
$TG_1/TG_1,T_1/C_1,$
$TG_1/TG_1,T_2/C_1\}$

$totalPath_{Q_{syn}} = 7$

$totalMatchPath_{Q_{sem},Q_{syn}} = 5$

Schema Similarity between Q_{sem} and Q_{syn}
$= \frac{\frac{5}{5}+\frac{5}{7}}{2}$

=0.86

Since the total matched paths of both structures (Q_{sem} and Q_{syn}) is equivalent to the total path of Q_{sem} (see 4. in Algorithm 4.10), this means that the query contents of Q_{sem} can be fully mapped to Q_{syn} despite the differences of their structures. In this case, Q_{syn} is classified as overwhelm match. An overwhelm match is preferred to partial match as it can fully convert the contents of Q_{sem}.

A partial match shall represent part of the information from the semantic query.

Although partial match is still allowed when exact match is not found. However, there is still a minimum rule where a partial match can be optimized. For example, priority can be placed on partial match that has higher score of unique path of constraint rather than target. This is because loosing information about constraint will affect the results, as content keywords are omitted. Whereas, loosing information about target will only affect the granularity of retrieval unit.

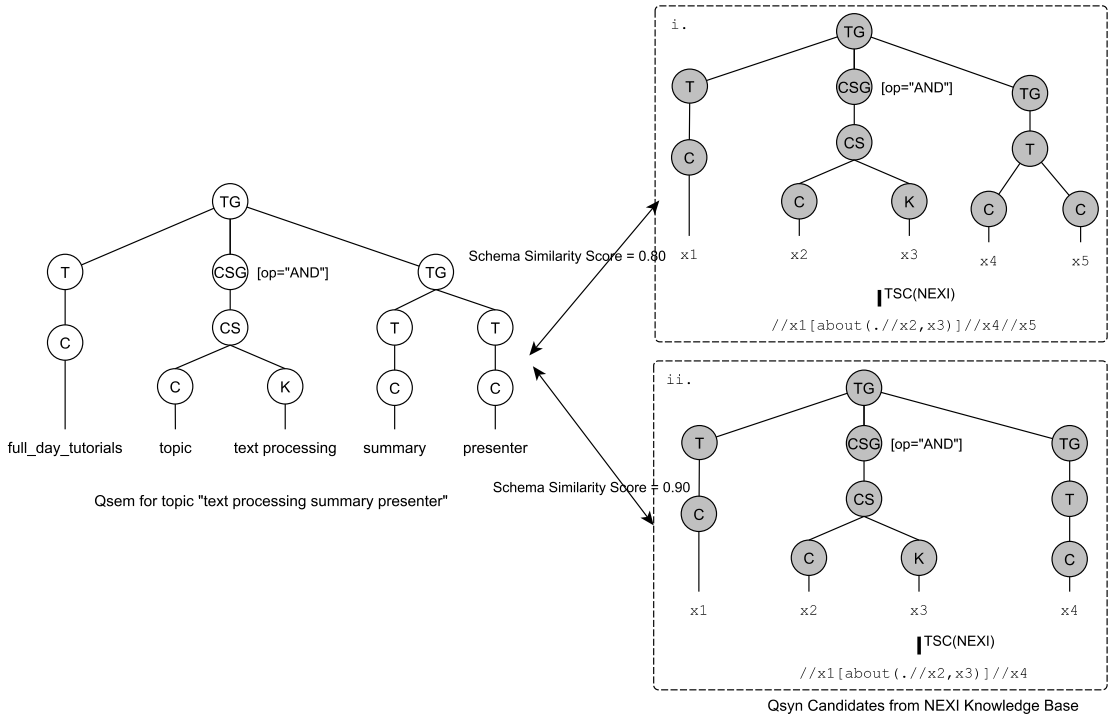


Figure 4.8: Case of Partial Match for Query Schema Matching

4.3.2 Query Content Mapping

Once a matched schema of syntax query structure is found, the contents of semantic query structure can be mapped to its query language form. The mapping is carried out by matching the path of a leaf node (containing content) of semantic query structure, with the path of a leaf node (containing variable) of syntax query structure. Path selection method like xpath can be used to identify the corresponding path of two query structures.

Example 4.8. *Let us revisit the query from example 4.6, "river view hotel singapore", its semantic query, Q_{sem}, and an exact matched syntax query, Q_{syn} of a query language type,*

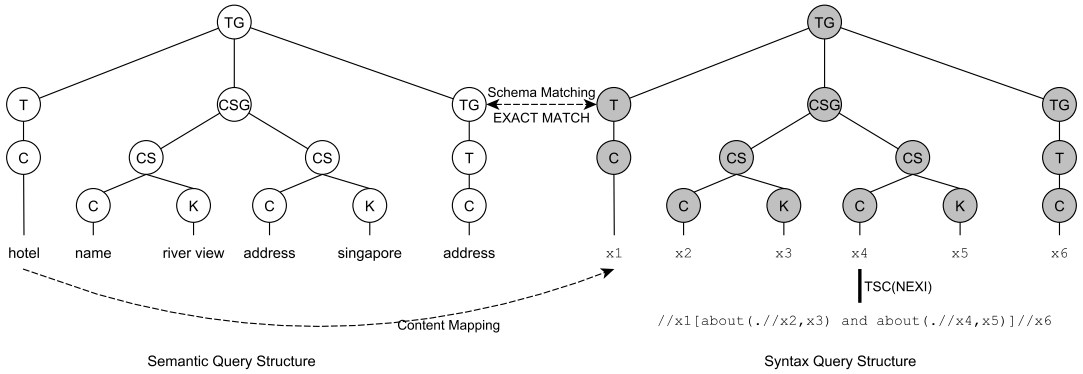


Figure 4.9: Content Mapping between Semantic Query and Syntax Query

NEXI (see figure 4.9).

1. *For example, to map the leaf content "hotel" from Q_{sem} to Q_{syn}, we first get the path of this leaf node at Q_{sem}.*

   getPath(Q_{sem},"hotel") = $TG_1/T_1/C_1$

2. *Then, we obtain the corresponding variable node from Q_{syn} by mapping the path from Q_{sem} to Q_{syn}.*

   mapPath(Q_{syn}, "$TG_1/T_1/C_1$") = $x1$

3. *After the variable node is found, mapping between the content node and variable node is formed.*

   formMapping("hotel",x1) = {"hotel",x1}

4. *After we have obtained all the corresponding variable nodes, we convert the contents to the query language string. A simple way to carry out content conversion is by placing the variables of M to STR as shown in figure 4.10. Hence, we obtain the structured query string for NEXI query language as follow.*

   STR = "//hotel[about(.//name,river view) and about(.//address,singapore)]//address"

Another Query Language The same conversion can be carried out for another query language. Let us consider another query language, *XMLFragment*. Given that an exact

Id	*M*
x1	hotel
x2	name
x3	river view
x4	address
x5	singapore
x6	address

Id	*STR*
0	//
1	*x*1
2	[about(.//
3	*x*2
4	,
5	*x*3
6	) and about(.//
7	*x*4
8	,
9	*x*5
10	)]//
11	*x*6

Figure 4.10: Mappings, *M* to Query String, *STR*, of *NEXI*

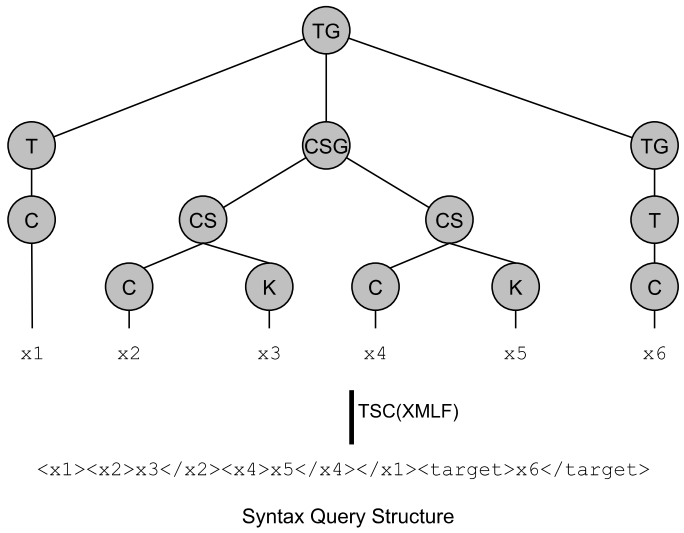


Figure 4.11: Syntax Query Structure for *XMLFragment*

match syntax query structure, Q_{syn} for this query language is available (as shown in figure 4.11). Similarly, we carry out content conversion by placing the variables of *M* in *STR* as shown in figure 4.12.

Hence, we obtain the structured query string for *XMLFragment* query language as follow.

STR = “<hotel><name>river view</name><address>singapore</address></hotel>

<target>address</target>”

Id	M
x1	hotel
x2	name
x3	river view
x4	address
x5	singapore
x6	address

Id	STR
0	$<$
1	$x1$
2	$><$
3	$x2$
4	$>$
5	$x3$
6	$</$
7	$x2$
8	$><$
9	$x4$
10	$>$
11	$x5$
12	$</$
13	$x4$
14	$></$
15	$x1$
16	$>< target >$
17	$x6$
18	$< /target >$

Figure 4.12: Mappings, M to Query String, STR, of $XMLFragment$

4.4 Query Selection

The last section of this chapter discusses the selection of interpreted queries. Although now we have obtained a number of interpreted queries, there are some issues like too many interpretations or bad interpretations. The main aim of query selection is to address these issues by methods such as carry out overall ranking, suggesting best query and finding cut off factors for too many interpretations. Revisit section 4.1.2, we have earlier stated that a context sub tree is the central of a retrieval unit, now we shall discuss the scoring of the context sub tree for determining query ranking.

4.4.1 Query Ranking

Although there are a number of factors that affect the scoring for query ranking, the first important score for an interpreted query is based on its context sub tree. And, the scoring of context sub tree is based on two factors, first, to find a context sub tree that gives the best retrieval unit, and second, to find the best term concept within that context sub tree in order to filter out irrelevant retrieval unit.

Context Sub Tree Ranking A good context sub tree reflects a good retrieval unit for a query. The features that we desire is basically based on these two guidelines, i) it is neither too big that it contains too much information, ii) it is not too small that it contains too little information resulting in non meaningful unit.

Based on the guidelines above, we present $C_{retrieval}(ST_{CTX}, Q)$, which is the confidence of a context sub tree to be the desired retrieval unit for a given query, Q, as follows:

$$C_{retrieval}(ST_{CTX}, Q) = \prod_{qt \in Q} score_{CTXPROX}(ST_{CTX}, Q) * score_{CTX}(ST_{CTX}),$$

where $score_{CTXPROX}(ST_{CTX}, Q)$ measures the importance of ST_{CTX} wrt. Q, and $score_{CTX}(ST_{CTX})$ measures the importance of ST_{CTX} in the collection, D. The first factor ($score_{CTXPROX}(ST_{CTX}, Q)$) reflects that the more frequent a query appears in a context sub tree, the more relevant the context sub tree is to the query. It also reflects that the smaller distance (edge distance) a query locates from the root of a context sub tree, the more semantically relevant they are. The second factor ($score_{CTX}(ST_{CTX})$) reflects that the more frequent a context sub tree appears in a collection, the most likely it is an important retrieval unit. Product is used in the first factor to ensure that all keywords are taken into considerations. Therefore, if a sub tree cannot contain all the keywords in the query, we will get a confidence score of 0.

Algorithm 4.11 shows the scoring and ranking of context sub trees using the confidence of a context sub tree, $C_{retrieval}(ST_{CTX}, Q)$. The algorithm returns a set of ranked context sub trees.

Example 4.9. *Now, we show how the confidence scoring differentiates between sub trees of different granularities. Consider the query, Q,"hotel river view", there are some candidates of context sub tree, such as*

$ST_{CTX_{hotel}} = \{hotel, I_{ST_{CTXhotel} \cap Q}\}$.

$ST_{CTX_{hotel\_information}} = \{hotel\_information,\ I_{ST_{CTXhotel\_information \cap Q}}\}.$

$ST_{CTX_{rates\_of\_hotel}} = \{rates\_of\_hotel,\ I_{ST_{CTXrates\_of\_hotel \cap Q}}\}.$

$ST_{CTX_{accommodation}} = \{accommodation,\ I_{ST_{CTXaccommodation \cap Q}}\}.$

The confidence score for each context sub tree wrt. the query is as follows.

$C_{retrieval}(ST_{CTX_{hotel}}, \{river\ view,\ hotel\}) = [0.133 * 1.000] * [\frac{17}{17}] = 0.133$

$C_{retrieval}(ST_{CTX_{hotel\_information}}, \{river\ view,\ hotel\}) = [0.286 * 1.000] * [\frac{1}{17}] = 0.019$

$C_{retrieval}(ST_{CTX_{rates\_of\_hotel}}, \{river\ view,\ hotel\}) = [0.286 * 1.000] * [\frac{1}{17}] = 0.019$

$C_{retrieval}(ST_{CTX_{accommodation}}, \{river\ view,\ hotel\}) = [0.100 * 0.500] * [\frac{2}{17}] = 0.006$

There are two parts of scoring in this example (shown in two square brackets). The first part shows scores obtained for each keywords from the first factor, $score_{CTXPROX}(ST_{CTX}, \{river\ view,\ hotel\})$. *Let us look at the first keyword. The first keyword, "river view", is a content keyword, hence its* $score_{CTXPROX}$ *is based on the density of the keyword (measuring frequency), and distance of the keyword (measuring depth). Due to its high density in sub trees,* $ST_{CTX_{hotel\_information}}$ *and* $ST_{CTX_{rates\_of\_hotel}}$*, the first keyword contributes to a higher score in these two sub trees compare to others.*

The second keyword, "hotel", is a structural keyword. Its $score_{CTXPROX}$ *is based on structural distance between the structure and the sub tree (measuring depth). If we look at "hotel" keyword in* $ST_{CTX_{accommodation}}$ *context sub tree, it is nested deeper in the sub tree compare to others. The deeper a keyword is nested, the less semantically relevant it is with the sub tree, the lower score it gets.*

The second part of confidence scores shows the overall importance of the sub tree. In this example, the $score_{CTX}(ST_{CTX})$ *factor is normalized with max frequency from this subset. We can see that in this collection, the* $ST_{CTX_{hotel}}$ *is more popular compare to the rest.*

Lastly, we rank the context sub trees by their confidence scores, that serve as the preliminary ranking for interpreted query later.

Algorithm 4.11 Rank Context Sub Tree

Input: unstructured query Q_U, a set of context sub tree candidate, $ST_{CTX_{cand}}$
Output: a set ranked context sub tree ST_{CTX}

▷ 1. Score Context Sub Tree by Confidence Score

for all $st_{CTX} \in ST_{CTX_{cand}}$ **do**
  for all $qt \in Q_U$ **do**
    $score_{CTXPROX_{qt}} \leftarrow score_{CTXPROX}(st_{CTX}, qt)$
  end for
  $score_{CTXPROX}(Q_U) \leftarrow \prod_{qt \in Q_U} score_{CTXPROX_{qt}}$
  $st_{CTX_{C_{retrieval}}} \leftarrow score_{CTXPROX}(Q_U) * score_{CTX}(st_{CTX})$
end for

▷ 2. Rank Context Sub Tree by Confidence Score

$ST_{CTX} \leftarrow sortCScore(st_{CTX_{C_{retrieval}}})$

Context Sub Tree Ranking Refining (with Concepts Weighting) The ranking of query can be further refined using the importance of concepts selected for query terms. This is because each concept selected for terms in query may also affect the selection of context sub tree. Especially in the case where the earlier factors, $score_{CTXPROX}(ST_{CTX}, Q)$ and $score_{ST_{CTX}}$, could not differentiate the sub tree. Considering this, we present $C_{retrieval}(ST_{CTX}, ST_{CTX_{CPT}}$ which is the confidence of a context sub tree with concepts weighting of terms in query, Q, as follows: $C_{retrieval}(ST_{CTX}, ST_{CTX_{CPT}}, Q)$

$= \prod_{qt \in Q, cpt \in CPT} score_{CTXPROX}(ST_{CTX}, Q) score_{CW}(ST_{CTX_{CPT}}, Q) * score_{CTX}(ST_{CTX})$,

where $score_{CW}(ST_{CTX_{CPT}}, Q)$ measures the importance of CPT wrt. $qt \in Q$ in ST_{CTX}.

Example 4.10. *From the previous example, we can see that the context sub tree, $ST_{CTX_{hotel\_information}}$ and $ST_{CTX_{rates\_of\_hotel}}$ have the same scores for its ranking. In this case, we can carry out further analysis using the weight of concept selected for the query term. Now, we show how the confidence scoring differentiates when concepts of terms are incorporated between these sub trees. Consider the query, Q,"hotel river view", and the $ST_{CTX_{hotel\_information}}$ and $ST_{CTX_{rates\_of\_hotel}}$ context sub trees,*

The new confidence score for each context sub tree wrt. the best concept of term in query is as follows.

$C_{retrieval}(ST_{CTX_{hotel\_information}}, ST_{CTX_{hotel\_information:\textbf{hotel}\cap river\ view}}, \{river\ view,\ hotel\})$

$= [(0.286 * \textbf{0.592}) * (1.000)] * [\frac{1}{17}] = 0.010$

$C_{retrieval}(ST_{CTX_{rates\_of\_hotel}}, ST_{CTX_{rates\_of\_hotel:\textbf{hotel}\cap river\ view}}, \{river\ view,\ hotel\})$

$= [(0.286 * \textbf{0.384}) * (1.000)] * [\frac{1}{17}] = 0.007$

Here, we can see that the context sub trees can be weighted based on the importance of the selected concept, "hotel" for the query term, "river view", given by the factor $score_{CW}(ST_{CTX_{CPT}}, Q)$. *As such, now we can refine the rank of the sub tree candidates based on this new scores.*

4.4.2 Query Ranking with User Confidence

In this section, we discuss how query ranking can be prioritized with user input. Since the a given query can consist of structural keywords (assume written correctly with some knowledge assistances), when these keywords are used, query ranking will give priority to these keywords. As such, we introduct a rank confidence scoring, C_{user} based on whether the concepts given in a query interpretation, QI, are suggested by user or system.

$$C_{user} = \frac{\sum concept_{USER}}{\sum [concept_{USER} \cup concept_{SYS}]}$$

For example, there are two queries, Q_1, "river view" and Q_2, "hotel river view". Both queries give the same interpretations,

$QI_1 = \{\text{hotel}_{ROOT:SYS}\}\{\text{name}_{SYS}\text{:river view hotel}\}$.

$QI_2 = \{\text{hotel}_{ROOT:USER}\}\{\text{name}_{SYS}\text{:river view hotel}\}$. Both query interpretations lead to the same query output, such as for language, NEXI,

STR_1 = "//hotel[about(.//name,river view)]", where $C_{user} = 0$

STR_2 = "//hotel[about(.//name,river view)]", where $C_{user} = 0.5$

Although both outputs are the same, however, the user confidence level of STR_2 is higher than STR_1.

4.4.3 Query Ranking with External Knowledge

Basically, just like any retrieval result, it is hard to determind a cut off point of the result list. Hence, most common approach to address this issue is to optimize the rank as good as possible, and leave the decision to user. Similarly, query ranking can also involve user feedbacks in the query formulation stage, before the actualy results are returned.

User Feedbacks In this framework, user feedback can be sought for both target and constraint of query interpretation. Such as, for a query, "hotel river view", user can be opted with suggesion options like "hotel information" and "rates of hotel", which are created from the target of different query interpretations. Once a user picks an option, it will refine the query to a smaller subset. User feedback can be interactive, allowing finer concepts options within the subset, to reach a final query.

Domain Knowledge Besides having user feedbacks, a predefined domain knowledge can also be used in query ranking. Domain knowledge containing a set of common object or element types and properties can be used to boost the rank of query. By referring to this knowledge, we can ensure that query's target that is too broad, e.g. grouping of types (e.g. authors, list) or incommon (or general) concepts (e.g. interested_places, instructions_for_authors) are ranked lower.

4.5 Summary

We have shown how our query transformation framework can be applied in interpreting, representing and mapping unstructured query to structured query language. In this chapter, we describe in detail how the framework handles issues arisen from the entire transformation process. For query interpretation (section 4.1), we have shown the

creation of context sub trees based on the knowledge generated from our context-based probabilistic approach. Then, we show interpretation of query based on these sub trees, where we focus on two kinds of concepts interpretations, the target and the constraint of the query. Following next is query representation (section 4.2), where the interpreted query is constructed as an intermediate query structure based on our defined intermediate query schema. The construction process involves constructing the interpreted query into a generic query structure, called the semantic query. Then, to generate the query as a structured query language, mapping of contents from semantic query structure to syntax query structure is carried out (section 4.3). Schema matching is employed to find the best syntax query. Lastly, we present several query selection (section 4.4) methods to optimize the usage of generated structured queries.

CHAPTER 5

EVALUATION

In this chapter, we present the evaluation of the Flexible Query Transformation (FQT) framework in structured retrieval environment. To evaluate the framework, the experiments are divided into algorithm evaluations, application evaluation and representation evaluation. Test collections used in the experiments cover structured collections for both data-centric and text-centric types, i.e. web sites (Special Interest Group of Information Retrieval (SIGIR)) and bibliography record (DBLP Computer Science Bibliography (DBLP)), and topics ranging from syntactic to real world data.

This chapter consists of five sections. We first introduce our evaluation goals and infrastructures. Then, we present experiments and discussions carried out on the framework from different aspects, i.e. query interpretation, query transformation and query representation. Lastly, we conclude by summarizing the outcomes and observations from the evaluation.

5.1 Introduction

In this thesis, the proposed FQT framework will be evaluated from various aspects to justify its usefulness and application. The three aspects are:

Algorithm One important evaluation on FQT is its capability of transforming information needs from unstructured query to structured queries. Thus, it is necessary to measure the effectiveness of query interpretation algorithms via the translated queries. The correctness of a translated query can be measured from a few points such as ability to identify the usage of contents and structural hints in a given query,

ability to suggest correct constraining structure (as filter for content keyword), ability to suggest targeting structure of structured query.

Application The second evaluation on FQT is its application in structured retrieval related tasks. In this evaluation, we will look at how good are the transformed structured queries. Then, we proceed to show the advantage of using transformed queries in retrieval task. For this, we measure the ability of FQT in suggesting improved queries to achieve better retrieval results, which can be based on the output of results given an information need.

Representation The last evaluation on FQT is carried out to measure its representation genericness, to support query transformation of multiple structured query types. For this, we show that the representation can be used to capture the information needs for different structured query languages. We also demonstrate how well the representation can be used to support transformation to multiple structured query languages via their knowledge bases.

Experiment Setup To carry out the evaluations in standard way, test collections are developed based on guidelines in information retrieval evaluation (Manning, Raghavan, & Schütze, 2008, Chapter 10), (Govert, Fuhr, Lalmas, & Kazai, 2006) and INEX (Kazai et al., 2003). The test collections used consist of four major components, i.e. a document collection, a set of information needs, a set of relevance assessments and performance metrics.

For the purpose of evaluation, we have developed a prototyped system in PHP 5 with MySQL 5.0 database. Experiments were performed on a 2.0GHz i7 machine with 4 GB RAM running Windows 7. A summary of the two main datasets used in the experiments are SIGIR Web and DBLP (see Table 5.1). The first data set has bigger average ele-

ment size (text centric), higher number of unique structure (heterogeneous), and higher document complexity. Whereas, the second data set has smaller average element size (data centric), higher number of unique structure (homogeneous), and lower document complexity.

Table 5.1: A Summary of Data Sets Statistics

Collection	Total XML	Element		Term		Doc Complexity
		Total	Size (byte)	Content	Structure	
SIGIR Web	133	8282	192	13453	457	62.3
DBLP	112154	1358021	93	541435	25	12.1

Detailed settings of each evaluation will be described in respective sections later.

5.2 Evaluation on Query Interpretation

5.2.1 Motivation

The experiment presented in this section is to check whether FQT is able to interpret correct structural information from information needs. The queries used in this experiment are unstructured queries (similar to Content Only (CO) query in INEX). Using our FQT framework, the unstructured queries will be transformed into structured queries (e.g. CAS query in INEX). Our main concern here is to find out whether FQT is able to infer correct structural information to build a compatible or better structured query that reflects similar information needs. A series of tests is carried out using different aspects of information needs, such as query length, types of needs and complexities of needs.

The experiment starts by comparing output of query transformation based on its query interpretation algorithms. Comparison is made between our query interpretation approach using context-based probabilistic model with baselines query interpretation models as follows.

- non-context approach (Kim et al., 2009), NCTX
- context with structure using IQD (Bao et al., 2010), CTX+S

- smallest lowest common ancestor approach (Petkova et al., 2009; J. Li et al., 2009), SLCA
- frequent subtree approach (Bao et al., 2010), FCA+D.

We tested the models on top of three standards information retrieval term scoring functions, i.e. BM25, Term Frequency Inverse Element Frequency (TFIEF) and Language Model (LM).

Hypothesis 5.1. *The accuracy of translated structured query using a context-based approach (i.e context-based probabilistic model) should be higher than non-context approach, especially in collection with higher structure complexities such as deeply nested contents.*

In addition, we want to see how the context-based probabilistic model fairs on different standard retrieval term weighting functions. Since the test collection consists of elements of different length (which is a standard characteristic for text-centric structured resource), we anticipate that our model works better when incorporating the term scoring functions (i.e. BM25, LM) that come with normalization compare to the one without normalization (i.e. TFIEF)

The experiment continues to compare the effectiveness query interpretation based on two types of common information needs (i.e. specific and general) used in information seeking processes. This comparison has driven us to make the following hypothesis.

Hypothesis 5.2. *A more detailed, i.e. specific information needs should give better accuracy in its translated structured query form compared to simpler one, i.e. general information needs.*

With regard to these information needs, we also evaluate whether the length of information needs and the usage of structural term in information needs have any effects on

the accuracy of the interpreted query. Therefore, the next hypotheses are as below.

Hypothesis 5.3. *Bigger query size (i.e. more terms) gives better hints in terms of user intention and search context during the processing of query interpretation, thus generate better structured query.*

Hypothesis 5.4. *Usage of structural keywords in information needs should also give better hints in terms of user intention and search constraints, thus generate better structured query.*

5.2.2 Test Collections

In this evaluation, the test collections consists of a text-centric data collections, a set of information needs in unstructured queries form, a set of relevance assessment consisting information needs in structured queries form, and performance metrics for the accuracy of translated queries.

5.2.2 (a) Data Collection

Evaluations were performed using structured document under domains of conference. This collection itself has characteristics such as different complexities of the document structures, diversities of its structure types and size of its elements/contents, therefore providing a diverse experimental setting for assessing the proposed query transformation framework.

SIGIR Web Collection The SIGIR 2008-2010 Web Sites Collection (referred as SIGIR Web thereafter) consists of the structured version of the three years conference site web pages. This is a collection has been developed since 2008 for the xml version of SIGIR conference site's web pages. It has a total of 133 web pages in xml. This collection is text centric as it is developed from text contents, and not generated from database. It has a

complex XML structure and each article contains conference contents of different length. On average an article contains 1234 XML nodes.

As this is a text centric collection, the semantic markups used in the collection are not confined to predefined schema, but more for the purpose of annotation of contents of the articles. They are meant for enabling meaningful contents over the web, rather than data exchange. There are various types of meaningful structures or markups in this collection, e.g. ontological based markups (article, workshop, tutorial, author etc.), logical based markups (abstract, introduction, body, paragraph etc.), topic based markups (xml retrieval, conference venue, accommodation etc.).

5.2.2 (b) Topic Set

The topics used in the evaluation are prepared in two manners, a synthetic set and a real user set. The synthetic set of topic is used to evaluate the efficiency of our algorithm such that various features of the algorithm can be justified. Nevertheless, for fairness of the evaluation, we have also included some real user topic set to demonstrate the applicability of the algorithm on real information needs.

For synthetic topic set, the topics are created by the author such that it can be used to test the features of query transformation. The topics cover variations in terms of information needs, like different query lengths, specifc needs, general needs, and so forth. The topics also cover functional tests with different information needs patterns.

For real user topic set, the topics are created by users who are familar with web search activity. The users were given a task to suggest topics based on a set of structured documents of the chosen collection. Although the collection has been fixed, the users were encouraged to create topics that reflects possible queries that they would use during normal search routine. At the same time, users were also asked to suggest possible results entry points that they would like see as answers for the created topics. The results would

be used for relevance assessment.

Type of Information Needs For both preparations of topics, the topics are further classified by its nature of specific or general.

- Specific: A topic that requests for a particular or a list of known objects. For these topics, relevant answers can be single or multiple elements, normally in the form of objects or entities like tel, movie, url, add etc. In these topics, users know what they are looking for and what answers they are expecting.

- General: A topic that requests for information which is non specific and general, covering a broader type of information. This topic normally results in more than one answer elements, whereby the returned element types can be of multiple types. E.g. given a topic asking for information about query representation in the domain of conference, the answers could be a workshop, a paper, a keynote, an abstract etc. Most of the time, for this topic type, users will learn about the topic by browsing and going through the information returned.

We show some examples of the topics in table 5.2 .

Table 5.2: Some topics for SIGIR Sites collection

Topic [**S**pecific/**G**eneral]	Description
room rate email river view [S]	I am looking for the room rate and email of River View Hotel.
text processing summary presenter [S]	I want to find out who is the presenter and what is the summary of text processing tutorial.
baeza-yates tutorial [S]	I am looking for tutorial presented by Baeza-Yates.
33rd Annual ACM SIGIR Conference sponsors [S]	Who are the sponsors for 33rd Annual ACM SIGIR Conference
probabilistic models [G]	I am looking for information about probabilistic models.
google industry track [G]	I am want to find out about Google's participate in industry track.

Topic Characteristics In addition to topic types, we also ensure that the evaluation topics have the common topic size, with an average of 2.29 keywords. According to Spink, Wolfram, Jansen, & Saracevic, 2001 and Teevan, Adar, Jones, & Potts, 2006, the average query length by web search users is 2.4 words and 2.7 words respectively. Further, as we also classify the topics into whether they contain implicit structural hints or not, in order to find out the benefit of such keywords in query interpretation.

Table 5.3: Topic Statistics (Query Interpretation Assessment)

Collection	Total	Size(T)		Keyword Type		Complexity	
		average	min,max	$T_C + T_S$	T_C	Specific	General
SIGIR Web	28	2.29	1,4	67.9%	32.1%	74.1%	25.9%

5.2.2 (c) Relevance Assessment

Once the topics for evaluation are created, it is also necessary to have a set of assessment to judge the outcome of experiment carried out on these topics. At the algorithm level, we measure topics based on the generated structured query. For this, user is required to provide the golden standard, i.e. an equivalent structured query, for the topic he creates. To make it easier for user, we let the users suggest the structural information required using an interface, rather than asking them to write in the form of structured queries syntax. Web page interfaces (corresponding to their structures resources) are used to let user suggest possibile focus points the correct information are located, i.e. Best Entry Point (BEP) that qualifies as the answer to his topic. BEP indicates where in a document that a user should start reading (Piwowarski et al., 2008; Reid, Lalmas, Finesilver, & Hertzum, 2006). For example, for topic *"text processing summary presenter"* in table 5.2, the evaluator has selected the entry points that resolve to these elements, i.e. $/article[1]/sigir2009[1]/full\_day\_tutorials[1]/summary[1]$, $/article[1]/sigir2009[1]/full\_day\_tutorials[1]/presenter[1]$ and $/article[1]/sigir2009[1]/full\_day\_tutorials[1]$. Once the relevant entry points are known,

we can obtain possible structures to generate structured queries for assessment. We will measure the structures accuracy of a query in terms of its entry concepts and term concepts.

Entry Concept As BEP refers to entry point of a particular physical element; at query level, it is more appropriate to generalize the entry point to the structure of an element, rather than referring to a specific element. We name this entry point as entry concept. Revisit the same topic, evaluator has specified that concepts "summary", "presenter", "full_day_tutorials" are all appropriate as entry point for the topic. There can be more than one concepts for each topic. This is because elements in XML are nested and varied in sizes, thus it is common to encounter situations where the concepts of both parent and child elements are relevant, but to a different extent.

The accuracy of an entry concept is measured based on the concept coverage. Concept coverage evaluates whether the entry concept is structurally correct or otherwise. Here we adopt a similar scale used for measuring component coverage in standard structured retrieval evaluation (see Manning et al., 2008). The coverage can be classified into four types, to indicate different weights for different level of concepts.

- Exact Coverage (cov_{exact}). This concept contains exactly what the topic is seeking.
- Too Broad (cov_{broad}). This concept contains what the topic is seeking, however it also contains other information.
- Too Small (cov_{small}). This concept contains what the topic is seeking, either partially, or not meaningful. E.g., an entry concept like $/presenter/last\_name$ for topic *"text processing summary presenter"* would be too small.
- No Coverage (cov_{no}). This concept does not contain what the topic is seeking.

Constraint Concept Besides entry concept, it is also necessary to measure the correctness of the constraint concept of a content term. This concept filters a term to a specific structure type such that other irrelevant structures can be omitted in the retrieval. Hence, if a user refines a term "andrew trotman" to the concept "author", other structures will not be considered during retrieval. The accuracy of a constraint concept can be classified into three categories as follows.

- Not Relevant (rel_{not}) This concept is not able to reflect the meaning of the content term.
- Somehow Relevant ($rel_{somehow}$) This concept can somehow reflect the meaning of the content term.
- Relevant (rel_{exact}) This concept is able to reflect the meaning of the content term.

It is necessary to include *srel* in the standard to handle a less rigid refinement of concepts used for a term. For example, for a topic where "title" is a relevant concept for a term "linguistic processing", a broader or less strict meaning like "paper", or "list of accepted papers" are also by some means relevant, and can be accepted as well.

5.2.2 (d) Performance Metrics

To assess the structured query generated by our query transformation framework, we measure the accuracy of its target concept and constraint concept using the following performance metrics.

Entry Concept Accuracy For assessing target concept, we compare the concept of the structured query with the entry concept specified by users. Each entry concept is scored

as follows.

$$C_{COV}(ec) = \begin{cases} 0 & \text{if } ec = cov_{no} \\ 0.5 & \text{if } ec = cov_{small} \\ 0.5 & \text{if } ec = cov_{broad} \\ 1 & \text{if } ec = cov_{exact} \end{cases}$$

To summarize the performance, a single-figure measure is used by taking the average of entry concepts for many topics. Given a topic, $q_i \in Q$, ec_{ij} is the set of entry concepts obtained from topic i, then the average over Q is

$$C_{COV_{AVG}}(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{n}\sum_{j=1}^{n} C_{COV}(ec_{ij}), \text{where } n \text{ is total } ec \text{ per } i$$

Constraint Concept Accuracy The accuracy of constraint concept is measured by its relevancy to the term in the context of its topic. Each term's concept is scored as follows.

$$C_{REL}(c) = \begin{cases} 0 & \text{if } c = rel_{not} \\ 0.5 & \text{if } c = rel_{somehow} \\ 1 & \text{if } c = rel_{exact} \end{cases}$$

Given a topic, $q_i \in Q$, t_{ij} is the set of content terms from topic i. c_{ij} is the first concept selected for content term t_{ij}. The average over Q is

$$C_{REL_{AVG}}(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{n}\sum_{j=1}^{n} C_{REL}(c_{ij}), \text{where } n \text{ is total term per } i$$

Note that c_{ij} is the first ranked concept for t_{ij}.

In order to analyse the ranked concepts, we employ $C_{REL}(c_{top_k})$ instead of $C_{REL}(c)$. The scoring of top_k concepts is as follows.

$C_{REL}(c_{top_k}) = 0$

$i = 1$

for $c_{rank_i} \leq c_{rank_k}$ **do**

$$C_{REL}(c_{rank_i}) = \begin{cases} 0 & \text{if } c_{rank_i} = rel_{not} \\ 0.5 & \text{if } c_{rank_i} = rel_{somehow} \\ 1 & \text{if } c_{rank_i} = rel_{exact} \end{cases}$$

if $C_{REL}(c_{rank_i}) > C_{REL}(c_{top_k})$ **then**

$C_{REL}(c_{top_k}) = C_{REL}(c_{rank_i})$

$i++$

end if

end for

This gives us

$$C_{REL_{AVG}}(Q,k) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{n}\sum_{j=1}^{n} C_{REL}(c_{top_k ij})$$

5.2.3 Effectiveness of Query Interpretation

In this section, we measure the effectiveness of the query interpretation in finding the target concept and constraint concept. Our context-based approach (referred as CTX thereafter) is compared with the non-context-based (referred as NCTX) and context-based with node specification (CTX+S) baselines.

5.2.3 (a) Interpreting Constraint Concepts

We show some cases of topics tested in Table 5.4. The table shows comparison of concepts found using CTX approach, and two baselines, NCTX, and CTX+S. These concepts are benchmark against the correct concepts suggested by user. From the experiment, we find that CTX is able to infer a better constraint concept for content term of a topic when a term has ambiguous concepts (e.g. T_2 and T_4). In such cases, our algorithm is able to suggest a relevant concept based on the context of the topic. Otherwise, for non ambiguous term (e.g. T_1), the outcome of constraint concepts would be similar to the baselines.

Table 5.4: Some Cases of Constraint Concepts Interpretation for Content Term in Query

	Topic	Content Term	User	CTX	NCTX	CTX+S
T_1	grand copthorne waterfront hotel address	grand copthorne waterfront hotel	hotel, name	name	location, name	-
T_2	bruce croft committee	bruce croft	senior_pc_ committee	senior_pc_ committee	bio, responder	senior_pc_ committee
T_3	bruce croft industry track	bruce croft	responder, bio	responder	bio, responder	-
T_4	presentation google	google	company	company	affliation	-

If we looked at the term, "grand copthorne waterfront hotel" in T_1, it is only related to one concept type in this collection, which is a "name" of a "hotel". Hence, for this kind of term, it will always have the same constraint concept since there is no ambiguous in its term usage. In fact, in this case, a filtering concept can even be omitted in the construction of structured query since it does narrow down the scope of term; whereas, for the term "bruce croft", it has different concepts describing its role in different parts of collection. In this case, CTX is able to utilize its local context analysis to scope down the constraint concepts to those relevant to a given topic. Therefore, the term is filtered with a concept, "senior_pc_committee" when we issue a topic that looks for information about committee (T_2), while it is filtered with a concept, "responder" when we issue a topic that looks for information about industry (T_3).

The cases show that NCTX is less accurate when a term has ambiguous concepts as it does not capture different concept usages within the same collection. The selected concept is based on the highest rank, such as for a term "google" for topic T_4, it still ranks concept, ""affiliation" higher rather than "company", because the former is a more frequent concept.

Let us revisit topic T_2, the result shows that CTX+S is able to infer senior_pc_committee as the constraint for "bruce croft". This approach is able to pick this concept as it includes factor of structure usage in query in its query inferring process. However, this model re-

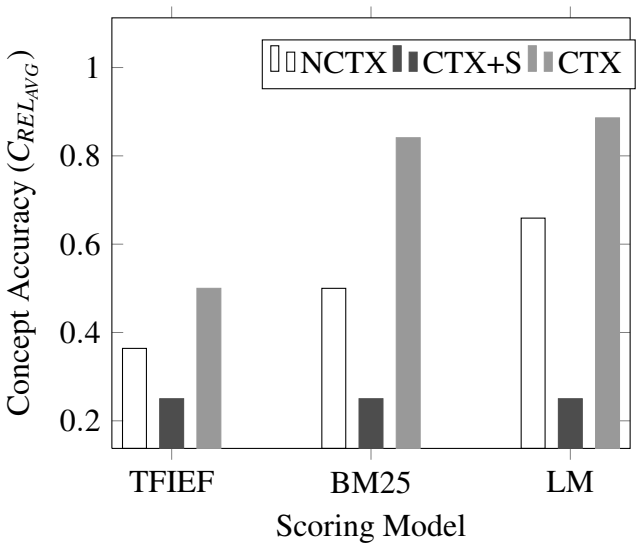


Figure 5.1: The Effect of CTX, CTX+S and CTX on Constraint Concept Selection Based on Top-1 Concept

quires the structure to be specified in query.

Table 5.5: Constraint Concept Accuracy ($C_{REL_{AVG}}$) Based on Top K Concepts for SIGIR Collection

Measure	$NCTX_{TFIEF}$	$CTX+S_{TFIEF}$	CTX_{TFIEF}
Top-1	0.364	0.250	0.500
Top-2	0.636	-	0.591
Top-3	0.727	-	0.841

Measure	$NCTX_{BM25}$	$CTX+S_{BM25}$	CTX_{BM25}
Top-1	0.500	0.250	0.841
Top-2	0.727	-	0.955
Top-3	0.773	-	0.955

Measure	$NCTX_{LM}$	$CTX+S_{LM}$	CTX_{LM}
Top-1	0.659	0.250	0.886
Top-2	0.682	-	0.955
Top-3	0.864	-	0.955

Stability of Concept Relevance To measure the performance of CTX in selecting constraint concepts, the average concept accuracy is taken based on a set of topics. Table 5.5 shows the results comparing CTX approach over NCTX. For demonstrating the stability of our approach on different IR scoring models, we carry out the evaluation of three most used scoring models in the structured retrieval literature, i.e. TFIEF, Okapi BM25

and LM. Our result shows that CTX is able to surpass its NCTX baselines in the overall accuracy as in Figure 5.1 and Table 5.5. From this result, we have few main observations.

- CTX shows its stability on various scoring models as in Figure 5.1. If we look at top-1 concept, when stronger scoring models are used like Okapi BM25 and LM, CTX improves in its overall accuracy, from 0.500 (for CTX_{TFIEF}) to 0.841 (for CTX_{BM25}) and 0.886 (for CTX_{LM}). This is due to element size normalization factor used in the latter retrieval models that emphasizes on direct term and concept relation such as <name>grand copthorne waterfront</name>, rather than indirect one like <description>..... grand copthorne waterfront is located .....</description>.

- In addition to top-1 concept, we are also interested to find out whether constraint concepts at rank 2 and 3 are relevant as shown in Table 5.5, as it could be helpful to include these concepts in the situation where user interaction is allowed. Along with its baseline, CTX is able to achieve better accuracy when top-2 and top-3 concepts are considered. The concept accuracies for all the scoring models are increased to 0.841 (for CTX_{TFIEF}) and 0.955 (for both CTX_{BM25} and CTX_{LM}) respectively.

- CTX+S has a lower accuracy because most topics do not use structural keywords to constrain the topic to search. Due to the very limited structure candidates (at most one or two used in a topic), we cannot see the differences concept scoring models here.

5.2.3 (b) Interpreting Target Concepts

In the experiment of finding target concepts, we compare our interpretation method with a popular baseline, SLCA and a recent improved method of finding subtree, FCA+D. From the experiment, we find that our query interpretation approach is able to find a better target concept in two situations. First, in a nested situation, for example, for topic

Table 5.6: Some Cases of Target Concepts Interpretation for Query

	Topic	User	CTX	SLCA	FCA+D
T_1	grand copthorne waterfront address	hotel, address	address	hotel	hotel
T_2	grand copthorne waterfront deluxe room rate	rate, room rate	room rate	hotel	hotel
T_3	wei che huang andrew trotman	paper	paper	authors	full papers, paper
T_4	trotman geva kaamps	workshop	organizers, workshop	organizers	workshops, workshop

T_4 in Table 5.6, the search term "trotman" is nested under multiple concepts like "workshops/workshop/organizers/organizer/name". When a seek content is located under such nested structures, the SLCA approach will return the nearest parents for all the terms, which gives us "organizers" for topic T_4. However, this concept is regarded as too small, as it will return less meaningful element. What this topic is seeking is actually type of concepts that can reflect the cooperations between these three persons, such as article, tutorial, workshop etc. Hence, in this case, the preferred concept would be "workshop". Compared to SLCA, CTX extends its targets selection to multiple levels of subtree, which enable us to obtain an addition target candidate, "workshop", which is structurally near to the query terms. For FCA+D, we can see that it tends to select target concepts which are higher in the hierarchy (e.g. "full papers" for T_3, "workshops" for T_4) due to it enforces less preference on concepts which are deeper (nested) in the hierarchy.

Second, when structural keywords are used in query, such as T_1 and T_2, our query interpretation algorithm can identify these keywords as target concepts. Using SLCA approach, a target concept is the root of the SLCA sub tree, whereas, our algorithm is able to handle a target concept that is contained within the sub tree. For example, for topic t_1, the query is looking for address of a hotel. Using SLCA or FCA+D, we obtain a sub tree rooted at "hotel". This sub tree contains both content keyword, "grand copthorne waterfront", and structural keyword, "address". There is no measure to utilize structural

Table 5.7: Target Concept Accuracy for SIGIR Collection

Measure $C_{COV_{AVG}}$	SLCA	FCA+D	CTX
Loose	0.574	0.629	0.759
Exact	0.185	0.296	0.667

*Loose measure includes exact, broad or small targets.

Table 5.8: Query Characteristic on Query Interpretation Performance

Query Characteristic		Target Concept Accuracy, $C_{COV_{AVG}}$	Constraint Concept Accuracy, $C_{REL_{AVG}}$
Info Needs Complexity	General	0.500	0.333
	Specific	0.868	0.894
Structural Hint	Without	0.500	0.500
	With	0.912	0.938
Query Size	1 term	0.400	0.333
	2 terms	0.722	0.875
	> 3 terms	0.923	0.929

keywords given in a query as target concepts. Our query interpretation approach addresses this limitation by introducing an algorithm that can suggest structural keywords used in query as target concepts within a sub tree.

To measure the overall performance of CTX in selecting target concepts, the average concept coverage is taken based on a set of topics. Table 5.7 shows the results comparing CTX approach over SLCA and FCA+D. In this test, we measure how accurate is the best suggested target concept compared to its baselines. Two measures are used to evaluate the accuracy of target concepts when they are assessed under either loose or strict manner. For both measures, we can see that our approach has higher accuracy compare to its baselines.

5.2.3 (c) The Effect of Query Characteristics

Based on the hypothesis made in section 5.2.1, we further explore to see how query characteristics affect the accuracy of an interpreted query. We have made three observations from the result in Table 5.8.

Complexity of information needs Query with specific information needs obtains better

accuracy for both target and constraint concepts compare query with generic information needs. This is because the search intention is clearer in the former, such as "grand copthorne waterfront address" (T_1, Table 5.6) compare to latter, such as "probabilistic models" (Table 5.2). When a *specific* query is given, there are more hints for query interpretation to find its target concept as well as constraint concept. This results in higher accuracy for *specific* query. Whereas when a *generic* query is given, there are often many possible suggested concepts. This increases the error rate as there may be non-relevant ones.

Usage of structural keywords Query that uses both content and structural keywords ($UQCAS$) gives better concept accuracy compare to query that uses content only keywords (UQC) for both target and constraint concepts. The main reason is that when structural keywords are used in query, our query interpretation algorithm will be able to identify these keywords in the query, and used them in a more effective way as either target concept or constraint concept; whereas for query without structural keywords, the selection of target concept or constraint concept is based on the query context, which may results in incorrect concepts selection.

Size of query A longer query gives better description of the query context thus give better concept accuracy compare to a shorter one. However, in this evaluation, we have only tested up to four query terms (each term can have more than one word). We have yet tested query with terms longer than four.

From these observations, we can conclude that FQT works best in its query interpretation in the condition where query i) has specific information needs, ii) using both content and structural keywords, and iii) containing more terms.

5.2.4 Summary of Query Interpretation Evaluation

This evaluation tested the query interpretation algorithm of FQT framework. To prove the effectiveness of our proposed algorithm with respect to the raised research question (Q2 in section 1.4), the experiment was carried out using a collection with higher structural complexities.

Our experimental results on query interpretation algorithm showed that query interpretation that uses query context factor, i.e. CTX and $CTX + S$, can suggest better constraints for keywords (Section 5.2.3 (a)). The main drawback of one of the baseline, $CTX + S$ is that it requires the constraint to be specified in a query, which we overcome it with CTX that can predict a constraint even it is not stated in a query. For target concept prediction, we found out that CTX works best when structures of desired element type are used implicitly as part of the keywords in query. CTX is able to identify these structures and use them as target concepts when formulating structured query (Section 5.2.3 (b)).

We also found that the CTX works better using a more advanced weighting models, i.e. LM and BM25, compared to TFIEF. The former are better in weighting terms in query against concepts, therefore contribute to higher accuracy in query concept selection (Section 5.2.3 (a)). In addition, we have found out that CTX shows better accuracy for topic with specific information needs and structural hints in it. For query size, more terms contribute to higher accuracy. However, this conclusion only applies to a maximum of 4 terms as it is longest query size of our topics (Section 5.2.3 (c)).

From this evaluation, we can conclude that query interpretation using the proposed context-based probabilistic model has the advantage when used with collection is more complex in terms of its document structures. With its contextual factor, it is able to utilize the structures more effectively for query interpretation in the query transformation process.

5.3 Evaluation on Query Transformation

5.3.1 Motivation

In this section, we want to evaluate the usefulness of FQT via transformed queries and the application of the queries in retrieval task. There are two main aspects that we want to find out regarding the transformed queries.

The precision and rankings of transformed queries The specification of information needs in unstructured query can be done in two manners, i.e. with or without structural keywords besides content keywords. And, depending on the information needs, the transformation of unstructured query may results in a set of structured queries rather than a single query. In this test, we are interested to check the accuracy of the translated queries, whether they are correctly transformed or otherwise.

For this purpose, we utilize two types of queries in the experiment. First query type has very specific information needs. In this case, the transformed queries set must be able to reflect the information needs that have been specified in unstructured form. Second query type has non specific or general information needs. For this case, the transformed queries set must be able to suggest every possible query which is relevant the information needs.

Further, we are also interested to find out how good is the framework in returning the first relevant query, which would be a desirable feature in system that requires fully automated transformation. In such system, the best transformed query will be automatically used for its retrieval results. For this test, we carry out the experiments on two different collection types, i.e. homogeneous structures and heterogeneous structures. This shall assist us in judging the type of collection that is suitable for such automated feature.

The effectiveness of transformed queries in retrieval task For application purpose, this experiment tests test the usefulness of the transformed queries in structured retrieval task.

Hypothesis 5.5. *The retrieval performance of structured query should be the same or higher than the performance of its unstructured query.*

To test this hypothesis, we compare the search results of query before transformation and query after transformation. For fairness, both queries are run using the same retrieval engine. Their search results are then compared using the precision and recall performance metrics.

5.3.2 Test Collection and Experimental Setup

For this evaluation, the test collection utilizes two data collections (i.e. heterogeneous and homogeneous), topic set for each collection, relevance assessment (i.e. whether an element is relevant or not relevant) for each topic set and performance metrics (Amer-Yahia & Lalmas, 2006).

5.3.2 (a) Data Collection

SIGIR Web Collection Same as section 5.2.2 (a).

DBLP Bibliography Collection The DBLP Computer Science Bibliography is a collection (referred as *DBLP Record* thereafter) developed by the University of Trier for computer science researchers to track the works or bibliographic details of their colleagues or others papers (Ley, 2009). This collection is data centric collection as it is originated from data in database. Each xml article features a DBLP record. Different from text centric, each article corresponds to an object type like conference paper, proceedings, journal article, web site etc. The contents of an object are created based on structures defined by the collection's DTD. Hence, its document structure is homogeneous type. For

example, a record object describing a conference paper, i.e. inproceedings, has author, title, pages, year, etc. In total, the DBLP data collection contains approximately 1.2 million records. A subset of the collection consisting 112154 xml records is used as our evaluation test collections.

5.3.2 (b) Topic Set

For the evaluation of query performance, the topics are prepared in similar manner as in previous evaluation. The characteristics of topics of both SIGIR Web and DBLP Record are reported in Table 5.9.

Table 5.9: Topic Statistics (Query Performance Assessment)

Collection	Total	Size(T)		Keyword Type		Complexity	
		average	min,max	$T_C + T_S$	T_C	Specific	General
SIGIR Web	23	2.04	1,3	69.6%	30.4%	73.9%	26.1%
DBLP Record	12	2.33	1,4	58.3%	41.7%	58.3%	41.7%

5.3.2 (c) Relevance Assessment

For query performance, we measure the relevance of xml elements retrieved wrt. a topic. To access whether xml elements are relevant or not for a given topic, two relevance dimensions are used (Manning et al., 2008, Chapter 10), i.e. component coverage and topical relevance (also known as exhaustivity and specificity). Component coverage measures whether a retrieved element has the correct coverage of information needs, which is neither too big (or high in the tree) or too small (too low in the tree). The coverage of component can be classified into four types, to indicate different weights for different level of components.

- Exact Coverage (cov_{exact}). This element contains exactly what the topic is seeking.
- Too Broad (cov_{broad}). This element contains what the topic is seeking, however it also contains other information.

- Too Small (cov_{small}). This element contains what the topic is seeking, either partially, or not meaningful.

- No Coverage (cov_{no}). This element does not contain what the topic is seeking.

Topical relevance measures the level of relevancy of an element wrt. what a topic is seeking. The relevancy of an element are classified into three categories as follows.

- Not Relevant (rel_{not}) This element is not relevant to the topic.

- Somehow Relevant ($rel_{somehow}$) This element is marginally relevant to the topic.

- Relevant (rel_{exact}) This element is fairly or highly relevant to the topic.

Combining both factors enable us to assess an element as partially relevant, instead of binary choices of relevant/non relevant. The combinations are quantified as follows using the quantification function, $Q_{REL,COV}(e)$.

$$Q_{REL,COV}(e) = \begin{cases} 0 & \text{if } e = \{cov_{no} \cap rel_{not}\} \\ 0.25 & \text{if } e = \{cov_{broad} \cap rel_{somehow}, cov_{small} \cap rel_{somehow}\} \\ 0.50 & \text{if } e = \{cov_{broad} \cap rel_{exact}, cov_{small} \cap rel_{exact}\} \\ 1.00 & \text{if } e = \{cov_{exact} \cap rel_{exact}\} \end{cases}$$

5.3.2 (d) Performance Metrics

Once we are able to assess whether elements retrieved for a topic is relevant or not, we can decide whether a transformed query is correct or otherwise. We regard a transformed query as correct as long as it contains one or more than one relevant elements that satisfy the information needs of given topic. Follow, we present the metrics used in evaluating the performance of a transformed query.

Precision of Transformation The precision of transformation metric measures the correctness of transformed structured queries. The precision of transformation of an unstructured query is given as, P_{QT}

$$P_{QT} = \frac{|Q_{S_{CORRECT}}|}{|Q_S|},$$

where Q_S is a set of structured queries generated from query transformation process, $Q_{S_{CORRECT}}$ is a set of correct structured queries, and $Q_{S_{CORRECT}} \in Q_S$.

The main drawback in this metric is we are only able to measure the structured queries that are successfully transformed (i.e. precision), but not be able to track the queries that are missed (i.e. recall).

Reciprocal Rank of Structured Query To evaluate the effectiveness of query ranking, a standard Information Retrieval (IR) metric called Reciprocal Rank (RR) (Tran, Wang, Rudolph, & Cimiano, 2009) is used. The metric measures the rank of generated structured queries by looking for the first correct structured query. The range of the value is from 0 to 1.

$$RR = \frac{1}{rank(Q_S)},$$

where Q_S is the first correct structured query.

For a set of queries, Q, Mean Reciprocal Rank (MRR) is used to obtain the average of ranks.

$$MRR(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|} RR(Q_i)$$

Precision, Recall and F-measure The evaluation on query transformation can be furthered measured by the structured retrieval results of generated structured queries. The common metrics for measuring structured retrieval results are precision, recall and F-measure (Baeza-Yates & Ribeiro-Neto, 1999; Amer-Yahia & Lalmas, 2006). In structured retrieval system, a set of elements are returned instead of documents, hence, both

precision and recall are adapted for elements. Precision, P_{SR} is defined by the fraction of the retrieved elements which is relevant. Whereas, recall, R_{SR} is defined by the fraction of the relevant elements that is retrieved.

$$P_{SR} = \frac{|E_{RELEVANT} \cap E_{RETRIEVED}|}{|E_{RETRIEVED}|},$$

where $E_{RETRIEVED}$ is a set of elements retrieved, $E_{RELEVANT} \cap E_{RETRIEVED}$ is a set of relevant elements retrieved.

$$R_{SR} = \frac{|E_{RELEVANT} \cap E_{RETRIEVED}|}{|E_{RELEVANT}|},$$

where $E_{RELEVANT}$ is a set of relevant elements, $E_{RELEVANT} \cap E_{RETRIEVED}$ is a set of relevant elements retrieved.

A weighted average for precision and recall can be computed using F-measure, F_{SR}, which is a single measure that trades of precision and recall scores.

$$F_{SR_{\beta=1}} = \frac{(\beta^2+1)P_{SR}R_{SR}}{\beta^2 P_{SR} + R_{SR}},$$

where $\beta < 1$ emphasizes on precision, while $\beta > 1$ emphasizes on recall, and $\beta = 1$ treats precision and recall equally.

5.3.3 Query Precision and Ranking

One Good Structured Query Before measuring the precision of generated queries and their rankings, we first get an overview of whether FQT is able generate a usable query. For this, we measure whether FQT is able to generate at least one correct query from a given unstructured query. A query is correct if it is able to return at least one relevant result (i.e. element) in retrieval task. As a result, out of all topics, FQT is able to generate

at least one correct query per topic (see Correct Queries in Table 5.10) except for the SIGIR Web topic set. For SIGIR Web specific topic set has a less percentage at 93.3% indicating there are topics where their transformed queries are all incorrect. This case happens when a given topic does not have exact match contents or structures with the data source.

Table 5.10: Transformed Queries Ranking and Precision

Dataset	Topic	Correct Queries (%)	MRR	P	P@5
SIGIR Web	Specific	93.3%	0.747	0.444	0.469
	General	100%	0.654	0.335	0.375
DBLP	Specific	100%	1.00	0.833	0.833
	General	100%	1.00	0.625	0.625

First Correct Rank Now, we proceed to measure the precision of generated queries and their rankings. First, we measure how good is the ranking of queries by checking on the rank of the first correct query. For each topic, each first correct query is scored using the RR metric. In the case where no generated queries are correct, RR is given the value 0. Table 5.10 shows the MRR (average value of RR for all topics) of topic sets for both SIGIR Web and DBLP data sets. We find that MRR for DBLP has perfect scores for both *specific* and *general* topic sets, but not for SIGIR Web, which only achieves 0.747 for *specific* topic set and even lower at 0.654 for *general* topic set. DBLP has such good result mainly because of the type of its xml source, which contains simple elements like article, inproceedings, book etc. These objects have little depth of hierarchical contents, and they do not contain nested (smaller) meaningful elements within them. Hence, there are much less structured query candidates per topic. Moreover, each candidate is already a well-defined element with no granularity problem. In this case, FQT can easily return a correct query in its top-1 position, giving us MRR of 1.000.

However, for SIGIR Web, FQT has returned a broad range of structured queries en-

compassing elements of various granularities. This happens because the xml source of SIGIR Web contains contents which are deeply nested, with irregular structures. Therefore, it is harder to predict which are the best entry points for structured queries. This causes some topics to rank *incorrect* structured queries first. In this test, we regard a generated structured query as *incorrect* even if the query results in either broader or smaller relevant answer. A lower MRR is also observed for *general* topics, which explain that a non-specific topic tends to create more structured queries candidates which may not be relevant, therefore affect the rank of correct queries.

Query Transformation Precision Besides query rank, we are also interested to find out how good is FQT is generating correct structured queries. Again, data set with simpler document structures, i.e. DBLP, has a higher precision compared to data set with complex structures, i.e. SIGIR Web. Our result in Table 5.10 shows that for overall precision, P, SIGIR Web, its precision values are 0.444 (specific) and 0.335 (general) each, which is much more lower than DBLP at 0.833 (specific) and 0.625 (general) each. Similar scores also observed for $P@5$. The low values of P for SIGIR Web topics are due to higher number of structured queries that have been suggested to users. There are two main reasons behind large number of structured queries, first, when topics have general information needs, they are opened to more possible element types. Second, when the topics seek is carried out on data set with complex document structures, which causes contents to be related to more structure types and granularities in this data set. When this happens, a smaller subset of queries can be proposed instead the entire list. One common technique is to select top-k queries. To ensure that this technique can be adopted in query list reduction, a better or similar result should be achieved (see $P@5$ Table 5.10), otherwise, we should still adopt the entire result list.

5.3.4 Structured Retrieval Performance

To verify structured queries can produce similar or better results compare to the original query, we compare the retrieval results of both queries. The test is conducted using the NEXI retrieval system (Trotman, 2009), which accepted both CO query and CAS query. This is inline with our experimental needs that compare topic (unstructured query) with its transformed structured queries. The former is submitted to NEXI system as CO query whereas the latter is submitted to the system as CAS query. The relevancy of returned elements are assessed based on the $Q_{REL,COV}$ metric.

Table 5.11: Top-1 Query Retrieval Performance

Dataset	Topic	Precision		Recall		F-Measure	
		SQ	UQ	SQ	UQ	SQ	UQ
SIGIR Web	Specific	0.687	0.056	0.629	0.611	0.570	0.099
	General	0.281	0.196	0.087	0.811	0.087	0.312
DBLP	Specific	1.000	0.042	1.000	0.500	1.000	0.076
	General	1.000	0.361	0.972	0.833	0.985	0.5

In this test, we want to see how good is the retrieval result of top-1 translated structured query compared to its original query. The F-measure results (see Table 5.11) show that the top-1 structured query (*SQ*) is better at retrieving relevant elements compared to its baseline (*UQ*) for all the topics of DBLP dataset. The same goes for *specific* topic set of SIGIR Web. However, for its *general* topic set, the top-1 suggested structured query has lower F score. From the results, we made the following observations.

- A lower F score of SIGIR Web *general* topics is due to its low recall score (0.087 for *SQ* compared to 0.811 for *UQ-C*). The main reason of a low recall is because a topic which is *general* has non-specific needs. Thus, each topic will have more than one correct structured queries. By only taking the first structured query, we can only cover partial of the correct results. For this topic type, *UQ* has better recall value since it includes every possible element candidates in its results, however, it also suffers from low precision because of the number of retrieved elements.

- For both data sets, the retrieval results for *specific* topics are better than *general* topics. This is because a *specific* topic tends to include structural keywords to indicate what it looks for specifically, therefore will filter off many non-relevant elements. This leads to better precision scores.

- Retrieval results on homogeneous data set (DBLP) are better than heterogeneous (SIGIR Web). The result also shows very high accuracy in precision and recall for DBLP, that suggests that the top-1 structured query by FQT on homogeneous data set is suitable to be adopted for application feature like “I am Feeling Lucky" (Wikipedia, 2012).

Before we end this section, we also want to see how good is the retrieval result of top-1 translated structured query compared to other query transformation work. For this, we also compare our framework with a related work that performs full query transformation to construct a NEXI structure query, AQRT (Automatic Query Refinement and Transformation by Petkova et al., 2009) on SIGIR Web. A subset of 10 queries where used in this test. The results in Table 5.12 shows a higher F-measure score for FQT compare to its baseline AQRT. It is consistent with our earlier result from query interpretation as AQRT is based on SLCA in its expansion and aggregation operators. Also, there is limitation of AQRT's ordering operator when it counters a query that looks for multiple types of elements. As for FQT, it could identify these keywords and propose them as structures of structured query, which in turn gives advantage to the performance of FQT.

Table 5.12: Structured Retrieval Performance Comparison

Framework	Precision	Recall	F-Measure
FQT	0.656	0.688	0.633
AQRT	0.406	0.438	0.383

5.3.5 Summary of Query Transformation Evaluation

This evaluation tested application of the translated query by measuring the retrieval results based on the top-1 translated structured query. We observed higher precision of the retrieval results for *specific* query type, which suggest that when writing query is written in a specific manner, it can lead to better focused results. As for data collections, a high precision and recall of results (≈ 1.000) were also recorded for DBLP. This indicates that our framework can actually transform unstructured to structured query almost perfectly for homogeneous collection. Our results for heterogeneous collection has a lower precision, which suggest that some correct structured queries are actually ranked lower (not located at the top-1). In particular, we found out that for *general* topics, user intervention may be involved to increase the precision as there are too many suggested structured queries (Section 5.3.4). Hence, this experiment suggests that FQT framework would probably works best when coupled with interactive information retrieval features for structures selection when used with heterogeneous collection.

5.4 Evaluation on Query Representation

The goal of the experiment in this section is to measure whether the proposed intermediate query representation can represent the information needs for the transformation of query from unstructured to structured form in a real world scenario. Evaluation carried out for intermediate query representation focuses on two aspects as follow.

- The ability of its semantic query structure (see Section 3.4.2 (a)) in capturing information needs specified by user for structured retrieval.

- The coverage of its syntax query structure (see Section 3.4.2 (b)) knowledge base in capturing mappings for transforming the information needs from structure form to query language form.

5.4.1 Data Sets

For this evaluation, we use information needs collections that have been created for the purpose of retrieval for structured resources. These collections have been chosen based on three criteria, i.e. creation fairness, domain diversities and query language types. For the fairness of information needs creation, we include topic collections which have been prepared as test suite (i.e. topics created under the guidelines of structured retrieval forum (INEX)) and query logs (i.e. topics created by system's user). To ensure that the representation can be applied to wider coverage of information needs complexities, the topic sets covers query languages of different types. In addition to these, the topics also include multiple domains to ensure the fairness of selected collections.

Table 5.13: Topic Collections

Name	Total	Query Language	Source Type	Domain
INEX IMDB	70	NEXI	Test Suite	Movie
INEX Wiki	100	NEXI	Test Suite	Articles
Geobase	90	XQuery	Query Log	Geography
SIGIR Sites	22	IQ	Test Suite	Web sites
DBLP	20	IQ	Test Suite	Bibliography

5.4.2 Performance Metrics

Query Representation Expressiveness The expressiveness of semantic query structure can be measured by checking whether it can represent the contents of queries used for structured retrieval. The performance metric used is *expressiveness*, which is the fraction of the structured queries in a topic set that can be expressed using the proposed semantic query structure. To get a rough estimation of the expressiveness, we run the evaluation based on topic sets that have been prepared as test suites for the purpose of evaluation of structured retrieval. We believe that these topic sets which have been created by a group of assessors can fairly represent the general information needs/queries. In addition, we have also included a topic collection which is a subset of real query logs.

Each query, q, in a topic collection, Q, is scored as follows.

$$REP(q) = \begin{cases} 1 & \text{if } q \text{ can be represented as } q_{sem} \\ 0 & \text{o}therwise \end{cases}$$

, where q_{sem} is the semantic query structure of q.

The expressiveness can be summarized per topic collection by taking the average of query score for all the topics in the collection. Let us denote Q as the topics of a collection. The average over Q is

$$REP(Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} REP(q_i)$$

Query Representation Effectiveness Besides measuring the expressiveness of the query representation, we also measure its effectiveness in handling the transformation to multiple query languages. The effectiveness of syntax query structure is measured based on the success rate of query mapping from a query structure represented in semantic query form to a query string in a structured language form. The performance metric used is *success rate*, which is the fraction of test query that can be translated into query language, X, successfully, given a knowledge base of syntax query structure for language, X. Given a knowledge base of query language, X, each query, q, in a topic collection, Q, is scored as follows.

$$CVRT(q) = \begin{cases} 1 & \text{if } q_{sem} \text{ can be converted to } q_X \\ 0 & \text{o}therwise \end{cases}$$

, where q_{sem} is the semantic query structure of q

and q_X is the structured query of q in language, X.

The effectiveness can be summarized per topic collection by taking the average of

query score for all the topics in the collection. Let us denote Q as the topics of a collection. The average over Q is

$$CVRT(Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} CVRT(q_i),$$

5.4.3 Expressiveness of Semantic Query Structure

We evaluated the expressiveness of semantic query structure using the topic sets mentioned in section 5.4.1. The ability of the semantic query structure to capture information needs for various type of domain has been shown from the diversities of the topics ranging from movies, web sites, bibliographies and geographical information. Although the collections are relatively small, they reflect a reasonable test bed as topics are originated from either test suites or subset of query logs. Both INEX IMDB and INEX Wiki are topics prepared by participants of the INEX forum. Whereas, the SIGIR Sites and DBLP topics are prepared in a similar manners by users familiar with both collections. For the Geobase, it is a query logs created by real users of a publicly available web interface (Jayapandian & Jagadish, 2008).

Table 5.14: Query Representation Expressiveness

Name	Total	REP (%)
INEX IMDB	70	100%
INEX Wiki	100	100%
Geobase	90	60%
SIGIR Sites	22	100%
DBLP	20	100%

A summary of the performance of query representation expressiveness is shown in Table 5.14. For SIGIR Sites and DBLP topics, we are able to express all the interpreted queries with our semantic query structure. A topic specifying specific need, T_{21}*: room rate email river view*, which has been represented as semantic query structure is shown in Figure 5.2.

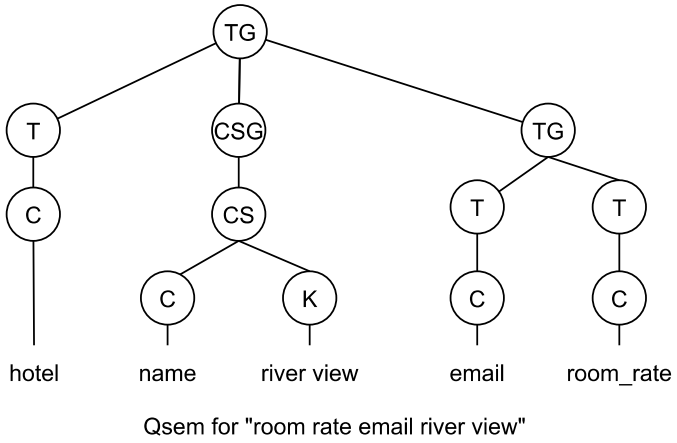


Figure 5.2: Representing Interpreted Query from SIGIR Sites Topics

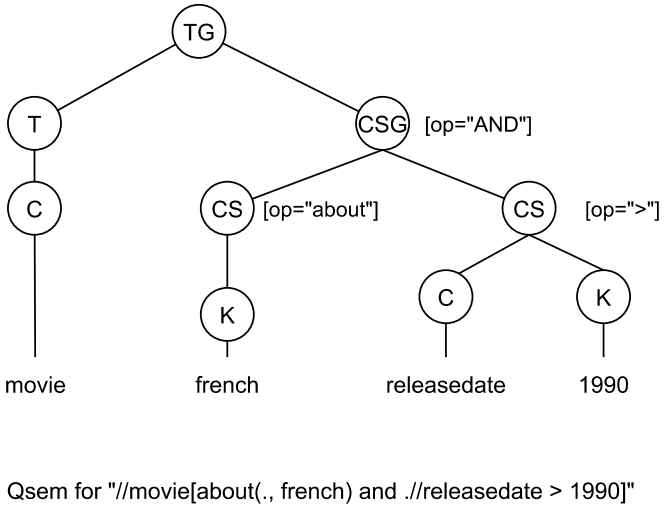


Figure 5.3: Representing Query Containing Constraint Operator.

For further evaluation on a larger topics set, we proceed to analyze both INEX IMDB and INEX Wiki topics. For both collections, we are able to express all the structured topics with our semantic query structure as well. A minor issue encountered for INEX IMDB is that one of its topic, $INEX_{2011111}$*: //movie[about(., french) and .//releasedate > 1990]*, requires the specification of operator, "greater than", i.e. "releasedate > 1990" in its information need. Although the semantic query structure is able to represent this information needs by capturing the operator as attribute of its constraint (see Figure 5.3), however, at the current stage, our framework do not carry out interpretation of operators keywords in unstructured query.

Lastly, we also tested the expressiveness of XQuery, which is a query language popu-

larly used in querying XML database. Among all the topics, 60% can be fully represented as semantic query structure. The percentage is lower for this topic collection as XQuery is a declarative language, where functional commands such as *let, count, max* etc. are used in topics when specifying information needs. For these topic cases, our semantic query structure can only represent partial information needs of the topics. Follow, we show some cases of topics that we are unable to fully represent in Figure 5.4.

Case When Topic Only Contain Concept/Structure Keywords In this case, a topic only specifies the type of structures to look for, without keywords that look up the contents of XML elements. An example of this case is seen in topic, T_{27}: let $e := document("geobase.xml")
//highest_point/elevation return <result> {max($e)} </result>. For this case, although we are to represent *highest_point* and *elevation* as target concepts, it is not a complete semantic query structure as it does not contain any constraint.

Case When Topic Contain Functional Commands In this case, a topic includes usage of functional commands. An example of this case is seen in topic, T_{51}: for $s in document("geobase.xml")//state where $s = "Rhode Island" return <result> {count($s/capital)} </result>. For this topic, we are able to represent the structure of "capital", but not the function to sum up the total elements of "capital". This topic illustrates the major limitation of our query representation, that makes it less flexible when used with XML DB systems.

Lastly, we are interested to see whether the semantic query structure is able to handle information needs of different complexities. Since the query complexities used for structured retrieval may be influenced by the type structured collections, i.e. whether they have well-defined schema or loosely defined structures, we include topic sets of two different INEX tracks, i.e. data centric track (INEX IMDB) and adhoc track (INEX Wiki) for this

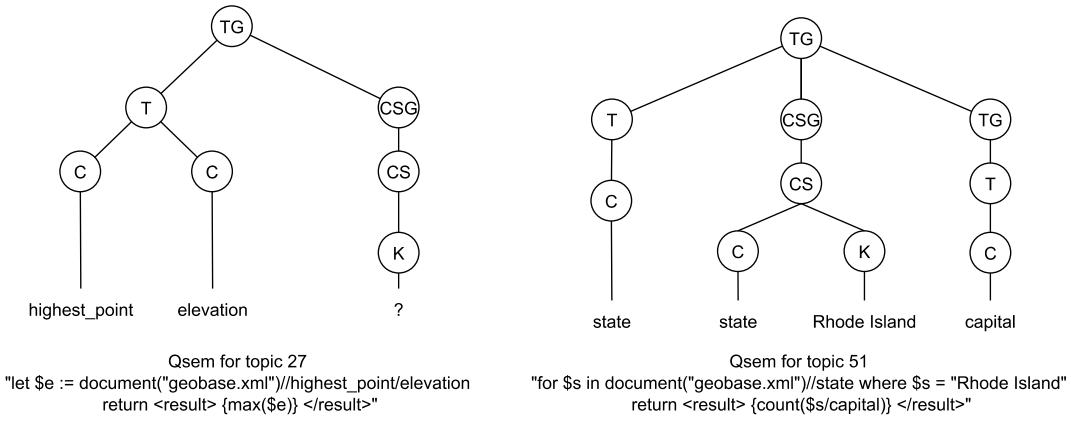


Figure 5.4: Limitations of Q_{sem} Representation for XQuery Cases

test. The distribution of complexities of the topics that are successfully expressed using semantic query structure is shown in Figure 5.5. For the former, its topics are richer in structures since users are more familiar with the structures/concepts used in its collection such as actor, plot, director, genre etc. As such, users are able to utilize them during query formulation. For the latter, the topics are simpler due to the types of structures/concepts which are too diversified, which prohibits users from using them in query formulation. Hence, as in Figure 5.5, we can see that the query complexities for INEX IMDB are actually slightly higher than INEX Wiki. See Appendix A for structured query complexity scoring.

In addition, the diversities of query complexities shown in Figure 5.5 also assure the fairness topics used in the experiment conducted in Table 5.14.

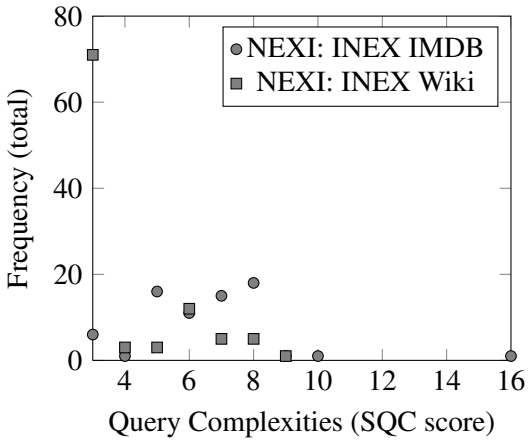


Figure 5.5: Expressiveness of Semantic Query Structure for Various Query Complexities

5.4.4 Coverage of Syntax Query Structure Knowledge Base

To convert the contents of query in semantic query structure form to query language string, we need to find its corresponding syntax query structure. For this, a knowledge base consisting syntax query structures is required. This section evaluates the stability of this approach and whether the knowledge base of syntax query structure is able to support the conversion to query language in query transformation framework.

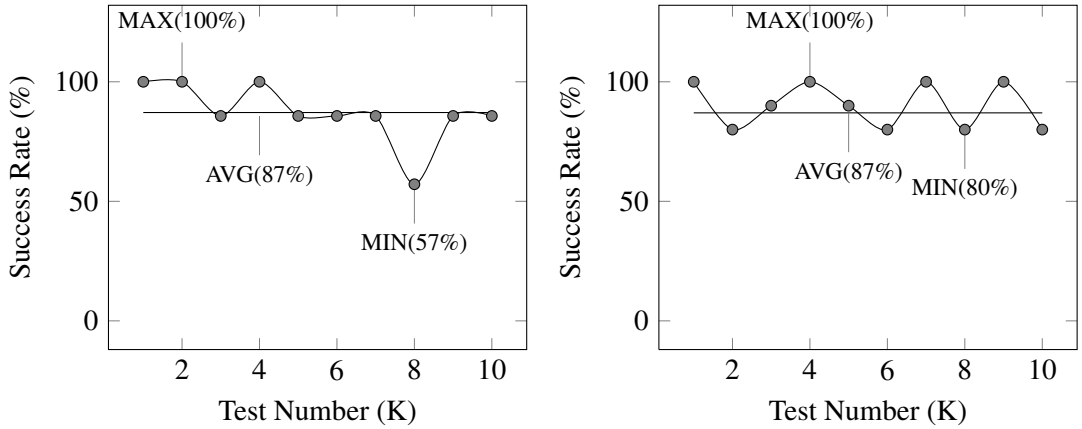
Stability of Knowledge Base For estimating the stability of knowledge base in supporting query conversion, a cross-validation technique (Kohavi, 1995) is used. This technique assesses how the results of performance will generalize to other data sets. The topic collection is partitioned into ten subsets. Nine sets are used as examples in knowledge base, and one set is used as test. This test was repeated 10 times, i.e. 10-fold cross-validation. This test is carried out using two NEXI topic collections. The performance of query conversion is shown in Table 5.15. The average success rate of query conversion is 86% for INEX IMDB and 87% for INEX Wiki collections. For both collections, we can see that the success rates for all the tests are scattered along the average line. An exceptional case of a rate of 57% is found in Figure 5.6(a), which contain higher number of unconverted queries. This is probably due to the size of topic set which is too small, leading to uneven distribution of queries for training and testing. This phenomenon is not seen in Figure 5.6(b) where we use a bigger topic set.

Our evaluation method for this section is adopted from Sumita & Iida, 1991 in their work of example-based machine translation where they achieve an average of 78% success rate in using examples database for translation. It is expected that our average success rate is higher as we are dealing with queries transformation which are definitely simpler with smaller scope compare to natural language translation.

Table 5.15: Query Conversion Success Rate for NEXI Knowledge Base

Name	KB Size (No. of Examples)	Test Size (No. of Query)	AVG CVRT* (%)
INEX IMDB	63	7	86%
INEX Wiki	90	10	87%

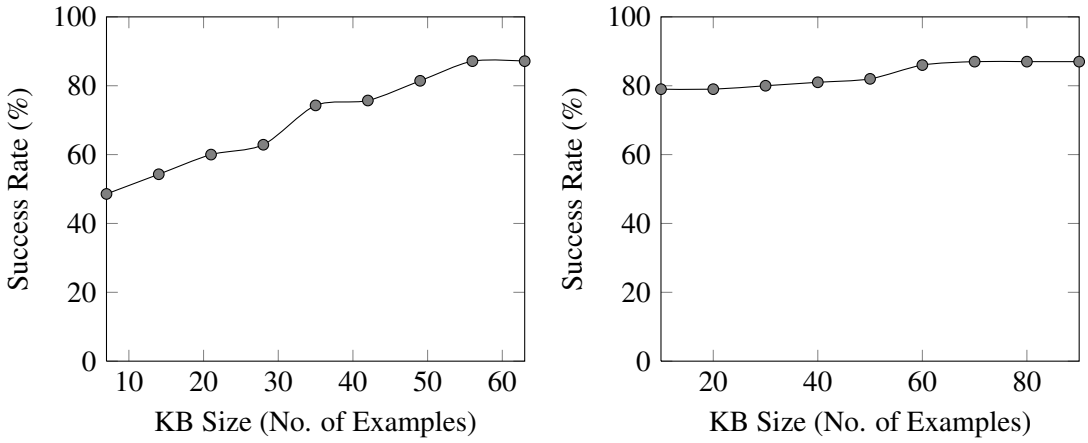
*Average value is taken over 10 fold cross validation success rate.



(a) NEXI Data Centric Topic Set (b) NEXI Text Centric Topic Set

Figure 5.6: The Success Rate of Query Formulation using K-Fold Cross-Validation Technique

Query Transformation Per Examples Figure 5.7(a) and Figure 5.7(b) show the relationship between the success rate of query formulation of an interpreted query and the number of examples. This graph shows that in general, for topics within the same collection, the more examples we have, the higher success rate of query transformation.



(a) NEXI Data Centric Topic Set (b) NEXI Text Centric Topic Set

Figure 5.7: The Effect of KB Size over Success Rate of Query Transformation

Cross Collections Test To ensure that a trained knowledge base can be used for cross collections queries, we carry out test on SIGIR topic set based on the knowledge base trained with INEX queries. For each topic that has been interpreted and represented in semantic query structure form, 78% of the topics were successfully transformed into NEXI language. The rest are partially transformed due to no exact counterpart of the query structure. In this case, the most similar query structure match is retrieved instead. A partially matched case can be seen in the semantic query structure for query, $T10$, "text processing summary presenter". The nearest syntax query structure, $NEXI35$, from NEXI knowledge base with the highest similarity score, 0.90, is proposed for partial transformation. In this test, although there are limitations in converting all the queries successfully, this is because of lacking of variation of query examples in knowledge base, which can easily be solved by adding more examples.

5.4.5 Summary of Query Representation Evaluation

This final evaluation demostrates that our proposed query transformation framework can be used to perform query transformation to create structured queries (e.g. NEXI, XQuery) by using a knowledge base of examples (e.g. structured queries of NEXI/ XQuery). We have also shown that the design of proposed intermediate query representation can handle more than one type of structured query languages. The experimental results verified that the success rate of query conversion improves with the number of examples in knowledge base. Hence, to further improve the success rate of query conversion, the knowledge base can be easily upgraded by putting appropriate examples.

5.5 Summary

This chapter presents the evaluation objective, methodology and results of our proposed FQT framework. The experiments focused on three aspects of the framework, which are the algorithm for query interpretation, the application of structured queries ob-

tained from the transformation process, and the scalability of query representation for extending transformation to multiple structured query languages. The experiments of query interpretation and transformation are performed on SIGIR Web and DBLP collections, whereas the experiments of query representation are performed using the topic collections from INEX and Geobase. For each evaluation, we tested our hypothesis empirically and draw conclusions based on the results we obtained from our experiments.

Query Interpretation

1. Improvised target concept selection (CTX > SLCA , CTX > FCA+D).
    - Able to differentiate structural hints used in query as target.
    - Able to use collection knowledge (via ctx distance and density factors) to suggest relevant target concepts.
2. Improved constraint concept selection (CTX > CTX+S, CTX > NCTX)
    - Able to suggest constraints even when it is not given.
    - Able to extend constraints to more choices along the structure path.
    - Able to use context of other keywords in a query to find relevant constraint.
3. Good for query with
    - Specific information needs (Specific:0.868 > General:0.500)
    - Structural hints (WITH:0912 > WITHOUT:0.500)
    - Longer query size (3TAbove: 0.932 > 2T:0.722 > 1T:0.400)

Query Transformation

1. Tested on retrieval task using the translated structured queries using NEXI retrieval system.

2. Overall, structured queries gives better retrieval results compared to unstructured queries.

3. Application suitability

    - Collection: Retrieval results on homogeneous collection are better than heterogeneous collection.
    - Topic: Retrieval results for specific topics are better than general topics.

Query Representation

1. Test collections comprising topics of various query language types, domains and sizes.

2. Two aspects tested, expressiveness (test the query structure) and effectiveness (test the conversion method).

    - Expressiveness: Can fully express information needs specified, except for Xquery data set (60%).
    - Effectiveness: Achieve satisfactory query conversion rate, i.e. 86% (IMDB: 70 examples) and 87% (Wiki: 100 examples) respectively compare to benchmark (78% Sumita & Iida 1991 for language translation).

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

Using unstructured query for structured retrieval offers greater flexibility to users in query writing, while they are still be able to explore the richer type of data with better semantic and structure. More semantically and structure rich data are available via common representation language like XML. These data are featured over in the web for the purpose of enabling meaningful contents. However, in order to effectively use these structures, a query must be able to include structures in its specification. As such, various structured query languages like XQuery, XPath, NEXI, XML fragment etc. have been created. However, due to the difficulties faced by users in writing syntactically and semantically correct queries, efforts to automate the construction of such queries have been carried out.

Automated query transformation is a desirable feature in the field of IR as it could integrate the benefits of structured retrieval to improve features of existing IR. Designing an effective query transformation framework is challenging as it involves problems of various phases from source query analysis, interpreting methods, query constructions into target language, query ranking etc. This thesis mainly resolves the flexibility issue of query transformation by combining probabilistic approach (in which query interpretation is based on collection statistics, and query construction is based on algorithms) with natural language approach (where query interpretation is based on context free parsing, and query construction is based on knowledge base).

In particular, this thesis has made two main contributions in its query transformation

approach, i.e. the design of a framework with better flexibility and a more effective query interpretation and construction.

6.1.1 Flexible Framework

The main contribution is a framework that formally defines query transformation that has better flexibility to be tuned to changes in structured retrieval features, especially its query languages and collections. The main strength of the framework design is that it separates the outcome of query interpretation from a strict query language type. In this thesis, the query construction algorithm creates an interpreted query which is free from a particular query language. The generated query is represented in a format known as *semantic query structure*. From the experiment in Section 5.4.3, we have shown that this structure can capture those information needs that are frequently used for structured retrieval.

The final step to generate the query in structured language form, is carried out by finding the matched query template known as *syntax query structure*. These templates were derived from the previous created queries, e.g. NEXI topics from INEX. We have shown that this approach is possible from the experiment in Section 5.4.4, where up to 87% (average success rate) of queries can be mapped to the correct templates. We have also demonstrated that the example-based method for building query templates knowledge base is feasible since we are only capturing structured query which relatively few templates are required. Moreover, the coverage of knowledge base can always be improved by simply adding more unique query examples into it. Such flexibility ensures that as long as the information needs of a source query is represented in *semantic query structure* form, disregards how it is interpreted, it can be mapped to a structured query language. This leaves room to allow additional methods to be incorporated on the query interpretation part later.

Within the framework, two knowledge capturing methods are proposed, i.e. a context-based probabilistic model is used to capture knowledge from collection for query interpretation, and an example-based query templates generator is used to capture knowledge for query language formulation. In this framework, we work on two different type of collections, in which we have broadly classified them into, i.e. simple collection (e.g each document presenting one *meaningful* element) and complex collection (e.g. each document has nested *meaningful* elements of multiple types). We have included the analysis of a complex collection in this thesis as we are encountering more structured resources with deeper hierarchy and greater element heterogeneity. The latter certainly requires more analysis when we want to suggest a good structure during query interpretation, e.g. finding a good concept for filtering a query term, finding a good element type for a query.

As such, we proposed a probabilistic model that includes context factor when suggesting a concept for a term in a query (Section 3.3.2). Without knowing the context factor, a suggested concept may not be appropriate even though it is the popular one in collection. We also take into consideration of including ancestor concepts as possible relevant concept for a term, due to some parent structures which may not be good enough to filter/constraint a term (Section 3.3.3). For the measurement of concept importance, our model is independent of a fixed basic term weighting models. This enable the selection of a different term weighting model based on the type collection. A simple, data centric collection can use a simpler term weighting model like TFIEF whereas collection that contains different length of texts in its element may opt for one with normalization like BM25 (Section 3.3.4 (a)).

For scalability to extend to another structured query language, we just need to create a template knowledge base for the query language, e.g. we can have an XQuery template knowledge base and a NEXI template knowledge base, and so forth. Each structured query language has its own template knowledge base, which can be flexibly shared and

updated with more examples to improve its mapping.

Lastly, to improve the reusability of our query transformation framework, we also provide formal definitions of the framework such that it can be well understood and implementable by other parties. Although there are a number of works that discusses query transformation, we have yet encounter with a work that formally defines the framework.

6.1.2 Improved Query Transformation

The second contribution of this thesis is the improvement made on various features of query transformation. We present our discussion based on the query transformation process, from the definition of source query to the application of our generated structured queries.

The source query of this framework can be specified with two types of keywords, i.e. content keyword that corresponds to the contents of elements, structural keyword that corresponds to the structure of elements. And, both keywords type can be written in totally unstructured form. The main strength of our query interpretation is it can suggest a structural keywords if it has not been specified in a query, whereas it can utilize it if it is specified. We have also designed the algorithm such that it can differentiate a structure specified as target or constraint (Section 4.1.3) as current works can only detect either one (Bao et al., 2010; Petkova et al., 2009). If a source query contains some linguistic structures, we still treat it as unstructured query. Main keywords will be detected using term identification, while other words like conjunction will be ignored.

Another strength of this framework is that it can fully convert a source query to an structured query. Although some works (Kim et al., 2009; Bao et al., 2010) prefer to bypass this step, by using the interpret query directly to retrieve a set of results, however, we find that this approach restricts the interpreted query to its internal retrieval model. When we construct the query into some standard query languages, e.g. NEXI, XQuery,

we are opened to wider adaption of framework. Optimization are be focused on query part, independent from their retrieval mechanisms. Up to date, we came across two works that carried out full transformation (Petkova et al., 2009; J. Li et al., 2009), to go the extra mile, we improve the query transformation such that it can be used to construct more than one structured query language type.

In order to ensure that the constructed query is useful, we have tested the performance of query interpretation based on suggested query target and query constraint. We showed that for collection with complex structures, our query interpretation using context-based probabilistic model surpasses its baselines (Section 5.2).

Finally, we have performed a real structured retrieval task using the transformed query. The error free query syntax has shown that automated transformation has its advantage compared to human written query. A comparison of the retrieval results between structured query and its original query shown that the former has higher precision especially for topics with specific needs (Section 5.3).

6.2 Limitations

Despites the contributions that we have made, there are some limitations that we have encountered in this thesis. In this section, we will discuss the shortcomings of this thesis as follows.

Domain Specific Query Interpretation The first limitation of this thesis is that its query interpretation works well if the collection is domain specific. The framework may face the problem of query misinterpretation when dealing with collection such as the world wide web. This is due to the number of different structures are too high in such open domain, and interpretation could be a tedious task when reasoning the correct structure. Nevertheless, with this limitation, we still able to exploit available document structures

more freely (compare to schema) within a domain but not as overwhelm as the entire structures subset of uncontrolled domain.

Under Utilization of Structured Query Functionalities The ultimate goal of query transformation is to be able to transform information needs from one form to another form of queries effectively. However, due to different structures complexitiy of two queries type, i.e. between the unstructured form of source query and the structured form of target query, the latter is often underutilized. This is because structured query languages are often more complex than keywords query. This happens especially for structured query like XQuery. For this query type, we can see that we are not able to exploit functions supported by the query language like count, max etc.

The Number of Structured Resources A good structured retrieval method is highly depending on the number and variation of the resources that we can test on. Although structured resources are become more available nowadays, the numbers are still far behind compared to unstructured documents. Hence, the application proposed query transformation are still limited to relatively little collections due the limitation of structured resources.

6.3 Future Works

We have made some significant progress into developing a flexible query transformation framework. However, there are still rooms for improvement towards the goal of achieving a comprehensive transformation solution. We conclude this thesis by discussing several directions for future work.

User Interaction In query transformation, a source query may have several structured queries due to the possibility of different concepts. When this happens, query ranking

would produce a list of ranked queries. Although all the suggested queries maybe relevant, we still find it hard to utilize them except for the top-1 query in which we can propose to user as the best query. Therefore, if users can involve at the stage of query selection, it could greatly help to improve the precision. Nevertheless, the interaction should be carried out in a user friendly and indirect manner, rather than letting them selecting a list of structured queries.

Linguistic Approach for Query Interpretation When a source query contains linguistic structure, e.g. a description of information needs, we find that simple linguistic parser or rules can be useful to identify structural and content keywords used in a query. It will save the cost of processing especially, when a keyword turns out to be both structural and content keywords using thesaurus matching method, we are required to use all the combinations as possible structured query candidates. However, we cannot rely on linguistic method alone as well, as we know that there can be quite a number of linguistic patterns that may appear in a query. We may not be able to define all of them. Thus, a combination of linguistic and collection thesaurus could be useful to address the issue.

Markups Classification Sometimes, a collection may consists of combination of different types of markups, including logical, linguistic (Zhao & Callan, 2008), thesaurus, categorical etc. Although all these markups may be meaningful, they serve different purposes. Combining them during a query transformation may result in a mix up interpretation. Therefore, we are interested to explore on how to classify such markups so that they can be used efficiently.

Structured Query to Structured Query Transformation Lastly, the outcome of our initial motivation of transforming unstructured query to structured query can actually be extended to allow transformation of one structured query to another type structured query.

This could be useful for internal usage among structured retrieval systems experts, where they can query systems that use other query languages automatically.

Appendices

APPENDIX A

EVALUATIONS AND BASELINES

This appendix describes some formulas used in evaluation.

A.1 Structured Query Complexity

Structure query complexity, *SQC* is an approximate measurement to calculate the complexity of a structured query based on three factors, i.e. the frequency of structural term, the frequency of content term, and the level of complexity such as total main(or sub) queries or total conditions.

$$SQC = f_{ST} + f_{CT} + L$$

, where f_{ST} is the total number of structure terms, f_{CT} is the total number of content terms, L is the total levels of complexity.

$f_{ST} \geq 0$, indicating that structure term is optional. $f_{CT} \geq 1$, indicating that there must be at least one content term. $L \geq 1$, indicating that there must be at least one main query or condition.

A.1.1 Examples

Here, we show some examples of *SQC* score on different types of structured queries.

NEXI

- Structured query = //movie[about(.//director ,“terry gilliam")]//actor[about(.//name, “benicio del toro") AND about(.//character, “dr gonzo")]
- Structure terms = {movie, director, actor, name, character}.

- Content terms = {terry gilliam, benicio del toro, dr gonzo}.
- Levels = {//movie[about(.//director ,“terry gilliam")], //actor[about(.//name, “benicio del toro") AND about(.//character, “dr gonzo")]}.
- $SQC = 5 + 3 + 2 = 10$

XQuery

- Structured Query = for $c in document(“geobase.xml")//city where $c/state = “Virginia" return <result>{$c/text()} </result>
- Structure terms = {city, state}.
- Content terms = {Virginia}.
- Levels = {for $c in document(“geobase.xml")//city where $c/state = “Virginia" return <result>{$c/text()} </result>}.
- $SQC = 2 + 1 + 1 = 4$

A.2 Term Weighting Models for XML Element

This section presents some basic term weighting models (Manning et al., 2008; Wang et al., 2007) that can be used with *Context-based Term Weighting* in the query transformation framework.

Requirements Here is a list of statistics required by the models. We denote t for a term, and e for an element. $score(e,t)$ is the score of a term in an element.

$tf(t,e)$ - the number of term, t, in an element, e.

$ef(t)$ - the number of elements containing term, t.

N_e - the total number of elements in a collection.

$len(e)$ - the size of element, e, measured by number of terms.

$len(col)$ - the size of collection, measured by number of terms.

avgl - the average length of elements in the collection, measured by taking the proportion of $len(col)$ against N_e.

Term Frequency This model is the basic of term scoring, it states that if an element mentions a query term more than others, it is likely to be more related to the query, and such that gets a better score.

$$score(e,t) = tf(t,e)$$

Term Frequency Inversed Document Frequency If a term appears in almost all elements, it looses its discriminative power, as all the elements would be relevant to this query term. This model extends the basic term frequency by a factor that controls the score of a term based on its occurrences in the collection. If a term occurs in more elements, the score is lower, and vice versa.

$$score(e,t) = tf(t,e) * log\frac{N_e}{ef(t)}$$

Okapi BM25 In a collection, the length of elements may not be consistent. This model introduces additional parameters to address this issue. The size of element is included this weighting scheme by taking the length of element, e against the average element length of the whole collection. k is a parameter to calibrate the weight of term frequency. b is another parameter to determine the weight of element length ($0 \leq b \leq 1$, b=1 means fully scaling a term weight by element length, b=0 means no length normalization) (Jones, Walker, & Robertson, 2000).

$$score(e,t) = \frac{(k+1) * tf(t,e)}{k * ((1-b) + b * \frac{len(e)}{avgl}) + tf(t,e)} * log\frac{N_e - ef(t) + 0.5}{ef(t) + 0.5}$$

Language Model This model uses the idea of an element is a good match to a query of the element is likely to generate the query. The intuition behind this model is that for each element, we want to find out what is the probabilistic of generating a query based on the words appear in this element. There are two parts of query estimation, the first estimates the query likelihood in an element and the second estimates the query likelihood in the collection. λ is a parameter to adjust the weight for the former and latter ($0 < \lambda < 1$) (Ponte & Croft, 1998).

$$score(e,t) = (1-\lambda)\frac{tf(t,e)}{len(e)} + \lambda\frac{tf(t,col)}{len(col)}$$

REFERENCES

Amer-Yahia, S., & Lalmas, M. (2006). Xml search: languages, inex and scoring. *SIGMOD Record*, *35*(4), 16-23.

Arampatzis, A., & Kamps, J. (2008). A study of query length. In S.-H. Myaeng, D. W. Oard, F. Sebastiani, T.-S. Chua, & M.-K. Leong (Eds.), *Sigir* (p. 811-812). ACM.

Baeza-Yates, R. A., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Bai, J., Nie, J.-Y., Cao, G., & Bouchard, H. (2007). Using query contexts in information retrieval. In *Proceedings of the 30th annual international acm sigir conference on research and development in information retrieval* (pp. 15–22). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1277741.1277747

Bao, Z., Lu, J., Ling, T. W., & Chen, B. (2010). Towards an effective xml keyword search. *IEEE Transactions on Knowledge and Data Engineering*, *22*, 1077-1092.

Barranco, C. D., Campaña, J. R., & Medina, J. M. (2005). Towards a xml fuzzy structured query language. In E. Montseny & P. Sobrevilla (Eds.), *Eusflat conf.* (p. 1188-1193). Universidad Polytecnica de Catalunya.

Bilotti, M. W., Ogilvie, P., Callan, J., & Nyberg, E. (2007). Structured retrieval for question answering. In *Proceedings of the 30th annual international acm sigir conference on research and development in information retrieval* (pp. 351–358). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1277741.1277802

Boag, S., Berglund, A., Chamberlin, D., Siméon, J., Kay, M., Robie, J., et al. (2007, January). *XML path language (XPath) 2.0* (W3C Recommendation). W3C. (http://www.w3.org/TR/2007/REC-xpath20-20070123/)

Bobed, C., Trillo, R., Mena, E., & Ilarri, S. (2010). From keywords to queries: discovering the user's intended meaning. In *Proceedings of the 11th international conference on web information systems engineering* (pp. 190–203). Berlin, Heidelberg: Springer-Verlag. Available from http://dl.acm.org/citation.cfm?id=1991336.1991359

Bray, T., Paoli, J., & Sperberg-McQueen, C. M. (1997). Extensible markup language (xml). *World Wide Web Journal*, *2*(4), 27-66.

Calado, P., Silva, A. S. da, Vieira, R. C., Laender, A. H. F., & Ribeiro-Neto, B. A. (2002). Searching web databases by structuring keyword-based queries. In *Cikm'* (p. 26-33).

Carmel, D., Efraty, N., L, G. M., Maarek, Y. S., & Mass, Y. (2002). An extension of the vector space model for querying xml documents via xml fragments. In *In workshop on xml and information retrieval, sigir*.

Carmel, D., Maarek, Y., Mandelbrod, M., Mass, Y., & Soffer, A. (2003). Searching xml documents via xml fragments. In *Proceedings of the 26th annual international acm sigir conference on research and development in informaion retrieval* (p. 151-158). New York: ACM.

Chamberlin, D. D. (2002). Xquery: An xml query language. *IBM Systems Journal*, *41*(4), 597-615.

Chi, C.-H., Ding, C., & Lam, K.-Y. (2002). Context query in information retrieval. In *Ictai* (p. 101-106). IEEE Computer Society.

Cohen, S., Mamou, J., Kanza, Y., & Sagiv, Y. (2003). Xsearch: a semantic search engine for xml. In *Proceedings of the 29th international conference on very large data bases - volume 29* (pp. 45–56). VLDB Endowment. Available from http://dl.acm.org/citation.cfm?id=1315451.1315457

Dopichaj, P. (2007). Improving content-oriented xml retrieval by exploiting small elements. In *Bncod workshops* (p. 68-74). IEEE Computer Society.

Fallside, D. C., & Walmsley, P. (2004, October). *Xml schema part 0: Primer second edition.* W3C Recommendation. Available from http://www.w3.org/TR/xmlschema-0/

Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (2002a). INEX: INitiative for the Evaluation of XML retrieval.
(http://www.is.informatik.uni-duisburg.de/bib/docs/Fuhr_etal_02a.html)

Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (Eds.). (2002b, December 9-11). *Proceedings of the first workshop of the initiative for the evaluation of xml retrieval (inex).* Schloss Dagstuhl, Germany.

Fuhr, N., Lalmas, M., Malik, S., & Szlávik, Z. (Eds.). (2005). *Advances in xml information retrieval, third international workshop of the initiative for the evaluation of xml retrieval, inex 2004, dagstuhl castle, germany, december 6-8, 2004, revised selected papers* (Vol. 3493). Springer.

Gonçalves, M. A., Fox, E. A., Krowne, A., Calado, P., Laender, A. H. F., Silva, A. S. da, et al. (2004). The effectiveness of automatically structured queries in digital libraries. In *Proceedings of the 4th acm/ieee-cs joint conference on digital libraries* (pp. 98–107). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/996350.996377

Graupmann, J. (2004). Concept-based search on semi-structured data exploiting mined semantic relations. In *Proceedings of the 2004 international conference on current trends in database technology* (pp. 34–43). Berlin, Heidelberg: Springer-Verlag. Available from http://dx.doi.org/10.1007/978-3-540-30192-9_4

Graupmann, J., Biwer, M., Zimmer, C., Zimmer, P., Bender, M., Theobald, M., et al. (2004). Compass: a concept-based web search engine for html, xml, and deep web data. In *Proceedings of the 30th international conference on very large data bases*

(Vol. 30, p. 1313-1316). VLDB Endowment.

Gövert, N., Fuhr, N., Lalmas, M., & Kazai, G. (2006). Evaluating the effectiveness of content-oriented xml retrieval methods. *Information Retrieval*, *9*(6), 699-722. Available from http://dx.doi.org/10.1007/s10791-006-9008-2

Hsu, W., Lee, M. L., & Wu, X. (2004). Path-augmented keyword search for xml documents. In *Proceedings of the 16th ieee international conference on tools with artificial intelligence* (pp. 526–530). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICTAI.2004.99

Huffman, S. B., & Baudin, C. (1997). Toward structured retrieval in semi-structured information spaces. In *Ijcai (1)* (p. 751-757). Morgan Kaufmann.

Itakura, K. Y., & Clarke, C. L. A. (2010). A framework for bm25f-based xml retrieval. In F. Crestani, S. Marchand-Maillet, H.-H. Chen, E. N. Efthimiadis, & J. Savoy (Eds.), *Sigir* (p. 843-844). ACM.

Jayapandian, M., & Jagadish, H. V. (2008). Automated creation of a forms-based database query interface. *PVLDB*, 695-709.

Jones, K. S., Walker, S., & Robertson, S. E. (2000, November). A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, *36*(6), 779–808. Available from http://dx.doi.org/10.1016/S0306-4573(00)00015-7

Kamps, J., Marx, M., Rijke, M. de, & Sigurbjörnsson, B. (2005). Structured queries in xml retrieval. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury, & W. Teiken (Eds.), *Cikm* (p. 4-11). ACM.

Kazai, G., Gövert, N., Lalmas, M., & Fuhr, N. (2003). The inex evaluation initiative. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, & G. Weikum (Eds.), *Intelligent search on xml data* (Vol. 2818, p. 279-293). Springer.

Kazai, G., Lalmas, M., & Vries, A. P. de. (2004). The overlap problem in content-oriented xml retrieval evaluation. In *Proceedings of the 27th annual international acm sigir conference on research and development in information retrieval* (pp. 72–79). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1008992.1009008

Kazai, G., & Trotman, A. (2007). Users' perspectives on the usefulness of structure for xml information retrieval. In *In proceedings of the 1st international conference on the theory of inofrmation retrieval* (pp. 247–260).

Kim, J. Y., Xue, X. B., & Croft, W. B. (2009). A probabilistic retrieval model for semistructured data. In *Ecir* (p. 228-239).

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on artificial intelligence - volume 2* (pp. 1137–1143). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from http://dl.acm.org/citation.cfm?id=

1643031.1643047

Lalmas, M., & Tombros, A. (2007). Evaluating xml retrieval effectiveness at inex. *SIGIR Forum*, *41*(1), 40-57.

Lehtonen, M. (2006, August). Designing user studies for xml retrieval. In *Proceedings of the sigir 2006 workshop on xml element retrieval methodology* (pp. 28–34). Dunedin, New Zealand: University of Otago.

Ley, M. (2009). Dblp - some lessons learned. *PVLDB*, *2*(2), 1493-1500.

Li, J., Liu, C., Zhou, R., & Ning, B. (2009). Processing xml keyword search by constructing effective structured queries. In Q. Li, L. Feng, J. Pei, X. S. Wang, X. Zhou, & Q.-M. Zhu (Eds.), *Apweb/waim* (Vol. 5446, p. 88-99). Springer.

Li, R., & Weide, T. P. van der. (2009). Language models for xml element retrieval. In S. Geva, J. Kamps, & A. Trotman (Eds.), *Inex* (Vol. 6203, p. 95-102). Springer.

Liu, Z., Walker, J., & Chen, Y. (2007). Xseek: A semantic xml search engine using keywords. In C. Koch et al. (Eds.), *Vldb* (p. 1330-1333). ACM.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

McHale, M. (1998, August). A comparison of wordnet and roget's taxonomy for measuring semantic similarity. In S. Harabagiu & J. Y. Chai (Eds.), *Proceedings of the coling/acl workshop on usage of wordnet in natural language processing systems.* Montreal, Canada: Association for Computational Linguistics, Morristown, NJ, USA. Available from http://www.citebase.org/abstract?id=oai:arXiv.org:cmp-lg/9809003

Ogilvie, P., & Callan, J. (2002). Language models and structured document retrieval. In N. Fuhr, N. Gövert, G. Kazai, & M. Lalmas (Eds.), *Inex workshop* (p. 33-40).

Ogilvie, P., & Callan, J. (2004). Hierarchical language models for xml component retrieval. In N. Fuhr, M. Lalmas, S. Malik, & Z. Szlávik (Eds.), *Inex* (Vol. 3493, p. 224-237). Springer.

Pal, S., & Mitra, M. (2007). *Xml retrieval: A survey.* Technical Report, CVPR. TR/ISI/CVPR/IR07-01.

Pehcevski, J., & Thom, J. A. (2005). Hixeval: Highlighting xml retrieval evaluation. In *In proceedings of the inex 2005 workshop* (pp. 43–57).

Pereira, F., Rajaraman, A., Sarawagi, S., Tunstall-Pedoe, W., Weikum, G., & Halevy, A. Y. (2009). Answering web questions using structured data - dream or reality? *PVLDB*, *2*(2), 1646.

Petkova, D., Croft, W. B., & Diao, Y. (2009). Refining keyword queries for xml retrieval by combining content and structure. In *Ecir* (p. 662-669).

Piwowarski, B., Trotman, A., & Lalmas, M. (2008, December). Sound and complete relevance assessment for xml retrieval. *ACM Trans. Inf. Syst.*, *27*(1), 1:1–1:37. Available from http://doi.acm.org/10.1145/1416950.1416951

Ponte, J. M., & Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st annual international acm sigir conference on research and development in information retrieval* (pp. 275–281). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/290941.291008

Reid, J., Lalmas, M., Finesilver, K., & Hertzum, M. (2006). Best entry points for structured document retrieval - part i: Characteristics. *Inf. Process. Manage.*, *42*(1), 74-88.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *In proceedings of the 14th international joint conference on artificial intelligence (ijcai-95* (pp. 448–453). Morgan Kaufmann.

Robertson, S., & Zaragoza, H. (2009, April). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, *3*(4), 333–389. Available from http://dx.doi.org/10.1561/1500000019

Salton, G., & Buckley, C. (1988, August). Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, *24*(5), 513–523. Available from http://dx.doi.org/10.1016/0306-4573(88)90021-0

Sandhaus, E. (2008). The new york times annotated corpus [Computer software manual]. Philadelphia.

Schenkel, R., Suchanek, F., & Kasneci, G. (2007, March). Yawn: A semantically annotated wikipedia xml corpus. In *Proceedings of 12th symposium on database systems for business, technology and the web of the german society for computer science.* (p. 277-291).

Sigmod. (2007). *Sigmod record in xml.* Available from http://www.sigmod.org/publications/sigmod-record/xml-edition ([Online; accessed 22 May 2013])

Somers, H. L. (1999). Review article: Example-based machine translation. *Machine Translation*, *14*(2), 113-157.

Spink, A., & Jansen, B. J. (2004). Web search: Public searching of the web. In (p. 77-99). Springer Netherlands.

Spink, A., Wolfram, D., Jansen, M. B. J., & Saracevic, T. (2001, February). Searching the web: the public and their queries. *J. Am. Soc. Inf. Sci. Technol.*, *52*(3), 226–234. Available from http://dx.doi.org/10.1002/1097-4571(2000)9999:9999<::AID-ASI1591>3.3.CO;2-I

Sumita, E., & Iida, H. (1991). Experiments and prospects of example-based machine translation. In D. E. Appelt (Ed.), *Acl* (p. 185-192). ACL.

Taha, K., & Elmasri, R. (2010). Xcdsearch: An xml context-driven search engine. *IEEE*

Trans. Knowl. Data Eng., *22*(12), 1781-1796.

Tannier, X. (2005). From natural language to nexi, an interface for inex 2005 queries. In *Inex* (p. 373-387).

Tannier, X., Girardot, J.-J., & Mathieu, M. (2005). Analysing natural language queries at inex 2004. In *Proceedings of the third international conference on initiative for the evaluation of xml retrieval* (pp. 395–409). Berlin, Heidelberg: Springer-Verlag. Available from http://dx.doi.org/10.1007/11424550_32

Teevan, J., Adar, E., Jones, R., & Potts, M. (2006). History repeats itself: repeat queries in yahoo's logs. In *Proceedings of the 29th annual international acm sigir conference on research and development in information retrieval* (pp. 703–704). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1148170.1148326

Theobald, M., Bast, H., Majumdar, D., Schenkel, R., & Weikum, G. (2008). Topx: efficient and versatile top-k query processing for semistructured data. *VLDB J.*, *17*(1), 81-115.

Tran, T., Wang, H., Rudolph, S., & Cimiano, P. (2009). 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* (pp. 405–416). Washington, DC, USA: IEEE Computer Society. Available from http://dx.doi.org/10.1109/ICDE.2009.119

Trotman, A. (2009). Narrowed extended xpath i. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems* (p. 1876-1880). Springer US.

Trotman, A., & Lalmas, M. (2006). Why structural hints in queries do not help xml-retrieval. In *Proceedings of the 29th annual international acm sigir conference on research and development in information retrieval* (pp. 711–712). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1148170.1148330

Trotman, A., Rocio Gomez Crisostomo, M. del, & Lalmas, M. (2009). Visualizing the problems with the inex topics. In J. Allan, J. A. Aslam, M. Sanderson, C. Zhai, & J. Zobel (Eds.), *Sigir* (p. 826). ACM.

Trotman, A., & Sigurbjörnsson, B. (2004a). Narrowed extended xpath i (nexi). In N. Fuhr, M. Lalmas, S. Malik, & Z. Szlávik (Eds.), *Inex* (Vol. 3493, p. 16-40). Springer.

Trotman, A., & Sigurbjörnsson, B. (2004b). Nexi, now and next. In N. Fuhr, M. Lalmas, S. Malik, & Z. Szlávik (Eds.), *Inex* (Vol. 3493, p. 41-53). Springer.

Voorhees, E. M. (1998). Variations in relevance judgments and the measurement of retrieval effectiveness. In *Proceedings of the 21st annual international acm sigir conference on research and development in information retrieval* (pp. 315–323). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/290941.291017

Wang, Q., Li, Q., & Wang, S. (2007). Preliminary work on xml retrieval. In *Inex* (p. 70-76).

Wang, Q., Ramírez, G., Marx, M., Theobald, M., & Kamps, J. (2011). Overview of the INEX 2011 data centric track. In S. Geva, J. Kamps, & R. Schenkel (Eds.), *Inex 2011 workshop pre-proceedings* (pp. 88–106).

Wikipedia. (2012). *Google search — wikipedia, the free encyclopedia.* Available from http://en.wikipedia.org/w/index.php?title=Google_Search&oldid=526339125 ([Online; accessed 5-December-2012])

Woodley, A., & Geva, S. (2004). Nlpx at inex 2004. In N. Fuhr, M. Lalmas, S. Malik, & Z. Szlávik (Eds.), *Inex* (Vol. 3493, p. 382-394). Springer.

Woodley, A., & Geva, S. (2006). Nlpx at inex 2006. In N. Fuhr, M. Lalmas, & A. Trotman (Eds.), *Inex* (Vol. 4518, p. 302-311). Springer.

Xu, Y., & Papakonstantinou, Y. (2008). Efficient lca based keyword search in xml data. In A. Kemper et al. (Eds.), *Edbt* (Vol. 261, p. 535-546). ACM.

Zenz, G., Zhou, X., Minack, E., Siberski, W., & Nejdl, W. (2009, September). From keywords to semantic queries-incremental query construction on the semantic web. *Web Semant.*, *7*(3), 166–176. Available from http://dx.doi.org/10.1016/j.websem.2009.07.005

Zhao, L., & Callan, J. (2008). A generative retrieval model for structured documents. In *Proceedings of the 17th acm conference on information and knowledge management* (pp. 1163–1172). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1458082.1458236

Zhao, L., & Callan, J. (2009). Effective and efficient structured retrieval. In *Proceedings of the 18th acm conference on information and knowledge management* (pp. 1573–1576). New York, NY, USA: ACM. Available from http://doi.acm.org/10.1145/1645953.1646175

Zwol, R. van, Baas, J., Oostendorp, H. van, & Wiering, F. (2006). Bricks: The building blocks to tackle query formulation in structured document retrieval. In M. Lalmas, A. MacFarlane, S. M. Rüger, A. Tombros, T. Tsikrika, & A. Yavlinsky (Eds.), *Advances in information retrieval, 28th european conference on ir research, ecir 2006, london, uk, april 10-12, 2006, proceedings* (Vol. 3936, p. 314-325). Springer.