Kikori-KS: An Effective and Efficient Keyword Search System for Digital Libraries in XML

Toshiyuki Shimizu[1], Norimasa Terada[2], and Masatoshi Yoshikawa[1]

[1] Graduate School of Informatics, Kyoto University
shimizu@soc.i.kyoto-u.ac.jp, yoshikawa@i.kyoto-u.ac.jp
[2] Graduate School of Information Science, Nagoya University
terada@dl.itc.nagoya-u.ac.jp

Abstract. Identifying meaningful document fragments is a major advantage achieved by encoding documents in XML. In scholarly articles, such document fragments include sections, subsections and paragraphs. XML information retrieval systems need to search document fragments relevant to queries from a set of XML documents in a digital library. We present Kikori-KS, an effective and efficient XML information retrieval system for scholartic articles. Kikori-KS accepts a set of keywords as a query. This form of query is simple yet useful because users are not required to understand XML query languages or XML schema. To meet practical demands for searching relevant fragments in scholartic articles, we have developed a user-friendly interface for displaying search results. Kikori-KS was implemented on top of a relational XML database system developed by our group. By carefully designing the database schema, Kikori-KS handles a huge number of document fragments efficiently. Our experiments using INEX test collection show that Kikori-KS achieved an acceptable search time and with relatively high precision.

1 Introduction

Large number of documents in digital libraries are now structured in XML. An XML document is a text marked up by using tags. Document fragments in an XML document are identified by using tags and may have nested document fragments. Identifying meaningful document fragments is a major advantage achieved by encoding documents in XML. In scholarly articles, such document fragments include sections, subsections, and paragraphs.

XML information retrieval (XML-IR) systems need to search document fragments relevant to queries from a collection of XML documents in a digital library. As the number of scholarly articles are increasing sharply, effective search systems are required. XML-IR systems can meet the growing demand for browsing only document fragments, such as sections or paragraphs, relevant to a certain topic.

There are several forms of queries for XML-IR systems. "XQuery 1.0 and XPath 2.0 Full-Text"[1], being developed by W3C, is an XML query language with full-text search capabilities. This form of query assumes users know XML

specific query languages, such as XPath [2] or XQuery [3], and the XML schema of documents. Although such an assumption is valid for a limited number of advanced users, we should consider many other end-users of XML-IR systems unfamiliar with XML query languages or XML schemas. As demonstrated by Web search engines, a set of keywords is a simple, intuitively understandable, yet useful form of queries, especially for unskilled end-users. Queries represented by a set of keywords are also versatile in that they can be issued against XML documents with heterogeneous schemas or without schema. In this paper, we present Kikori-KS, an XML-IR system for scholarly articles. Kikori-KS accepts a set of keywords as a query and returns document fragments in relevance order.

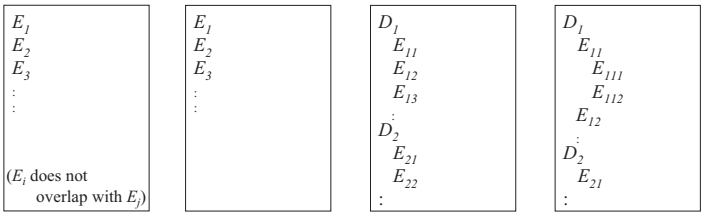
Developing an XML-IR system is a challenging task in many ways. Unlike conventional IR systems, XML document fragments, which are the search results of queries, can be nested each other. This feature raises an issue in the design of the user interface. We have developed a new user-friendly interface for Kikori-KS called the *FetchHighlight retrieval interface*. Because scholarly articles are an important granule, the FetchHighlight retrieval interface displays articles in relevance order, then for each article it shows document fragments in document order. The underlying database schema was designed to meet the requirements of the new user interface.

Furthermore, there are usually more document fragments than there are documents. For example, there are 16,080,830 document fragments against 16,819 documents in the XML collection used in our experiments. XML-IR systems need to process queries efficiently against a huge number of document fragments. Kikori-KS was implemented on top of a relational XML database system developed by our group [4]. In addition to tables as an XML database system, the database schema includes information on term weight. The system automatically translates sets of query keywords into SQL queries. We have designed a database schema specially tuned for fast query processing. Experiments show Kikori-KS returns query results, on average, 7.2 times faster than our previous method [5].

The rest of the paper is organized as follows. Section 2 explains the user interfaces we developed. Section 3 describes a conceptual database schema used in Kikori-KS, and we discuss how the database schema was refined and queries were processed to meet the requirements of the user interface. Section 4 presents the retrieval model used in Kikori-KS. Section 5 contains experimental results on retrieval precision and query processing time. Finally, Section 6 concludes the paper and presents future works.

2 User Interfaces

The granule of search targets for an XML-IR system are document fragments, which may be nested each other. This feature raises an important new issue when designing a user interface to browse search results. By considering users' needs for the output of a search, the INEX 2005 project [6] defined the following three strategies for element retrieval :



(a) Focussed (b) Thorough (c) FetchBrowse (d) FetchHighlight

Fig. 1. Interfaces for each strategy.

- **Focussed retrieval strategy**
  The system retrieves only focussed elements (i.e. non-overlapping elements) and ranks them in relevance order.
- **Thorough retrieval strategy**
  The system simply retrieves relevant elements from all elements without considering nestings and ranks them in relevance order.
- **Fetch and browse retrieval strategy**
  The system first identifies relevant documents (the fetching phase), and then identifies relevant elements within a fetched document (the browsing phase). The retrieved documents are initially ranked on the basis of their relevance, and then, the retrieved elements within a document are ranked in their relevance order. In the rest of this paper, we call this strategy the *FetchBrowse retrieval strategy*.

Figure 1 (a)-(c) show a naive and intuitive user interface image for each strategy. In the *Focussed retrieval strategy* (Figure 1 (a)) and the *Thorough retrieval strategy* (Figure 1 (b)), elements are displayed in descending order of their scores, i.e. element E_i is ranked higher than element E_j iff the score of E_i is higher than that of E_j. Additionally, in the Focussed retrieval strategy, E_i does not overlap with E_j for any E_i and E_j. In the FetchBrowse retrieval strategy (Figure 1 (c)), document D_i is ranked higher than document D_j iff the score of D_i is higher than that of D_j, and within the same document D_d, element E_{di} is ranked higher than element E_{dj} iff the score of E_{di} is higher than that of E_{dj}.

The three retrieval strategies of INEX were designed to evaluate XML-IR systems and were not necessarily intended to be used in designing user interfaces. From a practical point, we consider displaying search result elements aggregating by an XML document , which is a scholartic article in the case of INEX, is an effective display style for a user interface to have. The *FetchBrowse retrieval interface* is of that style. We also consider that displaying the search result elements in their document order (i.e. the occurrence order of start tags) is useful especially for document-centric XML. Taking this consideration into account, we have designed a new user interface, which we call the *FetchHighlight retrieval interface*. In the FetchHighlight retrieval interface, XML documents are first sorted in their relevance order, then relevant elements within the XML document

are displayed in document order. Figure 1 (d) shows a conceptual image for the *FetchHighlight retrieval strategy*. Elements are indented in accordance with their depth in the XML tree.

In addition, to facilitate user browsing and specify the position of the relevant document fragments easily, *outline elements* are output even if the score is 0, which have particular structural information that the administrator defined, together with relevant document fragments. Such outline elements in scholarly articles are sections, subsections and so on.

Figure 2 shows an example of the FetchHighlight retrieval interface. This example shows the search results for the keyword set "ontologies case study", which is one of the topics in INEX 2005. The anchor texts corresponding to document fragments with a high score are indicated by using a larger font. Users can easily recognize highly relevant document fragments even if the document score is not so high by browsing search results using the FetchHighlight retrieval interface (lower part in Figure 2). In addition, as the search results are ordered in document order, users can recognize the parts in the documents with many high relevant document fragments clustered. Users can browse the content of the document fragment highlighted within the document by clicking the corresponding anchor text (Figure 3).

We developed not only the FetchHighlight retrieval interface but also the *Thorough retrieval interface* and the *FetchBrowse retrieval interface*, which are the interfaces for the Thorough retrieval strategy and the FetchBrowse retrieval strategy, respectively. We did not implement the Focussed retrieval strategy because users lose some possible benefits of XML-IR [7]. The naive interface for the Focussed retrieval strategy is considered to be equivalent to the Thorough retrieval interface. However, the FetchHighlight retrieval interface can be regarded as a more user-friendly interface for the Focussed retrieval strategy because users can easily identify focussed elements using the FetchHighlight retrieval interface.

3 Implementation of Keyword Search on Relational Databases

3.1 Conceptual Database Design

We show how to store XML documents and term weight information, which is explained in Section 4, in a relational XML database system developed by our group [4]. Note that the granule of search targets for an XML-IR system are document fragments, which are elements or attributes. In this paper, we regard only elements as a document fragment for the sake of simplicity. The table schema of [4] is independent of the logical structure of XML documents. Kikori-KS uses the following schema:

- `Element (docID, elemID, pathID, start, end, label)`
- `Path (pathID, pathexp)`
- `Term (term, docID, elemID, tfipf)`

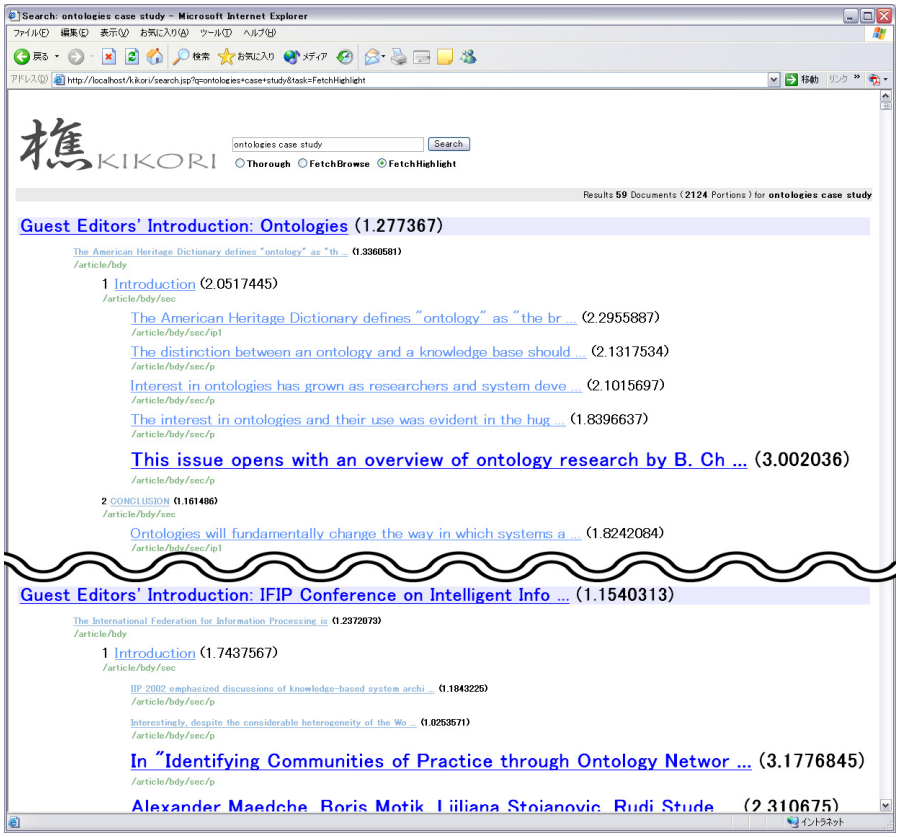


Fig. 2. FetchHighlight retrieval interface.

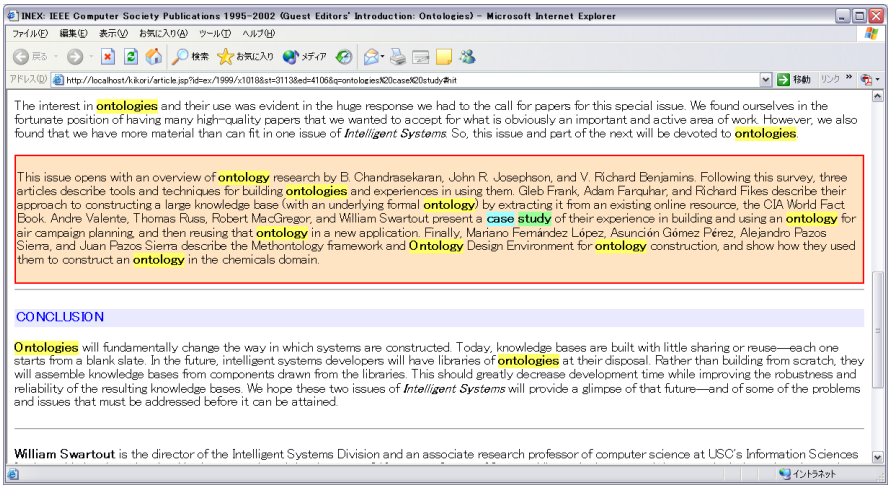


Fig. 3. Highlighted document fragment.

The `docID`, `elemID`, and `pathID` are identifiers that identify a document, an element, and a path, respectively. A pair of `docID` and `elemID` can uniquely identify an element in a whole XML document set. The `start` and `end` are the byte positions of the element and are used to clip the document fragment. The value of the `label` attribute is a short text representing the element. In scholarly articles, for example, section titles are used for the `label` values of the elements corresponding to sections. The `label` value is used as an anchor text in the user interface. The `pathexp` is the path expression from the root node and used when we want to take the structural information into account. The term weight in the element, whose detail is explained in Section 4, is stored in the `tfipf` attribute in the `Term` table.

3.2 Schema Refinement

We refined the conceptual relational database schema by considering the requirements of the user interfaces described in Section 2. The basic policy was to reduce query processing time by storing data redundantly.

1. Materialized view
   The search results in Kikori-KS need information across the `Term`, `Element`, and `Path` tables. Increasing speed of query processing is possible by creating a materialized view which is the result of joining these three tables.
2. Partition of `Term` table
   The `Term` table has vast amounts of rows. By partitioning the `Term` table with each term, we can access only the tables that a query really needs, and therefore achieving efficient query processing.

For each term, a table that has the following schema is constructed. We can identify the table using the table name with the term value. For term "xyz", the table name is `Term_xyz`.

- `Term_xyz (docID, elemID, tfipf, start, end, label, pathexp)`

In addition, in the case of the FetchHighlight retrieval strategy, we need outline elements. The system administrator predefines outline elements in such a way as selecting path information. Although outline elements can be retrieved dynamically during searches, we can process queries efficiently by selecting outline elements and constructing an `Outline` table in advance. We can also apply join operations in advance to the `Outline` table.

- `Outline (docID, elemID, start, end, label, pathexp)`

3.3 Query Translation

The input keyword set is translated into an SQL statement, and the system then calculates the score of each relevant element. The Kikori-KS system can automatically translate keyword sets into SQL statements to enable each retrieval strategy.

For each term in the keyword set, the system retrieves the corresponding `Term` table, then calculates the scores on the basis of the model described in Section 4. The FetchHighlight retrieval strategy and the FetchBrowse retrieval strategy need document scores. We considered that a document score is equivalent to the score of the root node in the document. In addition, the FetchHighlight retrieval strategy needs outline elements. The system retrieves the outline elements from the `Outline` table, then returns the union of the relevant elements and outline elements.

In addition to searching by simple keyword sets, Kikori-KS supports term restrictions by using mandatory term and negation. We can specify a mandatory term by using a "+" sign preceding the term and a negation by using a "-".

Kikori-KS appropriately arranges the results of SQL queries that perform the retrieval strategy the user selected and constructs a corresponding user interface.

4 Ranking Model

We explain the ranking model and term weighting method used in Kikori-KS in this section. The term weights are calculated in advance for fast query processing during searches.

The search results of the XML-IR systems should be ranked with scores that reflect relevance to the query. We used a *vector space model*, a model widely used for calculating scores. The score this model produces is the degree of similarity between the query vector and the document vector (element vector in the case of XML-IR) represented in a term space. We can use the dot product of query vector Q and element vector E for the degree of similarity.

$$Sim(Q, E) = \sum_{t \in Q, E} weight(t, Q) * weight(t, E) \tag{1}$$

We can use the term frequency of t in Q as the weight of the term in query ($weight(t, Q)$). As for $weight(t, E)$, which is the weight of term t in element E, *tf-idf*, which is the product of *tf* (term frequency) and *idf* (inverse document frequency), is an effective weighting scheme for unstructured documents. For structured documents, such as XML, several variants of *tf-idf* have been proposed [8–10]. The specificity of a term is calculated within the elements with the same *Category* in [9]. In the implementation of [9], *Category* is defined based on the path information from the root node. We call this method *ipf* (inverse path frequency) and use it in Kikori-KS. In addition, we adapted the method in [11] to XML documents and used the following formula for $weight(t, E)$.

$$weight(t, E) = \frac{ntf}{nel} * ipf \tag{2}$$

$$ntf = 1 + ln(1 + ln(tf)) \tag{3}$$

$$nel = \left((1 - s) + s * \frac{el}{avgel_p}\right) * (1 + ln(avgel_p)) \tag{4}$$

$$ipf = ln\frac{N_p + 1}{ef_p} \tag{5}$$

where ntf is the normalized term frequency (tf) of t, nel is the normalization factor that reflects the element length of E, and ipf is the specificity of term t within elements that share path p. Here, el is the number of terms in element E, p is the path of E from the root node, $avgel_p$ is the average el of elements that share path p, N_p is the number of elements that share path p, ef_p is the number of elements that term t occurs in the elements sharing path p, and s is a constant parameter and is usually set to 0.2 [11].

5 Experiments

We used the XML documents provided by INEX [6]. INEX is also developing topics, which are queries for XML document collection, and relevance assessments. We used the INEX-1.9 document collection, which is about 700 MB. INEX provides two types of queries: Content Only (CO) queries and Content And Structure (CAS) queries. We used 40 CO queries of INEX 2005 in our experiments. The experimental setup used is as follows: CPU: Intel Xeon 3.80 GHz (2 CPU), Memory: 4.0 GB, OS: Miracle Linux 3.0, and RDBMS: Oracle 10g Release1.

5.1 Effectiveness

We examined the precision of the search results using the Thorough retrieval strategy and the FetchBrowse retrieval strategy of Kikori-KS. Note that although the main focus of the Kikori-KS interface is the FetchHighlight retrieval interface, the precision of the FetchHighlight retrieval strategy cannot be directly measured using the INEX test collection. However, the precision of the FetchHighlight retrieval strategy is considered to be equivalent to the FetchBrowse retrieval strategy because the only difference between the strategies is the appearance of the user interface.

Many research groups in INEX are evaluating the effectiveness of each XML retrieval system. Although many metrics have been proposed and used for evaluations, we used precision/recall graph because it seems to be general one. Figures 4 and 5 show the results of Kikori-KS with ones by other INEX participants. The position of Kikori-KS is relatively high especially using the FetchBrowse retrieval strategy (Figure 5) and so using the FetchHighlight retrieval strategy as well.

5.2 Efficiency

We examined the processing time of SQLs translated from 40 CO queries and measured the average time for the Thorough retrieval strategy, the FetchBrowse retrieval strategy, and the FetchHighlight retrieval strategy. We retrieved only the top 1,500 results as stipulated by INEX.

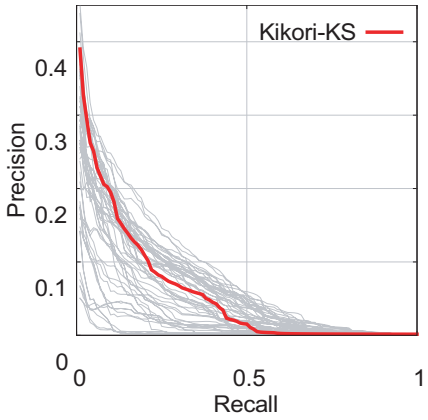


Fig. 4. Precision/Recall of Thorough.

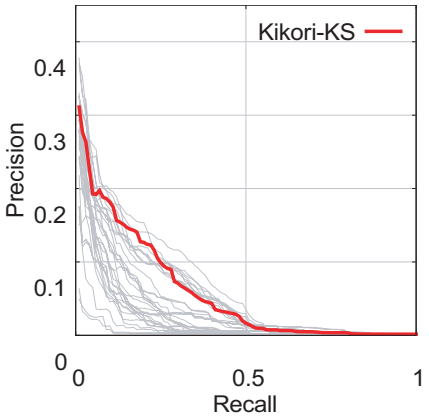


Fig. 5. Precision/Recall of FetchBrowse.

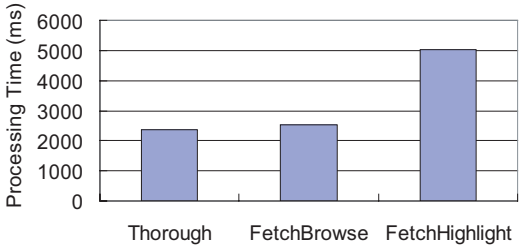


Fig. 6. Processing time.

Kikori-KS achieved a processing time of about 2.5 seconds for the Thorough retrieval strategy or the FetchBrowse retrieval strategy and about 5.0 seconds for the FetchHighlight retrieval strategy. We consider this an acceptable search time.

An efficient top-k query processing method for XML-IR was proposed in [12]. The method's main focus is CAS queries and whose target is increasing speed of searches using the Thorough retrieval strategy. In the case of CAS queries, we can use structural information to narrow down candidate results, and process faster than CO queries. The processing time indicated in Figure 6 is the time when we retrieved the top 1,500 results. The processing time of Kikori-KS is comparable with the time of [12]. Kikori-KS can handle the FetchBrowse retrieval strategy with a comparable time using the Thorough retrieval strategy.

The FetchHighlight retrieval strategy needs a union with the outline elements, and the processing time is about as twice as that for the Thorough retrieval strategy or the FetchBrowse retrieval strategy. If outline elements are not needed, the processing time is equivalent to that of the FetchBrowse retrieval strategy.

6 Conclusions

We introduced Kikori-KS, a system which we can use to search XML documents by using a set of keywords. This system allows us to browse search results with a user-friendly FetchHighlight retrieval interface. XML documents and their term weight information were stored in a relational XML database system, and the schema was refined for increasing speed of query processing. Using an INEX test collection, we confirmed that Kikori-KS can handle a keyword set query in an acceptable time and with relatively high precision.

Future work includes developing storage schema and weighting methods for phrase searches and introducing content and structure (CAS) searches.

References

1. W3C: XQuery 1.0 and XPath 2.0 Full-Text (2006) http://www.w3.org/TR/xquery-full-text/.
2. W3C: XML Path Language (XPath) Version 1.0 (1999) http://www.w3.org/TR/xpath.
3. W3C: XQuery 1.0: An XML Query Language (2006) http://www.w3.org/TR/xquery/.
4. Yoshikawa, M., Amagasa, T., Shimura, T., Uemura, S.: XRel: a path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology **1** (2001) 110–141
5. Fujimoto, K., Shimizu, T., Terada, N., Hatano, K., Suzuki, Y., Amagasa, T., Kinutani, H., Yoshikawa, M.: Implementation of a high-speed and high-precision XML information retrieval system on relational databases. In: Advances in XML Information Retrieval and Evaluation. Volume 3977 of Lecture Notes in Computer Science., Springer-Verlag (2006) 254–267
6. INEX: INitiative for the Evaluation of XML Retrieval (2005) http://inex.is.informatik.uni-duisburg.de/2005/.
7. Clarke, C.L.A.: Controlling overlap in content-oriented XML retrieval. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil (2005) 314–321
8. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: A semantic search engine for XML. In: Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany (2003) 45–56
9. Grabs, T., Schek, H.J.: ETH Zürich at INEX: Flexible information retrieval from XML with PowerDB-XML. In: Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval, Schloss Dagstuhl, Germany (2002) 141–148
10. Amer-Yahia, S., Curtmola, E., Deutsch, A.: Flexible and efficient XML search with complex full-text predicates. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, USA (2006) 575–586
11. Liu, F., Yu, C.T., Meng, W., Chowdhury, A.: Effective keyword search in relational databases. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, Chicago, USA (2006) 563–574
12. Theobald, M., Schenkel, R., Weikum, G.: An efficient and versatile query engine for TopX search. In: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway (2005) 625–636