NAIST-IS-DD1161007

Doctoral Dissertation

A Study on Effective and Efficient XML Element Retrieval Considering Document Updates

Atsushi Keyaki

February 6, 2014

Department of Information Science
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Atsushi Keyaki

Thesis Committee:

Professor Hirokazu Kato	(Supervisor)
Professor Yuji Matsumoto	(Co-supervisor)
Professor Jun Miyazaki	(Tokyo Institute of Technology)
Associate Professor Kenji Hatano	(Doshisha University)
Assistant Professor Goshiro Yamamoto	(Co-supervisor)

A Study on Effective and Efficient XML Element Retrieval Considering Document Updates*

Atsushi Keyaki

Abstract

Extensible Markup Language (XML) is a markup language for structured documents that has become the de facto format for data exchange. The data formats of many applications, such as Wikipedia articles and office documents, are defined as XML formats. A large number of XML documents are available on the Web, and we expect this trend to continue in the future.

Search system users have to find out information they need by themselves, because most of existing search systems return a list of documents as search results. It takes a large effort to find it out from long-length documents. In addition, there is a possibility that users cannot find useful information nevertheless they spent long time on the documents. This information seeking process is much cost in information retrieval. On the other hand, users of XML element search systems need not to find out information they need by themselves because XML element search systems return a list of elements which satisfy users' information need. Therefore, the framework of an XML element search system can reduce the cost in information retrieval, which is the reason why XML element retrieval techniques are useful and worth working on.

There are two main streams for researches of XML element retrieval techniques, i.e., 1) attaining effective search for satisfying accurate information retrieval, and 2) attaining efficient search for fast query processing. In order to satisfy 1), we proposed a scoring method to identify informative XML elements and a reconstruction method of search results to identify the most appropriate

*Doctoral Dissertation, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1161007, February 6, 2014.

granularity of XML elements as search results. Our experimental evaluations showed our proposed methods overwhelmed existing methods in search accuracy.

Then, we also try to handle document updates of an XML search system. This is because document updates need to be managed when we come to think of a practical use of search systems. If document updates are not handled in a search system, users cannot obtain appropriate search results, which reduces the usefulness of the search system. We propose to extend a function of incremental updates of indices to general XML element search systems, with filters to reduce the update cost by eliminating unimportant elements and terms. Moreover, we apply a method for integrating path expression which estimates accurate global weights in term calculation. We confirmed the proposed methods update indices in short time without a drop in search accuracy.

As an output of these researches, we developed a practical XML element search system which achieves accurate search and fast query processing with satisfying immediate reflection of document updates. An expected application of XML element retrieval techniques is Web document especially HTML document which is one of the the most common data format of Web documents. To fill the gap between XML element retrieval and element retrieval for HTML documents, say HTML element retrieval, we proposed methods for adapting (XML) element retrieval techniques to HTML documents. These are 1) a reconstruction method for resolving disagreement between a logical document structure and a physical document structure, and 2) a filter to eliminate outside of the main content (sub–content). The findings from our experiments are that the reconstruction method improved search performance while the filter removed elements composed of sub–content properly.

Keywords:

Extensible Markup Language (XML), element retrieval, accurate retrieval, fast sarch, document update, indexing, statistics

文書の更新を考慮した高精度かつ高速な XML 部分文書検索に関する研究*

欅 惇志

内容梗概

W3C によって策定された Extensible Markup Language (XML) はデータ交換の標準フォーマットとして広く利用されており，近年は様々なアプリケーションのデータ，Wikipedia 記事や Office 文書など多くの用途に利用されている．そのため，現在までに膨大な数の XML 文書が蓄積され，今後ますます多くの XML 文書が作成されると予想される．

従来の検索エンジンのうちのほとんどは，検索結果として文書のリストを提示する．そのため，ユーザは各文書中から欲しい情報を自ら発見する必要があるが，記述量の多い文書から求める情報を抽出する際には大きな労力を必要とし，また，長時間を費やしても欲しい情報を含まない可能性もあり，これらの作業はユーザにとって大きな負担である． それに対して，XML 部分文書検索では，検索結果として文書のうちユーザが必要とする情報が記述された箇所を特定して提示することを目指す．従って，ユーザの検索時の労力を軽減することが可能な，非常に有用な検索技術である．

既存の XML 部分文書検索に関する研究では，主に，1) ユーザが求める必要十分の内容を検索結果として提示することを目指す高精度な検索と，2) ユーザに対して高速に検索結果を提示することを目指す高速な検索が取り組まれてきた．我々は高精度な検索を実現するために，情報量の多い部分を発見するためのスコアリング手法と，文書中の検索結果として最適な部分を特定するための検索結果構築手法の提案を行った．評価実験の結果，提案手法は従来の手法と比較してより高精度に検索が可能であるという結果が得られた．

また，我々は検索システムの実運用を想定した場合には必ず発生する文書の更新への対応を目指した．なぜなら，これら文書の更新に対応しなかった場合には，

*奈良先端科学技術大学院大学 情報科学研究科 専攻 博士論文, NAIST-IS-DD1161007, 2014 年 2 月 6 日.

検索システムはユーザに対して適切な検索結果を提示することはできず，その結果，検索システムの利便性が低下するためである．我々は，一般的な構成の XML 部分文書検索システムに差分更新機能を搭載させるべく新たな索引構造を定義し，更新対象のうち不要なデータを除外するためのフィルタを提案した．更に，索引語の重み算出時において正確な大域的重み算出のためのパス式統合手法の提案を行った．評価実験の結果，提案手法を適用することで，検索精度の低下を抑制しつつ，極めて短時間で索引の更新を実現した．

これらの研究の成果物として，正確な検索と高速なクエリ処理，そして文書の更新の即座の反映を満たした実用的な XML 部分文書検索システムを開発した．XML 部分文書検索技術の期待される応用対象として Web 文書が存在する．XML 部分文書検索と，Web 文書の中でも代表的なファイルフォーマットである HTML 文書に対する部分文書検索の間に存在する差異を解決するため，HTML 文書に対して部分文書検索技術の適用を目的とした文書の整形手法，即ち，1) 文書の論理構造と物理構造の不一致解消のための文書の再構造化手法，2) 本文以外のサブ・コンテンツ除外のためのフィルタの提案を行った．評価実験から，再構造化手法によって計算句性能が向上し，フィルタによってサブ・コンテンツの除外が確認された．

キーワード

Extensible Markup Language (XML), 部分文書検索，高精度 XML 検索，高速 XML 検索，文書の更新，索引，統計量

Acknowledgments

I warmly thank my supervisor, Professor Hirokazu Kato. His comments, based on the view point of a different research area, are quite useful to train my skills to abstract my idea. Nevertheless my major is different from his major, he welcomed to his laboratory. Moreover, he always cared me warmly. I could stay IMD Lab in a comfortable way thanks to his gentle manner. I also thank that I could learn about computer vision and human interface to some extent. It is good opportunity for me to turn my eyes to other directions of research.

I owe deepest thanks to Professor Jun Miyazaki at University of Tokyo Institute of Technology for his appropriate supervision. He gave me many valuable advice and comments on both my research and attitude for being a good researcher. He has always guided me in patience since my B.S. degree, even when I have not had sufficient knowledge and skill to discuss with him. He kindly repeated the discussion many times until I got understood. I will always keep in mind that research need to be based on a practical problems as one of many lessons given by him.

I appreciate Associate Professor Kenji Hatano at Doshisha University. The experiences during my B.S. degree and M.S. degree at Doshisha University became the basis of my current research. I also appreciate him for encouraging me in many ways. In particularly, I really thank that he recommended me to go on to this school. I'm sure that this three years made me improved. Then, I also thank that he gave me many useful comments after my graduation from his laboratory.

I would like to thank my thesis co-supervisor Professor Yuji Matsumoto. Thank you very much for reviewing my thesis and for the insightful comments and suggestions that helped me to improve the overall quality of this thesis.

A special gratitude to Assistant Professor Goshiro Yamamoto. Discussions with him on not only how to do a good research but also how to manage a laboratory are full of fun. I am also grateful to Assistant Professor Takafumi Taketomi for his valuable and sharp comments to my study.

I am also indebted to Assistant Professor Toshiyuki Shimizu at Kyoto University for his kindness and help with my research and other fellowship applications.

I also would like to show my gratitude to Associate Professor Toshiyuki Amagasa at University of Tsukuba for his honest comments. These comments prompted me to improve my presentation.

I also appreciate Professor Haruo Yokota at University of Tokyo Institute of Technology. He always asked me many critical questions on conferences.

I'm very thankful with associate researcher Yuki Arase and Principal researcher Jun'ichi Tsujii at Microsoft Research Asia for their supervision during my stay at Microsoft Research Asia. These days were really exciting and enjoyable.

I also owe my special thanks to Associate Professor Tetsuya Sakai at Waseda University for encouraging me to apply for study abroad. Without your suggestion, I would not have gone through amazing experiences.

My thanks are also extended to all current and former members of IMD Lab. In special, I cannot thank enough Jaakko Hyry, Marc Ericson Santos, Max Krichenbauer, and Luiz Gustavo Moreira Sampaio for proofreading some Chapters of my thesis. Your suggestions and corrections were indeed a great help.

A kindly thanks to Yuichiro Fujimoto for sharing challenging time with me as the only other Japanese Ph.D. course student. Our talks and your attitude for research always made me feel better and encouraged.

I would like to thank the Lab's secretary Ms. Makiko Ueno for her help with official documents and other bureaucracies.

I am very grateful to the Japan Society for the Promotion of Science (JSPS) for adopting me as JSPS Research Fellow (DC2). During my Ph.D. program, I was able to have a comfortable student life thanks to the financial support.

This acknowledgment is extended to all members of my family, whose support give me enough strength to overcome all obstacles. Especially, I would like to send

my word to my parents. I respect my father Kazumori Keyaki from the bottom of my heart for his unique way of thinking. I will try to follow my mother Hiroko Keyaki's example being kind with the others.

Contents

List of Figures

List of Tables

List of Abbreviations

DB	DataBase
HTML	HyperText Markup Language
IR	Information Retrieval
NTCIR	NII Testbeds and Community for Information access Research
NLP	Natural Language Processing
RDB	Relational DataBase
RDBMS	Relational DataBase Management System
INEX	Initiative for the Evaluation of XML Retrieval
W3C	World Wide Web Consortium
XML	Extensible Markup Language

CHAPTER 1

Introduction

1.1. Background and motivation

Nowadays, the cyber world has changed for Internet users. The users can not only utilize information on the World Wide Web (WWW, Web) by W3C [1], but also generate and publish information as a content generator. As a result, a tremendous amount of data is stored on the Web, which is represented as the information explosion era or the big data era. In this situation, it is almost impossible to find out information they need from the Web without any tools. Therefore, the users use search systems to find the information efficiently and effectively.

The most popular and commonly used search system is a document search system. It plays a necessary role in our daily life. However, it is not true that document search system users are completely satisfied with state-of-the-art document search systems. A survey on the trustworthiness of Web search systems [2] reports that users think it is a waste of time for deciding documents to browse from a list of documents retrieved as search results. During information seeking process, many Web search users utilize the title and the result snippet[1] of

[1]A result snippet is a short summary of each document. It is discussed in Section 2.2.4.1 in

each document. However, they also think that these kinds of information are not enough for efficient browsing, and they want to utilize information other than title and result snippets.

Moreover, not only text documents but also many kinds of data such as images, music (audio) files, and movies appear on the Web these days. In addition, users' information need is becoming more and more diverse. These cause appearances of many kinds of search systems, i.e., an image search system, a music search system, a movie search system, and so on. Each search system is useful as far as users' information need is satisfied within a specific data format domain. These search systems specified to a certain data format domain cannot be used efficiently when the users' information need is satisfied across two or more data format domains, and the search system is expected to target multiple data formats at the same time.

From the discussion above, a novel framework of information retrieval, beyond document retrieval, is required. This is why an Extensible Markup Language [3] (XML) search system attracts a lot of attentions recently. XML is a markup language for structured documents that has become the de facto format for data exchange. This means data defined by different data formats can be exchanged with each other only if these data formats are based on XML. These data are targeted at the same time with an XML search system. The data formats of many applications, such as Wikipedia [4] articles and office documents, are defined as XML-based data formats. A large number of XML documents are available on the Web, and we expect this trend to continue in the future.

There are several search techniques for XML documents, but we mainly focus on XML element retrieval [5] among these techniques. We would like to show the advantage of an XML element search system. As we mentioned before, search system users have to find out information they need by themselves, because most of existing search systems return a list of documents as search results. It takes a large effort to find it out among long documents. In addition, there is a possibility that users cannot find useful information nevertheless they have already spent a long time on the documents. This information seeking process is expensive in information retrieval. On the other hand, users need not find out information they need by themselves because XML element search systems return a list of

more detail.

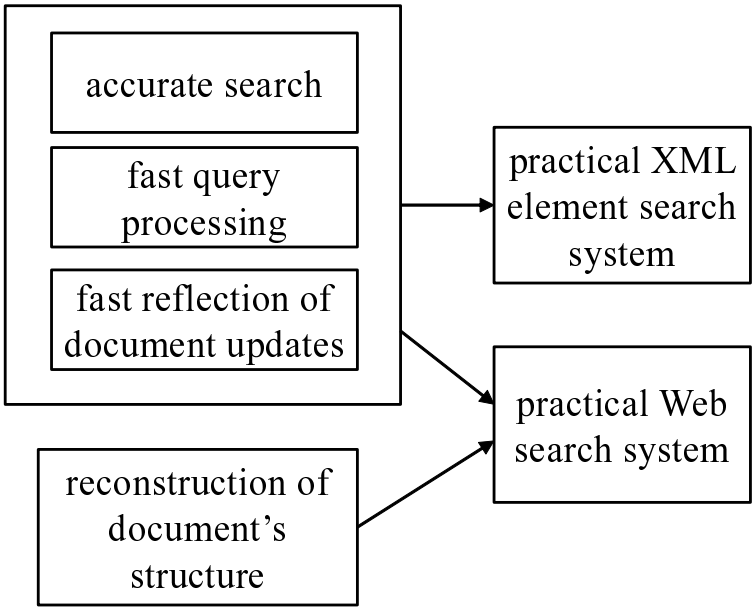


Figure 1.1. Research goal

elements[2] which satisfy the users' information need. We believe this is true because it is reported that relevant descriptions and appropriate granular elements largely agree with each other [6]. Therefore, the framework of an XML element search system can reduce the cost in information retrieval, which is the reason why XML element retrieval techniques are useful and worth working on.

1.2. Research Problems and Solutions

First of all, we show the research goal of this thesis in Figure 1.1. Hereafter, we would like to explain the research goal with each components for the goal.

1.2.1 Accurate XML Element Retrieval

There are two main streams for researches of XML element retrieval techniques, i.e., 1) effective search for satisfying accurate information retrieval, and 2) effi-

[2] An element is a partial document (sub-document) of an XML document. We explain element in Section 2.2.4 more fully.

cient search for fast query processing. Accuracy is the most important aspect for information retrieval systems, because fast search systems that return inaccurate search results quickly are not useful for users. Accordingly, we firstly work on attaining accurate search.

Concerning 1), existing techniques for effective XML retrieval are often derived from methods for document retrieval. In the research field of document retrieval, the traditional approach to find relevant documents out is as follows: first, calculate a term weight of each term in a document with a term-weighting scheme; next, compute a score for each document using these term weights. The approach for XML element retrieval is the same except that a term-weighting scheme is specialized to XML element retrieval. Although these approaches have been successful, these are not enough to attain effective "XML element retrieval" because a gap between the goal of XML element retrieval and that of document retrieval exists. A document search system tries to retrieve a relevant document which contains descriptions satisfying users' information need. By contrast, an XML element search system tries to retrieve a relevant element which satisfies users' information need. This means the XML element search system needs to identify important descriptions in the XML document. Accordingly, the requirement for XML element retrieval is more difficult than that of document retrieval.

Moreover, existing studies don't show a clear and effective strategy for returning search results when several elements in an XML document are judged relevant. The approach that these existing studies apply is just to return an element which is regarded as the most relevant. It is not always true that such element is the most appropriate search result for users.

In addition, existing approaches treat every element as independent search target in term calculation. However, elements in the same document are somewhat influential with each other.

Therefore, we proposed a scoring method to identify informative XML elements, a reconstruction method of search results to identify the best granular XML elements as search results, and a scoring method to identify useful elements with statistics of related elements.

On the other hand, there are many studies focused on 2). Their approaches are mainly classified into two; a) removing useless data before stored into the database, and b) applying an efficient query processing method. Since these

scopes are not our main concern, we basically utilize existing studies regarding 2). However, note that these techniques are related to our study discussed in the next section.

1.2.2 XML Element Retrieval System Considering Document Updates

Document updates need to be managed when we come to think of a practical use of search systems. As such, we also try to handle document updates of an XML element search system. If document updates are not handled in a search system, users cannot obtain appropriate search results, which reduces the usefulness of the search system. Thus, the index of a search system need to be updated according to document updates. When we update an index in the simple manner, that is, rebuilding an index from scratch and treating all data as update targets, update efficiency is very low.

In addition, global weights, i.e. the statistics computed with all documents in the search system, may not be accurate when only a few number of documents is indexed or when global weights change drastically.

To solve the problems related to handling document updates, we propose to extend a function of incremental updates of indices to general XML element retrieval systems, with filters to reduce the update cost by eliminating unimportant elements and terms. Moreover, we apply a method for integrating path expression which estimates accurate global weights in term calculation.

1.2.3 Expansion into Web Documents

As a result of these studies mentioned above, we try to develop a practical XML element search system which achieves accurate search and fast query processing with satisfactory of fast reflection of document updates as Figure 1.1 denotes. There is still a gap between a practical XML element search system and a practical information retrieval system, because the main target of Web document search systems is not XML documents but HyperText Markup Language [7] (HTML) documents. XML documents and HTML are different from each other in some features, although they both originate from Standard Generalized Markup Language [8] (SGML) which is a format of structured document. While XML is

defined for managing the content, HTML is defined for displaying the documents on a Web browser. Accordingly, HTML documents have following features:

1. the document structures of HTML documents are less well-formatted compared with XML documents,

2. there is disagreement between the logical structure of the document contents and physical document structure, and

3. HTML documents contain many of outside of main main–content (sub–content) which are irrelevant for a query.

These features can be harmful for effectiveness of XML element retrieval.

Then, in adapting (XML) element retrieval techniques to HTML document, or HTML element retrieval, we adopt a complement tool for HTML tags, reconstruct a document structure of HTML documents to resolve disagreement between a logical document structure and a physical document structure, and apply a filter to eliminate sub–content (elements).

1.3. Contributions

The main contributions of this thesis are:

- **Accurate XML element retrieval methods**

  We propose 1) a scoring method considering requirements for results snippets, 2) a reconstruction method to identify the best granularity for search results, and 3) a scoring method considering statistics of related elements. Experimental evaluations showed that the proposed methods exceeded a traditional baseline approach in search accuracy. Particularly, the result reconstruction method contributes most to a rise in search accuracy. The most accurate setting of the proposed methods surpasses the state-of-the-art approach of document retrieval.

- **Methods for fast incremental updates**

  We propose 1) an index structure for fast term calculation, 2) two filters for eliminating useless data, and 3) an estimation method of statistics. As a result of experiments, we attained incremental updates of indices in a

short time when document updates occurs. Moreover, we diminished the drop in search accuracy. Furthermore, search accuracy of the proposed method showed equal to grater than that of rebuilding from scratch approach combined with the reconstruction method mentioned above.

- **An expansion of XML element retrieval techniques into Web documents**

  We propose 1) a reconstruction method to conform documents' logical structure to physical documents' structure, and 2) a filter to eliminate sub–content. The experiments yield that the proposed filter removed elements composed of sub–content, which improved search accuracy. Likewise, the reconstruction method improved search performance. Moreover, it also turned out that search accuracy is higher and less laborious with the framework of element retrieval than these of document retrieval.

1.4. Outline of the thesis

This thesis consists of seven chapters. Chapter 2 provides an overview of information retrieval and XML followed by XML element retrieval. It also introduces some notations and definitions used throughout the thesis.

Chapter 3 describes related studies which are composed of studies of XML element retrieval, XML keyword search, dynamic updates of data, and content comprehension of HTML documents.

Chapter 4 focuses on our proposed accurate XML element retrieval methods, whereas Chapter 5 describes a fast incremental indexing method.

In Chapter 6, an expansion of XML element retrieval techniques into Web documents is discussed.

Chapter 7 summarizes this thesis, with discussions about the contributionsand possible future work.

CHAPTER 2

Preliminaries

XML element retrieval are strongly related to two techniques, namely information retrieval and XML search. Thus, this Chapter explains an overview of information retrieval techniques for text documents, followed by XML including XML element retrieval techniques. We also talk about overview of databases systems.

2.1. Information Retrieval

2.1.1 Overview

As a consequence of the Internet's spread and a performance improvement of computers, a variety of data such as text document, music files, and movies appear on the Web. Many approaches are proposed for each type of data to achieve effective and efficient information retrieval. In this overview, we focus on text document retrieval because other information retrieval techniques are based on it.

Figure 2.1 depicts the model of text document retrieval [9]. From a search system user's perspective, the user converts his/her information need to a format which can be input into a computer. The converted information need is called a *query* which is usually a set of keywords in a text document search system. A computer neither interprets a query nor documents generated by humans. Conse-

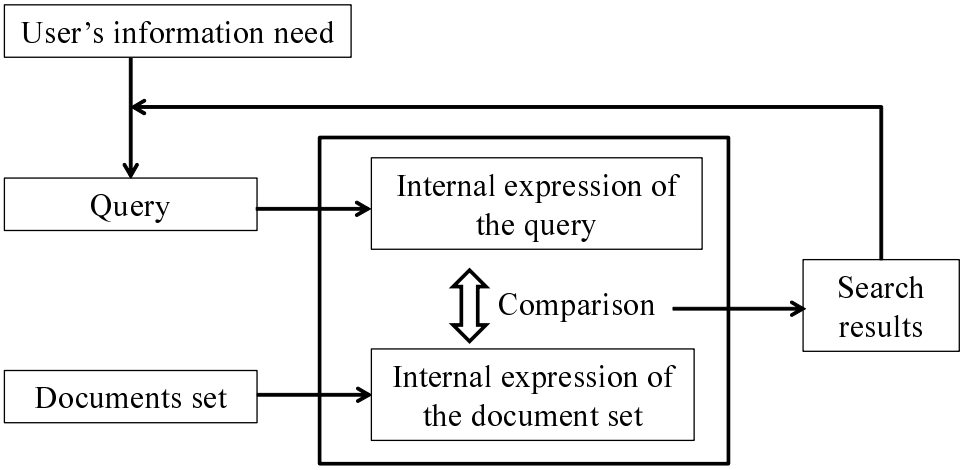


Figure 2.1. Model of information retrieval

quently, both a query and documents set are converted into an internal expression for enabling efficient comparison with each other. As a result of the comparison, each document is assigned a relevancy score to the query. A document is highly relevant if the document has a high relevancy score. The search system returns relevant documents as search results in descending order of their relevancy.

During the process of converting words into an internal expression, such words that are useless to identify relevant documents are eliminated. Prepositions, pronouns, auxiliary verbs, conjunctions, and grammatical articles are applied, and these are classified as function words. On the other hand, words such as nouns, verbs, adjectives, and adverbs are classified as content words. Hereafter, we call a content word as an *index term*, or simply *term*. Then, even in content words, there are some uninformative words called stop words. These words are removed too. SMART stop list [10] is well-used in search systems[1].

Furthermore, many search systems apply stemming process with a query and documents. In a search system, each term is treated just as a symbol. This means that words belonging to different parts of speech or having different tenses are judged completely to be different words even though they are originally derived from the same word. For example, *retrieve*, *retrieved*, *retrieval* are different in terms of surface expression. In many cases, information that these terms con-

[1]Words in the list are described in Appendix **A**

vey are not so different with each other. Therefore, some search systems apply stemming process to omit differences in surface expression. Porter's stemming algorithm [11] is commonly used in information retrieval researches.

2.1.2 Traditional Document Retrieval Techniques

In this section, we describe some kinds of traditional document retrieval techniques.

2.1.2.1 Term-weighting-based Scoring Function

At that time the Web appeared, only reliable and informative documents exist on the Web. Under the circumstances, returning all documents containing a user-issued query keyword is useful enough to extract information a user needs. In other words, a document containing a query keyword is judged relevant, while other document not containing any of query keywords is judged irrelevant. The scoring function is called the boolean model [9].

Along with the growth of the Web, a wide variety of documents appeared. As a result, a huge number of documents which are irrelevant to the information needs of the user are generated. Thus, the boolean model scoring is already inadequate for a search system. Accordingly, more high-performance scoring functions are proposed for attaining effective information retrieval.

TF-IDF [12] is one of the most popular term weighting scheme for document search with the vector space model. Both local weight and global weight are considered in TF-IDF. Local weight is calculated by the term frequency (TF) of each term in each document. This is based on the idea that an event occurring frequently in a document is important. Then, global weight uses inverse document frequency (IDF) of each term in all documents. This is based on the idea that an event is also important if the event seldom occurs in an entire document set. The term weight in TF-IDF is derived by the product of TF and IDF. Concretely, a weight of term t in a document d, $w_{tfidf}(d,t)$, is calculated as follows:

$$w_{tfidf}(d,t) = tf_{d,t} \cdot idf_t, \quad idf_t = 1 + \log \frac{N}{df_t} \tag{2.1}$$

Where $tf_{d,t}$ is the number of t in d, N is the total number of the documents in a document set, and df_t is the number of documents containing t.

A document score is calculated by term weights of query keywords. Let T be a set of query keywords, a score of d, S_{tfidf_d}, is calculated as:

$$S_{tfidf}(d) = \sum_{t_i \in T} w_{tfidf}(d, t_i) \tag{2.2}$$

Although TF-IDF achieves effective search, it has a tendency to give a long document a high score. To avoid this shortcoming, normalized TF-IDF [12] which normalizes based on document length has been proposed. A weight of t in d, w_{ntfidf}, is calculated with:

$$w_{ntfidf}(d, t) = \frac{tf_{d,t}}{dl_d} \cdot idf_t, \;\; idf_t = 1 + \log \frac{N}{df_t} \tag{2.3}$$

where dl_d is the document length (the number of the terms in d).

In the same manner as that of TF-IDF, a score of d in normalized TF-IDF, S_{tfidf_d}, is calculated as follows:

$$S_{ntfidf}(d) = \sum_{t_i \in T} w_{ntfidf}(d, t_i) \tag{2.4}$$

TF-IDF and normalized TF-IDF are the heuristic approaches. Whereas, Okapi's BM25 [13] is based on the classical probabilistic model. Okapi's BM25 leverages on statistics, not only the ones used in TF-IDF and normalized TF-IDF, but also average length of documents in a document set. A weight of t of d, $w_{bm25}(d, t)$, is calculated with the following:

$$w_{bm25}(d, t) = \frac{(k_1 + 1) tf_{d,t}}{k_1((1 - b) + b\frac{dl_d}{avdl}) + tf_{d,t}} \cdot \log \frac{N - df_t + 0.5}{df_t + 0.5} \tag{2.5}$$

where $avdl$ is the average length of documents in a document set. Therefore, a score of d, $S_{bm25}(d)$, is calculated as:

$$S_{bm25}(d) = \sum_{t_i \in T} w_{bm25}(d, t_i) \tag{2.6}$$

BM25F [14] is also proposed for field search as an expansion of Okapi's BM25. Information retrieval techniques for field search are used for structured document. BM25F gives a weight to each field (tag) for considering the importance of a field. For example, give a high weight to a `title` tag which assumes that a document

has high possibility to be relevant when a query keyword appears in the `title` field. A score of d, $S_{bm25f}(d)$, is calculated as follows:

$$S_{bm25f}(d) = \sum_{t_i \in T} fw_{f,t_i} \cdot w_{bm25}(d, f_i, t_i) \tag{2.7}$$

where f_i is the field that t_i appears and fw_{f,t_i} is a weight of f_i.

Statistical language model techniques have been developed in the fields of speech recognition and machine translation. Recently, these techniques have been introduced into the field of information retrieval. In particular, the query likelihood model [15] (QLM) is well studied and achieves significant results, although it is reported that an accurate model cannot be estimated when a document length is small [16]. In the term-weighting scheme, the score of each document is the product of the occupancy probabilities of the query keywords as shown in Eq. (2.12). This means that non-zero values are computed only for the documents containing all the query keywords. To resolve this problem, a term of query keywords is assigned a non-zero value (generally very small value) when a query keyword does not appear in a document. This process is called smoothing techniques, which is often used in QLM. Smoothing values are computed with a background language model calculated with an entire document set.

Based on a survey paper about QLM [16], there are two smoothing methods, i.e., linear interpolation and Dirichlet smoothing. It is said that the Dirichlet smoothing can compute a document's relevancy score accurately even when document length is short. In addition, there is a method that combines linear interpolation and Dirichlet smoothing as a two-stage smoothing method. However, the method cannot overwhelm the either smoothing method.

We show the way of calculating relevancy score with QLM. A probability that t is generated from d (i.e., a term weight of t in d), $\hat{P}_{mle}(t|M_d)$ ($w_{qlm}(d,t)$), is calculated as follows:

$$w_{qlm}(d,t) = \hat{P}_{mle}(t|M_d) = \frac{tf_t}{dl_d} \tag{2.8}$$

To apply the smoothing techniques, Eq. (2.8) is transformed. Eq. (2.9) represents the linear interpolation and Eq. (2.10) represents the Dirichlet smoothing.

$$w_{qlm}(d,t) = \omega \hat{P}_{mle}(t|M_d) + (1-\omega)\hat{P}_{mle}(t|M_b) \tag{2.9}$$

$$w_{qlm}(d,t) = \frac{\hat{P}_{mle}(t|M_d) + \mu \hat{P}_{mle}(t|M_b)}{dl_d + \mu} \tag{2.10}$$

$$\hat{P}_{mle}(t|M_b) = \frac{\sum_{d \in C} tf_{d,t}}{\sum_{d \in C} dl_d} \tag{2.11}$$

where ω and μ are given parameters, M_b is a background model, and C is a document set.

Then, the two-stage smoothing method is calculated as follows:

$$w_{qlm}(d,t) = \omega \frac{\hat{P}_{mle}(t|M_d) + \mu \hat{P}_{mle}(t|M_b)}{dl_d + \mu} + (1-\omega)\hat{P}_{mle}(t|M_b) \tag{2.12}$$

At last, a score of d, $s_{qlm}(d,t)$, is calculated as follows:

$$s_{qlm}(d) = \hat{P}(T|M_d) = \prod_{t_i \in T} w_{qlm}(d,t_i) \tag{2.13}$$

2.1.2.2 Network-analysis-based Scoring Function

Besides scoring methods based on term weighting, there also exists scoring methods based on network analysis to identify important documents. The Web can be interpreted as a graph. In more concrete terms, document and hyperlink represent node and directed edge with arrow, respectively. There are two types of hyperlinks, i.e., an incoming link and an outgoing link. PageRank [17] and HITS [18] are known as famous scoring functions based on network analysis. Note that some researches such as [19] and [20] leverages both a term weighting factor and a network analysis factor.

2.2. XML

In this section, we explain about XML and it's related techniques.

2.2.1 Prominent Structured Documents

Figure 2.2 depicts some of the prominent structured documents. XML is a markup language and one of such structured documents. It is composed of commonly used functions of Standard Generalized Markup Language [8] (SGML) which is also a markup language. A structured document, including XML, has meta information of a certain concept and semantics by adding a pair of start and end tags. A start tag starts with a left angle bracket(<) and ends with right angle bracket (>).

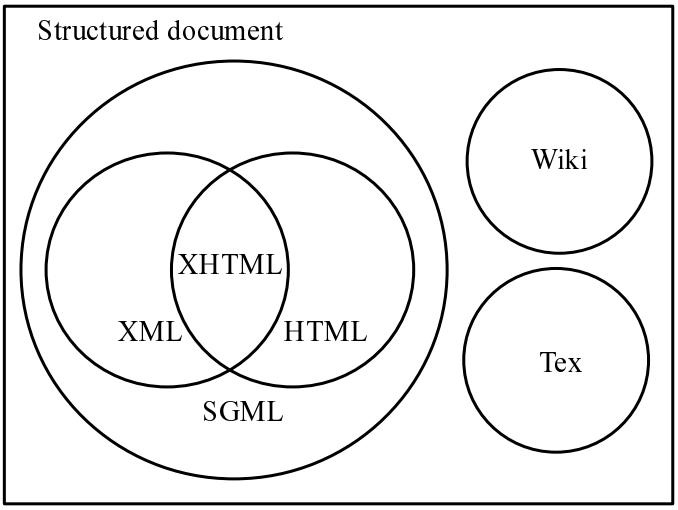


Figure 2.2. Structured documents

Meta information is annotated between the angle brackets. An end tag is similar to a start tag, except that an end tag has a slash (/) right after the left angle bracket. We show a concrete example representing a person' name as follows: <person_name>Atsushi KEYAKI</person_name>.

HyperText Markup Language [7] (HTML) is another popular and widely used markup language. HTML is defined to display decorated contents on a browser. HTML tags are classified into two groups, that is, tags representing document structure and tags representing emphasis [21]. To classify them, it is recommended to use Cascading Style Sheets [22] for the later tags.

Both XML and HTML are subsets of SGML. Particularly, HTML defined by XML is called as Extensible HyperText Markup Language [23] (XHTML). Basically, HTML need to be translated according to the definition of XML, although there are some rules in translating HTML into XHTML.

Likewise, as other example of structured documents, Wiki is content management system which is editable via a Web browser. Tex is also an example of structured documents. It is mainly used for publishing.

	data-centric XML	document-centric XML
shema	simple	complicated
structure	regular	irregular
text	short (word)	long (sentence)
content	homogenous	heterogenous
mixed content	little	much
sibling order	insignificant	significant

Table 2.1. Comparison between data-centric XML and document-centric XML

2.2.2 Targeted XML Documents

There are two types of XML documents: data-centric XML and document-centric [24]. Although there is no explicit boundary between them, some trends exist. Table 2.1 compares data-centric XML and document-centric XML, which is quoted from [25]. The most critical factor among the trends is that data-centric XML mainly contains single or compound words in its text nodes, while document-centric XML tends to contain one or more sentences in its text nodes. That causes a difference in searching strategy for each type of XML.

The other features of data-centric XML are that it simple schema, regular structure, and homogenous content. In addition, its content is a little mixed and sibling order is insignificant. One of the most typical examples of data-centric XML is DBLP [26] which is a Web site of computer science bibliography. Figure 2.3 depicts an example of data-centric XML. The XML document represents publication data, and each text node is defined by a context, that is, a path expression. A path expression is concatenated tags from the root to the node. Studies investigating data-centric XML primarily focus on searching query keywords efficiently. It is said that the users' information need for data-centric XML are relatively simple. That is the reason why a sub-tree is relevant for a query only if the sub-tree contains all query keywords in its descendant nodes. Thereby, search techniques for data-centric XML documents are called XML keyword search.

In contrast, document-centric XML has complicated shema, irregular structure, and heterogenous content. Also, its content is more mixed and sibling order is significant. Scientific articles and Wikipedia articles are document-centric XML

documents as shown in Figures 2.4-2.6. Existing studies toward document-centric XML are performed for effective XML element retrieval, because users' information need for document-centric XML are more complex and a search system needs to identify relevant descriptions from each document. Accordingly, search techniques for document-centric XML are called XML information (element) retrieval.

Since we focus more on proposing information retrieval techniques for structured documents rather than proposing efficient keyword search algorithms, our main targeted XML documents are document-centric XML documents.

2.2.3 Search Methods for XML documents

There are two ways for searching an XML document. One is specifying document structure, while the other is specifying a keyword in an XML document.

Many of the requests for data-centric XML are to extract a sub-tree of an XML document by a structural constraint. Thus, a document structure of an XML document need to be managed in an efficient way. There are wide research approaches such as structural joins [27, 28], indexing [29, 30, 31, 32, 33, 34, 35, 36], and node labeling [37, 38] for this purpose.

On the other hand, querying with a keyword is very important with document-centric XML as well as a structural constraint. In consequence, XML Path Language [39] (XPath) and XQuery [40] defined by W3C handle XML full-text search. XPath is a query language to extract a node from an XML document by specifying element name, attribute name, and string value (text). We show some XPath syntax in Table 2.2.

Whereas, XQuery, also a query language, is an expansion of XPath. It provides higher expressiveness with a FLWOR format query. Proposing efficient processing algorithms of XPath and XQuery is also active research area [41, 42, 43].

2.2.4 XML Element Retrieval

In this section, we describe the concepts of XML elements and queries in XML element retrieval.

XPath syntax	semantics
nodename	selecting all nodes with the name "nodename"
/	selecting from the root node
//	selecting nodes in the document from the current node that match the selection no matter where they are
//*nodename*	selecting all *nodename* elements no matter where they are in the document
.	selecting the current node
..	selecting the parent of the current node
@	selecting attributes
*	matching any element node

Table 2.2. Examples of XPath syntax

2.2.4.1 Comparison of Document Search and Element Search

Here, we explain the difference between XML element retrieval systems and well-used document retrieval systems. When many of document retrieval systems propose a list of relevant documents, the systems additionally provide users with result snippets [15], which are summaries of each document, approximately 50 words in length. Result snippets are generated by a text extraction technique that extracts the text that is nearby the query keywords. Search-system users utilize the result snippets located around the result when deciding which documents are worth browsing. Despite the fact that many search systems rely on result snippets, not all result snippets help them decide which documents to browse. This is because result snippets do not consider the context between the extracted text; as a consequence, some result snippets do not make sense [2]. Moreover, length of relevant descriptions differs from document to document. It suggests that not always result snippets are composed of all relevant descriptions without any irrelevant descriptions.

On the other hand, the main purpose of an XML element retrieval is to extract the relevant descriptions (elements) from a query and propose them in descending order of their relevancy scores. XML element retrieval systems can propose a list that contains the relevant descriptions from a query, while many document re-

trieval systems propose a list that contains relevant documents for a query. Hence, users do not have to spend time seeking out the relevant parts that satisfy their information need. This feature saves users' time and energy during information retrieval.

There are some other approaches to extract important descriptions in documents such as automatic summarization [44], information extraction [45], and passage retrieval [46].

Automatic summarization techniques are cultivated in the research field of natural language processing (NLP) to identify important descriptions in a document. It is reported that summarization accuracy with news articles is very high to the extent of achieving practical use level [47].

Information extraction techniques aim to extract specific structure (pattern) characters with a certain rule created manually or automatically. NLP techniques and data mining techniques are applied for accomplishing the end.

A passage retrieval technique is one of the information retrieval techniques focused on extracting only relevant descriptions from a document like XML element retrieval techniques. Information unit of a passage retrieval technique has some variations, e.g., HTML tags similar to XML element retrieval techniques, fixed width of window, and a bunch of descriptions describing the same topic. XML element retrieval techniques are different from passage retrieval techniques in some point, XML element retrieval techniques utilize a document structure in term calculation. Moreover, considering the idea of granularity of a document structure.

Compared with these approaches, one of the advantages of XML element retrieval is that search results are self-contained, which is one of the requirements for result snippets [15], because it considers the context of the sentences by utilizing document structures.

2.2.4.2 XML Element

We give specific examples in Figures 2.4–2.6 to define XML elements. Figure 2.4 illustrates XML documents. Each document is assigned a document identifier (DID). Figure 2.5 depicts trees abstracted from Figure 2.4. An XML document can be presented in a tree structure, which helps to understand the structure of the document. Each element is assigned an element identifier (EID), which is

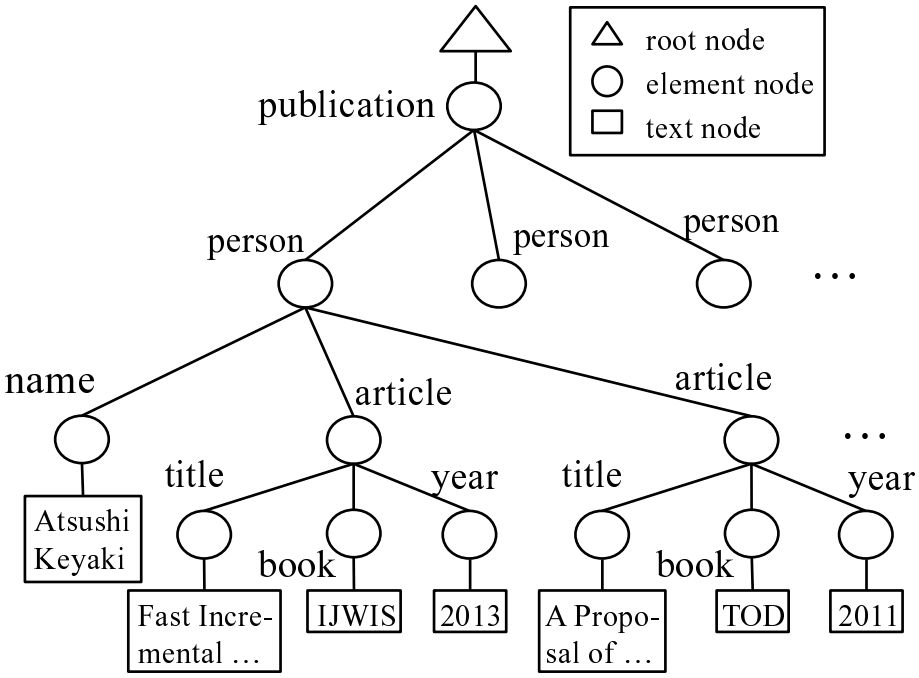


Figure 2.3. Example of data-centric XML document

assigned in document order. We can identify an element using its DID and EID.

A pair of start and end tags represents an XML element node within an XML tree, and the nested structure of XML elements represents ancestor–descendant relationships. Each element in Figure 2.6 is the text that comprises a set of text nodes within the XML tree in Figure 2.5. In this concrete example, the `article` node, which represents the entire document, has all of the text nodes as its descendant text nodes, while the `body` node also has its descendant text nodes. This demonstrates why there are overlapping XML elements in XML documents. In short, each XML element has an inclusion relation, i.e., an ancestor-descendant relationship. Moreover, an ancestor element of an element is called a larger element, and a descendant element of an element is called a smaller element. We describe the path expression (PE) of each element.

Authors add a structure to a document: e.g. chapters, sections, and paragraphs. We utilize these structures to identify the best material for satisfying the information need for users. Some structures are meaningless, so elements defined by those structures are inappropriate as search results, and some existing studies

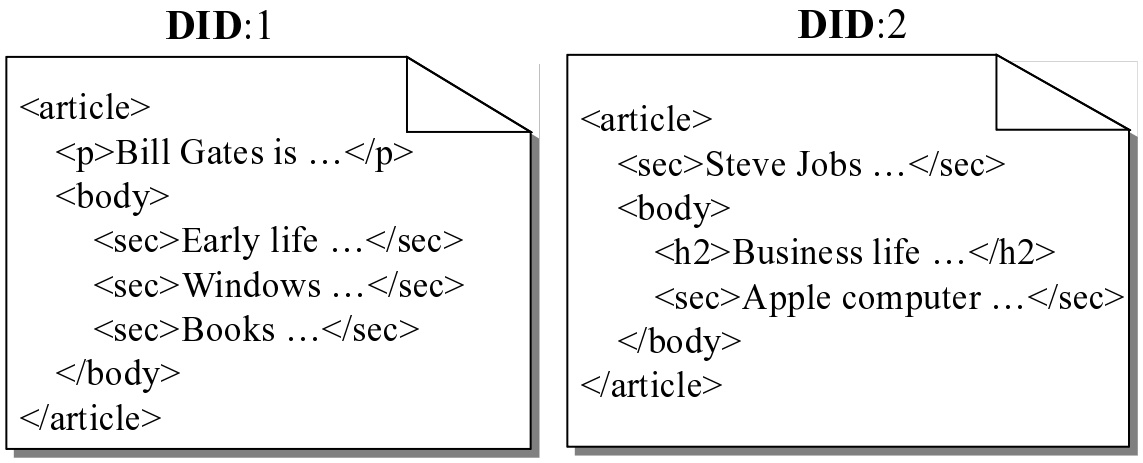


Figure 2.4. (Document-centric) XML document

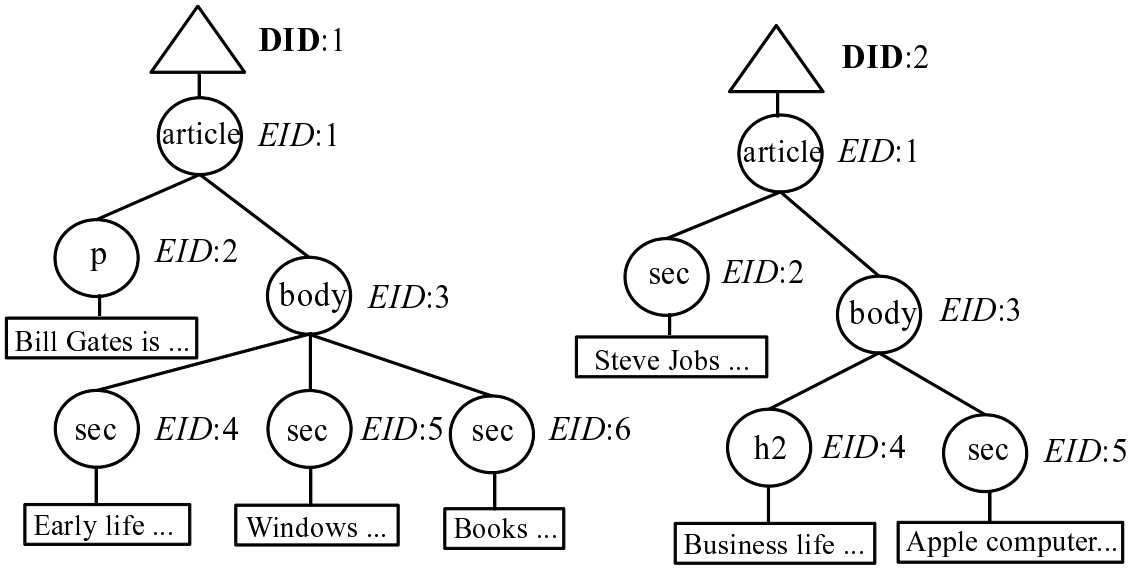


Figure 2.5. XML tree

[48], [49] include attempts to eliminate these. Suppose that a user seeks information from Document 1 about "Early life . . . ", "Windows . . . ", and "Books · · · ". XML element retrieval systems try to present the user Element 3 of Document 1, because that element contains all of the information that the user needs and no

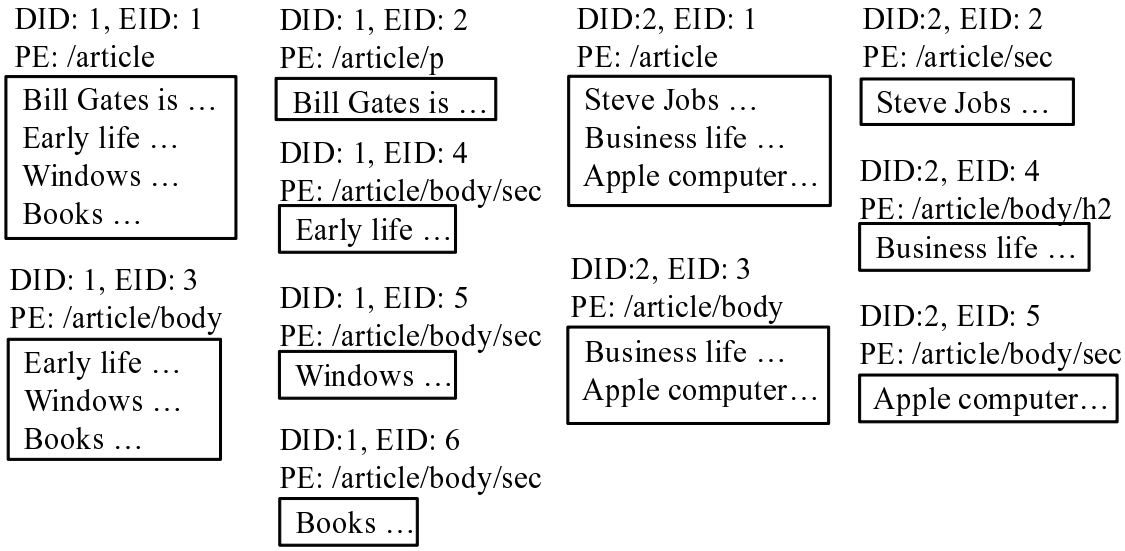


Figure 2.6. XML element

extra information.

2.2.4.3 History of XML Element Retrieval

Initiative for the Evaluation of XML Retrieval (INEX) project [50] launched in 2002, and is the largest ongoing project for XML element retrieval. Test collections provided by the INEX project are widely used for evaluating the effectiveness of the XML element retrieval systems. The project also carries out a competition using XML documents generated by the scientific articles and the Wikipedia articles[2], or document-centric XML.

The INEX project requires search systems to return a list ranked by their relevancy scores as search results. In addition, search systems extract 1,500 or fewer XML elements for each query. In the past, some existing studies do not remove overlapping elements and return a naïvely ranked list that is sorted in descending order of the XML elements' scores. We call such a list a *simple ranked list*. On the other hand, most studies have reported damage to search accuracy because of overlapping search results [51]: therefore, a ranked list without overlapping XML

[2]Wiki format articles are exchanged into XML format

elements is returned from recent XML element search systems. We call such a ranked list without overlaps *a non-overlapped ranked list*.

2.2.4.4 Ad Hoc Track of INEX

Ad hoc track has three kinds of tasks. These are the Focused task, the Relevant in Context task, and the Best in context task. Note that none of the tasks accepts overlapping elements in search results.

At first, in the focused task, relevant elements are to be extracted as much as a search system can. Unless search results do not contain overlapping elements, multiple elements can be extracted from a single document.

Secondly, the Relevant in Context task also allow a search system to return multiple elements without overlapping. A different point compared with the focused task is that elements need to be returned per a document in the Relevant in Context task.

Finally, the third task is the Best in Context task. In this task, the best entry point to start reading a document need to be specified. The evaluation measure considers the distance between the specified best entry point and the ground truth (manually judged) best entry point.

Since we believe that the most important aspect in satisfying users' information need is to show necessary and sufficient information as search results, we mainly focus on the Focused task.

2.2.4.5 INEX Test Collections for Ad Hoc Track

We performed some experiments using two test collections: the INEX 2008 test collection and the INEX 2010 test collection [52, 53].

The INEX test collections consist of three components: (1) the INEX document collection, (2) the INEX topics, and (3) the INEX relevance assessments. The INEX document collection is a Wikipedia XML corpus based on a snapshot of Wikipedia in English. The INEX 2008 test collection collected approximately 660,000 articles in early 2006, while the INEX 2010 test collection collected 2,660,000 articles in late 2008. Moreover, the INEX 2010 document set is composed of longer articles and are annotated additional semantic markups.

The topics of INEX 2008 and INEX 2010 include 68 queries and 115 queries,

respectively. We use all of the topics in the experiments, unless otherwise noted. Each query is represented as narrowed-extended XPath I [54] (NEXI). These topics (and assessment results) were created by INEX participants. In the topic creation process, the participants described narrative, title, NEXI, and description based on the same information need. The narrative explains not only what information is being sought, but also the context and motivation of the information need, i.e., why the information is being sought and what work-task it might help to solve. This is because precise recording of the narrative is important for scientific repeatability; there must exist, somewhere, a definitive description of relevant and irrelevant to the user. The title is NEXI without any structural constraints. The description remarks one or two sentence natural language definition of the information need. We show an example of a topic as follows:

```
<topic id="544" ct_no="6">
  <title>meaning of life</title>
  <castitle>
    //article[about(., philosophy)]//section[about(., meaning of
      life)]
  </castitle>
  <description>What is the meaning of life?</description>
  <narrative>I got bored of my life and started wondering what the
    meaning of life is. An element is relevant if it discusses the
    meaning of life from different perspectives, as long as it is
    serious. For example, Socrates discussing meaning of life is
    relevant, but something like ‘‘42’’ from H2G2 or ‘‘the meaning
    of life is cheese’’ from a comedy is irrelevant. An element must
    be self contained. An element that is a list of links is
    considered irrelevant because it is not self-contained in the
    sense that I don’t know in which context the links are given.
  </narrative>
</topic>
```

Generally speaking, search system users browse only the top results in the

search results [2]. This means that the most important challenge is to obtain high accuracy within the top-ranked XML elements. For this reason, the INEX project regards iP[.01], which means interpolated precision at recall level 1%, as the formal measure of the evaluation of a search system. The INEX project also uses mean average interpolated precision (MAiP). MAiP is an evaluation measurement that calculates the average of the (mean) interpolated precision at each recall level. The INEX assessment tool divides the recall levels into 101 levels.

Moreover, focused task in the ad hoc track, which tracks effective XML element retrieval, aims to identify the most appropriate granular element of a search result, because the track tries to reveal the capability of an XML element retrieval. Nevertheless, document retrieval has been actively studied in recent years, compared with element retrieval, in parts of the INEX project. This is because it is difficult to identify the most appropriate XML element, and it is relatively easier to attain more accurate search results by returning the entire document as an XML element, even though it contains some irrelevant parts. In fact, many of the top-ranked search systems in INEX official results are document search systems [52] over the years. However, one of our goals is to return the most appropriate XML element in one XML document in order to save users' time and effort. Therefore, we attempt to attain accurate XML element retrieval.

The INEX relevance assessments are evaluation tools for XML element retrieval. Using the INEX relevance assessments, an XML search system can be evaluated on the basis of some evaluation measures by inputting a ranked list into the evaluation tools. We use this in our experimental evaluation. Although the official measure for the focused task in an ad hoc track is iP[.01], we also show MAiP in order to reveal the overall effectiveness of our proposed method.

2.2.4.6 Queries Used in INEX

There are two ways of expressing information need into a query: through keywords and through document structure. A query entirely composed of query keywords is called a content-only (CO) query, whereas a query composed of pairs of query keywords and a constraint on the document structure is called a content-and-structure (CAS) query.

CO queries are used just as in traditional information retrieval for text documents. Users can submit CO queries even if they do not know the structures

of the documents that are retrieved. In contrast, CAS queries utilize one of the most significant features of structured documents: i.e. document structure. With a CAS query, a user can obtain specific results with regard to granularity and content.

We give a specific example of a CO query and a CAS query. These are expressed in the narrow extended XPath I (NEXI) [54] query language. A CO query `//*[about(., "Windows")]` means that the candidate search results are elements containing "Windows". Elements 1, 3, and 5 of Document 1 can be search results in Figure 2.6.

A CAS query: `//article[about(.,"Steve")]//sec [about(., "Apple")]` is more complex. Let us focus on the first half of the query, `//article[about(., "Steve")]`, which means that candidates for this part are elements that contain "Steve" and whose path expressions end with an `article` tag. The second half of the query, `//sec[about(., "Apple")]`, means that candidate search results are elements that contain "Apple" and whose path expressions end with a `sec` tag. The search results are elements that satisfy the latter constraint and whose ancestor elements satisfy the former constraint. The only element satisfying the query constraints is Element 5 of Document 1 in Figure 2.6.

2.3. One Click Access Task (1CLICK)

One Click Access task [55, 56] (1CLICK) is one of the tasks of NII Testbeds and Community for information access Research (NTCIR). The goal of 1CLICK task is to return single textual output which satisfy users' information need instead of a ranked result list. Moreover, submitted systems are expected to present important pieces of information first and to minimize the amount of text the user has to read. Although there are English task and Japanese task, we mainly discuss English task.

In the 1CLICK-1 task [55], four types of queries (CELEBRITY, LOCAL, DEFINITION, and QA) are selected by 1CLICK-1 organizers based on findings from a mobile query log study [57]. Whereas, eight types of queries (ARTIST, ACTOR, POLITICIAN, ATHLETE, FACILITY, GEO, DEFINITION, and QA) are used in 1CLICK-2 [56] because more fine-grained query classification is appropriate for representing users' information need. 1CLICK task supposes both

desktop search environment and mobile search environment. For each query, at most 1,000 characters can be extracted for desktop run and 280 characters for mobile run.

There are three source types as follows:

- **Mandatory**
  organizers provided Web search results and their page contents for each query. Participants may use only this information to generate search results.

- **Oracle**
  organizers provided a list of relevant pages for each query, which are a subset of the pages provided for Mandatory runs. Participants can use the data either wholly or partially to generate search results. If this data set is used in any way at all, the run is considered an Oracle run.

- **Open**
  participants may choose to search the live Web on their own to generate search results. Any run that does not use the oracle data but uses at least some privately-obtained Web search results is considered an Open run, even if it also uses the pages provided for Mandatory runs.

Note that participants can use external knowledge sources such as WordNet [58] or publicly available resources such as Wikipedia to summarize given page contents.

1CLICK-2 organizers provide 1) a set of documents, 2) a set of queries, and 3) nuggets (ground truth) as a test collection. Nuggets are generated by 1CLICK-2 organizers.

A set of queries are extracted from the query logs issued to the popular Web search system. Then, a set of documents are top-ranked 500 documents that the search system returns as search results. Moreover, nuggets are relevant text for a query. For example, part of nuggets for the query "michael jackson death" are "murray tried revive", "realized he needed to call help", and "prescribed drugs to Jackson". Concerned with the evaluation measure, submitted search results are evaluated with weighted recall, S–measure, T–measure, and S#–measure considering the match position.

Based on the discussion above, the goal is similar to that of XML element retrieval in terms of that both try to return only relevant descriptions for a query

as search results. On the other hand, there are some differences between them, namely, their targeted documents and their formats of search results. The targeted documents of 1CLICK task are HTML documents, while those of XML element retrieval are XML documents. In addition, search results of 1CLICK task are nuggets, whereas, those of XML element retrieval are elements. Consequently, 1CLICK task is largely information extraction task rather than automatic summarization task like XML element retrieval is. Furthermore, Since one of the guidelines for XML element retrieval is that a search result need to be self-contained, the search results for XML element retrieval are generally become larger than that of 1CLICK.

2.4. Database Systems

A number of kinds of database systems are used for research and business uses. These database systems are classified into some groups, for example, relational database [59] (RDB), XML database [60], Key-Value Store (KVS) database.

RDB has the largest share of users among database systems. RDB manages data as a two dimensional table structure where a set of data is arranged sideways and each set of data is arranged longways. A set of data, called a tuple, contains some attributes. Moreover, attributes enabling to identify a tuple are defined as primary keys. A query is represented in a SQL format. When a SQL query refers to two or more tables, each table is joined with others to return a retrieved relation. RDB has such an advantage that data can be easily imported from other sources such as comma separated values (CSV). As a disadvantage, it is relatively difficult to alter a table definition once it is defined. Prominent relational database management systems (RDBMS) are Oracle Database [61], IBM DB2 [62], Microsoft SQL Server [63]. Note that we can treat originally XML data easily and efficiently with XRel [64].

XML database specializes in storing XML data. Nowadays many applications employ XML as a data format. As a result, a huge number of XML data was generated. XML databases are developed and implemented for efficient management of these XML data. XML database has high expressiveness, and its data structure is flexible, which is a large merit of an XML database. Fast query processing of XPath and XQuery is required because an XML database stores XML data. Open

source product BaseX [65] and Toshiba TX1 [66] are outstanding XML databases. In addition, some RDBMS including ones previously mentioned have extension for handling XML data.

KVS attracts much attention recently because of its high affinity with a distributed system. The data structure of KVS is just a pair of a key and a value. There are some remarkable distributed KVS database systems, e.g., Google Bigtable [67] and Amazon Dynamo [68]. The storage method of versatile database software library BerkeleyDB [69] is also KVS. With satisfying high speed indexing of data and query processing, it can be used like RDBMS because a user can define a comparison function for data having the same key and it supports even multiple key option. Moreover, a function of secondary index reduces data overlaps in indices, which cause cut down of disk size and I/O cost.

CHAPTER 3

Related Studies

In this chapter, we explain effective and efficient XML retrieval. We also discuss studies focused on updates in search systems.

3.1. XML Element Retrieval

In this section, we explain related studies of XML element retrieval techniques. The main target of the techniques is document-centric XML documents such scientific articles and Wikipedia articles as shown in Figure 2.4.

3.1.1 Accurate XML Search

The most important goal of XML element retrieval is highly accurate searches. The mainstream approach to extracting relevant elements is as follows: first, calculate a term weight for each element by using a term-weighting scheme; next, compute a score for each element using these term weights.

Term-weighting schemes for XML element retrieval are often derived from studies on document retrieval. Both of these are composed of three types of factors: local weights that are statistics derived from each document (element); global weights that are statistics derived from all document in a document set;

and constant values (coefficients and parameters). Local weights and constant values are easy to calculate and refer to because local weights are computed for a newly inserted element. However, it is difficult to calculate global weights on demand because the entire document set must be scanned to compute these.

The most significant difference between document retrieval and XML element retrieval is the method for computing global weights. Term-weighting schemes in document retrieval assume that every document has the same attribute and belongs to the same class. Thus, global weights are calculated using all documents. However, in XML element retrieval, elements are assigned to classes. Global weights are calculated for elements of the same class. There are different ways to classify elements. One approach is to classify elements by path expression. In Figure 2.6, since Elements 4, 5, and 6 of Document 1, and Element 4 of Document 2 all have the same path expression `/article/body/sec`, the global weights are calculated using these elements.

Alternatively, elements with the same tag can be placed in the same class. Because Elements 4, 5, and 6 of Document 1 and Elements 2 and 4 of Document 2 all have the `sec` tag, the global weights are calculated using these elements, as depicted in Figure 2.6. We use classification based on path expression in our system, because this is reportedly more accurate [70].

There are several kinds of term-weighting schemes for XML element retrieval; e.g. TF-IPF [71, 72], BM25E [73], and the query likelihood model for XML element retrieval [74] (QLMER). BM25E is regarded as a more effective term-weighting scheme than TF-IPF. Actually, most of the top-ranked search systems at INEX use BM25E [5]. However, no exhaustive comparison between BM25E and QLMER has been explored. We therefore examine the potentials of these term-weighting schemes in this thesis.

TF-IPF is a path expression-based scoring method that is extended by the well-known TF-IDF [12, 72] approach for document search. A term weight of TF-IDF is the product of a term frequency (TF) in a document as a local weight and inverse document frequency (IDF) as a global weight, while that of TF-IPF is the product of TF and inverse path frequency (IPF). Concretely, let T be the set of query keywords, e be an XML element, p be a path expression of e. A term weight of term $t \in T$ in e of TF-IPF, $w_{tfipf}(e,t)$, and a score of e, $S_{tfipf}(e)$, is

calculated as:

$$w_{tfipf}(p,e,t) = \frac{tf_{e,t}}{el_e} \cdot ipf_{p,t}, \;\; ipf_{p,t} = 1 + \log \frac{N_p}{pf_{p,t}} \tag{3.1}$$

$$S_{tfipf}(e) = \sum_{t_i \in T} w_{tfipf}(p,e,t_i) \tag{3.2}$$

where $tf_{e,t}$ is the number of occurrences of t in e, el_e is the length of e (the total number of terms in e), N_p is the number of elements of which path expressions are p, and $pf_{p,t}$ is the number of elements containing t of which path expressions are p.

As one of variations of TF-IPF, TF-IAF [75] consider not only statistics of element but also statistics of query. In more detail, TF-IAF uses the inverse abbreviated-path frequency (IAF) as a global weight instead of IPF. Let qs be a structural constraint of a query. A term wight of term $t \in T$ in e of TF-IAF, $w_{tfiaf}(qs,t)$, and a score of e, $S_{tfiaf}(e)$, is calculated as follows:

$$w_{tfiaf}(qs,t) = qtf_t \cdot iaf_{qs,t}, \;\; iaf_{qs,t} = 1 + \log \frac{N'_{qs}}{apf_{qs,t}} \tag{3.3}$$

$$S_{tfiaf}(e) = \sum_{t_i \in T} w_{tfipf}(p,e,t_i) \cdot w_{tfiaf}(qs,t_i) \tag{3.4}$$

where $qtf(t)$ is the number of frequency t in T, $N'(qs)$ is the number of elements of which satisfy qs, and $apf(p,t)$ is the number of elements containing t of which path expressions satisfy qs. Note that hereafter we call TF-IAF as query structure scoring method (QS).

BM25E [73] is a probabilistic model. In term calculation of the classic term-weighting scheme TF-IPF, statistics on the occurrence frequencies of terms are utilized. Conversely, BM25E utilizes not only the statistics but also average element length of elements the same path expression. The term weight of BM25E, $w_{bm25e}(p,e,t)$, and a score of e, $S_{BM25E}(e)$, is calculated with the following:

$$w_{bm25e}(p,e,t) = \frac{(k_1+1)tf_{e,t}}{k_1((1-b)+b\frac{el_e}{avel_p})+tf_{e,t}} \cdot \log \frac{N_p - pf_{p,t} + 0.5}{pf_{p,t} + 0.5} \tag{3.5}$$

$$S_{bm25e}(e) = \sum_{t_i \in T} w_{bm25e}(p,e,t_i) \tag{3.6}$$

where $avel_p$ is the average length of the elements with p and k_1, b are given parameters.

There are some scoring functions based on the statistical language model for XML element retrieval including QLMER. In the term-weighting scheme of QLMER, the score of each element is the product of the occupancy probabilities of the query keywords as shown in Eq. (3.11). Thus, the zero probability problem also occurs in QLMER, and smoothing techniques are also adopted. Note that smoothing values are computed not with a document (element) model but with a background language model, which is applied for an entire document set.

There are two kinds of smoothing methods, i.e., linear interpolation and Dirichlet smoothing. It is reported that the latter mentioned smoothing can compute an element's relevancy score accurately even when the document length is short [16]. Therefore, we assume the Dirichlet smoothing is more effective in the context of XML element retrieval because many elements in it are relatively short. However, most studies select linear interpolation [76] [77] [78] [79], except for [80] which uses Dirichlet smoothing.

In addition, the background model for existing studies including [80] is commonly computed with all documents in a document set, although path expression-based approach is revealed to be more accurate for XML element retrieval.

In short, there has been no adequate studies which investigates the best setting for QLMER. Accordingly, we survey effectiveness of varying settings of smoothing methods in Section 4.6.1.2.

We show the way of term calculation with QLMER.

A probability that t is generated from e (i.e., a term weight of t in e), $\hat{P}_{mle}(t|M_e)$ ($w_{qlmer}(p, e, t)$), is calculated as:

$$w_{qlmer}(p, e, t) = \hat{P}_{mle}(t|M_e) = \frac{tf_{e,t}}{el_e} \tag{3.7}$$

To apply the smoothing techniques, Eq. (3.7) is transformed. Eq. (3.8) represents the linear interpolation and Eq. (3.9) represents the Dirichlet smoothing. Moreover, we denote two equations for computing background models in Eq. (3.10) which is collection-based background model and Eq. (3.11) which is path expression-based background model.

$$w_{qlmer}(p, e, t) = \omega \hat{P}_{mle}(t|M_e) + (1 - \omega)\hat{P}_{mle}(t|M_b) \tag{3.8}$$

$$w_{qlmer}(p, e, t) = \frac{\hat{P}_{mle}(t|M_e) + \mu \hat{P}_{mle}(t|M_b)}{el_e + \mu} \tag{3.9}$$

$$\hat{P}_{mle}(t|M_b) = \frac{\sum_{e \in C} tf_{e,t}}{\sum_{e \in C} el_e} \tag{3.10}$$

$$\hat{P}_{mle}(t,p|M_b) = \frac{\sum_{e \in p} tf_{e,t}}{\sum_{e \in p} el_e} \tag{3.11}$$

where ω and μ are given parameters, M_b is a background model, and C is a document set.

The following equation is two-stage smoothing method with the path expression-based background model.

$$w_{qlmer}(p,e,t) = \frac{\hat{P}_{mle}(t|M_e) + \mu(\omega \hat{P}_{mle}(t,p|M_b) + (1-\omega)\hat{P}_{mle}(t,p|M_b))}{el_e + \mu} \tag{3.12}$$

Finally, a score of e, $s_{qlmer}(p,e,t)$, is calculated with:

$$S_{qlmer}(e) = \hat{P}(T|M_e) = \prod_{t_i \in T} w_{qlmer}(p,e,t_i) \tag{3.13}$$

As another approach, machine learning techniques, particularly learning to rank algorithm [81], is also adapted to XML element retrieval [82, 83, 84]. The largest benefit of the learning to rank approach is its highly accurate search performance. In contrast, shortcomings are that some of the manual judgment data are required before learning process starts because learning to a rank technique is supervised approach, and appropriate features, which vary by each document set, used in learning process need to be selected. Additionally, every machine learning technique has the aspect of the learning process taking a relatively long time to estimate parameters.

Since it is very common to use a weight computed by a term-weighting scheme as a feature of machine learning process, improving effectiveness of a term-weighting scheme is still a quite important research topic. Moreover, our proposed system in this thesis requires fast term calculation because the proposed system handles document updates in real time as we discuss in Chapter 5 deeply. Therefore, scoring with a machine learning approach is not the most suitable choice for the proposed system.

On the other hand, network-analysis-based scoring functions are also not proper in this situation as well as machine learning approaches. Hyperlink information is utilized when calculating document score with network-analysis-based scoring functions. However, hyperlinks from other documents to a newly inserted

document is not generated just after the document insertion. This means an accurate score of the documents cannot be computed in real time, which is a fatal problem for a search system considering document updates sensitively. Moreover, an effective way for element-granule search with network-analysis-based approach has not been proposed yet.

According to the discussion above, we adopt a term-weighting-based scoring method for the proposed system. Note that integrating a term-weighting-based approach with a machine learning approach or network-analysis-based approach may bring more accurate search performance, and it remains as one direction of future work.

3.1.2 Fast XML Search

Although the most important requirement of XML element retrieval is enabling effective searches, fast query processing is also required by a system user.

To attain efficient XML element retrieval, various approaches have been taken, such as:

- applying top-k algorithms to return search results quickly, and
- compressing and reducing data to suppress the index size to minimize the amount of data scanned in query processing.

Many top-k searches have been proposed [85]. There are two conditions for efficient query processing: 1) term weights are calculated before query processing begins, and 2) terms are sorted in descending order of weight. This means that we only need to scan highly ranked terms in query processing. Note that some query processing methods also utilize an index for a random scan, which is used to refer to the weight of an arbitrary term in any element.

Some studies [86], [87] have used term-weighting schemes [73] for effective searches. Theobald et al. proposed two types of indices and a top-k algorithm for efficient searches [86]. One type is for scoring an element in query processing, and the other type is for checking a structural constraint on a query. They also proposed cost-based query processing, which identifies an effective moment to check the structural constraints and determines which query keyword is reasonable to process.

Trotman et al. proposed a low-cost method of data compression and selection [87].

In these studies, the aim was to retrieve elements satisfying users' information need by retrieving elements from a fixed document set; i.e. document updates were not considered.

Kikori [88] provides not only an efficient and effective search, but also a user interface. Kikori is based on the XRel [64] which splits an XML document into five tables, `document`, `path`, `element`, `term` (`token`), and `attribute` tables, to store XML document into RDBMS. The interface of the Kikori displays search results in many ways which correspond to each task of the Ad hoc track of INEX denoted in Section 2.2.4.4.

3.1.3 Data Cleansing Techniques

Information of document structure is useful to reveal the best description for users' information need. However, some structures are meaningless and elements defined by these structures are inappropriate as search results. Thus, some existing studies try to eliminate them [48], [49].

In fact, removing useless or low-scored elements is effective for accurate XML retrieval. Although every granularity of XML elements should be treated as search targets, effectiveness of the results decreases sharply if a search system returns uninformative XML elements. Extremely small XML elements are often not suitable for search results; Hatano et al. noted that when such meaningless XML elements are removed, the search accuracy improves [89]. Furthermore, our previous study suggests that large XML elements are also inappropriate for search results [90].

3.1.4 Important Sentence Extraction

We will explain a study of important sentence extraction. Since identifying relevant descriptions from a document is similar to extracting an important sentence from the document, knowledge from studies focused on important sentence extraction is expected to be useful for achieving even our goal.

In the past, many studies have focused on important sentence extraction. One application of the technique is a result snippet. Takami et al. generate result snippets [91] with using natural language processing techniques to extract

important sentences from a document. The direction of their and our studies are the same because both try to utilize the knowledge of important sentence extraction for attaining accurate information retrieval.

There are three requirements for results snippets [15], that are,

- **Requirement 1**
  maximally informative about the query,
- **Requirement 2**
  self-contained enough to be easy to read, and
- **Requirement 3**
  short enough to fit within the normally strict constraints on the space available for summaries.

Accordingly, we propose a scoring method which assigns higher score to elements satisfying these three requirements.

3.2. XML Keyword Search

In this section, we explain related studies of XML keyword search techniques. The main target of the techniques is data-centric XML documents such DBLP as shown in Figure 2.3.

Although we are primarily interested in search techniques for document-centric XML documents, some studies for data-centric XML documents are related to our research.

Data-centric XML documents generally describe only one term in their text nodes. Therefore, studies investigating data-centric XML primarily focus on searching query keywords. The existing research efforts to attain efficient XML element searches usually utilize the lowest common ancestor (LCA) approach [92]. As a part of this approach, the LCA itself may originate as the top-level common ancestor of arbitrary nodes in an XML tree; however, it is generally defined as the deepest node containing all of the query keywords in its descendants. Research involving LCA and XML elements [93, 94, 95, 96, 97, 98, 99] shows significant results related to efficient XML keyword search; however, such techniques do not

perform well in the context of accurate XML searches. In other words, the retrieval accuracy of XML search systems decreases when we use LCAs as the most appropriate XML elements for a given query [100].

To address this problem, research efforts have also tried to identify and extract more relevant XML elements from the sub-tree whose root node is an LCA. XSeek [100] is one such solution that produces a meaningful LCA (MLCA), which classifies and analyzes XML tags by using XML schema and the positions of the query keywords. In the case of XSeek, the nodes related to a query are selected and extracted in the order of their relevance to a query. Another approach, eXtract [101], is an expansion of MLCA and infers a user's search purpose by analyzing queries. In eXtract, queries are classified into two cases: (1) extracting an explicit search target, and (2) extracting the neighbors of the search target.

From the results of the studies, identifying the most appropriate XML element is very difficult, yet important, for XML element search. The purpose of the approaches in these studies is similar to ours to return the best results. However, our method returns not only the LCA, but also any node, because we do not think that the LCA condition is enough for the most appropriate XML element. Moreover, we do not need to utilize foreign information such as XML schema, because we try to identify the appropriate element by using the relationships between the elements.

Tanabe et al. [102] argued that a single XML document generally contains both data-centric content and document-centric content. That is the motivation why they focus on integrating XML keyword search techniques for data-centric XML documents and information retrieval techniques for document-centric XML documents to gain useful search results.

3.3. Dynamic Updates of Data

The handling of document updates is especially important in Web search systems because documents are constantly inserted, deleted, and modified. When documents are updated, useful search systems should treat them as search targets immediately. If systems present search results based on a past snapshot of the Web, the content of the Web documents may since have changed. Search systems should reflect the current state of the Web and handle dynamically changing Web

documents.

Recently, some techniques for handling document updates have been proposed. Chen et al. [103] tackled this challenge in the field of information extraction. They reported that a long processing time is required to apply information extraction techniques to document collections when document updates occur. As a result, a delay occurs before information extracted from the updated documents is available. To shorten the delay, they proposed a method for recycling the intermediate results of past snapshots. Neumann et al. [104] also effectively utilized the information of past snapshots, but with Resource Description Framework (RDF) data.

Ren et al. [105] preserved not only the latest graph data but also past snapshots to trace the transition of the graph. Our study is different from theirs to the extent that we present the information along with the latest state of the Web.

The aforementioned studies utilized the intermediate results of past snapshots. Hence, we also utilize those or existing indices. We incrementally update existing indices when new documents are inserted. In addition, Web search systems are expected to maintain high performance with a low update cost. In the case of text retrieval, high search accuracy should also be maintained.

There has been no adequate study focused on incremental updates in XML element retrieval with effective and efficient query processing. Therefore, this is the first study to tackle the problem. Although some researches have focused on incremental updates of an inverted index [106], [107], [108], they proposed index data structures of indices and physical storage methods. Our study differ from their studies because we introduce a function for incremental updates of indices for several purposes in XML element retrieval by proposing an efficient method of data management.

3.4. HTML Documents Content Comprehension and Classification

A study [21] reported that tags in structured documents are largely classified into two groups; 1) tags surrounding self-contained content and 2) tags enabling separate content. In this thesis, we define tags of 1) as structural tags. Concrete

examples of structural tags are `HEAD`, `BODY`, and `P` tags of HTML. These tags are quite commonly used and can be meaningful clues for identifying useful and appropriate granular elements.

In addition, some HTML tags defined in HTML5 [109] such as `ARTICLE`, `SECTION`, `NAV`, and `ASIDE` tags are also structural tags. `SECTION` tag can be used nested while each of the other tags represent specific context or semantic. This means that a physical document structure agreeing to a logical structure of the document can be generated with these tags. However, we propose a method which works well even when we cannot utilize these newly defined tags, because these tags has not widely used yet.

In contrast, tags of 2) are used for representing decoration, attribute, and specific idea. To enumerate some examples, `B`, `FONT`, `I` tags are applicable. Most of HTML tags are classified into 2), because HTML is defined for the sake of being used for displaying with a browser. The fact that the number of tags of 1) is small suggests that it is more difficult to identify the most appropriate granular elements.

There are some kind of tags which perform a boundary between one topic and another [110]. To show some tangible examples, these tags are Heading tags (`H1`–`H6` tags), `HR` tag, and `BR` tag. These tags are leveraged to split content according to a topic.

A classification approach in separating HTML content based on actual appearance also exists [111]. A merit of the approach is that parts which are originally dissimilar in terms of document structure can be gathered into the same group if these parts are similar enough in regards to the appearance on a browser.

On the other hands, a study [112] propose a method to extract main–content from news articles based on un-supervised learning. The key idea of the study is that the blocks of sub–content are the same or strongly similar to each other, while the blocks of main–content are different from each other.

CHAPTER 4

Accurate XML Element Retrieval Beyond Traditional Term-Weighting Schemes

This chapter describes our proposed methods for accurate XML element retrieval.

Many of existing studies mainly focus on the precise calculation of term weights. The traditional approach is that term-weighting schemes for document retrieval are extended to XML element retrieval. However, there is a gap between document retrieval and XML element retrieval in that relevant descriptions need to be identified in XML element retrieval. Therefore, we consider the importance of not only term perspective but also element perspective while considering the requirement for the important sentence extraction in Section 4.1.

Furthermore, these existing studies did not consider the approach for returning search results, and only a few studies discuss an overlap between elements in an XML document [51]. In these studies, some XML search systems simply extract the element with the highest score. This means that the other elements are unconditionally discarded from the search result candidates. These kinds of systems do not always return a useful search result, because these elements not being considered can cause element granularity to be extremely large or extremely small. An extremely small element does not make sense in isolation, even though search results contain relevant descriptions, while extremely large elements many

contain irrelevant descriptions, too. Therefore, we need to judge which XML elements are most appropriate. For this purpose, we consider the text size of the elements and their inclusion relations. More detail is discussed in Section 4.2.

Moreover, existing scoring functions for XML element retrieval calculate a relevant score of each element independently. However, we believe that elements belonging to the same document have some sort of relationship. Accordingly, we hypothesize that a useful element is identified by using statistics of related elements. A scoring method leveraging statistics of ancestor and descendant elements is explained in Section 4.3.

Section 4.4 describes our approach for integrating the proposed methods discussed in Sections 4.1 to 4.3. Furthermore, 4.5 describes the implementation of the proposed system.

In Section 4.6, we conduct some experimental evaluations to measure the effectiveness of the proposed methods.

4.1. A Scoring Method Considering Requirements for Result Snippets

Based on the three requirements for result snippets denoted in Section 3.1.4, we define three challenges in order to adapt the requirements to XML element retrieval. They are as follows:

- **Challenge 1**
  even though an element contains many of query keywords, the element is not appropriate when only a specific kind of query keywords is occurring,
- **Challenge 2**
  an element contains many of irrelevant descriptions for a query, the element is not appropriate, and
- **Challenge 3**
  an extremely large element is not appropriate.

Concerned with the Challenge 1, we consider the number of distinct query keywords. Because query keywords may have numerous meanings, it is often

difficult to identify a proper one. One solution is to consider the co-occurrence of query keywords. If an XML element contains several distinct query keywords in its text nodes, we can assume that the XML element is closely related to the meaning of the given query keywords. Thus, we propose a query keyword scoring method (QK) as follows:

$$S_{QK}(e) = \frac{dqk_e}{Q} \tag{4.1}$$

Where $S_{QK}(e)$ is a QK score of element e, dqk_e is the number of the distinct query keywords in e, and Q is the total number of the query keywords of a query.

In terms of the Challenge 2, the ratio of relevant descriptions for a query to irrelevant descriptions needs to be considered. Then, to handle the Challenge 3, the text size (element length) of each element needs to be considered. We regard these challenges are solved with existing XML element retrieval techniques, because some of them normalize with an element length, while others penalize extremely large and small elements. Above all existing XML element retrieval techniques, QS is the most appropriate one. The scoring method not only solve these challenges, but also satisfy the Requirement 1 that maximally informative about the query.

We finally combine QK and QS to consider the requirement more strictly. In order to combine these two values, a mathematical function is usually used. There are many mathematical functions which can combine multiple values [113]; however, we try to use the function of their product which is the simplest one in the same spirit of traditional document retrieval systems. Therefore, a score of query-oriented scoring method (QO) is calculated as follows:

$$S_{QO}(e) = S_{QK}(e) \cdot S_{QS}(e) \tag{4.2}$$

4.2. Result Reconstruction Method

Search systems should return the smallest-possible XML elements for the given query; however, recently, document search has been regarded as more effective than XML element search (as demonstrated by INEX). We believe that XML element search enables users to save time and energy in their information retrieval tasks and could be very convenient. Therefore, we aim to propose a method to find XML element search results whose length is appropriate.

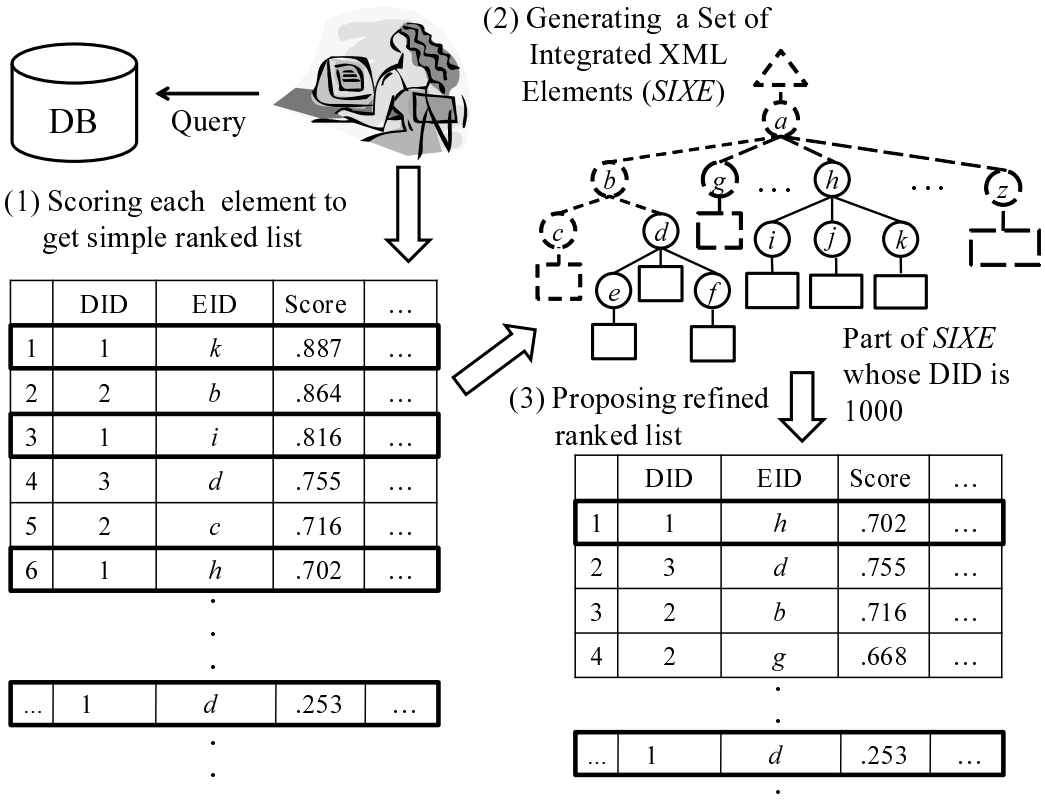


Figure 4.1. Overview of our proposed method

In our approach, we show a *refined ranked list*, which is composed of the relevant and informative XML elements. As shown in Figure 4.1, an overview of our method is as follows:

(1) We first score each XML element by using a scoring method to obtain a simple ranked list.

(2) We extract the XML elements from a simple ranked list and generate a set of relevant XML elements by considering the limitations of the XML element length and the reconstruction of the XML elements.

(3) To present a refined ranked list, we rank the XML elements extracted from Step (2); we re-rank and remove the XML elements in the simple ranked list and incorporate them into the refined ranked list.

Hereafter, we denote the set of relevant XML elements generated in Step (2) as the *Set of Integrated XML Elements* (*SIXE*).

4.2.1 Selecting an Effective Scoring Method

Our method starts with a simple ranked list. Therefore, the search accuracy of the proposed method depends on the quality of the given list. Selecting a suitable scoring method for an XML element search is the key to acquiring a reliable simple ranked list.

We assume the following two conditions as follows:

- a simple ranked list is highly accurate in terms of its MAiP, and
- a simple ranked list contains XML elements of varying lengths.

With regard to the first condition, a simple ranked list with high MAiP contains many relevant elements, regardless of whether or not it is ranked in the appropriate order. Therefore, we need to utilize a method with high MAiP.

With regard to the second condition, we want a simple ranked list that is composed of elements of varying lengths, i.e., from small-length elements to large-length elements, because the most appropriate parts in the XML document are not specific granular elements. In other words, the appropriate granularity differs from document to document. Besides, we cannot extract the appropriate granular elements if a simple ranked list contains only specific granular elements. Thus, we assume that a simple ranked list should contain XML elements of varying lengths.

To examine which list satisfies the condition, we utilize the notion that there is a relationship between the granularity and the text size of an element. Note that we consider the ratio of the text size of an element to the granularity of the element. In this case, the standard deviation becomes large if a simple ranked list contains XML elements of varying lengths. From the above discussion, we will explore the standard deviation for the second condition.

While we try to generate a simple ranked list that contains XML elements of varying lengths, Kamps et al. noted that large-sized XML elements tend to be more effective [52] compared with middle-sized or small-sized ones; however, problems can occur while extracting such large-sized elements. Consider an XML document in which several relevant elements exist. Using the tree structure shown

in Figure 4.1, we consider the sub-trees with the root nodes d and h to be the relevant elements. When we extract a sub-tree that contains all of the relevant elements, it might also include numerous irrelevant elements. For example, if we choose a sub-tree with root node a, it contains numerous irrelevant elements, including c and g. If the scoring method used gives a higher score to large-sized XML elements, result a may rank higher than d and h. To solve this problem, we should extract multiple relevant XML elements from the XML document. In particular, we need to extract differently sized XML elements in order to determine the appropriate granularity of XML elements. Given this approach, we focus on MAiP and the length of the retrieved XML elements. We performed a preliminary experiment, as described in Section 4.6.1.2, to verify whether or not our assumption is true.

4.2.2 Generating a Set of Integrated XML Elements

To generate an *SIXE* which are composed of appropriate granular elements as search results, we extract the relevant XML elements from the simple ranked list generated in Step (1) of our method. As discussed in Section 4.2.1, large XML elements might contain irrelevant descriptions and decrease search accuracy.

One baseline approach for generating a non-overlapped ranked list of XML elements is to repeatedly extract the XML elements from a simple ranked list in descending order of their rank, unless an overlap occurs. The overlapped XML elements are simply discarded and ignored.

This operation continues as long as either a candidate of the XML elements remains in the search results or the number of extracted XML elements reaches a predefined upper limit[1].

On the other hand, we aim to reconfigure XML elements in a simple ranked list to produce results that are better than those of the established baseline. As noted in Section 3.1.1, we should consider how to identify and extract XML elements of appropriate lengths in order to attain a more accurate XML element search. We should also consider how to handle overlapping results, which we ignored in the baseline approach.

From the above discussion, we derive the following requirements:

[1]In our experimentation, we extracted 1,500 or fewer XML elements for each query.

- **Requirement 1**
  Because traditional search results include several large XML elements, we should impose an extraction limit on the element size.

- **Requirement 2**
  The extracted XML elements are appropriately abbreviated and reconstructed to resolve the overlap problem.

4.2.2.1 Extraction Limit

To satisfy Requirement 1, we need to limit the size of the extracted XML elements to an *extraction limit* (EL). Large-sized elements tend to contain more kinds of contents, which means that such elements may have parts that are irrelevant to users' information need. Thus, we suppose that the text size of the relevant parts in an XML document is lesser than a certain value. We limit the text size of the extracted XML elements for each XML document.

In our previous study, it turns out that it is difficult to identify the most appropriate granular elements only with document structure information because document structures of the same length elements are quite different with each other [114]. For this reason, we leverage not granular but text size of elements in finding the most appropriate granular elements. Eventually, we extract multiple relevant XML elements from an XML document, as long as their sizes are properly restricted.

To set the limit, we consider two approaches: (1) the value of the limit is dependent on the text size of each document, and (2) the value of the limit is independent of the text size of each document. In approach (1), the document-size-dependent condition, we assume that the relevant parts can be represented as a certain ratio of each XML document size. Accordingly, we limit the extracted text size for each XML document by defining EL of an XML document D as follows:

$$EL_D = \alpha \cdot |D| \tag{4.3}$$

where $|D|$ is the size of the XML document D, and α ($0 \leq \alpha \leq 1$) is the ratio of the size of the relevant element.

In approach (2), the document-size-independent condition, there may be XML documents of varying lengths, i.e., there are both large-sized documents and small-

sized documents. In this situation, EL derived from approach (1) could be too small a threshold for informative search results if there are extremely small-sized XML documents. Hence, we set EL as a constant value in approach (2) . We define EL as follows:

$$EL_D = \beta \tag{4.4}$$

where β is the parameter of the text size, which is the maximum number of relevant parts in an XML document.

Given this definition, we extract the XML elements from a simple ranked list when the text size of the XML elements in the *SIXE* is less than EL. This process repeats until the size exceeds EL.

In a report of users' behavior about Web browsing [115], the average time of browsing a Web page is less than a minute. In addition, the average text reading speed in English is approximately 1,000 to 1,200 characters per minute [56]. Thus, we hypothesis that users' tolerance for reading each result, that is, the best parameter for EL, is at most 1,200.

In this thesis, we explore the best parameter for EL in Section 4.6.1.3. However, the best parameter may not be able to be explored in practical situation. Thus, we set 1,200 as default value for EL in case the parameter cannot be tuned preliminary. Note that the parameter can be tuned with R–precision measured by manual judgment, even though there is no ground truth.

4.2.2.2 Reconstruction of Elements

To satisfy Requirement 2, we need to arrange the extracted XML elements such that the *SIXE* contains useful search results. To generate a non-overlapped ranked list for the baseline approach, we simply eliminate the overlapping XML elements. This may prevent us from extracting the relevant XML elements. For example, in Figure 4.2, we assume that the XML element rooted at node c is the most relevant one in the tree; however, we cannot extract c if we have already extracted d.

To address this problem, we search for larger XML elements and overwrite them. As a result, these relevant elements are all contained in the *SIXE*, while the existing approaches extract the elements with the higher score. As these overwrite operations are applied, the XML element lengths in the *SIXE* increase; therefore, the overwrite operation is executed only when Requirement 1 is satisfied.

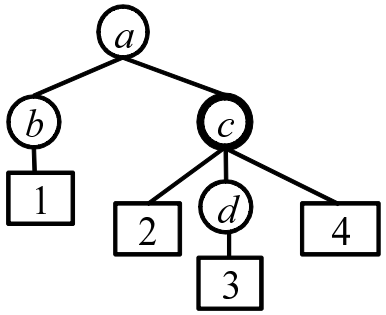


Figure 4.2. Example of *overwrite* elements

	NodeID	Score
1	*d*	high
2	*b*	middle
3	*c*	middle
4	*e*	low
5	*a*	low

Re-score *c* with *d*

	NodeID	Score
~~1~~	~~*d*~~	~~high~~
2	*c*	**upper middle**
3	*b*	middle
4	*e*	low
5	*a*	low

Figure 4.3. Overview of Bottom-Up scoring

Again, consider Figure 4.2. Suppose that c has been extracted. If a is extracted, a overwrites c, because it is larger than c. Furthermore, we discard d, because it is smaller than c.

4.2.3 Generating a Refined Ranked List

After generating an *SIXE*, we compute a score to each XML element in the *SIXE* and finally generate a refined ranked list. The simplest way to score these XML elements is to use the initial score, which is used in Step (1) of our method, although it is possible to use other scoring functions.

4.3. A Scoring Method with Statistics of Related Elements

In this section, we try to find out more informative elements for achieving high accuracy especially at lower recall level (top-rank search results).

As a result of SIXE proposed in the previous section, we obtain a set of elements which are composed of elements of appropriate granularity as search results. We then try to rank them in an effective way.

The simplest way to generate a refined ranked list is just to use the initial score, as adopted in Section 4.2.2.2. However, such an approach that treats each element as independent search target is not different from existing approaches. We reconstruct search results while considering overlapping relations among elements, since we belive it leads to better search performance. Thus, we hypothesize that leveraging statistics of related elements is useful to identity more informative elements.

In the following subsections, we propose two scoring methods for generating a refined ranked list: (1) *bottom-up* scoring, which utilizes the statistics of the descendant XML elements in order to score an ancestor XML element, and (2) *top-down* scoring, which utilizes the statistics of an ancestor XML element in order to score the descendant XML elements.

4.3.1 Bottom-Up Scoring

The *overwrite* operation introduced in Section 4.2.2.2 is executed when overlaps exist in the *SIXE*. In the overwrite operation, an ancestor XML element is extracted in place of its descendants. The ancestor should be ranked lower in a simple ranked list as compared to its descendants, implying that the refined ranked list also treats the ancestor XML element as a lower rank if the initial scores are used. In other words, a descendant element that was originally proposed earlier can be proposed later as a part of the ancestor element. We show a concrete example in Figures 4.2 and 4.3. When we generate a non-overlapped ranked list with the proposed approach, we first extract d and b. Next, we extract c and remove d from the list because of overwriting. As a consequence, the text nodes in c are proposed as a search result after the text nodes in b are proposed.

Regarding the text node in d, it should be proposed before the text node in b is proposed. However, the proposed order of these text nodes is reversed as a result of overwriting. We think that this may damage the search accuracy. Therefore, we define a scoring method in which an element has elements with a higher score among its descendants as a bottom-up scoring method (BU).

When we calculate the BU score of an XML element, we should consider the statistics of its descendant elements. Conversely, it is not appropriate that XML elements with low scores are ranked high. Therefore, we must integrate these initial scores properly. To re-score a descendant element with the statistics of its ancestor element, we consider two approaches: (1) integrating the scores of the ancestor element and the descendant element by a constant fraction, and (2) integrating the score by a ratio of the lengths of the ancestor XML element and its descendants. Furthermore, we also consider an approach which is a mixture of the former approaches, that is, (3) integrating the scores by a constant ratio after they are re-scored with a ratio of the length of the two elements.

Here, we discuss how to calculate BU by (1), a constant ratio. Let f_a be an ancestor XML element and f_d be the descendant element with the highest score. We define the bottom-up (BU) scoring function as:

$$s_{BU}(f_a) = \gamma \cdot s(f_d) + (1 - \gamma) \cdot s(f_a) \tag{4.5}$$

where $\gamma (0 \leq \gamma \leq 1)$ is the ratio of the effects on an ancestor element.

Next, we denote the approach for (2). Portions of the text nodes in an ancestor XML element are composed of descendant XML elements; the scores of the text nodes in these descendants affect those in the ancestors. Therefore, the BU score should be calculated using the initial score, as well as the ratio of the lengths of the ancestor XML element and that of its descendants:

$$s_{BU}(f_a) = \frac{1}{2}\frac{|f_d|}{|f_a|} \cdot s(f_d) + \frac{1}{2}\frac{|f_a| - |f_d|}{|f_a|} \cdot s(f_a) \tag{4.6}$$

where $|f_d|$ is the length of f_d, f_a is the length of f_a, $s_{BU}(f_a)$ is the bottom-up score of f_a, $s(f_a)$ is the initial score of f_a, and $s(f_d)$ is the initial score of f_d. Note that both initial scores are divided by two to adjust the weight.

Finally, we present an equation to calculate the BU score with approach (3), which is a mixture of approaches (1) and (2):

$$s_{BU}(f_a) = \gamma\frac{|f_d|}{|f_a|} \cdot s(f_d) + (1 - \gamma)\frac{|f_a| - |f_d|}{|f_a|} \cdot s(f_a) \tag{4.7}$$

Note that Equation 4.6 is equal to Equation 4.7 when $\gamma = 0.5$. We did some experiments to explore the best approach and its parameter of γ in Section 4.6.1.4.

4.3.2 Top-Down Scoring

As we discussed, the number of the distinct query keywords identify informative elements. On the other hand, the larger XML elements contain more query keywords, indicating that larger XML elements tend to be ranked higher. In other words, we might overlook smaller XML elements, even if they are informative. To cope with this problem, we propose a scoring method that is independent of the XML element size.

If an XML element contain numerous distinct query keywords, the element is identified as informative one. We suppose that a descendant element of the informative element is also informative because the descendant element takes over features of ancestor elements. Therefore, we consider top-down scoring method (*TD*) by calculating the ratio of the number of distinct query keywords contained in an XML element to that of its top-level ancestor, i.e., the entire document.

Let f be a scored XML element and D_f be an XML document associated with f. We define the *top-down* (*TD*) scoring function as

$$s_{TD}(f) = s(f) \cdot count(D_f) \tag{4.8}$$

where $s(f)$ is the initial score of f, and $count(D_f)$ is the number of distinct query keywords in D_f.

The two methods *BU* and *TD* can be integrated. We call this mixture-scoring method *BU-TD*. We calculate the *BU* score first and then re-score with *TD*, because the *BU* method occurs with overwriting.

4.4. Integrated Use of Scoring Functions

In this section, we explain integrated use of scoring functions proposed in Sections 4.1, 4.2, and 4.2.3.

4.4.1 Integration Procedure of Each Scoring Function

We proposed four approaches of scoring functions:

1. *QO* (*QS*·*QK*)
2. *SIXE*
3. *BU*
4. *TD*

Note that since *QO* is the product of *QS* and *QK*, there are five distinct scoring functions in total.

Some combinations of two or more scoring functions can be integrated because some of the scoring functions are independent from each other. In addition, the integration result changes with the applied order of scoring functions even though the combination of the scoring functions is the same. If all permutations are possible, there are many variations of integrated use. However, in reality possible integrated methods are restricted because of limitations regarding the combination constraint and applied order constraint.

To enumerate constraints, for example, *BU* need to be used with *SIXE* at the same time because *BU* is the scoring function executed during overlap operation of *SIXE*. As another example, since *QS* is applied in term calculation, there are only two options about *QS*, namely, applying or not. In other words, there is no option about the applied order. Meanwhile, since *TD* is based on *QS*, it is not natural that both scoring functions are applied in a variation of integrated methods. Based on these constraints, there are 19 kinds of integrated methods as follows.

- **Two scoring functions**
  QS-QK (*QO*), *QS-SIXE*, *QS-TD*, *QK-SIXE*, *SIXE-QK*, *SIXE-BU*, *SIXE-TD*, *TD-SIXE*
- **Three scoring functions**
  QS-QK-SIXE (*QO-SIXE*), *QS-SIXE-BU*, *QS-TD-SIXE*, *QK-SIXE-BU*, *SIXE-BU-QK*, *SIXE-BU-TD*, *TD-SIXE-BU*
- **Four scoring functions**
  QS-QK-SIXE-BU (*QO-SIXE-BU*), *QS-SIXE-BU-QK*, *QS-SIXE-BU-TD*, *QS-TD-SIXE-BU*

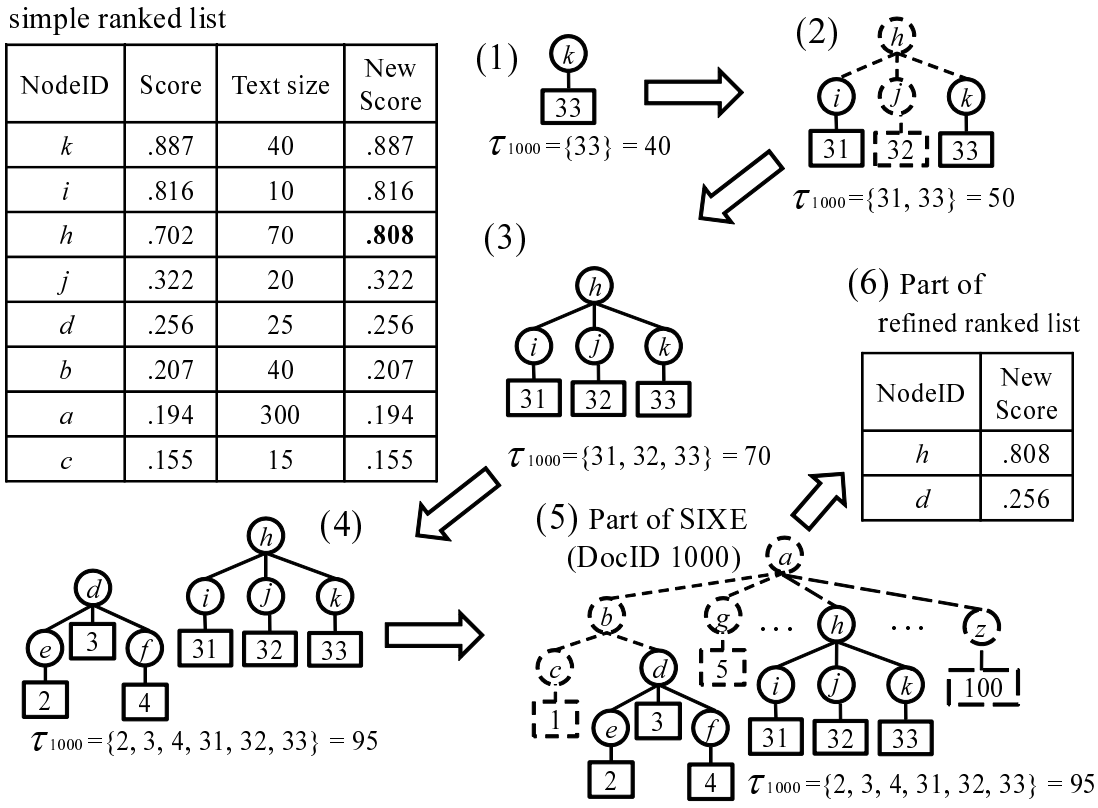


Figure 4.4. Example of generating a refined ranked list for an XML document

In Section 4.6, we investigated search accuracies of every single function and all variations of integrated methods.

4.4.2 Example of Generating SIXE and a Refined Ranked List

In summary, we illustrate an example of generating a part of an *SIXE* in which the document ID is 1,000 and its corresponding refined ranked list uses *BU* scoring. Figure 4.4 provides a graphical view of this example. Note that we use Equation 4.6 to calculate the *BU* score.

Suppose that $\alpha = \frac{1}{3}$, and $|D_{1000}| = 300$. Then, $EL_{1000} = \alpha \cdot |D_{1000}| = 100$. Next, we introduce τ_{1000}, which is the total length of the extracted XML elements from document ID 1,000.

We first obtain a simple ranked list that is calculated in Step (1). The obtained XML elements are shown in the left table of Figure 4.4. For the sake of simplicity, we assume that the list contains only the XML elements whose document ID is 1,000.

We extract the XML element with the highest score, which is node k, from the simple ranked list. Because the text length of k is 40 ($< EL_{1000}$), k is extracted. This extraction process continues, because τ_{1000}, which contains text node 33, is less than EL_{1000}. Therefore, i is selected next, because i has the second-highest score in the simple ranked list. Thus, i is extracted, because τ_{1000}, which contains text nodes 31 and 33, equals 50 ($< EL_{1000}$). Node h is the next candidate to be extracted. Because i and k are the descendants of h, they are overwritten by h. In particular, i and k are removed from the *SIXE* and h is added. At this point, τ_{1000} becomes 70, which is still less than EL_{1000}.

Next, *bottom-up* scoring is applied. Function (2) is used to score node $h(s_{BU}(h) = \frac{40}{70} \cdot 0.887 + \frac{70-30}{70} \cdot 0.702 = 0.808)$.

Following Figure 4.4 further, d is also extracted. Nodes b and c fail to be extracted, because τ_{1000} exceeds EL_{1000}. In the end, the *SIXE* is formed by nodes d and h.

Finally, the refined ranked list is constructed by adding the scores calculated via *bottom-up* scoring into the *SIXE*. In the same manner, we generate the complete *SIXE* for all of the XML documents and construct the final refined ranked list in the descending order of their scores.

4.5. Implementation

The proposed method is implemented with RDB. We used XRel [64] to transform the XML document into a RDB format. Although XRel splits the XML document into five tables, the proposed system only utilizes `element` and `term` tables for query processing. The schema of the tables is as follows.

```
CREATE TABLE element (
    did number,
    eid number,
```

```
    st_pos number,
    ed_pos number,
    path VARCHAR2(512),
    pathexp VARCHAR2(512),
    numTerm number,
    PRIMARY KEY (did, eid)
);

CREATE TABLE term (
    did number,
    eid number,
    term VARCHAR2(256),
    weight float(23),
    PRIMARY KEY(did, eid, term)
);
```

The `element` table contains information about elements. Attributes of the `element` table are documentID (`did`), elementID (`eid`), start position of the element in the document (`st_pos`), end position of the element in the document (`ed_pos`), full-path (`path`), path expression (`pathexp`), the number of terms in the element (`numTerm`). Primary Keys are `did` and `eid`. Offset information, or `st_pos` and `ed_pos`, is leveraged to for overlapping relationships between elements. Then, full-path information is used when search results are shown, while a path expression is utilized to check whether an element satisfies a structural constraint.

On the other hand, the `term` table stores information related to a term. Attributes of the `term` table are documentID (`did`), elementID (`eid`), term (`term`), and term weight (`weight`). Primary keys are `did`, `eid`, and `term`. A term weight is calculated with an arbitrary term-weighting scheme.

Since the INEX topics used in experimental evaluations are written in NEXI, these NEXI queries need to be translated into SQL format. We semi-manually rewrite NEXI queries into SQL queries. Concrete examples of queries for baseline, QS, QK, QO can be found in Appendix **B**.

4.6. Experimental Evaluations

We conducted some experimental evaluations for confirming the effectiveness of the proposed methods with INEX test collections.

Note that the PC we used for the experiments runs Oracle Enterprise Linux 5.5. It has four Intel Xeon X7560 CPUs (2.3GHz), 512GB of memory, and a 4.5TB disk array. The database used in query processing is constructed with Oracle Database 11g Release 2.

We go through following pre-processes as follows:

1. removing attributes, comments, and special characters of XML documents,
2. removing the stop words by SMART stop list (listed in Appendix **A**), and
3. applying stemming step by Porter [11].

4.6.1 Preliminary Experiments

Before evaluating the search accuracy of an *SIXE*, we choose the term-weighting scheme for the initial search in Section 4.2.1, and also set the parameters α and β for EL in Section 4.2.2.1. In the next sections, we perform preliminary experiments to find the best term-weighting scheme for the initial search and the best parameters with the INEX 2008 test collection. We use the same term-weighting scheme and parameters with the experiments of the INEX 2010 test collection.

4.6.1.1 Smoothing Method for Query Likelihood Model for Element Retrieval Techniques

In choosing the best term-weighting scheme in next section, the best smoothing method for QLMER.

We therefore conducted some preliminary experiments with some conditions to choose the best parameters for each smoothing method. Concretely, we used five smoothing methods, i.e., the linear interpolation (collection-based smoothing and path expression-based one), the Dirichlet smoothing (collection-based smoothing and path expression-based one), and two-stage smoothing.

Table 4.1 shows the best tuned search accuracies of the smoothing methods. It shows that the linear interpolation with the path expression-based smoothing

	iP[.01]	MAiP
linear interpolation (collection, $\omega = 0.5$)	.5194	.1087
linear interpolation (path expression, $\omega = 0.5$)	.5211	.1130
Dirichlet smoothing (collection, $\mu = 1100$)	.5612	.1370
Dirichlet smoothing (path expression, $\mu = 1100$)	.6038	.1487
two-stage (path expression, $\omega = 1.0$, $\mu = 1100$)	.6130	.1487

Table 4.1. Effects of a smoothing method on search accuracy

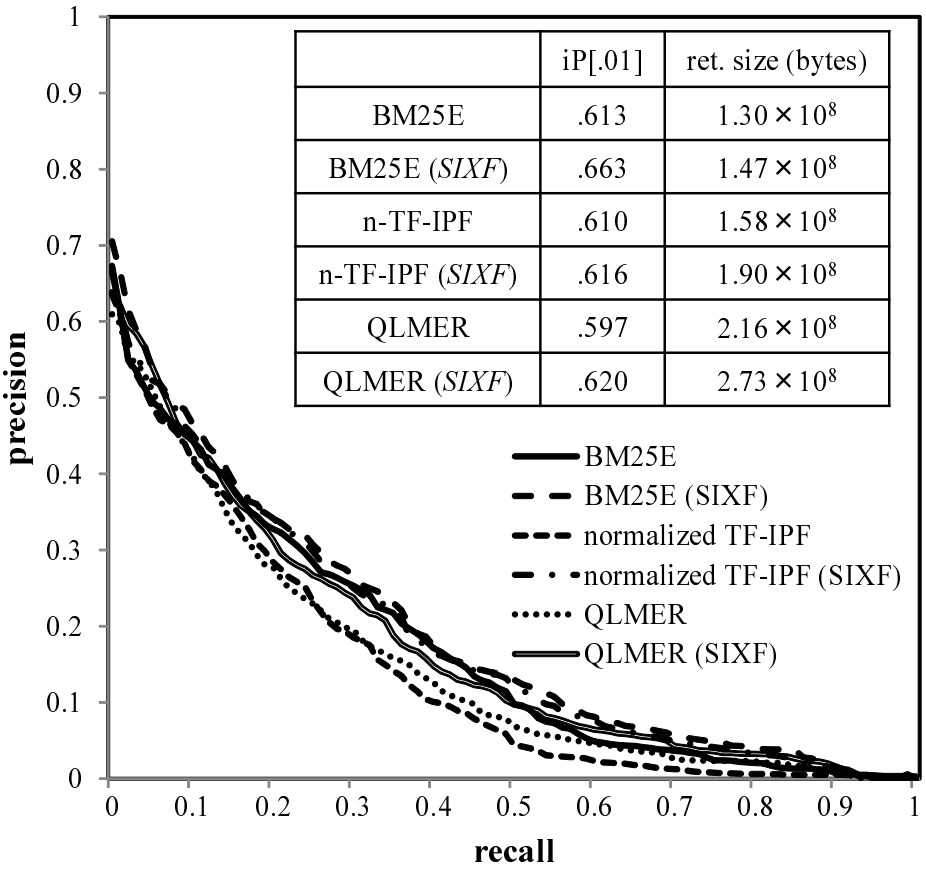
	normalized TF-IPF	BM25E	QLMER
MAiP	.1399	.1679	.1487
Standard deviation	1.30×10^{-3}	1.41×10^{-3}	1.53×10^{-3}

Table 4.2. Standard deviation and effect of *SIXE*

is more effective than the collection-based one. Similarly, the Dirichlet smoothing with the path expression-based smoothing is more effective than the collection-based one. Therefore, path expression-based smoothing is more effective in search accuracy. Then, two-stage smoothing cannot overwhelm the either smoothing method as well as document retrieval [16].

4.6.1.2 Choosing Term-weighting Scheme for Initial Search

In Section 4.2.1, we assumed that a simple ranked list that is suitable for *SIXE* has two conditions: (1) a simple ranked list is highly accurate in terms of its MAiP, and (2) a simple ranked list contains many sizes of XML elements. Accordingly, we performed two preliminary experiments to reveal the most suitable scoring method and to evaluate whether our assumption is true or not. In the former experiment, we compared the search accuracy of the following three scoring methods: the normalized TF-IPF [72], BM25E [73], and QLMER [74]. Because BM25E requires a weight to be assigned to each tag, we simply set the weight of all tags to 1. The parameters for BM25E are tuned with the INEX 2008 test collection ($k_1 = 2.5, b = 0.85$) and used for both test collections. Likewise, smoothing method and its parameters for QLMER is set based on the experiments conducted in the previous section.



	iP[.01]	ret. size (bytes)
BM25E	.613	1.30×10^8
BM25E (*SIXF*)	.663	1.47×10^8
n-TF-IPF	.610	1.58×10^8
n-TF-IPF (*SIXF*)	.616	1.90×10^8
QLMER	.597	2.16×10^8
QLMER (*SIXF*)	.620	2.73×10^8

Figure 4.5. Comparison of scoring methods

Table 4.2 shows that both MAiP and the standard deviation of BM25E are higher than those of the normalized TF-IPF. Thereby, BM25E should be a more suitable scoring method for our method *SIXE* than the normalized TF-IPF, if our assumption is valid. Meanwhile, although the MAiP of BM25E is higher than QLMER, the standard deviation of BM25E is lower than QLMER, which makes it difficult to predict which scoring method is more suitable for the proposed reconstruction method.

To confirm it, we apply *SIXE* to normalized TF-IPF, BM25E, and QLMER. The result of the experiment is illustrated in Figure 4.5. The figure shows that the iP[.01] of BM25E (*SIXE*) is higher than that of the normalized TF-IPF (*SIXE*) and QLMER (*SIXE*). Also, the rate of increase between the original method and *SIXE* is highest with BM25E.

α	0.1	0.2	0.3	0.4	0.5
iP[.01]	.447	.548	.626	.605	.595
MAiP	.0291	.0505	.0768	.0950	.108
α	0.6	0.7	0.8	0.9	1.0
iP[.01]	.603	.601	.609	.600	.662
MAiP	.123	.132	.138	.140	.237

Table 4.3. iP[.01] at each α for EL

According to the results of experiments, our assumption is correct, and the scoring method attaining highly MAiP and returning variously sized XML elements is suitable for our goal. This suggested that such a scoring method can return a more appropriate granularity of XML elements when an overwrite operation runs. We conclude that we use BM25E as a scoring method for the initial search in the following experiments.

4.6.1.3 Tuning Parameter for *EL* of *SIXE*

We performed other preliminary experiments to determine the parameter EL, as described in Section 4.2.2.1, before evaluating our method *SIXE*. First, we performed experiments to evaluate the effectiveness of EL on the document-size-dependent condition, which assumes that the ratio of the relevant parts in an XML document is lower than a certain value. Because EL depends on α, we evaluate it with iP[.01] and MAiP by changing α from 0.1 to 1 by steps of 0.1 in the experiment. Table 4.3 shows iP[.01] and MAiP at α. In this regard, α is the ratio of the relevant parts in an XML document. The experiment shows that the best value of α is 1.0, because iP[.01] and MAiP are the best. This fact indicates that a size limitation on the document-size-dependent condition is not appropriate. Moreover, the result also shows that some XML documents should be returned as a whole document.

We subsequently performed a preliminary experiment to explore the search accuracy of EL on the document-size-independent condition, which assumes that the relevant text in an XML element is lower than a constant value. Table 4.4 shows iP[.01] and MAiP at β. In this regard, β is the text size of the relevant

β	100	200	300	400	500	600	700
iP[.01]	.482	.536	.595	.629	.649	.648	.649
β	800	900	1000	2000	3000	4000	5000
iP[.01]	.655	.656	.663	.659	.659	.660	.661

Table 4.4. iP[.01] at each β for EL

parts in an XML document. It shows that iP[.01] is the highest when $\beta = 1,000$. This result agrees with the estimation that the best parameter for EL is less than 1,200.

From this result, we use EL on the document-size-independent condition with $\beta = 1,000$.

4.6.1.4 Tuning Parameter for *BU*

In this section, we investigate the best tuned parameter for BU described in Section 4.3.1. Since BU is applied during overwrite operations of *SIXE*, BU is used with *SIXE*.

There are two options for BU as follows: (1) integrating the scores of the ancestor element and the descendant elements by a constant fraction, (2) integrating the scores by a ratio of the lengths of the ancestor XML element and that of its descendant elements, and (3) combining approaches (1) and (2).

We evaluated the search accuracy by changing γ, which is the ratio of the effect of an ancestor element from 0.0 to 1.0 by 0.1 steps, using Equation 4.5 (Table 4.5) and Equation 4.7 (Table 4.6). We do not have to examine the search accuracy of Equation 4.6 individually, because the result of Equation 4.6 is equal to that of Equation 4.7 when $\gamma = 0.5$.

The results of the two tables show that each iP[.01] of approach (3) is higher than that of approach (2); besides, the iP[.01] of approach (3) with $\gamma = 0.6$ is higher than that of $\gamma = 0.5$. On the basis of these results, the most effective approach for BU is calculated by approach (3), which integrates the scores by a constant ratio after re-scoring with a ratio of the length of the two elements with $\gamma = 0.6$. Therefore, we use Equation 4.7 with $\gamma = 0.6$ in the latter experiments.

γ	0.0	0.1	0.2	0.3	0.4	0.5
iP[.01]	.398	.409	.427	.452	.479	.498
MAiP	.102	.105	.106	.109	.115	.120
γ	0.6	0.7	0.8	0.9	1.0	
iP[.01]	.520	.551	.575	.576	.578	
MAiP	.126	.130	.134	.135	.137	

Table 4.5. iP[.01] at each γ for BU (Equation 4.5)

γ	0.0	0.1	0.2	0.3	0.4	0.5
iP[.01]	.461	.485	.512	.550	.605	.665
MAiP	.108	.115	.126	.142	.165	.189
γ	0.6	0.7	0.8	0.9	1.0	
iP[.01]	.669	.667	.626	.665	.665	
MAiP	.190	.190	.190	.189	.189	

Table 4.6. iP[.01] at each γ for BU (Equation 4.7)

4.6.2 Evaluations for Each Scoring Function

We evaluated search accuracy and retrieved byte size of each of the proposed scoring functions, namely QO(QS and QK), *SIXE*, and *TD*. Note that we do not evaluate BU here because BU cannot be used singularly. We set BM25E as a baseline method.

Table 4.7 depicts that *SIXE* and *TD* improved search accuracies compared to baseline, while QO decreased the search accuracy.

In terms of search accuracy of a single scoring function, *SIXE* showed the most effective scoring function. From this result we can see that integrating elements to extract a larger element for more relevant descriptions is effective in advancing search accuracy.

Although QO decreased search accuracy, both of the scoring functions in QO, or QS and QK, improved search accuracy. This result indicates that QS and QK has negative influences on each other. Therefore, we need to be careful in combining these scoring functions in order not to extract useful search results.

	baseline	*QO*	*QS*	*QK*	*SIXE*	*TD*
iP[.01]	.6131	.6069	.6236	.6146	.6628	.6239
MAiP	.1679	.1721	.1720	.1767	.1946	.1535
retrieved	1.26×10^8	1.73×10^8	1.47×10^8	1.68×10^8	1.47×10^8	1.33×10^8

Table 4.7. Comparison among the propoed scoring functions

Futheremore, *TD* can be used as a single scoring function, although *TD* was originally proposed to be used with *SIXE*. It turned out that TD improved search accuracy. From the fact that *TD* is even more accurate than *QK*, our notion that descendant elements take over features of ancestor elements seems to be true.

4.6.3 Evalutions for Integrated Methods

We continuously survey the effectiveness of integrated methods each of which is composed of possible permutations of the proposed scoring functions. The results of the evaluations are shown in Table 4.8.

All the integrated methods except *QS-TD* improved search accuracies compared with the baseline approach. Especially, every integrated method which applies *SIXE* improved search accuracy, which suggests that *SIXE* contributes to a rise in search accuracy.

BU which cannot be evaluated in the experiments conducted in the previous section improved search accuracies in many integrated methods applied *BU*. Additionally, it seems that *TD* is suitable to be applied after *SIXE*, because *SIXE-BU-TD* and *QS-SIXE-BU-TD* are more accurate than *SIXE-BU* and *QS-SIXE-BU*, respectively. Conversely, *TD* before *SIXE* may decreases search accuracy because *QS-SIXE* is better than *QS-TD-SIXE* regarding search accuracy.

In all variations of the integrated methods, *SIXE-BU-TD* is the most accurate. This result agrees with tendencies discussed above.

From the observations obtained through experiments, *SIXE* highly contributes to search accuracies, and *BU* and *TD* also lead to a more accurate search.

Accordingly, we conduct exhaustive experiments on them with both INEX 2008 test collection and INEX 2010 test collection.

	iP[.01]	MAiP	byte	rate of increase
SIXE-BU-TD	.6686	.1934	1.43×10^8	1.086
SIXE-BU	.6653	.1888	1.43×10^8	1.081
QS-SIXE-BU-TD	.6585	.1918	1.43×10^8	1.070
QS-SIXE-BU	.6555	.1886	1.43×10^8	1.065
QS-SIXE	.6540	.1901	1.43×10^8	1.062
TD-SIXE-BU	.6442	.1860	1.74×10^8	1.047
SIXE-TD	.6435	.1869	1.43×10^8	1.045
TD-SIXE	.6430	.1874	1.74×10^8	1.045
SIXE-BU-QK	.6370	.1813	1.43×10^8	1.035
SIXE-QK	.6363	.1826	1.43×10^8	1.034
QK-SIXE-BU	.6362	.1807	1.79×10^8	1.034
QK-SIXE	.6354	.1824	1.79×10^8	1.032
QS-SIXE-BU-QK	.6300	.1815	1.43×10^8	1.023
QS-QK-SIXE-BU	.6297	.1813	1.83×10^8	1.023
QS-QK-SIXE	.6294	.1831	1.83×10^8	1.022
QS-TD-SIXE-BU	.6258	.1723	2.04×10^8	1.017
QS-TD-SIXE	.6254	.1742	2.04×10^8	1.016
baseline	.6156	.1679	1.26×10^8	1.000
QS-TD	.6109	.1435	1.59×10^8	0.992

Table 4.8. Comparison among integrated methods

4.6.4 Further Experiments with *SIXE*, *BU*, and *TD*

As summarized in Table 4.9, all of *SIXE-BU*, *SIXE-TD*, and *SIXE-BU-TD* improved the search accuracies compared with both baseline and *SIXE* in the INEX 2008 test collection. Note that *SIXE-TD* increased significantly the size of the retrieved XML elements. This is an unpleasant result, because we believe that the XML element search should produce search results that are small rather than large, because larger should be uncomfortable for users. Conversely, *SIXE-BU* and *SIXE-BU-TD* did not increase the size of the retrieved XML elements as much. Furthermore, *SIXE-BU-TD* can search most accurately among these scoring methods.

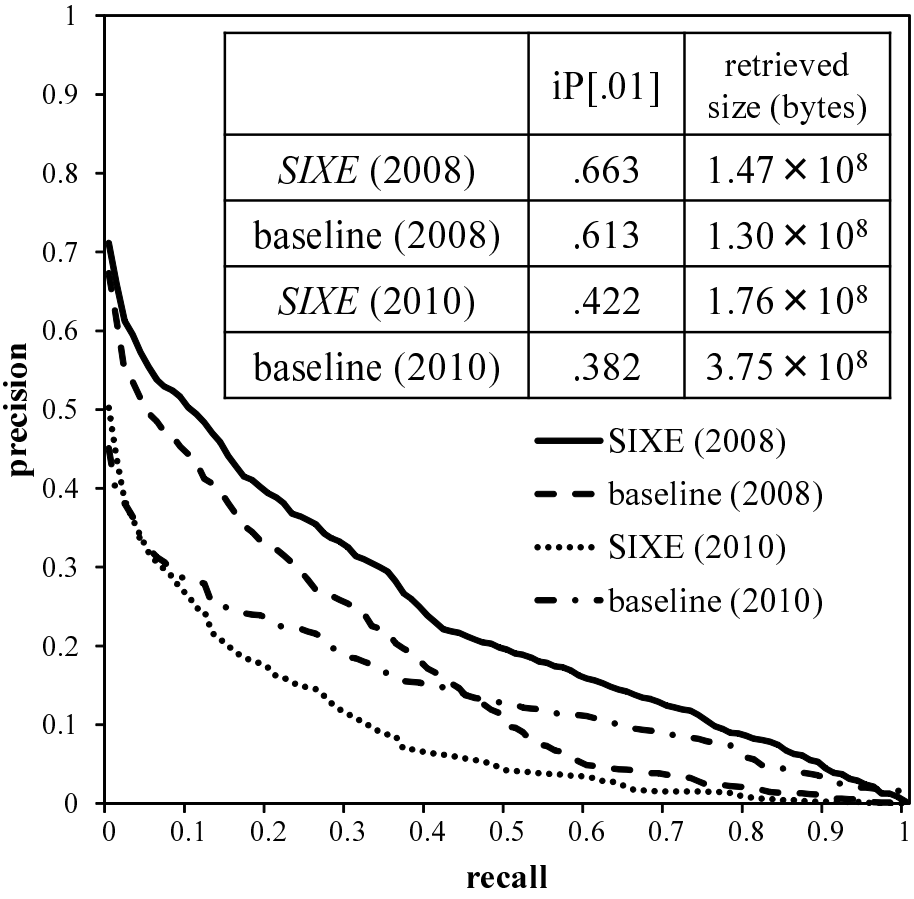
INEX 2008	baseline	*SIXE*	*SIXE-BU*	*SIXE-TD*	*SIXE-BU-TD*
iP[.01]	.613	.660	.668	.662	.669
MAiP	.171	.240	.191	.242	.196
retrieved size (bytes)	1.30×10^8	1.43×10^8	1.43×10^8	2.28×10^8	1.43×10^8
INEX 2010	baseline	*SIXE*	*SIXE-BU*	*SIXE-TD*	*SIXE-BU-TD*
iP[.01]	.382	.422	.437	.384	.410
MAiP	.144	.0930	.0952	.138	.117
retrieved size (bytes)	3.75×10^8	1.76×10^8	1.76×10^8	3.68×10^8	1.57×10^8

Table 4.9. Effect of scoring methods INEX 2008 and INEX 2010

Therefore, *SIXE-BU-TD* is the most suitable method for our proposal. Finally, the iP[.01] of the proposed method is improved by 9% compared to that of the baseline.

Likewise, we examined search accuracies of the baseline approach and the proposed methods in INEX 2010 test collection. These accuracies decreased compared with these of INEX 2008 in the same manner as other participants (see Table 5.7). This is because INEX 2010 test collection contains four times larger articles and additional semantic markups compared with INEX 2008 test collection. These features make accurate search more difficult.

Then, all variations of the proposed methods improved search accuracies compared with the baseline approach. In both test collections, *SIXE* quite contributes to improving search accuracy as Figure 4.6 shows. On the other hand, only *SIXE-BU* could improve the search accuracy compared with *SIXE* in the INEX 2010 test collection. There are two possibilities why the search accuracy of *SIXE-TD* decreased. First, it is not always true that an element in an XML document that contains many kinds of query keywords is effective. Second, we could not generate good *SIXE*, and it may contain some irrelevant elements. From the fact that both the iP[.01] and the MAiP of the baseline in the INEX 2010 test collection are lower than that in the INEX 2008 test collection, the simple ranked list may contain some irrelevant elements. Therefore, we should remove such irrelevant elements from the simple ranked list to tackle the lower accuracy compared with the INEX 2008 test collection.



	iP[.01]	retrieved size (bytes)
SIXE (2008)	.663	1.47×10^8
baseline (2008)	.613	1.30×10^8
SIXE (2010)	.422	1.76×10^8
baseline (2010)	.382	3.75×10^8

Figure 4.6. Comparison of our reconstruction method versus the baseline

Furthermore, we compared our approach with other approaches in which INEX plays a part. Table 5.7 compares our proposed method (*SIXE-BU-TD* with the INEX 2008 and *SIXE-BU* with the INEX 2010) with three other competitive XML search systems [52]. Only four systems (including ours) were evaluated in the INEX 2008 test collection. In the experiments, our proposed method provides the highest precision when the recall level is less than or equal to iP[.01]. Because the official measure for the focused task is iP[.01], our system has the highest level of accuracy.

We also compared our method with the other top-three teams in the INEX 2010 test collection in [53]. The result shows that our method has a higher score

INEX 2008	iP[.00]	iP[.01]	iP[.05]	iP[.10]	MAiP
Doshisha Univ.	.7092	.6691	.5606	.5046	.2417
Renmin Univ. of China	.5969	.5969	.5815	.5439	.2486
Queensland Univ. of Technology	.6232	.6220	.5521	.4617	.2134
Univ. of Amsterdam	.6514	.6379	.5901	.5280	.2261
INEX 2010	char prec	iP[.01]	iP[.05]	iP[.10]	MAiP
Doshisha Univ.	.3884	.1822	.0382	.0000	.0088
Univ. Pierre et Marie Curie	.4125	.1012	.0385	.0000	.0076
Univ. of Helsinki	.3435	.1186	.0273	.0000	.0069
Lia Univ. of Avignon	.3434	.1500	.0000	.0000	.0077

Table 4.10. Comparison of four INEX participant search systems including our proposed system

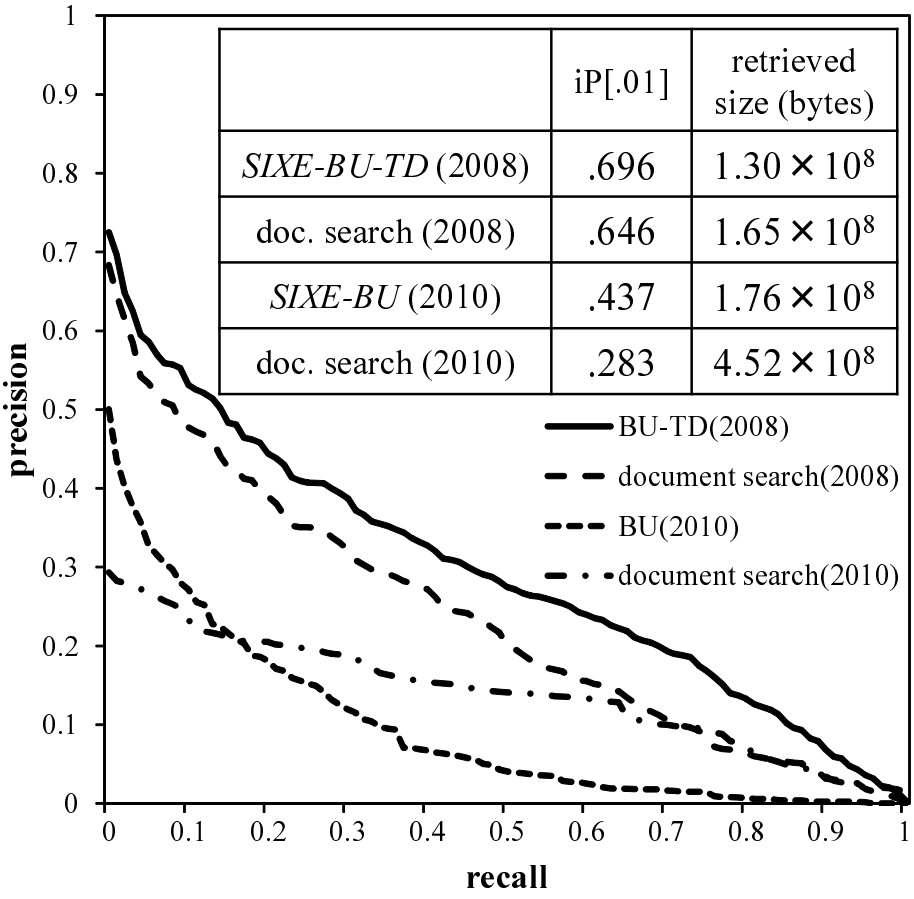
in iP[.01][2].

4.6.5 Comparisons to Document Search

We finally compared our method (*SIXE-BU-TD* with the INEX 2008 and *SIXE-BU* with the INEX 2010) with a traditional document search, because a document search is often reported as being more effective than an XML element search [52]. In a document search, a set of the entire XML documents is returned as a search result generated by BM25E. In this experiment, we used all queries that return the entire XML documents in both of the INEX test collections.

As shown in Figure 4.7, the results of our experiments showed that all of the interpolated precisions at each recall level of the method *SIXE-BU-TD* (2008) are higher than those of the document search (2008). This indicates that an XML element search is more effective than a document search. On the other hand, *SIXE-BU* (2010) also overwhelmed the document search (2010) in recall levels including iP[.01]. Therefore, we conclude that an XML element search can be

[2]INEX official runs requires search systems to extract 1,000 characters per a query in INEX 2010. To compare with the other systems, we show the search accuracy of the proposed method that extracts 1,000 characters per a query. This is the reason why the result is very different from Table 4.9.



	iP[.01]	retrieved size (bytes)
SIXE-BU-TD (2008)	.696	1.30×10^8
doc. search (2008)	.646	1.65×10^8
SIXE-BU (2010)	.437	1.76×10^8
doc. search (2010)	.283	4.52×10^8

Figure 4.7. Comparison of XML element search and document search

useful.

Furthermore, we note that the retrieved text size of our proposed method is substantially smaller than that of a document search. As such, we present more focused content to users, which saves them time and energy.

CHAPTER 5

Fast Incremental Indexing with Accurate and Fast XML Element Retrieval

In Chapter 4, we proposed accurate XML element retrieval approaches through evaluations with two test collections. Combined with efficient query processing algorithm [86] and data compression and selection method [87], the proposed method attains both accurate search and fast query processing. To achieve the goal mentioned in Figure 1.1, we need to make more effort. Hereinafter, we focus on fast reflection of document updates for accomplishing a practical XML element search system.

Web documents are frequently updated; i.e. inserted, deleted, or modified. In particular, Wikipedia articles are updated 4000 to 8000 times per hour[1]. Information retrieval systems are expected to present search results based on the latest content on the Web, especially as new topics are added to documents. Without handling updates, a search system cannot find newly inserted documents, and it ranks documents based on obsolete information, which reduces the effectiveness. Thus, we add a function for handling document updates to the existing techniques for XML element retrieval.

When document insertions and modifications occur, scores of these inserted

[1] `http://www.wikichecker.com/editrate/`

and modified documents need to be calculated immediately. In this situation, term-weighting-based scoring is appropriate because it takes long time to calculate accurate scores with network-analysis-based scoring as we mentioned in Section 3.1.1.

Accordingly, there are two goals for this study as follows:

1. reflecting document updates to a search system as soon as possible, and
2. computing scores (term weights) accurately.

To achieve the first goal, two requirements are needed as follows:

- building an index which can be updated incrementally according to document updates, and
- highly efficient indexing.

The mainstream approach for updating an index is to construct a new index periodically *from scratch* while discarding the existing one. It may take a long time to retrieve updated documents if constructing a new index is costly. Incremental updates are required to shorten this delay.

Our proposed system has a function of incremental updates of the index. We believe that this is the first study focused on fast incremental updates of indices in effective and efficient XML element retrieval systems. Although Google [116] supports fast incremental updates with effective and efficient query processing, its approach differs from ours. Google analyzes the link information of Web pages to find important pages, whereas our study utilizes text information. We can apply our approach to other structured documents apart from the Web, even if these do not have link information.

The proposed system can insert, delete, and modify the index by expanding existing an index structure. The update efficiency of the index is low because a number of update targets is treated as update targets in XML element retrieval compared with document retrieval. Thus, we propose two kinds of filters for reducing update cost by eliminating unnecessary update targets.

Concerned with the second goal, term weights cannot be calculated accurately in some situations such as:

- when a search system does not store sufficient amount of documents, and

- when new topic emerge drastically.

This is because some kinds of statistics used in calculating a term weight are *global weights* that are aggregate statistics derived from all documents in a document set. Thus, global weights are difficult to calculate in the some situations mentioned above. We need a method to approximate accurate global weights with an insufficient number of documents.

In the latter part of this Chapter, we evaluated the effectiveness and efficiency of our approaches through experiments with two cases: the static statistics case in which topics rarely change, and the dynamic case in which new topics are added frequently.

5.1. Fast Incremental Updates of Indices

General XML element retrieval systems [86], [87] have functions for index construction and query processing. They rebuild an index from scratch when document updates occur. This means it tales long time to build a new index when the amount of accumulated documents is large. In contrast, our proposed system has capabilities for document insertion, deletion, and modification to reflect document updates immediately. Moreover, we propose two filters which reduce update cost for attaining fast incremental updates.

5.1.1 Expansion of Existing Functions

The proposed system use an arbitrary term weighting scheme. Based on the experiments in Chapter 4, we use BM25E [73], although we also refer to QLMER [74] to show even statistical language models are handled with our framework.

5.1.1.1 Structures of Indices

We show the structures of the proposed indices in Figure 5.1. In many existing studies, the term weights stored in the indices are calculated beforehand, and structural constraints can be checked with these. The proposed indices inherit these capabilities but also contain global weights to calculate a term weight immediately, which is essential for fast incremental updates.

- `Term` (DID, EID, term, term weight, Path ID, element length)
- `Tag-term` (DID, EID, tag, term, term weight, Path ID, element length)
- `RS` (DID, EID, term, term wight)
- `Path` (Path ID, path expression)
- `GW-Path-term` (Path ID, term, frequency [, values of back-ground language model for QLMER])
- `GW-Path` (Path ID, frequency, total length)
- `Term-filter` (tag, term, threshold value)

Figure 5.1. Structure of the indices

As in the related studies [86], [87], the structures of the indices are defined in an RDB format. Primary keys are underlined. The `Term`, the `Tag-term`, the `RS`, and the `Path` indices are used for efficient and effective query processing as in other studies.

In the `GW-Path-term` and the `GW-Path` indices, the global weights are indexed for enabling fast term calculation. In Eq. (3.5), (2.10), and (2.11), $pf_{p,t}$, N_p, $avel_p$, and $\hat{P}_{mle}(t|M_b)$ are global weights.

We discuss the `Term-filter` index below in Section 5.1.3.2.

5.1.1.2 Top-k Searches

The proposed system has a function for top-k searches [85] to enhance its usability in fast query processing. To return search results, only the top k tuples are retrieved for each term (a pair of tag and term for CAS queries). The term weights in the tuples are summed for each element to calculate scores. Furthermore, a weight in an arbitrary term in any element can be gained with a random scan when we need to calculate exact scores for search results. We can attain not only efficient query processing but also effective query processing with the random scan.

A CO query retrieves the `Term` index whereas a CAS query retrieves the `Tag-term` sequentially to extract candidate search results in query processing.

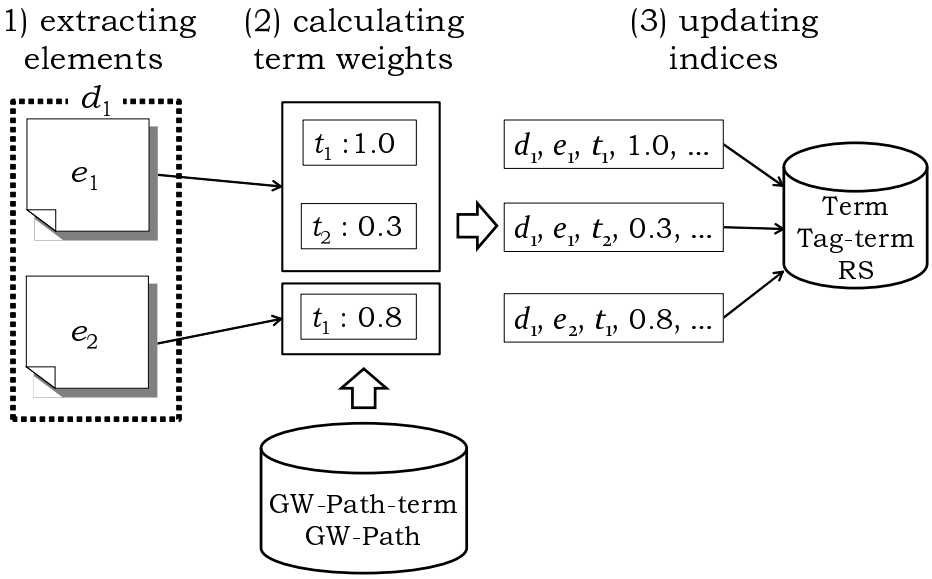


Figure 5.2. Architecture of the simple approach

Accurate scores are calculated for elements by a random scan with the `RS` index. Note that tuples in the `Term` index are grouped by term in descending order of term weight, whereas tuples in the `Tag-term` index are grouped by pair of tag and term. When a CAS query contains two or more structural constraints, the path expressions of elements must be checked to determine whether these satisfy the query constraints.

5.1.2 Handling Document Updates

5.1.2.1 Document Insertion

When a document is inserted, the updating process is conducted as follows:

(1) extracting elements from the inserted document,

(2) calculating term weights for the elements, and

(3) updating indices.

Figure 5.2 describes each process in detail.

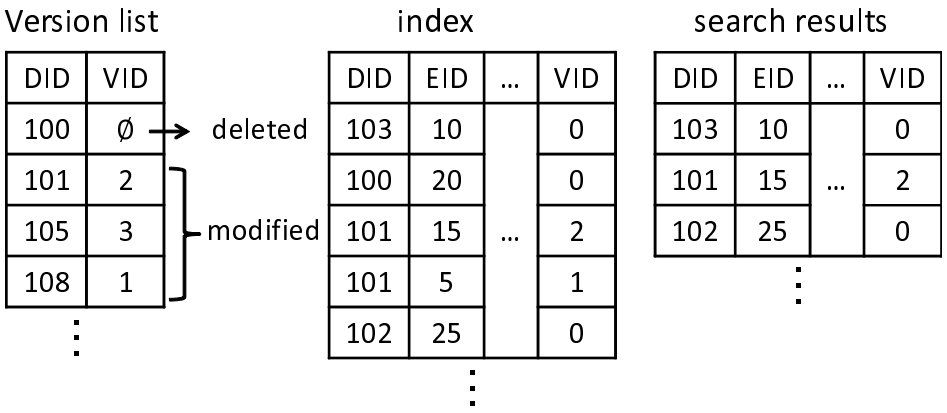


Figure 5.3. the version list and query processing

First, the document is parsed and elements are extracted. As a result, elements e_1 and e_2 are extracted.

Second, the term weights of t_1 and t_2 in e_1, and t_1 in e_2 need to be calculated. As we explained in Section 3.1.1, term weighting schemes for XML element retrieval are composed of three types of factors: local weights that are statistics derived from each document (element); global weights that are statistics derived from all document in a document set; and constant values (coefficients and parameters). Local weights and constant values are easy to calculate and refer to because local weights are computed for a newly inserted element. On the other hand, it is difficult to calculate global weights on demand because the entire document set must be scanned to compute these. Therefore, term weights are calculated immediately with the `GW-Path-term` and the `GW-Path` indices.

We only need to store all kinds of global weights in the indices when using another term-weighting scheme requiring other statistics.

Finally, the `Tag-term`, the `Term`, and the `RS` indices are updated incrementally after the term weights are calculated.

Note that an entire set of documents can be updated at once to reduce the I/O cost.

5.1.2.2 Document Deletion

When a document is deleted, there is a high cost to find and delete all tuples related to the document because the tuples are spread across the indices. We therefore take another approach to reduce the cost of the deletion. We manage the DIDs of deleted documents instead of deleting tuples in the indices. Then, we simply ignore the tuples of the DIDs in query processing. With this approach, we can reflect the deletion immediately.

We prepare a version list to manage the deleted documents. The list contains pairs. Each pair contains the DID of the deleted document and the version identifier (VID) with its value marked as $\emptyset$. We overwrite the VID as $\emptyset$ when the DID of the deleted document is contained. Specifically, Document 100 in Figure 5.3 has been deleted because the VID of Document 100 is $\emptyset$.

The tuples of documents deleted in the indices are eliminated when the load average is low. After eliminating the tuples, the DIDs of the documents deleted in the version list are also eliminated.

5.1.2.3 Document Modification

The modification process is achieved through the deletion and insertion processes. In more detail, we delete all tuples related to the modified document and insert the latest version of the document to handle the modification. We also utilize the version list to manage the version of the document, because there is a high cost to delete the tuples of a modified document immediately. To enable fast modification, we only target the tuples of the latest version in query processing.

Note that the granularity of modification is document granularity, because some problems arise with element granularity. One of the problems is the size of the version list. The overhead in query processing become greater when we manage not documents but elements. Another problem is the difficulty in mapping old structure to new structure when the document structure changes. These are the reasons that we adopt the document as the granule of modification.

The modification process is conducted as follows: first, when a modification occurs, the version list is scanned to determine whether the DID of the modified document is contained. If the DID exists in the version list, 1 is added to the VID; otherwise, the DID of the modified document and its VID value of 1 are inserted.

For example, Document 101 in Figure 5.3 has gone through modification twice because its VID is 2.

Second, the `Term`, the `Tag-term`, and the `RS` indices are updated in the same manner as for document insertion. As shown in Figure 5.1, each tuple contains a VID whose value is the same as that written in the version list. Note that the VID of the first document inserted is 0.

Finally, each tuple is checked to determine whether the tuple is valid in query processing based on the VID. The tuple is the latest when the VID of the tuple is the same as that of the modified document in the version list. Moreover, the tuple is also the latest when the DID of the modified document is not contained in the version list. In contrast, the tuple is invalid when the VID of the tuple is smaller than that of the modified document in the version list. We give a specific example of the validation check in Figure 5.3. The DID of the first tuple in the index is 103, and the version list does not contain that DID. Thus, the first line is valid. The document of the second tuple has been deleted, because the DID of this tuple is contained in the version list and its VID is $\emptyset$. The third tuple is the latest, because the VID of this tuple is the same as the VID corresponding in the version list to the Document 101. Similarly, the fourth tuple is not the latest, because the VID of this tuple is less than that of the VID corresponding to the Document 101.

Old versions of tuples in the indices are removed when the load average is low. In this regard, the VID of the latest version of a tuple is rewritten as 0, and the DID of the deleted document is removed from the version list.

5.1.3 Filters for Reducing Update Cost

We propose two kinds of filters for selecting important elements and terms to index.

It is obvious that we can reduce update cost with these filters. However, search accuracy will be reduced if we remove elements and terms relevant to any query. This would violate the first requirement. To avoid a decrease in search accuracy, we should decide carefully which elements and terms can be removed.

5.1.3.1 Element Filter

We propose an *element filter* to remove unnecessary elements. Past studies [49, 90] led to the fact that middle granule elements are moderate and the most appropriate as search results, because extremely large granule elements (e.g. whole documents) tend to contain irrelevant descriptions and extremely small granule elements cannot satisfy the information need by themselves. Hereinafter, we attempt to remove extremely small elements, since identifying these is easier. Note that a user can search a term in an extremely small granular element to be removed, because ancestor elements of the removed element also contain the same text.

It is essential to define what extremely small elements are. Many of the Web documents include table-of-contents or reference information, which basically consists not of sentences but of keywords. These descriptions cannot satisfy an information need directly, although they can serve as navigational information. Since one requirement for text summarization is that "information should be self-contained" [15], we remove any element that cannot be understood by itself.

In addition, path expressions that rarely appear in the document set cannot be calculated accurately, as discussed above in Section 5.2.2. It is reasonable to remove these possibly harmful elements for search accuracy.

Based on the discussion above, we define three conditions of removed elements as follows:

(1) elements containing few terms (threshold τ_{el}),

(2) elements with deep path expressions (threshold τ_{depth}), and

(3) elements with rare path expressions (threshold τ_{Zipf}).

Note that we need to seek appropriate thresholds to remove only irrelevant elements for retaining search accuracy.

Regarding the first condition, the terms in the information other than the body text including table-of-contents and reference information, contain few terms also in the elements. Actually, study [49] reports that search accuracy improves when short elements are removed.

It is reported that the average sentence length in plain English is 15 to 20 words [117]. Then we hypothesize that eliminating an element shorter than 15 words does not prevent a user to extract useful information because we suppose that the minimum granularity for conveying useful information is a sentence. We validate whether the hypothesis is true or not in Section 5.3.2. On the other hand, we set 15 as default value for the threshold value τ_{el} in case the best parameter for τ_{el} cannot be explored with automatically or manually.

In the second condition, elements with deep path expressions are eliminated. Tables or lists in HTML have a tendency to be nested deeply. The value itself of each cell is not important, because it is meaningless without further information. We therefore regard these elements as irrelevant.

The threshold value τ_{depth} is decided by upper limit depth of elements which are assigned high score with the proposed method. In consequence, recall are not affected by the filter because the filter does not remove elements possibly to be search results. Concretely, we measure the depth of highly ranked elements, and count the number of elements by element depth. Then, τ_{depth} is set as the value that are larger than or equal to the depth of most of these highly ranked elements. We conduct a experiment for investigating the value in Section 5.3.2.

Many elements are removed when there are many documents deeper than the threshold value. Otherwise, few elements are removed when the number of the elements whose depth is less than the threshold value is small. In short, effectiveness of the filter changes with statistics of elements depth in a document set. Note that the threshold value can be computed without any manual judgment. Since the threshold value differs from one document set to another, we do not apply the element filter with element depth condition unless τ_{depth} is set[2].

Regarding the third condition, we use Zipf's law [118] to obtain the threshold of median frequency f, which is computed as follows:

$$f = \frac{\sqrt{8F_1 + 1} - 1}{2} \tag{5.1}$$

where F_1 is number of the path expressions appearing only once in the document set. Preliminary experiment for the threshold value τ_{Zipf} is also examined in

[2]As discussed in Section 5.3.2, τ_{depth} has a possibility to be set along with document updates even though τ_{depth} is not set when an XML element retrieval system starts working.

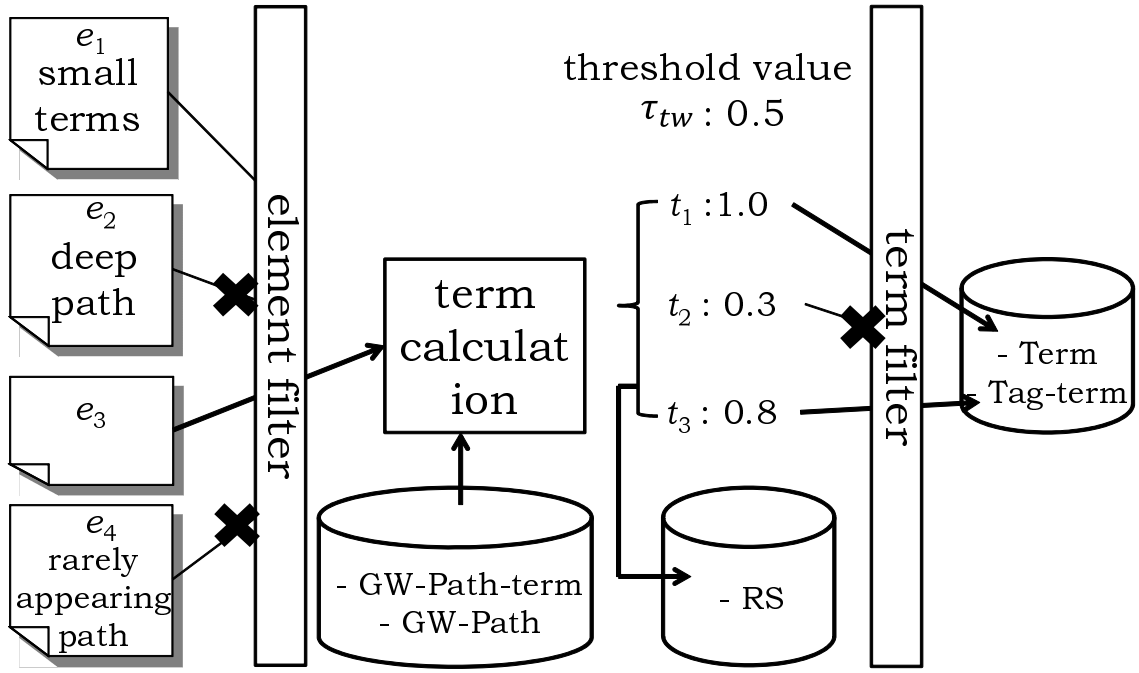


Figure 5.4. The element filter and the term filter

Section 5.3.2. In the same manner as τ_{depth}, the element filter with path frequency condition is not applied unless τ_{Zipf} is set.

Figure 5.4 illustrates the behavior of the element filter. Suppose that four elements, e_1, e_2, e_3, and e_4, are extracted from inserted documents. Elements e_1, e_2, and e_4 are eliminated by the element filter because e_1 is too short, the path expression of e_2 is too deep, and the path expression of e_4 rarely appears. As a result, only e_3 is chosen as a target.

5.1.3.2 Term Filter

Although there are many candidate search results, only a few elements are presented as search results. Therefore, we suppose that search accuracy is not significantly affected even though indices do not contain terms with low weights.

Based on this idea, we remove the unimportant terms with the term filter. The thresholds τ_{tw} are defined as the term weights of the nth largest term for each pair of tag and term contained in the indices. These values are stored in the

`Term-filter` index so that they can be looked up quickly.

Note that we apply the term filter only to the `Term` and `Tag-term` indices to enable accurate calculation of the score for elements with the `RS` index. In addition, we do not apply the filter when the number of tuples of the pair of tag and term is less than n.

The strategy for choosing n is as follows:

- The value for n need to be set large not to decrease search accuracy. At least, m is set larger than the number of search results for a query, i.e., 1500 in this thesis. This is because Top-k search cannot be processed properly when n is smaller than 1500, which cause a drop in search accuracy.
- When n is too large, the cost for applying the term filter exceeds the effect of the term filter.

Accordingly, n is set small value to maximize the effect of the term filter as long as the filter affects search accuracy. In Section 5.3.2, we investigate the appropriate value for n. Note that the term filter does not work before n is set. In addition, the value of n can be tuned with R–precision measured by manual judgment, even though there is no ground truth.

Figure 5.4 shows an example of how the term filter works. Suppose that τ_{tw} is 0.5 and there are three terms to insert into the `Tag-term`, the `Term`, and the `RS` indices. We use the single value of τ_{tw} for simplicity although τ_{tw} differs for each pair of tag and term. Terms t_1 ($1.0 > \tau_{tw}$) and t_3 ($0.8 > \tau_{tw}$) are successfully indexed with the `Term`, the `Tag-term`, and the `RS` indices because they are greater than τ_{tw}. In contrast, term t_2 ($0.3 < \tau_{tw}$) is only indexed with the `RS` index because it is less than τ_{tw}.

5.2. Estimating Accurate Global Weights

5.2.1 Effects of Incremental Updates

We examine the effectiveness and efficiency of incremental updates of indices. Note that we used the INEX 2008 test collection in the experiments.

The PC that we used for the experiments runs Oracle Enterprise Linux 5.5. It has four Intel Xeon X7560 CPUs (2.3GHz), 512GB of memory, and a 4.5TB disk

ratio of the initial documents (%)	iP[.01]	MAiP	update time (ms/doc)	total index construction time (h)	agreement ratio (%)
from-scratch	.664	.213	7.7	7.7	1.0
90	.652	.202	2.4	9.5	.18
70	.648	.196	3.7	8.9	.15
50	.644	.202	4.9	8.4	.13
30	.618	.194	6.0	8.0	.12
10	.589	.168	6.9	7.5	.093

Table 5.1. The results of the simple approach

array. The indices were implemented using BerkeleyDB in GNU C++.

5.2.1.1 Experimental Procedure

We define an index before incremental updates take place as an *initial index*. We distinguish between documents used to construct initial indices (*initial documents*) and documents used to update indices (*update documents*). Here, we assume that the statistics of the documents are static, i.e., the statistics of the initial documents and the update documents are the same. For this purpose, we randomly sampled documents in order to distinguish between them. In Section 5.3.4, we consider a more complex case in which the statistics of the documents change dynamically.

All documents are processed through the stop-word and stemming steps before the construction of the initial indices begins. The procedure is as follows: first, the initial documents are parsed to calculate term weights and the initial indices are constructed; then, the update documents are obtained for updating indices incrementally. All data in the `GW-Path-term` and the `GW-Path` indices are scanned in the main memory during updates. Then, the update documents are parsed and the `Term`, the `Tag-term`, and the `RS` indices are updated incrementally.

5.2.1.2 Evaluation Results

We investigated search accuracies, update efficiency per document, and total time of index construction by changing the percentage of initial documents within the

document set, as indicated in Table 5.1. For example, when the ratio is 30%, the initial indices are constructed using 30% of the documents in the set, and the indices are updated using the remaining 70% of the documents. When the ratio of initial documents is 100%, updates of the indices do not take place (*no-update*).

Table 5.1 shows that incremental updates reduce search accuracy, which demonstrates that global weights cannot be computed accurately using only a subset of the documents. To make the incremental update practical, we need to solve the problem of inaccurate global weights.

We additionally conducted an experiment to survey how global weights change during index update process. The most right column in Table 5.1 shows the agreement ratio of global weights in initial index compared with ones for *from-scratch*. Even when the ratio of the initial documents is 90%, the agreement ratio is only 0.18. From this observation, it is important to follow a change of global weights. Updating not only the `Term`, the `Tag-term`, and the `RS` indices, but also the `GW-Path-term` and the `GW-Path` indices is reasonable. Global weights are expected to be gotten more accurate gradually with updates of the `GW-Path-term` and the `GW-Path` indices. However, not always a sufficient number of documents are indexed in computing accurate global weights. Moreover, global weights need to be updated drastically when statistics of documents change dynamically. Therefore, a method that calculate accurate global weights with a limited number of documents.

The average time for incremental updates is 53.4 ms per document when the ratio of initial documents is 50%, whereas the time required to construct indices from scratch (*no-update*) is 42.1 ms per document. This suggests that the update efficiency decreases as the ratio of initial documents increase. As a result, indexing may take a long time when we update a number of documents.

5.2.2 Integrating Path Expression for Accurate Global Weights

We attempt to calculate global weights accurately using a limited number of documents. Since these are calculated within elements having the same path expression, we cannot obtain appropriate statistics for a path expression appearing rarely in the document set. We therefore consider a more effective approach. Specifically, we integrate path expressions having a similar property to expand

1: /article/sec
2: /article/sec/sec
3: /article/sec/emp/sec
4: /article/emp/sec
5: /article/emp/sec/sec

Figure 5.5. Examples of path expressions

the elements in the same class. To accomplish this, we integrate path expressions of the similar attribute.

To integrate path expressions, we regard a path expression as an array of tags and identify the path expressions that are similar to each other in terms of the appearance order and appearance frequencies of tags. As a result of the integration, we eliminate classes that do not contain enough elements to calculate accurate global weights.

In addition, the cost to adopt these methods is small, because these approaches simply calculate a frequencies and check the order of tags in a path expression. We can ignore the harmful effects on update efficiency. We now explain three integration methods:

1) set-of-tags method (ST),

2) bag-of-tags method (BT), and

3) order-of-tags method (OT).

5.2.2.1 Set-of-Tags Method (ST)

Tags in structured documents are separated into two groups. One represents structural classifications such as `article` and `sec` tags. The other indicates semantics, ideas, attributes, and specific contents such as `person`, `emp`, and `table` tags. These two groups of tags are supposedly independent in their appearance. This suggests that a combination of tags can generate two or more path expressions. It is not always appropriate that these path expressions are placed into

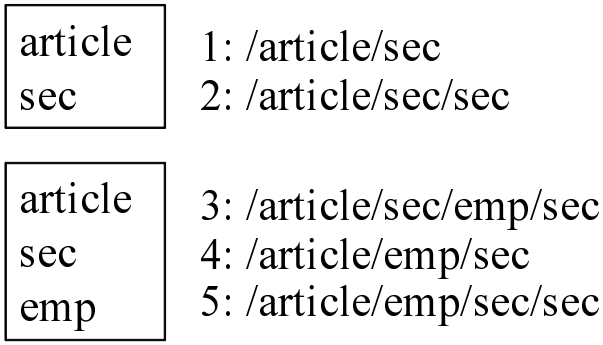


Figure 5.6. An example of classification in ST

different classes. This is why we focus on relaxing the appearance order and frequencies of tags in path expressions to integrate similar path expressions.

The set-of-tags (ST) method relaxes both the appearance order and frequencies of tags in path expressions. Accordingly, we consider only the names of the tags. We classify path expressions composed of the same tag names as members of the same class.

Classification of the path expressions in Figure 5.5 is shown in Figure 5.6. The first two path expressions are in the same class because they are both composed of `article` and `sec` tags, while the other three path expressions are in the same class because they are all composed of `article`, `sec`, and `emp` tags. The global weights of the elements with the first two path expressions are calculated together, and the global weights of the elements with the other path expressions are calculated together.

5.2.2.2 Bag-of-Tags Method (BT)

The bag-of-tags (BT) method relaxes only the appearance order of tags in path expressions. We do not consider the order of tags from the perspective of the bag-of-words concept.

Classification of the path expressions in Figure 5.5 is shown in Figure 5.7. We first enumerate the names and frequencies of tags in each path expression to integrate the path expressions classified as members of the same class. As a result, we integrate the third and fifth path expressions because both have one `article`,

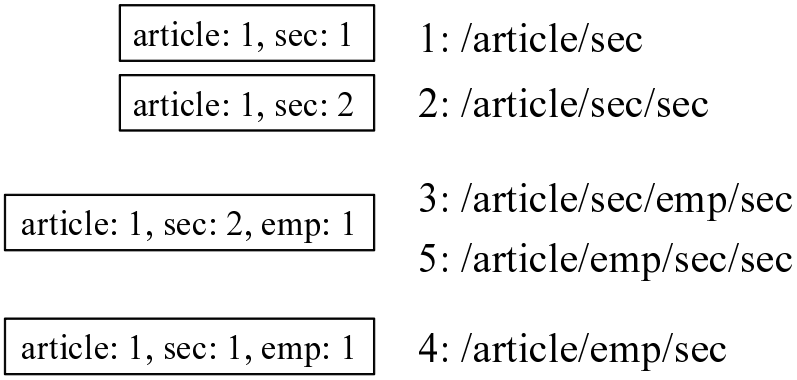


Figure 5.7. An example of classification in BT

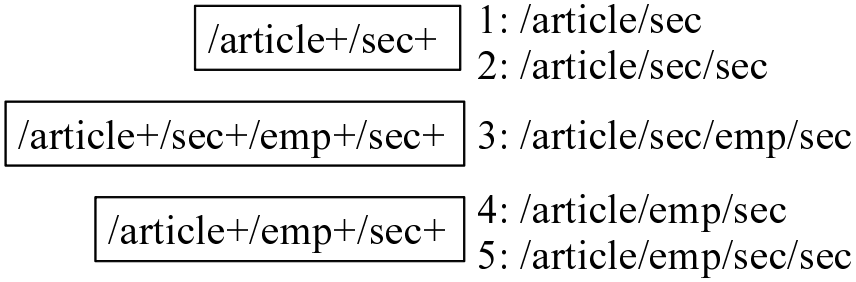


Figure 5.8. An example of classification in OT

two `sec`, and one `emp` tags.

5.2.2.3 Order-of-Tags Method (OT)

The order-of-tags (OT) method relaxes only the appearance frequencies of sequential tags in a path expression. In some path expressions, a tag appears consecutively two or more times; for example, `col` tags in `table` of HTML. In this case, even if the frequencies of a tag appearing consecutively are different, we suppose that the features of a path expression are not much different because the semantics of each tag are fixed. Therefore, if consecutive tags are the same, such tags can be aggregated.

Classification of the path expressions in Figure 5.5 is shown in Figure 5.8. Note

that `sec` tags appear consecutively in the second and fifth path expressions. The first and second path expressions are integrated, because these have have one or more `article` tags followed by one or more `sec` tags. The fourth and fifth path expressions are also integrated, these have one or more `article` tags followed by one or more `emp` tags, and one or more `sec` tags.

5.3. Experimental Evaluations

5.3.1 Experimental Design

With the simple incremental update system as the baseline, we investigate whether the applying two filters and integrating path expressions are effective for searching accurately and efficient for updating the indices or not.

The experimental environment is the same as that used in Section 5.2. The proposed methods are evaluated using two document sets; one with static statistics, which means that its topics rarely change; and the other with dynamic statistics, which means that new topics are regularly added.

Our proposed approaches admit some variations. There are four ways of calculating global weights: the default method, which is classification based on path expression; the set-of-tags method (ST); the bag-of-tags method (BT), and the order-of-tags method (OT). There are three parameters in the element filter: the element length threshold τ_{el}, the path depth threshold τ_{depth}, and Zipf's threshold τ_{Zipf}. By examining the effectiveness of each approach, we can choose the best setting.

In our experimental procedure, we first ran some preliminary experiments to tune the parameters of the element filter and term filter. Next, with these tuned parameters, we measured the average update time per document, the index size, and the search accuracy for each variation of the proposed methods. We used the document set with static statistics and chose 50% as the ratio of initial documents. Finally, we confirmed the effectiveness of the proposed methods by using the document set with dynamic statistics.

τ_{el}	10	15	20	25	30	35
iP[.01]	.633	.664	.640	.639	.640	.617

Table 5.2. Accuracies with changing τ_{el}

τ_{depth}	3	4	5	6	7
highly ranked elements	.69	.89	.97	.99	1.00
all elements	.19	.44	.69	.88	.96
highly ranked elements (50%)	.76	.93	.98	1.00	1.00
highly ranked elements (30%)	.76	.93	.98	1.00	1.00
highly ranked elements (10%)	.76	.93	.98	.99	1.00

Table 5.3. Depth of Path expressions and the ratio of elements

5.3.2 Preliminary Experiments for the Element Filter and Term Filter

The element filter eliminates the elements that have extremely short elements, extremely deep path expressions, and rarely appearing path expressions. In this section, we describe some experiments that we conducted to decide the thresholds.

According to the results listed in Table 5.2, we set τ_{el} to 15; namely, in terms of search accuracy, the best value for the element-length threshold is 15. This result agrees to our hypothesis.

Table 5.3 shows the proportion of elements whose depth of path expressions is less than or equal to τ_{depth}. We measured the proportion for highly ranked elements, concretely top 1,500 elements, obtained in our previous study described in Chapter 4, and for all elements in the test collection. We set τ_{depth} to 6, because most of highly ranked elements are less than or equal to 6. This means that we ignores any element whose depth is six or more. Since the depth of 12% of all elements is more than or equal to 6, these elements can be removed.

We additionally conducted a experiment for exploring τ_{depth} dynamically. We measured three more variations of the same statistics. These are calculated with 50%, 30%, and 10% of all documents. As we can see from the result, relatively precise results can be gained even in the early stage of document accumulation.

n	*no filter*	1500	5000	10000	30000
iP[.01]	.639	.629	.631	.641	.635
update time (ms/doc)	53.4	21.5	36.9	48.8	69.4

Table 5.4. Effects of the term filter with changing n

We also investigated the threshold of Zipf's law. We computed median frequency of the path expressions by using Eq. 5.1. We set τ_{Zipf} to 166, which ignores any element whose path expression appears 166 or fewer times in the initial index.

In analogy with the element filter, the term filter eliminates terms whose weights are below the threshold. We conducted an experiment to decide the threshold for the term filter by measuring iP[.01] and update time with changing n, as shown in Table 5.4. Since search accuracy did not decreased and update time was faster than that of *no filter*, we set n to 10000 or τ_{tw}, which ignores the terms whose weights are less than the 10,000th largest weight of each pair of tag and term.

5.3.3 Evaluations of the Document Set with Static Statistics

We measured the average update time per document, the size of indices, and the search accuracy with each variation of the proposed methods, as indicated in Table 5.5. Note that in the case of *no-update*, or constructing a new index from scratch, the average update time replaces the construction time of the initial indices. Note that elements whose length is less than τ_{el} are removed from all results, even those of *no-update*[3].

5.3.3.1 Effects of the Proposed Methods

Compared with the iP[.01] of the baseline system, those of ST, BT, and OT are improved. In particular, ST is the most effective method for calculating accurate global weights and is 2.57% more accurate than the baseline. In addition, the

[3]This is the reason why accuracy of *no-update* is different from baseline in Chapter 4.

run ID	update time (ms/doc)	disk size(GB)	iP[.01]	MAiP
no-update	(42.1)	111	.664	.213
baseline	53.4	111	.639	.200
rnd_flt (10%)	49.8	107	.635	.168
rnd_flt (20%)	45.1	100	.603	.154
rnd_flt (30%)	41.2	95	.612	.148
rnd_flt (40%)	37.2	92	.612	.141
rnd_flt (50%)	33.8	88	.594	.137
rnd_flt (60%)	30.0	77	.569	.137
ST	53.3	111	.655	.199
BT	53.2	111	.652	.206
OT	53.2	111	.653	.207
τ_{el}	45.5	97	.646	.202
τ_{depth}	49.1	106	.651	.204
τ_{Zipf}	51.1	109	.649	.198
elem filter	40.1	94	.651	.204
term filter	48.8	104	.641	.196
two filters	39.3	89	.652	.201
ST_filters	40.1	88	.662	.204

Table 5.5. Effects of the proposed approaches

update efficiencies of these methods are almost equal.

All components of the element filter (i.e. τ_{el}, τ_{depth}, and τ_{Zipf}) save update cost without reducing search accuracy. The combination of τ_{el}, τ_{depth}, and τ_{Zipf} is the most effective of all possible combinations and yields 23.6% faster updates than the baseline approach. We used this setting for the element filter in the subsequent experiments.

ㄴ Hereinafter, we also investigated performance of a random filter (rnd_flt) which removes updated elements randomly for proving effectiveness of the element filter. The random filter varies a cut-off rate (a percentage of removed element) from 10% to 60% at interval of 10%. As the cut-off rate increases, up-

date efficiency improves and disk size of the indices reduces, while search accuracy decreases. When comparing the element filter with the random filter at the same efficiency level in regard to update time, concretely cut-off rates are 30% and 40%, the element filter achieves more accurate search performance. From this result, usefulness of the element filter is obvious.

The term filter also reduces the update cost by 6.70% without sacrificing search accuracy compared with the baseline approach. Next, we evaluated the combination of the two filters. This approach performs better than either of single filters in terms of both update efficiency and search accuracy. The update efficiency is improved by 26.3%.

We confirm whether the proposed two filters affect recall or not because some elements are removed through the two filters. Figure 5.9 is a precision-recall curve of the baseline and two filters. As we can see from the figure, these three methods draw almost the same curves. Consequently, search accuracy is little affected by the two filters.

The former experiments showed that the search accuracy improved with the path expression integrating method and the update efficiency improved with two filters. Then, we combined ST and the two filters as ST_filters. The search accuracy improved by 3.73% compared with the baseline, while the update efficiency improved by 24.9%.

In terms of query efficiency, each method takes 1.5 seconds to 2.0 seconds per query. This should be acceptable for users. Finally, we can attain fast incremental updates of indices with an effective and efficient search.

5.3.3.2 Search Accuracy Combined with Reconstruction Method

The best setting of the proposed method, ST_filters, is less effective than *no-update*, although search accuracy is improved with the proposed methods. Thereby, we adapted the reconstruction method proposed in Chapter 4 into the incremental update system to improve search accuracy more.

We investigated search accuracies of *no-update*, baseline, *SIXE* with ST_filters, *BU* with ST_filters, *TD* with ST_filters, and *BU-TD* with ST_filters. Note that we set EL as 1000, which is the same value used in Chapter 4.

Table 5.6 showed *SIXE* (ST_filters) and *BU* (ST_filters) overwhelmed in terms of search accuracies of iP[.01], while *TD* (ST_filters) and *BU-TD* (ST_filters)

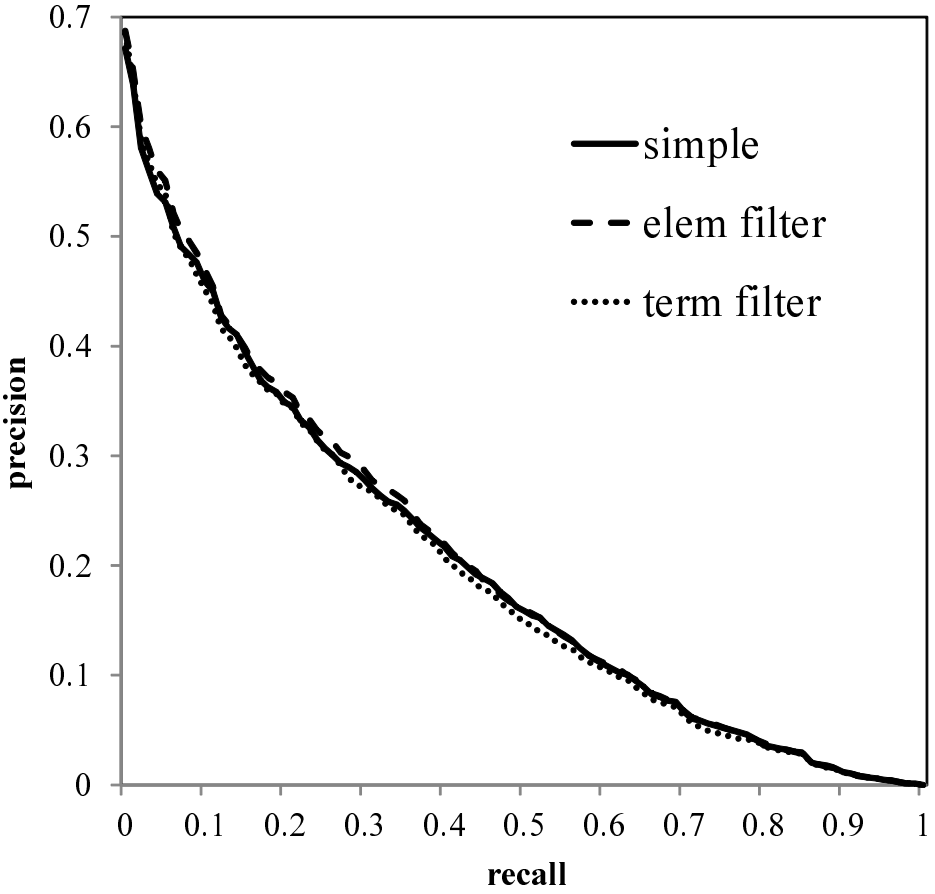


Figure 5.9. Effect on recall by the two filters

decreased their search accuracies.

At last, we compared the proposed method *BU* (ST_filters) which has the highest accuracy with other INEX participants. As a result, none of them is superior to the proposed method at the value of iP[.01]. The advantages of the proposed method are that not only it searches accurately but also it reflect document updates in a short time.

Regarding query efficiency, reconstruction process take approximately 0.5 seconds. The total query processing time ends up 2.0 seconds to 2.5 seconds, which still keeps a practical use level.

The conclusion is that the combinations of the reconstruction method, path expression integration, and the two filters achieve accurate search accuracy and

	iP[.01]	MAiP
no-update	.670	.214
ST_filters	.662	.204
SIXE (ST_filters)	.671	.227
BU (ST_filters)	.674	.217
TD (ST_filters)	.653	.215
BU-TD (ST_filters)	.653	.215

Table 5.6. Effects of a reconstruction method

Team	iP[.00]	iP[.01]	iP[.05]	iP[.10]	MAiP
BU (ST_filters)	.6898	.6736	.5647	.4789	.2167
Renmin Univ. of China	.5969	.5969	.5815	.5439	.2486
Queensland Univ. of Tech.	.6232	.6220	.5521	.4617	.2134
Univ. of Amsterdam	.6514	.6379	.5901	.5280	.2261

Table 5.7. Comparison with other INEX participants

fast query processing with immediate updates of indices.

5.3.4 Evaluations of the Document Set with Dynamic Statistics

In the previous evaluations, we assumed that the term distribution and term statistics are static. However, new topics can emerge suddenly on the Web and may change the term distribution drastically. Here we artificially assemble a document set with dynamic statistics to investigate the effectiveness of the proposed methods.

In this set, the initial documents do not include a certain topic but the updated documents do include the topic. We outline the steps to evaluate as follows: (1) identify documents on a certain topic, (2) construct the initial index using the other documents, and (3) update the indices incrementally using the documents related to the topic.

We utilized the categories in Wikipedia to judge whether a document belongs

Category name	CQ	CW
Technology and applied sciences	18	54
Culture and the arts	20	51
Natural and physical sciences	9	24
Society and social sciences	4	13
History and events	4	11
Philosophy and thinking	3	8
General reference	3	7
Health and fitness	2	7
People and self	3	6
Geography and places	2	5
Mathematics and logic	0	0
Religion and belief systems	0	0

Table 5.8. Category and Query

to a certain topic. Wikipedia has many categories of various sizes: twelve major categories are listed in Table 5.8. We separated 68 queries into the twelve categories. Each query contains from one to five query keywords, and we obtained a keyword set for each category. Since the categories "Technology and applied sciences" (*technology* for short) and "Culture and the arts" (*culture* for short) include relatively large numbers of queries (category queries, or CQs) and query keywords (category keywords, or CW), we used these categories in the evaluation. We assigned a document to a certain category if the document contains the category keywords. Note that these category keywords are stemmed.

- **Category words of *technology***
  aircraft, applied, automobil, aviat, bay, bletchlei, break, car, code, colossu, compani, comput, databas, detect, engin, expert, file, filter, format, graphic, imag, inform, instal, intrus, invent, java, languag, linux, manag, mechan, metadata, mine, motor, museum, network, nikola, open, oper, park, patent, program, raid, record, retriev, rotari, secur, social, sourc, storag, system, tata, tesla, virtual, wireless

- **Category words of *culture***

# of indexed doc. ($\times 10^4$ doc.)	*technology* (iP[.01])	
	baseline	ST_filters
37 (25% updated)	.524	.532
47 (50% updated)	.525	.548
56 (75% updated)	.546	.563
66 (100% updated)	.578	.592

# of indexed doc. ($\times 10^4$ doc.)	*culture* (iP[.01])	
	baseline	ST_filters
31 (25% updated)	.456	.517
43 (50% updated)	.506	.560
54 (75% updated)	.501	.585
66 (100% updated)	.496	.587

Table 5.9. Effects on emerging a new topic

acquisit, africa, al, basketbal, berber, bilingu, childbirth, children, classic, countri, cultur, danc, dish, europ, european, fiction, film, food, franc, game, guitar, hors, instrument, japanes, keyboard, languag, mahler, museum, nba, north, person, picasso, player, portugues, produc, region, rule, scienc, scrabbl, song, spanish, style, symphoni, tap, tast, terracotta, tradit, typic, vegetarian, vodka, wine

We used the category queries only to examine the effectiveness of the proposed methods, because we focus on the effects of term distributions with dynamically changing statistics. In this situation, we assumed that users expect an effective search to be available as soon as new topics are added to the collection.

The numbers of documents in the initial indices of *technology* and *culture* are 280,000 and 200,000, respectively. We evaluated the effects of the changing statistics at four points during the updates. After the updates, the number of indexed documents reached 660,000 for both categories.

Table 5.9 lists the iP[.01] of each category for the baseline and ST_filters. For both categories, the proposed methods attained better search accuracies than the baseline. In particular, ST_filters increased the search accuracies rapidly even when the number of update documents was small.

5.4. Further Discussion

From the results of the experiments, it turned out that the proposed system update in minute time without decreasing search accuracy even though statistics of terms drastically change. However, it has some limitations; 1) weights of terms

are not exactly accurate and 2) the size of indices become enlarged along with document insertion, deletion, and modification. These matters imply that rebuilding indices from scratch periodically is also required as well as incremental updates of indices. Therefore, come to think of practical use scenario, two series of indices are constructed where one series of indices is incrementally updated during rebuilding the other series of indices from scratch.

Henceforth, we discuss the scalability of the proposed method. In building a scale-out XML element search system, we suppose the situation as follows:

- Data of great magnitude are dispersed in multiple nodes. According to assignment rule, a part of data is stored into a certain node. As an example of the assignment rule, the assigned node of a term is decided by the initial letter of the term.

- There exists a data storage such as BerkeleyDB on the disk of each node. Search index can be used only within a node, although the search index cannot be used beyond the node. We suppose that the structure of the storage is the same as the proposed index structure. In addition, data is sorted by descending order of term weight with using the search index.

- Since the amount of main memory of each node is limited, the main memory can be used only for cache of the data storage. Note that it is possible to allocate some area of the main memory for version management of data. Furthermore, the main memory also contains global weights related to a certain term which is mapped to the node, as well as threshold values for the term filter.

- Since the performance of computational capacity of each node is limited, a finite number of requests are processed per second with each node.

In this situation, every single node of the distributional system need to satisfy two requirements as described below:

- **Requirement 1**
  Update process need to be handled rapidly. The update process include term calculation and updates of data storage, and

- **Requirement 2**
  There is a possibility that data in multiple nodes is utilized in query processing. Even that situation, query processing with short response need to be achieved by efficiently collecting data widespread throughout the multiple node.

With regard to the Requirement 1, when a document is inserted, elements are obtained from the document. Only elements which get through the element filter are treated as update targets. Each term in the elements with related information, i.e., local weights, is delivered to a certain node corresponding to the term. In the node, a weight of the term is calculated with the global weights contained in the main memory. For efficient storage updates, unimportant terms are eliminated with the term filter. In consequence, fast term calculation is achieved with the proposed index structure, while efficient updates of storage is accomplished with the proposed filters, namely, the element filter and the term filter.

Concerning the Requirement 2, a node request other nodes to read/write data when the node refer to data in other nodes. Under this circumstances, common tasks such as generating search results from collected data and deciding a node for a term to be stored when document updates take place are executed on a lightly load node. Accordingly, load balancing is distributable.

In reading data from storage, sequential scan is mainly adopted because it is expected that the performance of random scan is quite inefficient in this scenario. Thus, search index in each node play an important role. Consequently, efficient Top-k search is attained for a single term. In other words, a set of terms sorted by their weights can be gained by term. Then, calculating element scores with the term weights and proposing search results are executed on a lightly load node as we mentioned above.

One more challenge remains to be unsolved. It is how to efficiently manage conflict when multiple requests try to access the same node. One solution is that a data is forbidden to be accessed by two or more processes at the same time. To carry out this solution, the status of version information of a data in the main memory changes to *in use* while the data is read/written.

As a result, the proposed methods can be applied for satisfying these requirements with minor change. The conclusion is that the proposed methods have scale-out capability.

CHAPTER 6

Expansion of XML Element Retrieval into HTML Documents

In the past, XML element retrieval techniques were investigated with scientific articles and Wikipedia articles in the INEX project. Consequently, an accurate XML element retrieval system is coming true. As a next step, since XML is not the only data format which is applicable with (XML) element retrieval, it is expected that these techniques are adapted into Web document which is also one of structured documents as same as XML document. A scope of XML element retrieval is spread drastically if an effectiveness for HTML documents is proved, because Web document is a very common data format.

In expanding XML element retrieval techniques into HTML document which is a representative data format of Web documents, characteristics of XML and HTML are quite different because of their intended purposes. We enumerate characteristics of HTML document as follows:

1. many HTML documents are not invalid in tag consistency,
2. logical structure in terms of content and physical document structure do not agree with each other, and
3. there are many parts which do not satisfy users' information need.

Concerning (1), consistency of start and end tags need to be assured (every start tag corresponds to its end tag) when expanding XML element retrieval techniques into structured documents for treating elements properly. Generally speaking, most of XML documents are well-formatted in terms of tag consistency because they are mainly used for data management. In fact, many of XML parsers only accept XML documents of which tag consistencies are well-formatted. In conversely, the majority of HTML documents are not perfectly well-formatted in their tag consistency. This is caused by the fact that ordinary Web browsers interpret and display a HTML document even though tag consistency of the HTML document. To resolve the problem, a tag balancer tool is utilized often. One of the most well used tag balancer tools CyberNeko HTML Parser [119] which automatically complements the incomplete tags.

In regard to (2), many of hand written documents (even some of automatically generated documents) have logical document structure of content. For example, a document is composed of some chapters. Likewise, each chapter has some sections, and each section also has some sub-sections. This chapter composition information is definitely useful to understand document content. XML element retrieval techniques exploit these information to identify an element which satisfies user's information need, because a document structure of an XML document is fundamentally based on a logical document structure of content. In other words, a document structure need to be defined according to logical document structure of content.

In contrast, most of HTML tags perform text decoration, e.g., changes of font size, color, and style. In consequence, a document structure of HTML document does not reflected logical structure of content, which prevent expansion of XML element retrieval techniques into HTML documents.

Related to (3), a HTML document is often composed of multiple parts (elements). Concretely, a typical structure of Weblog articles is shown in Figure 6.1. It contains not only main–content but also further information such as page title, a list of past entries, a list of link information, a category information of entries, and a set of tag cloud. Hereinafter, we call these further information as sub–content.

Generally, these sub–content are just hyperlinks to other documents. Thus, users' information need is not satisfied only with these elements, although these are

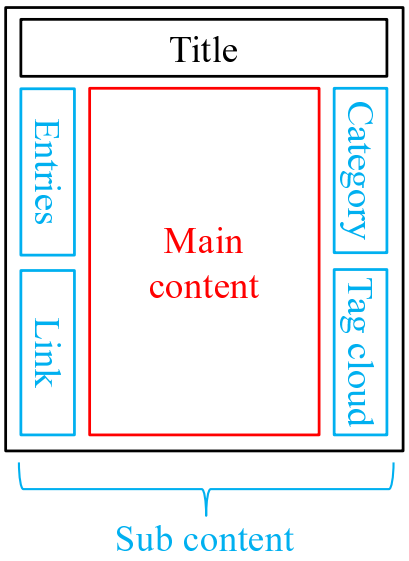


Figure 6.1. Typical structure of a Web document

useful for exploratory information retrieval. Nevertheless these elements should not be ranked high in search results, these elements where an unitary term often occurs frequently tend to be assigned high relevancy score with a term-weighting-based scoring function. Therefore, we need to identify such an element composed of sub-content to be removed from search results. Note that we mainly focus on sub–content which harms search performance although there are some other sub–content such as a profile information of the author, a calendar, and a search box.

To resolve the problem raised by the characteristics of HTML documents, we format HTML documents; 1) reconstructing document structure according to logical structure of content, and 2) removing unimportant elements.

Concerned with 1), a study [110] reports that some HTML tags represents a boundary between one topic to another. Based on knowledge from the study, we reconstruct HTML documents for achieving high agreement between a physical document structure and a logical structure of content.

As a solution for 2), we leverage occurrence information of `A` tags to eliminate content outside of main body, which is based on an hypothesis that elements described for the purpose of transferring to other documents have more `A` tags.

6.1. Reconstruction of a Document Structure

In this section, we will discuss the reconstruction method to resolve the disagreement between a logical structure and a physical structure of the document.

A logical structure of a document is composed of table of contents like chapters, sections, and paragraphs. In contrast, only the granularity of paragraph can be defined with `P` in HTML documents. This causes a disagreement between the logical structure and physical structure.

On the other hands, it is reported that Heading tags (`H1-H6`) perform a boundary between one topic to another [110]. We suppose that a level of Heading tag (the number in a Heading tag) somewhat expresses a logical structure of content, because a level of Heading tag is set along with a degree of importance of content[1]. In short, we hypothesize that a more important Heading tag intends a larger granular topic while a less important Heading tag intends a smaller granular topic.

Information between a pair of start and end Heading tags (we call this as a Heading tag, for short) is just title of a heading. It is general that content just after a Heading tag is about the title of the heading. We therefore reconstruct a HTML document according to a level of a Heading tag, because we expect that we can extract a logical structure of content with Heading tag information. Note that we insert a Container Heading tag for the goal.

- **Start tag**
  When a Heading tag (`H`x) appears, a start Container Heading tag (`CH`x) is inserted just before the Heading tag.

- **End tag**
  When a level of a newly appeared Heading tag (`H`x) is the same or smaller than that of previously appeared Heading tags, an end Container Heading tag (`CH`y) is inserted just before a start Container Heading tag (`CH`x). Note that the level of `BODY` is smaller than any Heading tag.

We show a concrete example of reconstruction process with Figure 6.2. The original HTML document of left figure is reconstructed into the HTML of right

[1]The more important content is, the smaller the number is.

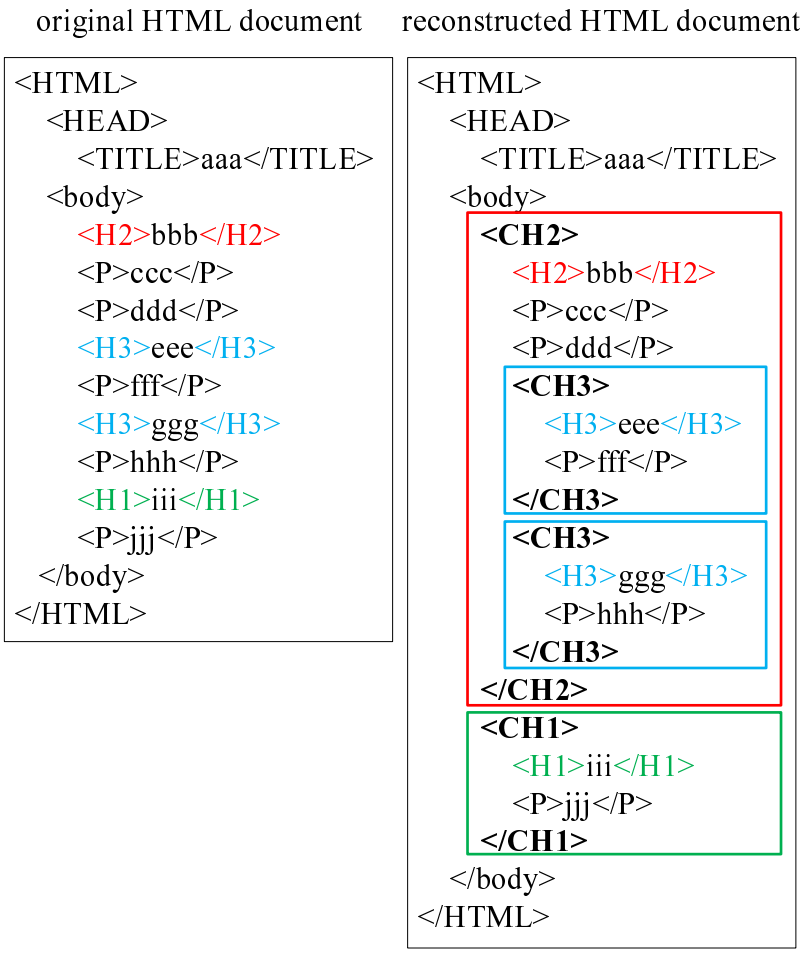


Figure 6.2. Reconstruction HTML documents

figure. There are four Heading tags in the HTML document, i.e., H2, H3, H3, and H1 tags. First of all, a H2 tag appears. In this time, a start CH2 tag is inserted just before the H2 tag. Next, a start CH3 tag is inserted just before the H3 tag, because the level of the H3 tag is larger than that of the previously appeared H2 tag. Then, a H3 tag appears again. Since the levels of the newly appeared H3 tag and the previously appeared H3 tag are same, an end CH3 tag is inserted followed by a start CH3 tag is inserted. A H1 tag comes last. The level of the H1 is smaller than the previously appeared H2 and H3 tags. Accordingly, end CH3 and end CH2 tags are inserted followed by a start CH1 tag. The reconstruction process is completed when an end CH1 tag is inserted just before an end BODY tag.

Some other tags such as UL, OL, DL, TABLE, FORM, and DIV have possibilities that they are also regarded as structural tags. Managing these tags is one of our future work.

Likewise, BR and HR separate contents, which may be a clue of reconstruction. This is also a part of our future work.

6.2. Eliminating Elements of Outside of Main Body

Elements composed of sub–content, or a list of past entries, a list of link information, a category information of entries, and a set of tag cloud, do not satisfy users' information need. We try to propose a filter which removed these uninformative elements.

Removing elements with small element length is enough for filtering out unimportant elements with the scientific articles and the Wikipedia articles [49]. On the other hand, elements we are trying to remove here are not always small in their element length. We need to identify with another approach.

Come to think of trends of elements composed of a list of entries or a list of link information, these are hyperlinks for transferring to other pages. Since these elements are expected to contain many A tags, we remove elements which contain many A tags as sub–content. We enumerate candidate statistics for measuring a quantity of A tag.

- **A probability of occurrence of A tag:**
  A frequency of A tag normalized by element length.

- **A ratio of nodes rooted with A:**
  A ratio of the number of elements rooted with A tag to the total number of the elements in an XML tree.

- **A ratio of text size in A tag:**
  A ratio of text size in A tag to the total text size of the element.

With regard to the first statistics, it is not always true that every element with high probability is sub–content. For example, such main–content that includes many figures linking to original figures may be misjudged as sub–content. Thereby, this measurement is not the most appropriate one.

Concerned with the second statistics, elements contain many `A` tags are judged as sub–content. However, not every element is marked up with other tags besides `A` tag. Since it is natural that main–content contain hyperlinks, the second statistics also may cause misjudgment.

Let us move to the third statistics. It considers only `A` tags which are used for hyperlinks. Additionally, the statistics is independent of other tags besides `A` tag. Therefore, it is expected that the statistics judge sub–content properly. This is the reason why we adopt the third statistics in our proposed filter.

A sub–content score of element e, $S_A(e)$, is calculated as follows:

$$S_A(e) = \frac{C_A}{C_e} \tag{6.1}$$

where C_A is the total number of characters in all descendant elements rooted with a `A` tag of e, C_e is the total number of characters in e. Note that elements of which score exceed $\tau (0 \leq \tau \leq 1)$ are removed as unimportant elements.

6.3. Experimental Evaluations

In this section, we report the effect of the proposed reconstruction method and set the threshold value for the proposed sub–content filter, followed by evaluating search performance of the proposed methods.

Since we would like to confirm the effectiveness of XML element retrieval techniques for Web documents, we used Web (HTML) documents and Web queries for the experiments, i.e., 1CLICK-2 test collection [56].

We go through following pre-processes. Note that former three processes are the same as the ones for XML documents.

1. removing attributes, comments, and special characters of HTML documents,
2. removing the stop words by SMART stop list (listed in Appendix **A**)
3. applying stemming step by Porter [11].
4. validating corresponding relations in tags with CyberNeko HTML Parser [119].

Tag name	the number of tags (%)
HC1	33217 (.092)
HC2	113300 (.31)
HC3	116787 (.32)
HC4	60276 (.17)
HC5	26819 (.074)
HC6	10880 (.030)

Table 6.1. Inserted Heading Container tag

degree of deepening	the number of documents (%)
0 (not deepened)	3639 (.14)
1	5194 (.19)
2	8300 (.31)
3	6912 (.26)
4	2509 (.093)
5	346 (.013)
6	28 (.0010)

Table 6.2. Degree of deepening of document structures

In the same manner as the former chapter, the PC that we used for the experiments runs Oracle Enterprise Linux 5.5. It has four Intel Xeon X7560 CPUs (2.3GHz), 512GB of memory, and a 4.5TB disk array. The indices were implemented using BerkeleyDB in GNU C++.

6.3.1 Evaluations of the Reconstruction Method

We investigated how document structures changed along with the reconstruction method. Table 6.1 shows the numbers and percentages of inserted Heading Container tags. These number correspond to the frequencies of Heading tags. As we can see from the table, `H2` and `H3` tags appeared frequently. Note that the average number of inserted Heading Container tag per document is 13.4.

Next, we examined the degree of deepening of document structure with Head-

sub–content score	accuracy
0.1-0.3	50%
0.3-0.5	60%
0.5-0.7	80%
0.7-0.9	100%
0.9-	100%

Table 6.3. Discriminant accuracy of the sub–content filter

ing Container tags. To show some tangible examples, the degree of deepening of a document is 2 when the document includes `H2` and `H3` tags, while that is 6 when Heading tags appear in order from `H1` to `H6`. We report degrees of deepening of document structures and their numbers and ratios in Table 6.2. Heading tag is appeared in more than 80% documents, and the average number of the degree of the deepening is 2.23.

It turned out that new granular elements apart from paragraph are defined by focusing Heading tags. However, this also means an increase of search targets. Compared with the original documents, 25% of more elements are generated. We need to consider the way to control the explosion of newly generated elements when we target Web scale. It remains one of our future work.

6.3.2 Evaluation of Sub–content Filter

We explored whether the sub–content filter can remove sub–content such as a list of past entries, a list of link information, a category information of entries, and a set of tag cloud, or not.

We randomly extracted 10 elements with each range of the sub–content score as Table 6.5 shows. We manually judged if each element is main–content or sub–content. In consequence, every element of which sub–content score is higher than 0.7 is judged sub–content. We therefore set 0.7 as threshold value τ for the sub–content score. Note that 9% of elements are removed as sub–content.

6.3.3 Effect of XML Element Retrieval Techniques for HTML Documents

6.3.3.1 Experimental Design

We used the 15 queries of DEFINITION type which aim to extract relevant information exhaustively. This is because we suppose that the type is relatively compatible with element retrieval compared with a query which has obvious information to be extracted.

In generating search results, we firstly calculate a relevancy score of each element with BM25E [73]. Note that we adopt the tag-based approach for global weight calculation, because the number of documents is not enough for calculating accurate global weights with the path expression-based approach. In addition, we eliminate elements shorter than 15 in their length, and apply the integration method proposed in Chapter 4 when an overlap occurs.

Next, we explain the evaluation method. We manually extract relevant descriptions from documents that search results belong to. During the process, we did not assign the importance of descriptions but evaluate whether the descriptions are relevant or not[2].

Evaluation measures are agreement rate (AR), exhaustiveness rate (ER), F–measure of AR and ER (FM), average text size, and standard variation. Let e be a retrieved element as search result and D_e be a document that e belongs to. The value of agreement rate, exhaustiveness rate, and harmonic average are calculated as follows:

$$AR = \frac{size_{e,r}}{size_e} \tag{6.2}$$

$$ER = \frac{size_{e,r}}{size_{D_e,r}} \tag{6.3}$$

$$FM = \frac{2 \cdot AR \cdot ER}{AR + ER} \tag{6.4}$$

where $size_{e,r}$ is text size of relevant descriptions in e, $size_e$ is text size of e, $size_{D_e,r}$ is text size of relevant descriptions in D_e. Note that these values are average of 15 queries.

[2]The process is maximally conformed to that of INEX based on the experiments that the author have participated the INEX project for several years.

	@1			@5			@10		
	AR	ER	FM	AR	ER	FM	AR	ER	FM
ELEM	.503	.233	.176	.498	.305	.298	.512	.351	.344
REC	.517	.282	.230	.545	.372	.424	.586	.360	.418
DOC	.382	.800	.458	.372	.733	.413	.394	.727	.504

Table 6.4. Agreement rate, exhaustiveness rate, and F–measure

	average text size	standard variation
ELEM	2288.754	2576.560
REC	1717.870	1292.166
DOC	6411.987	1486.383

Table 6.5. Average text size and standard variation at top-10

6.3.3.2 Results of the Experiments

We evaluated search performances at three revels, namely, top-1, top-5, and top-10 as Table 6.5. Comparison methods are element search approach (ELEM) which generates search results in the way described above, reconstruction approach (REC) which applies the reconstruction method with ELEM, and document search (DOC). Note that both ELEM and REC adopt the sub–content filter.

The results tell that the agreement rates of both ELEM and REC are higher than that of DOC at all revels. On the other hand, the exhaustiveness rate of DOC is higher than those of ELEM and REC. Meanwhile, largely F–measure of DOC is the highest except that F–measure of REC at top-5 is higher than that of DOC. In consequent, the element retrieval approach is good at agreement rate.

Moreover, both agreement rate and exhaustiveness rate improved at all levels with the reconstruction method. As a result of multiple comparison, there is statistical difference between REC-ELEM at significance level 5%, while there are statistical differences between both REC-DOC and ELEM-DOC at significance level 1%.

In terms of average text size and standard variation at top-10 as we show in Table 6.5, the average text size of DOC is much larger than those of ELEM and

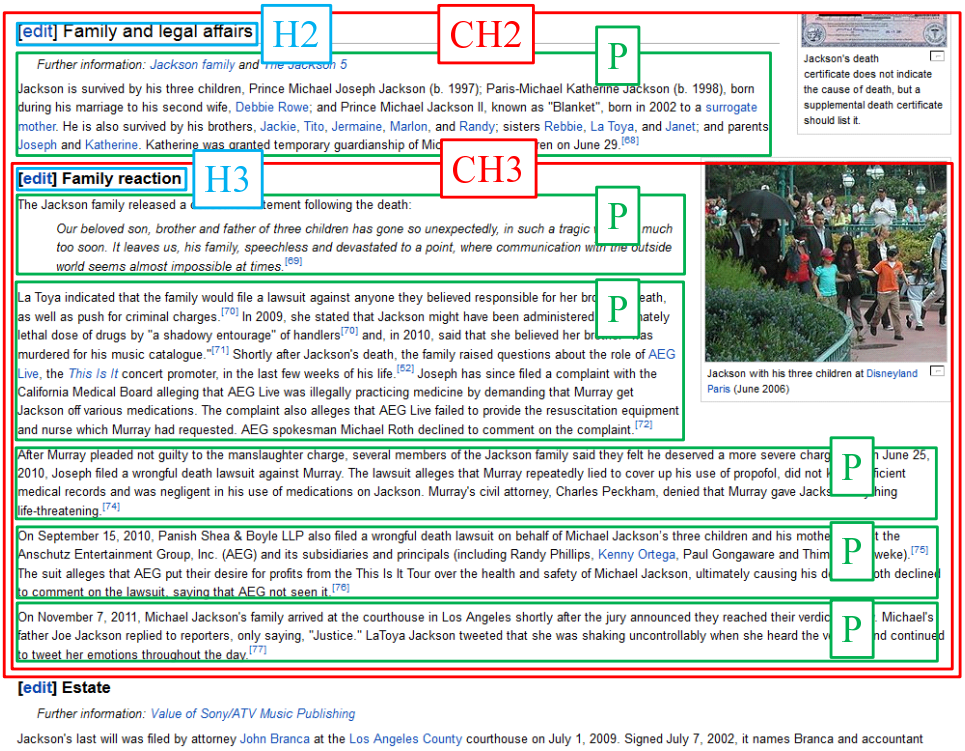


Figure 6.3. Multiple paragraphs are aggregated into one element

REC. The reason why the average text size and the standard variation of ELEM is higher than those of REC is that entire documents are returned as search results with ELEM when appropriate granular elements cannot be returned. Since the less the amount of retrieved search results become, the less laborious, REC is more helpful for users.

6.3.3.3 Positive and Negative Examples of the Reconstruction Method

As a result of experiments described in the previous section, the reconstruction method largely improved search accuracy. Figure 6.3 depicts the example of positive effect of the reconstruction method. Newly defined elements, i.e., `CH2` and `CH3` are composed of multiple paragraphs, which enables the proposed system to return middle granular elements.

In more detail, as shown in Figure 6.4 which depicts the simplified source code of the document, the document has flat structure. To extract descriptions satis-

```
<H2>Family and legal affairs</H2>
<P>Further information: Jackson family and The Jackson 5<P>
<P>Jackson is survived ... Michael's three children on June 29.<P>
<H3>Family reaction</H3>
<P>The Jackson family released ... following the death:</P>
<P>Our beloved son, ... almost impossible at times.</P>
<P>La Toya indicated that ... comment on the complaint.</P>
<P>After Murray pleaded ... anything life-threatening.</P>
<P>On September 15, 2010, ... saying that AEG not seen it.</P>
<P>On November 7, 2011, ... her emotions throughout the day.</P>
```

Figure 6.4. Simplified original HTML document

```
<CH2>
  <H2>Family and legal affairs</H2>
  <P>Further information: Jackson family and The Jackson 5<P>
  <P>Jackson is survived ... Michael's three children on June 29.<P>
  <CH3>
    <H3>Family reaction</H3>
    <P>The Jackson family released ... following the death:</P>
    <P>Our beloved son, ... almost impossible at times.</P>
    <P>La Toya indicated that ... comment on the complaint.</P>
    <P>After Murray pleaded ... anything life-threatening.</P>
    <P>On September 15, 2010, ... saying that AEG not seen it.</P>
    <P>On November 7, 2011, ... her emotions throughout the day.</P>
  </CH3>
</CH2>
```

Figure 6.5. Simplified reconstructed document

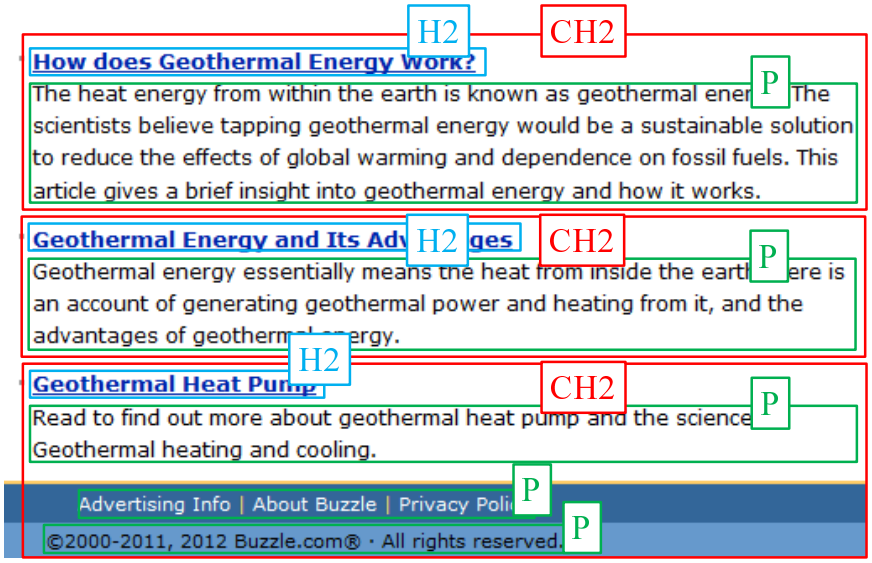


Figure 6.6. Newly generated element including sub–content

fying users' information need, it is the often the case that each sentence denoted as `P` tag is not useful. On the other hand, Figure 6.5 shows the the simplified source code of the reconstructed document. Consequently, newly defined granular elements, namely `CH2` and `CH3`, larger than paragraph and smaller than entire document can be returned as a search result.

On the other hands, the reconstruction method causes harmful effect in some cases. Concretely, the newly generated element generated by the Heading tag appearing last in a document may include sub–content and footer information. This decreases search performance. To avoid this, we need to regulate these unimportant parts as a part of our future work.

Moreover, we observed one more problem that we cannot not remove some sub–content with the sub–content filter from reconstructed documents. It seems that the appropriate threshold value for the sub–content filter changes by granularity of element. Precisely, even elements composed of sub-content may have lowly sub–content score for larger granular elements. There are some candidate approaches for resolving the problem, namely, smoothing with element length, utilizing rendered visual information, and machine learning for discriminating sub–content.

CHAPTER 7

Conclusions

7.1. Summary of this thesis

In this thesis, we worked on developing a practical XML element search system as well as a practical Web search system. There are three requirements for the practical XML element search system, namely, accurate search, fast query processing, and fast reflection of document updates. In addition, one more requirement is needed for the practical Web search system, that is, an expansion of XML element retrieval techniques into Web documents.

As main contributions towards this requirements, we proposed a method for accurate XML element retrieval and fast and incremental update method. We also proposed a method to be easily applied XML element retrieval techniques.

With regard to the accurate XML element retrieval, existing studies mainly focus on proposing accurate term-weighting scheme extended by document retrieval term-weighting schemes. This means that these existing studies calculate element score independently and do not consider overlapping relationships in an XML document. Moreover, there is no adequate study which discusses the way of returning non-overlapped ranked list from a simple ranked list. Accordingly, we proposed a scoring method to identify informative XML elements, a reconstruc-

tion method to identify the best granularity of XML elements as search results, and a scoring method to find informative elements with considering statistics of ancestor or descendant element. Our experimental evaluations showed that our proposed methods exceeded existing methods in search accuracy. Particularly, the reconstruction method improved search accuracy most.

Concerned with the fast and incremental update method, there are two goals for that. One is achieving fast incremental updates of index and the other is to calculate term weights accurately. Regarding the first goal, existing studies need to rebuild index from scratch because existing studies do not consider document updates. It takes long time when the number of documents accumulated in a search system is large. Moreover, update target cost is expensive when all data is treated as update targets. Therefore, we proposed a index structure for enabling incremental updates. We also proposed the element filter and the term filter which eliminate unnecessary elements and terms, respectively. With the index structure and these filters, we attained fast incremental updates.

Concerning second goal, accurate term weight cannot be computed when the number of documents is insufficient or statistics of a document set change drastically. This is caused by inaccurate global weights. We therefore proposed an estimation method for accurate statistics even when the statistics of a documents set changes drastically.

As a result of this research, we developed an XML element search system which achieves accurate search and fast query processing while satisfying fast updates of documents. In other words, we successfully developed a practical XML element retrieval system.

Furthermore, we worked on accomplishing a practical Web search system. For the purpose, we adapted XML element retrieval techniques to HTML documents to develop an HTML element search system. There are some features of HTML documents compared with XML documents, i.e., the document structures of HTML documents are not well-formatted, there is disagreement between the logical structure of the document contents and physical document structure, and HTML documents contain many of uninformative contents. These features may prevent adapting XML element retrieval techniques to HTML documents. Since the first feature is resolved with the existing tool, we proposed a method to reconstruct HTML documents hinted by some tags which represent boundary

of contents for diminishing the disagreement. Similarly, we proposed the filter for eliminating unimportant elements based on the ratio of the hyperlinks of an element. As a result of experimental evaluations, the filter properly removes unimportant elements, while the reconstruction method improved search performance with decreasing text size of search results. Moreover, it is revealed that the framework of element retrieval can return more focused and accurate search results compared with those of document retrieval. On the other hands, the reconstruction method may generate elements including uninformative descriptions, which cause a drop in search performance.

7.2. Future Work

7.2.1 For More Accurate XML Element Retrieval

For more accurate search performance in an XML element retrieval system, integrating a term weighting scheme with a machine learning technique and/or a network-analysis-base scoring function appears feasible. To achieve this, an adaptation method from network-analysis-base scoring functions for XML element retrieval needs to be proposed.

7.2.2 Finding Good Trade-off Between Incremental Update and Rebuilding from Scratch

Regarding an XML element retrieval system with incremental updates, we need to investigate a good time to switch between incremental updates and rebuilding from scratch. With knowledge from the presented research, a more practical XML element retrieval system is expected to be developed.

7.2.3 Developing a Practical Element Retrieval Web Search System

To expand (XML) element retrieval techniques into HTML documents, there are still some challenges to be resolved. These are to investigate the effect of the reconstruction method more fully, to consider other tags besides Heading tags for

identifying a logical structure of the documents, to propose a refined sub–content filter which remove even large granular sub–content.

Furthermore, in expanding element retrieval techniques into the Web in a real sense, not only element retrieval techniques are applicable to HTML documents, but also an enormous number of documents, literally Web scale, need to be managed in an efficient way. A solution for to this is adapting the proposed method to a large scale distributed environment for scaling out. Furthermore, in an XML element retrieval system, there are many processes, in each of which sub-processes are executed independently. Accordingly, it seems that it is effective to execute these processes with Graphics Processing Units (GPU) which have higher performance in parallel processing compared with CPU for scaling up.

Publication List

Journal Papers

1. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "A Proposal of Accurate XML Element Retrieval Considering Document Updates", IPSJ Transactions on Databases (TOD), Vol. 6, No.4, pp.1–16, September 2013.

2. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "Fast Incremental Indexing with Effective and Efficient Searching in XML Element Retrieval", International Journal of Web Information Systems (IJWIS), Vol.9, Iss.2, June 2013.

3. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Result Reconstruction Approach for More Effective XML Element Search", International Journal of Web Information Systems (IJWIS), Vol.7, Iss.4, pp.360–380, December 2011.

4. Atsushi Keyaki, Kenji Hatano, Jun Miyazaki: "A Proposal of a Reconstruction Method to Return Well-informative Search Results", IPSJ Transactions on Databases (TOD), Vol.4, No.1, pp.1–13, March 2011.

5. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Query-oriented XML Fragment Search Approach on A Relational Database System", Journal of

Digital Information Management, Digital Information Research Foundation, Vol. 8, No. 3, pp.175–180, June 2010.

International Conferences and Workshops (reviewed)

1. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "A Path expression-Based Smoothing of Query Likelihood Model for XML Element Retrieval", Proceedings of the 1st ACIS International Symposium on Applied Computing and Information Technology (IIAI-AAI ACIT 2013), pp.296–300, Matsue, Japan, December 2012.

2. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "Fast and Incremental Indexing in Effective and Efficient XML Element Retrieval Systems", Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services (iiWAS2012), pp.157–166, Bali, Indonesia, December 2012.

3. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Relaxed Global Term Weights for XML Element Search", Comparative Evaluation of Focused Retrieval, Volume 6932 of LNCS, Springer, pp.71–81, 2011.

4. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Result Reconstruction Approach for More Effective XML Fragment Search", Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), pp.115–123, Paris, France, November 2010.

5. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Scoring Method of XML Fragments Considering Query-Oriented Statistics", Proceedings of the 2nd International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2009), pp.65–76, London, United Kingdom, August 2009.

International Conferences and Workshops (not reviewed)

1. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "XML Element Retrieval@1CLICK-2", Proceedings of the 10th NTCIR Conference, Chiyoda, Japan, June 2013.

2. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Result Reconstruction Method for Effective XML Fragment Search at INEX 2010", INEX 2010 Workshop Pre-Proceedings, pp.65–76, Vught, Netherlands, December 2010

3. Atsushi Keyaki, Jun Miyazaki, and Kenji Hatano: "A Method of Generating Answer XML Fragment from Ranked Results", INEX 2009 Workshop Pre-proceedings, pp.65–76, Brisbane, Australia, December 2009.

Domestic Conferences and Workshops (reviewed)

1. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "Fast Updates on Indices for Accurate XML Element Retrieval Systems", Proceedings of WebDB Forum 2012, A1-1, Tokyo, November 2012. received student encouragement award (in Japanese)

Domestic Conferences and Workshops (not reviewed)

1. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, and Hirokazu Kato: "An Adaptation of XML Element Retrieval Techniques to Web Documents", The 6th Forum on Data Engineering and Information Management (DEIM 2014), A1-4, Awaji, March 2014. (in Japanese) (to appear)

2. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takafumi Taketomi, and Hirokazu Kato: "Update-Aware Accurate XML Element Retrieval", The 5th International Workshop with Mentors on Databases, Web

and Information Management for Young Researchers (iDB Workshop 2013), Sapporo, July 2011. [closed]

3. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, and Hirokazu Kato: "An Improvement of XML Element Retrieval Considering Document Updates", The 5th Forum on Data Engineering and Information Management (DEIM 2013), A1-2, Koriyama, March 2013. received best paper award (PhD session) and student presentation award (in Japanese)

4. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, and Hirokazu Kato: "An Evaluation and a Refinement of Filters Based on Statistics of XML Elements for Reducing Update Cost of Indices", The 4th Forum on Data Engineering and Information Management (DEIM 2012), D3-2, Maiko, March 2012. (in Japanese)

5. Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, and Hirokazu Kato: "An Approach of Dynamic Maintenance of Indices for XML Element Retrieval", IPSJ SIG Technical Report, 2011-DBS-153,No.32, pp. 1-8, November 2011.

6. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Relaxed Global Term Weights for XML Element Search", The 3rd International Workshop with Mentors on Databases, Web and Information Management for Young Researchers (iDB Workshop 2011), Kyoto, August 2011. [closed]

7. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Method of Relaxation Global Weight for XML Element Search", The 3rd Forum on Data Engineering and Information Management (DEIM 2011), E6-6, Izu, February 2011. (in Japanese)

8. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Proposal of Scoring Method for Reconstructed XML Fragments", WI2-2010-40, pp.35-40, Sado, September 2010. (in Japanese)

9. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Construction Method with Using XML Information Retrieval Technique", The 72nd National Convention of Information Processing Society of Japan', 6ZC-7, pp.121-122, 6ZC-7, Tokyo, March 2010. (in Japanese)

10. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "A Method of Presenting Search Results Based on the Reconstructing Ranked Results", The 2nd Forum on Data Engineering and Information Management (DEIM 2010), C7-2, Awaji, March 2010. (in Japanese)

11. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Advantages of XML Fragment Retrieval Method Considering Query-Oriented Statistics", IPSJ SIG Technical Report, Vol.2009-DBS-148/2009-FI-95, No.1, pp.1-8, Kobe, July 2009. received student encouragement award (in Japanese)

12. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki: "Algorithm for Extracting Important Parts from Structured Documents", The 1st Forum on Data Engineering and Information Management (DEIM 2009), B5-4, Kakegawa, March 2009.(in Japanese)

13. Atsushi Keyaki, and Kenji Hatano: "A Proposal for Extracting Significant Parts of Structured Documents Using Positions of Query Keywords", Proceedings of Information Processing Society of Japan Kansai Branch Conference 2008, pp.125-128, Kyoto, October 2008.(in Japanese)

Award

1. DEIM 2013, Best paper award (PhD session), 2013

2. DEIM 2013, Student presentation award, 2013

3. WebDB Forum 2012, Student encouragement award, 2012

4. NAIST Excellent Student Scholarship Program (NAIST Top Scholarship Program), 2011

5. IPSJ (Information Processing Society of Japan) DBS, Student encouragement award, 2009

Appendix

A. SMART Stop List

a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, asking, associated, at, available, away, awfully, b, be, became, because, become, becomes, becoming, been, before, beforehand, behind, being, believe, below, beside, besides, best, better, between, beyond, both, brief, but, by, c, c'mon, c's, came, can, can't, cannot, cant, cause, causes, certain, certainly, changes, clearly, co, com, come, comes, concerning, consequently, consider, considering, contain, containing, contains, corresponding, could, couldn't, course, currently, d, definitely, described, despite, did, didn't, different, do, does, doesn't, doing, don't, done, down, downwards, during, e, each, edu, eg, eight, either, else, elsewhere, enough, entirely, especially, et, etc, even, ever, every, everybody,

everyone, everything, everywhere, ex, exactly, example, except, f, far, few, fifth, first, five, followed, following, follows, for, former, formerly, forth, four, from, further, furthermore, g, get, gets, getting, given, gives, go, goes, going, gone, got, gotten, greetings, h, had, hadn't, happens, hardly, has, hasn't, have, haven't, having, he, he's, hello, help, hence, her, here, here's, hereafter, hereby, herein, hereupon, hers, herself, hi, him, himself, his, hither, hopefully, how, howbeit, however, i, i'd, i'll, i'm, i've, ie, if, ignored, immediate, in, inasmuch, inc, indeed, indicate, indicated, indicates, inner, insofar, instead, into, inward, is, isn't, it, it'd, it'll, it's, its, itself, j, just, k, keep, keeps, kept, know, knows, known, l, last, lately, later, latter, latterly, least, less, lest, let, let's, like, liked, likely, little, look, looking, looks, ltd, m, mainly, many, may, maybe, me, mean, meanwhile, merely, might, more, moreover, most, mostly, much, must, my, myself, n, name, namely, nd, near, nearly, necessary, need, needs, neither, never, nevertheless, new, next, nine, no, nobody, non, none, noone, nor, normally, not, nothing, novel, now, nowhere, o, obviously, of, off, often, oh, ok, okay, old, on, once, one, ones, only, onto, or, other, others, otherwise, ought, our, ours, ourselves, out, outside, over, overall, own, p, particular, particularly, per, perhaps, placed, please, plus, possible, presumably, probably, provides, q, que, quite, qv, r, rather, rd, re, really, reasonably, regarding, regardless, regards, relatively, respectively, right, s, said, same, saw, say, saying, says, second, secondly, see, seeing, seem, seemed, seeming, seems, seen, self, selves, sensible, sent, serious, seriously, seven, several, shall, she, should, shouldn't, since, six, so, some, somebody, somehow, someone, something, sometime, sometimes, somewhat, somewhere, soon, sorry, specified, specify, specifying, still, sub, such, sup, sure, t, t's, take, taken, tell, tends, th, than, thank, thanks, thanx, that, that's, thats, the, their, theirs, them, themselves, then, thence, there, there's, thereafter, thereby,

```
therefore, therein, theres, thereupon, these, they, they'd, they'll,
they're, they've, think, third, this, thorough, thoroughly, those,
though, three, through, throughout, thru, thus, to, together, too,
took, toward, towards, tried, tries, truly, try, trying, twice, two,
u, un, under, unfortunately, unless, unlikely, until, unto, up,
upon, us, use, used, useful, uses, using, usually, uucp, v, value,
various, very, via, viz, vs, w, want, wants, was, wasn't, way, we,
we'd, we'll, we're, we've, welcome, well, went, were, weren't, what,
what's, whatever, when, whence, whenever, where, where's,
whereafter, whereas, whereby, wherein, whereupon, wherever, whether,
which, while, whither, who, who's, whoever, whole, whom, whose, why,
will, willing, wish, with, within, without, won't, wonder, would,
would, wouldn't, x, y, yes, yet, you, you'd, you'll, you're, you've,
your, yours, yourself, yourselves, z, zero
```

B. Examples of SQL format query

We semi-manually transformed NEXI query: `//article[about(., philosophy)]` `//section[about(., meaning of life)]`

B.1 SQL Format Query of Baseline Approach

```
SELECT r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm,
    SUM(r0.s1 + r1.s2) AS score
FROM (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, SUM(t.weight) AS s1
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('philosophi')
```

```
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos) r0, (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, e.path,
        e.numTerm, SUM(t.weight) AS s2
    FROM  element e, term t
    WHERE e.pathexp LIKE '#%/article#%/section'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('mean', 'life')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm
) r1
WHERE r0.did = r1.did
    AND r0.st <= r1.st
    AND r1.ed <= r0.ed
GROUP BY r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm
ORDER BY score DESC;
```

B.2 SQL Format Query of *QS*

```
SELECT r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm,
    SUM(r0.s1 * (1 + LN(c00.count/c01.count)) +
    r1.s2 * (1 + LN(c10.count/c11.count))) AS score
FROM (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, SUM(t.weight) AS s1
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('philosophi')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos) r0, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
```

```
        SELECT e.did, e.eid
        FROM element e
        WHERE e.pathexp LIKE '#%/article'
        GROUP BY e.did, e.eid)
) c00, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
        FROM element e, term t
        WHERE e.pathexp LIKE '#%/article'
            AND e.did = t.did
            AND e.eid = t.eid
            AND t.term IN ('philosophi')
        GROUP BY e.did, e.eid)
) c01, (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm,
        SUM(t.weight) AS s2
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article#%/section'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('mean', 'life')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm
) r1, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
        FROM element e
        WHERE e.pathexp LIKE '#%/article#%/section'
        GROUP BY e.did, e.eid)
) c10, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
```

```
        SELECT e.did, e.eid
        FROM element e, term t
        WHERE e.pathexp LIKE '#%/article#%/section'
            AND e.did = t.did
            AND e.eid = t.eid
            AND t.term IN ('mean', 'life')
        GROUP BY e.did, e.eid)
) c11
WHERE r0.did = r1.did
      AND r0.st_pos <= r1.st_pos
      AND r1.ed_pos <= r0.ed_pos
GROUP BY r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm
ORDER BY score DESC;
```

B.3 SQL Format Query of *QK*

```
SELECT r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm,
    SUM(r0.s1 + r1.s2) * (r0.inc + r1.inc) as score
FROM (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, SUM(t.weight) AS s1,
        COUNT(t.term) AS inc
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('philosophi')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos) r0, (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, e.path,
        e.numTerm, SUM(t.weight) AS s2, COUNT(t.term) AS inc
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article#%/section'
```

```
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('mean', 'life')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm
) r1
WHERE r0.did = r1.did
    AND r0.st_pos <= r1.st_pos
    AND r1.ed_pos <= r0.ed_pos
GROUP BY r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path,
    r1.numTerm, r0.inc + r1.inc
ORDER BY score DESC;
```

B.4 SQL Format Query of *QO*

```
SELECT r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path, r1.numTerm,
    SUM(r0.s1 * (1 + LN(c00.count/c01.count)) + r1.s2 *
    (1 + LN(c10.count/c11.count))) * (r0.inc + r1.inc) AS score
FROM (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, SUM(t.weight) AS s1,
        COUNT(t.term) AS inc
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('philosophi')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos
) r0, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
        FROM element e
```

```
        WHERE e.pathexp LIKE '#%/article'
        GROUP BY e.did, e.eid)
) c00, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article'
        AND e.did = t.did
        AND e.eid = t.eid
        AND t.term IN ('philosophi')
        GROUP BY e.did, e.eid)
) c01, (
    SELECT e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm,
        SUM(t.weight) AS s2, COUNT(t.term) AS inc
    FROM element e, term t
    WHERE e.pathexp LIKE '#%/article#%/section'
        AND e.eid = t.eid
        AND e.did = t.did
        AND t.term IN ('mean', 'life')
    GROUP BY e.did, e.eid, e.st_pos, e.ed_pos, e.path, e.numTerm
) r1, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
        FROM element e
        WHERE e.pathexp LIKE '#%/article#%/section'
        GROUP BY e.did, e.eid)
) c10, (
    SELECT CAST(COUNT(*) AS numeric) AS count
    FROM (
        SELECT e.did, e.eid
        FROM element e, term t
```

```
        WHERE e.pathexp LIKE '#%/article#%/section'
            AND e.did = t.did
            AND e.eid = t.eid
            AND t.term IN ('mean', 'life')
        GROUP BY e.did, e.eid)
) c11
WHERE r0.did = r1.did
      AND r0.st_pos <= r1.st_pos
      AND r1.ed_pos <= r0.ed_pos
GROUP BY r1.did, r1.eid, r1.st_pos, r1.ed_pos, r1.path,
    r1.numTerm, r0.inc + r1.inc
ORDER BY score DESC;
```

Bibliography

[1] World Wide Web Consortium. `http://www.w3.org/`. Accessed in November, 2013.

[2] Satoshi Nakamura. Trustworthiness Analysis of Web Search. *Journal of Japanese Society for Artificial Intelligence*, 23(6):767–774, 2008. (in Japanese).

[3] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). `http://www.w3.org/TR/REC-xml/`, 2008. Accessed in November, 2013.

[4] Wikipedia –The Free Encyclopedia–. `http://en.wikipedia.org/wiki/Main_Page`. Accessed in November, 2013.

[5] Jaap Kamps, Shlomo Geva, Andrew Trotman, Alan Woodley, and Marijn Koolen. Overview of the INEX 2008 Ad Hoc Track. In *Advances in Focused Retrieval*, volume 5631 of *Lecture Notes on Computer Science*, pages 1–28. Springer Berlin, 2008.

[6] Jaap Kamps and Marijn Koolen. On the Relation between Relevant Passages and XML Document Structure. In *in Proc. of the 30th ACM SIGIR*, 2007.

[7] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. `http://www.w3.org/TR/html401/`, 1999. Accessed in November, 2013.

[8] Charles F. Goldfarb, Edward Mosher, and Raymond Lorie. SGML. `http://www.w3.org/MarkUp/SGML/`, 1979. Accessed in November, 2013.

[9] Kenji Kita, Kazuhiko Tsuda, and Masami Shisibori. *Algorithm of Information Retrieval*. Kyoritsu Shuppan, 2002. (in Japanese).

[10] G. Salton. *The SMART Retrieval System–Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[11] M.F.Porter. An Algorithm for Suffix Stripping. *Readings in Information Retrieval*, pages 313–316, 1997.

[12] Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Journal of Information Processing and Management*, 24(5):513–523, 1988.

[13] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. *The Third Text Retrieval Conference (TREC-3)*, pages 109–126, 1995.

[14] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *Proceedings of the 13 ACM International Conference on Information and Knowledge Management*, pages 42–49, 2004.

[15] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[16] Koji Eguchi. Trends and Issues in Probabilistic Language Models for Information Retrieval. *IEICE Transactions on Information and Systems*, J93-D(3):157–169, 2010.

[17] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[18] Jon M. Kleinberg. Improvement in TF-IDF Scheme for Web Pages based on the Contents of Their Hyperlinked Neighboring Pages. *ACM Computing Surveys (CSUR)*, 31(5):56–68, 1999.

[19] Kazunari Sugiyama, Kenji Hatano, Masatoshi Yoshikawa, and Shunsuke Uemura. Improvement in TF-IDF Scheme for Web Pages based on the Contents of Their Hyperlinked Neighboring Pages. *Systems and Computers in Japan*, 36(14):56–68, 2005.

[20] Hao ming Wang, Martin Rajman, Ye Guo, and Bo qin Feng. NewPR-Combining TFIDF with Pagerank. In *Proc. of the 16th International Conference, Athens, Greece*, volume 4132 of *LNCS*, 2006.

[21] Takashi Tokuda and Keishi Tajima. Classification of XML Tags according to Their Roles in Document Structure. *DBSJ Journal*, 8(1):1–6, 2009. (in Japanese).

[22] Elika J. Etemad. Cascading Style Sheets (CSS) Snapshot 2010. `http://www.w3.org/TR/CSS/`, 2011. Accessed in November, 2013.

[23] XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). `http://www.w3.org/TR/xhtml1/`, 2002. Accessed in November, 2013.

[24] Henk Blanken, Torsten Grabs, Hans-Jörg Schek, Ralf Schenkel, and Gerhard Weikum. *Intelligent Search on XML Data: Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *Lecture Notes on Computer Science*. Springer-Verlag, 2003.

[25] Toshiyuki Shimizu. *A Study on Document-Centric XML Search*. PhD thesis, Kyoto University, 2008.

[26] Michael Ley, Marc Herbstritt, Marcel R. Ackermann, Oliver Hoffmann, Michael Wagner, Stefanie von Keutz, and Katharina Hostert. The DBLP Computer Science Bibliography. `http://www.informatik.uni-trier.de/~ley/db/index.html`. Accessed in November, 2013.

[27] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal XML pattern matching. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 310–321, 2002.

[28] Haifeng Jiang, Hongjun Lu, Wei Wang, and Beng Chin Ooi. XR-Tree: Indexing XML Data for Efficient Structural Joins. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, pages 253–263, 2003.

[29] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistrucutred databases. In *Proc. of the 23rd International Conference on Very Large Data Bases*, pages 436–445, 1997.

[30] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proc. of the 27th International Conference on Very Large Data Bases*, pages 361–370, 2001.

[31] Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *Proc. of the 27th International Conference on Very Large Data Bases*, pages 341–350, 2001.

[32] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases. *ACM Transactions on Internet Technology*, 1(1):110–141, 2001.

[33] Raghav Kaushik, Philip Bohannon, Jeffrey F. Naughton, and Henry F. Korth. Covering Indexes for Branching Path Queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 133–144, 2002.

[34] Wei Fan Haixun Wang, Sanghyun Park and Philip S. Yu. ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 110–121, 2003.

[35] Praveen Rao and Bongki Moon. PRIX: Indexing and Querying XML using Prufer Sequences. In *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, pages 288–300, 2004.

[36] Haixun Wang and Xiaofeng Meng. On the Sequencing of Tree Structures for XML Indexing. In *Proceedings of the 21th International Conference on Data Engineering (ICDE 2005)*, pages 372–383, 2005.

[37] Xiaodong Wu, Mong Li Lee, and Wynne Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In *Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)*, pages 66–78, 2004.

[38] Patrick O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, and Nigel Westbury. ORDPATHs: Insert-friendly XML Node Labels. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 903–908, 2004.

[39] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0 (Second Edition). `http://www.w3.org/TR/xpath20/`, 2011. Accessed in November, 2013.

[40] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language (Second Edition), year = 2011, note = Accessed in November, 2013,. `http://www.w3.org/TR/xquery/`.

[41] Sihem Amer-Yahia, Chavdar Botev, and Jayavel Shanmugasundaram. TeXQuery: A full-text Search Extension to XQuery. In *Proceedings of the 13th international conference on World Wide Web*, pages 583–594, 2004.

[42] Raghav Kaushik, Rajasekar Krishnamurthy, Jeffrey F. Naughton, and Raghu Ramakrishnan. On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 779–790, 2004.

[43] Toshiyuki Shimizu and Masatoshi Yoshikawa. Full-Text and Structural Indexing of XML Documents on B+ Tree. *IEICE Transactions on Information and Systems*, E89-D(1):237—247, 2006.

[44] Inderjeet Mani. *Automatic Summarization*. John Benjamins Publishing, 2001.

[45] Peggy M. Andersen, Philip J. Hayes, Alison K. Huettner, Linda M. Schmandt, Irene B. Nirenburg, and Steven P. Weinstein. Automatic Extraction of Facts from Press Releases to Generate News Stories. In *Proc. of the 3rd Conference on Applied Natural Language Processing*, pages 170–177, 1992.

[46] Marcin Kaszkiel and Justin Zobel. Passage retrieval revisited. In *Proc. of the 20th ACM SIGIR*, pages 178–185, 1997.

[47] Manabu Okumura and Hidetsugu Nanba. *Automatic Text Summarization*. OHM Publishing, 2005. (in Japanese).

[48] Fang Huang, Stuart Watt, David Harper, and Malcolm Clark. Compact Representations in XML Retrieval. In *Formal Proc. of INEX 2006 Workshop*, volume 5631 of *LNCS*, 2007.

[49] Kenji Hatano, Hiroko Kinutani, Toshiyuki Amagasa, Yasuhiro Mori, Masatoshi Yoshikawa, and Shunsuke Uemura. Analyzing the Properties of XML Fragments Decomposed from the INEX Document Collection . In *Advances in XML Information Retrieval*, volume 3493 of *Lecture Notes on Computer Science*, pages 168–182. Springer Berlin, 2005.

[50] Initiative for the evaluation of xml retrieval. `http://inex.mmci.uni-saarland.de/`, 2013. Accessed in November, 2013.

[51] Gabriella Kazai, Mounia Lalmas, and Arjen P. de Vries. 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*, pages 72–79, 2004.

[52] Jaap Kamps, Shlomo Geva, Andrew Trotman, Alan Woodley, and Marijn Koolen. Overview of the INEX 2008 Ad Hoc Track. In *INEX 2008 Workshop Pre-proceedings*, pages 1–28, 2008.

[53] Paavo Arvola, Shlomo Geva, Jaap Kamps, Ralf Schenkel, Andrew Trotman, and Johanna Vainio. Overview of the INEX 2010 Ad Hoc Track. In *INEX 2010 Workshop Pre-proceedings*, pages 11–40, 2010.

[54] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *Advances in XML Information Retrieval*, pages 16–40, 2005.

[55] Tetsuya Sakai, Makoto P. Kato, and Young-In Song. Overview of NTCIR-9 1CLICK. In *Proc. of the 9th NTCIR Conference*, 2011.

[56] Makoto P. Kato, Matthew Ekstrand-Abueg, Virgil Pavlu, Tetsuya Sakai, Takehiro Yamamoto, and Mayu Iwata. Overview of the NTCIR-10 1CLICK-2 Task. In *Proc. of the 10th NTCIR Conference*, 2013.

[57] Jane Li, Scott Huffman, and Akihito Tokuda. Good abandonment in mobile and PC internet search. In *Proc. of the 32th ACM SIGIR*, 2009.

[58] the National Science Foundation. WordNet –A lexical database for English–. `http://wordnet.princeton.edu/`, 2013 (last update). Accessed in November, 2013.

[59] Yoshifumi Masunaga. *Introduction to Relational databases*. Science-sha, 1991. (in Japanese).

[60] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[61] Oracle. Oracle Database. `http://www.oracle.com/jp/products/database/overview/index.html`. Accessed in November, 2013.

[62] IBM. DB2. `http://www-01.ibm.com/software/data/db2/`. Accessed in November, 2013.

[63] Microsoft. SQL Server. `http://www.microsoft.com/en-us/sqlserver/default.aspx`. Accessed in November, 2013.

[64] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel: A Path-based Approach to Storage and Retrieval of XML Documents using Relational Databases. *ACM Transactions on Internet Technology*, 1(1):110–141, 2001.

[65] Open source product. XBase. `http://basex.org/`. Accessed in November, 2013.

[66] Toshiba. TX1. `http://www.toshiba-sol.co.jp/pro/xml/`. Accessed in November, 2013.

[67] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proc. of the 7th Symposium on Operating System Design and Implementation*, 2006.

[68] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Proc. of the 21st ACM SIGOPS symposium on Operating systems principles*, 2007.

[69] Oracle. BerkeleyDB. `http://www.oracle.com/us/products/database/berkeley-db/overview/index.html`. Accessed in November, 2013.

[70] Benjamin Piwowarski and Patrick Gallinari. A Bayesian Framework for XML Information Retrieval: Searching and Learning with the INEX Collection. *Journal of Information Retrieval*, 8(4):655–681, 2005.

[71] Torsten Grabs and Hans-Jörg Schek. PowerDB-XML: A Platform for Data-Centric and Document-Centric XML Processing. In *Proceedings of the First International XML Database Symposium*, volume 2824 of *Lecture Notes on Computer Science*, pages 100–117. Springer Berlin, 2003.

[72] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective Keyword search in Relational Databases. In *Proceedings of the 2006 ACM SIGMOD international Conference on Management of Data*, pages 563–574. ACM, 2006.

[73] Wei Liu, Stephen Robertson, and Andrew Macfarlane. Field-Weighted XML Retrieval Based on BM25. In *Advances in XML Information Retrieval and*

Evaluation, volume 3977 of *Lecture Notes on Computer Science*, pages 161–171. Springer Berlin, 2006.

[74] Atsushi Keyaki, Jun Miyazaki, Kenji Hatano, Goshiro Yamamoto, Takefumi Taketomi, and Hirokazu Kato. A Path expression-Based Smoothing of Query Likelihood Model for XML Element Retrieval. In *Proceedings of the 1st ACIS International Symposium on Applied Computing and Information Technology (IIAI-AAI ACIT 2013)*, pages 296–300, 2013.

[75] Kenji Hatano, Sihem Amer-yahia, and Divesh Srivastava. Document-Scoring for XML Information Retrieval using Structural Condition of XML Queries. In *IEICE technical report*, pages 13–18, 2007. DE2007-117.

[76] Paul Ogilvie and Jamie Callan. Hierarchical Language Models for XML Component Retrieval. In *Formal Proc. of INEX 2004 Workshop*, volume 3493 of *LNCS*, 2005.

[77] Paul Ogilvie and Jamie Callan. Parameter Estimation for a Simple Hierarchical Generative Model for XML Retrieval. In *Advances in XML Information Retrieval and Evaluation*, volume 3977 of *Lecture Notes on Computer Science*, pages 211–224. Springer Berlin, 2006.

[78] Jinyoung Kim, Xiaobing Xue, and W. Bruce Croft. A Probabilistic Retrieval Model for Semistructured Data. In *Proc. of the 31th ECIR*, pages 228–239, 2009.

[79] Jaap Kamps, Maarten de Rijke, and Börkur Sigurbjörnsson. Length Normalization in XML Retrieval. In *Proc. of the 27th ACM SIGIR*, pages 80–87, 2004.

[80] Jovan Pehcevski, James A. Thom, and S.M.M. Tahaghoghi. RMIT university at INEX 2005: Ad hoc track. In *Formal Proc. of INEX 2005 Workshop*, volume 3977 of *LNCS*, 2006.

[81] Tie-Yan Liu. Learning to Rank for Information Retrieval,. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[82] Nicolas Usunier, David Buffoni, and Patrick Gallinari. Ranking with Ordered Weighted Pairwise Classification. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1057–1064, 2009.

[83] David Buffoni, Nicolas Usunier, and Patrick Gallinari. LIP6 at INEX'09: OWPC for ad hoc track. In *INEX 2009 Workshop Formal Proceedings*, volume 6203 of *Lecture Notes on Computer Science*, pages 59–69, 2011.

[84] David Buffoni, Nicolas Usunier, and Patrick Gallinari. LIP6 at INEX ' 10: OWPC for Ad Hoc Track. In *INEX 2010 Workshop Formal Proceedings*, volume 6932 of *Lecture Notes on Computer Science*, pages 61–71, 2011.

[85] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys (CSUR)*, 40:1–58, 2008.

[86] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. TopX: Efficient and Versatile Top-k Query Processing for Semistructured Data. *The VLDB Journal*, 17(1):81–115, 2008.

[87] Andrew Trotman, Xiang-Fei Jia, and Shlomo Geva. Fast and Effective Focused Retrieval. In *INEX'09 Proceedings of the Focused retrieval and evaluation*, pages 229–241, 2009.

[88] Toshiyuki Shimizu, Norimasa Terada, and Masatoshi Yoshikawa. Development of an XML Information Retrieval System Using Relational Databases . *IPSJ Transactions on Databases*, 48(11):224–234, 2007. (in Japanese).

[89] Kenji Hatano, Hiroko Kinutani, Masahiro Watanabe, Yasuhiro Mori, Masatoshi Yoshikawa, and Shunsuke Uemura. Keyword-based XML Portion Retrieval: Experimental Evaluation based on INEX 2003 Relevance Assessments. In *Proceedings of the Second Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 81–88, 2004.

[90] Atsushi Keyaki, Jun Miyazaki, and Kenji Hatano. A Method of Generating Answer XML Fragment from Ranked Results. In *INEX 2009 Workshop Pre-Proceedings*, pages 563–574, 2009.

[91] Shinya Takami and Katsumi Tanaka. Web-snippet Generation Suitable for Search Purpose in Web Search Results. *IPSJ Transactions on Databases*, 49(4):1648–1656, 2008. (in Japanese).

[92] Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proceedings of the 17th International Conference on Data Engineering*, page 321. IEEE, 2001.

[93] Sihem Amer-Yahia and Mounia Lalmas. XML Search: Languages, INEX and Scoring. *SIGMOD Record*, 35(4):16–23, 2006.

[94] Norbert Fuhr and Kai Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–180, 2001.

[95] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-Free XQuery. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 72–83, 2004.

[96] Vagelis Hristidis and Nick Koudas. Keyword proximity search in xml trees. *IEEE Transaction on knowledge and data engineering*, 18(4):525–539, April 2006.

[97] Yu Xu and Yunnis Papakonstaninou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 527–538. ACM, 2005.

[98] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective Keyword Search for Valuable LCAs over Xml Documents. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 31–40, 2007.

[99] Vagelis Hristidis and Nick Koudas. Keyword Proximity Search in XML Trees. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(4):525–539, 2006.

[100] Ziyang Liu and Yi Chen. Identifying Meaningful Return Information for XML Keyword Search. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 329–340. ACM, 2007.

[101] Yu Huang, Ziyang Liu, and Yi Chen. Query Biased Snippet Generation in XML Search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 315–326. ACM, 2008.

[102] Tsubasa Tanabe, Toshiyuki Shimizu, and Masatoshi Yoshikawa. Effective Keyword-Based XML Retrieval Using the Data-Centric and Document-Centric Features. In *Proceedings of the 8th Asia Information Retrieval Societies Conference (AIRS 2012)*, pages 427–436, 2012.

[103] Fei Chen, Xixuan Feng, Christopher Ré, and Min Wang. Optimizing Statistical Information Extraction Programs Over Evolving Text. In *Proceedings of the 28th International Conference on Data Engineering (ICDE 2012)*, 2012.

[104] Thomas Neumann and Gerhard Weikum. xRDF3X: Fast Querying, High Update Rates, and Consistency for RDF Databases. In *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB2010)*, pages 256–263, 2010.

[105] Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On Querying Historical Evolving Graph Sequences. In *Proceedings of the 37th International Conference on Very Large Data Bases (VLDB2011)*, pages 726–737, 2011.

[106] Anthony Tomasic, Héctor García-Molina, and Kurt Shoens. Incremental Updates of Inverted Lists for Text Document Retrieval. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 289–300, 1994.

[107] Nicholas Lester, Justin Zobel, and Hugh E. Williams. In-Place versus Re-Build versus Re-Merge: Index Maintenance Strategies for Text Retrieval Systems. In *Proceedings of the 27th Australasian conference on Computer science*, pages 15–23, 2004.

[108] Giorgos Margaritis and Stergios V. Anastasiadis. Low-cost Management of Inverted Files for Online Full-Text Search. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 455–464, 2009.

[109] Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, and Ian Hickson. HTML5. `http://www.w3.org/TR/html5/`, 2014. Accessed in November, 2013.

[110] Seung-Jin Lim and Yiu-Kai Ng. Converting the Syntactic Structures of Hierarchical Data to Their Semantic Structures. *Information Organization and Databases*, 579:343–355, 2000.

[111] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting Content Structure for Web Pages based on Visual Representation. In *Proc. of the 5th ACM APWeb*, pages 406–417, 2003.

[112] Mitsuo Yoshida and Makio Yamamoto. Primary Content Extraction from News Pages without Training Data. *DBSJ Journal*, 8(1):29–34, 2009. (in Japanese).

[113] Mark Montague and Javed A. Aslam. Condorcet Fusion for Improved Retrieval. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pages 538–548, 2002.

[114] Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki. Algorithm for Extracting Important Parts from Structured Documents. In *Proc. of the 1st Forum on Data Engineering and Information Management*, 2009. (in Japanese).

[115] Jakob Nielsen. How Long Do Users Stay on Web Pages? `http://www.nngroup.com/articles/how-long-do-users-stay-on-web-pages/`, 2011. Accessed in November, 2013.

[116] Google. `http://www.google.com`. Accessed in November, 2013.

[117] Martin Cutts. *Oxford Guide to Plain English*. Oxford University Press, 2010.

[118] M. E. Maron. Automatic Indexing: An Experimental Inquiry. *Journal of the ACM*, 8:404–417, 1961.

[119] Marc Guillemot Andy Clark. Cyberneko html parser (1.9.19). `http://nekohtml.sourceforge.net/index.html`, 2013. Accessed in November, 2013.