

Effective Keyword Search in XML Documents Based on MIU

Jianjun Xu¹, Jiaheng Lu², Wei Wang¹, and Baile Shi¹

¹ Department of Computing and Information Technology,
Fudan University, China
{xjj, weiwang1, blshi}@fudan.edu.cn

² Department of Computer Science,
National University of Singapore
lujiaheng@comp.nus.edu.sg

Abstract. Keyword search is an effective approach for most users to search for information because they do not need to learn complex query languages or the underlying structures of the data. This paper focuses on effective keyword search in XML documents which are modeled as labeled trees. We first analyze the problems caused by the refinement of result granularity during XML keyword search and then propose to partition an XML document into XML fragments with the granularity of Minimal Information Unit (MIU). Furthermore, we present efficient index structures and the corresponding search algorithms. Finally, our comprehensive experiments demonstrate the benefits of our method over previously proposed methods in terms of result quality, index size and execution time.

1 Introduction

Keyword search is now the most popular information discovery method because the user does not need to learn any query language, or know the underlying structure of the data. Google, as the most famous search engine, provides keyword search on the HTML documents of World Wide Web. When the user just inputs several keywords on its simple-style homepage, Google can return all the HTML documents that are associated with these keywords. Therefore, the search engines of this kind can greatly facilitate naïve users to search for information on WWW.

With XML gradually becoming the *de facto* standard for data representation and exchange, keyword search on XML documents has become an important research direction. Although an XML document can be regarded as an HTML document, and thus the existing search engine techniques can be used for XML keyword search, the XML document has its own characteristics to be exploited. The biggest difference between HTML keyword search and XML keyword search lies in the granularities of their results. HTML keyword search returns the whole HTML document. The tags in the HTML document are just display instructions with no semantic information, so it is difficult to partition an HTML document into fragments. On the contrary, the tags in the XML document contain certain semantic information, indicating the meaning of the data nested. Therefore, we can return just the fragments of the XML document

associated with the keywords, rather than the whole XML document. In XML keyword search, the granularity of the search result is refined from document into element, which is called Refinement of Result Granularity. Refinement of result granularity is very useful and effective in searching on large XML documents, because it can help the user filter out a great deal of irrelevant information. Besides, because XML documents in most practical applications are often far larger than HTML documents in size, if we just return to the user the whole XML document, the high cost of network transfer will greatly degrade the search performance.

However, refinement of result granularity also gives rise to several problems. First, if some result element is partitioned from the XML document and returned to the user, it will be semantically incomplete with its context lost. Consider a sample XML document shown in Figure 1. Suppose that the user inputs a single keyword “Ullman”, for he wants to search for the information about Ullman’s publications. In this situation, all the known researches will return to the user the elements directly containing “Ullman”, such as the second *author* element in Figure 1, i.e. <author>Jeffrey D. Ullman</author>. Nevertheless, for the user, the context information is insufficient in such a simple result. He cannot understand whether Ullman is the author of a book or that of a paper, needless to say what on earth this publication is. Obviously, such search results will never satisfy the user. Therefore, how to make the search results semantically complete with necessary context information becomes a pressing problem to be solved.

```

<publi cation>
  <books>
    <publi sher>Prentice Hall</publi sher>
    <book>
      <title>Database Concepts</title>
      <author>Davi d M Kroenke</author>
    </book>
    <book>
      <title>A First Course in Database Systems</title>
      <author>Jeffrey D. Ullman</author>
    </book>
    ...
  </books>
  ...
</publi cation>

```

Fig. 1. A Sample XML Document

Second, the search results gained through using the existing techniques still contain much irrelevant information. Still consider the XML document in Figure 1. Suppose that the user wants to search for the information about Ullman’s books published by Prentice Hall, and thus he may input the keywords “Ullman Prentice Hall”. All the known researches return the smallest elements containing “Ullman”, “Prentice” and “Hall” in the XML document, such as the first *books* element. Obviously, such a result will not satisfy the user as well, because it contains all the information of the books published by Prentice Hall. Thus, what the user really wants is still submerged in the sea of irrelevant information. Of course, in comparison with returning the whole XML document, it is a big step forward to simply return the first *books*

element, because all the information of the books published by other presses has been filtered out. Yet, all in all, it is still a problem how to achieve a finer result granularity to filter out irrelevant information.

Third, refinement of result granularity may lead to invalid search results. The commonest situation is that different keywords are matched with different parts of the XML document. Because there is no or weak relationship among these parts, such search results mean little to the user. The root cause of the problem lies in that the keywords belong to different entities, among which there exists weak relationship. Thus, how to identify entities in the XML document and their relationships to reduce invalid search results as much as possible becomes another challenge.

This paper makes the following contributions: (1) Based on the analysis of the problems caused by refinement of result granularity, we give the definition of Minimal Information Unit (MIU) and present the algorithm of partitioning the XML document into MIUs. (2) Regarding MIU as the granularity of indexing and searching, we design efficient index structures and the corresponding search algorithms. (3) We conduct an extensive experimental study with real-life as well as synthetic XML data sets to validate the effectiveness and efficiency of our method. Our results demonstrate significantly improved result quality and search performance.

The rest of this paper is organized as follows. In Section 2, we review related work about XML keyword search. In Section 3, we present the data model that we use and the query semantics. Then, Section 4 states the definition of MIU in the XML document and the algorithm of partitioning the XML document into MIUs. Section 5 presents efficient index structures and the corresponding search algorithms. Section 6 contains experiments that show the effectiveness and efficiency of our method. Finally, Section 7 concludes this paper.

2 Related Work

At the present time, there are mainly two directions in the researches on XML keyword search. One is to add full-text search features and ranking to accurate XML query languages such as XML-QL[5, 6, 7]. The advantage of these languages after extension is the accuracy of the search results. However, all these languages are not suitable for naïve users, because they need to learn the syntax of complex query languages, and know the structures of XML documents.

The researches in the other direction make good use of the characteristics of the XML document to carry out keyword search. These researches mainly include: XRANK[1], XKeyword[2], XSearch[3] and XKSearch[4]. XRANK is the first to realize that the granularity of the search result can be element. And it puts forward DIL algorithm, which merges the lists of inverted index items to find out all the elements containing all the keywords. XKeyword is the extension of keyword search in the relational databases—DISCOVER[8]. Regarding an XML document as a graph, XKeyword transforms XML keyword search into proximity search among the keywords in the graph. From the angle of semantics, XSearch solves, to some degree, the problem of invalid search results. XKSearch brings forward ILE algorithm, which is used to search for all the SLCA (Smallest Lowest Common Ancestor) in the XML tree. ILE algorithm outperforms DIL algorithm when the search contains the

keywords with significantly different frequencies. The SE variant is tuned for the case where the keywords have similar frequencies.

The above-mentioned researches all make contributions to the development of XML keyword search in the second direction, but most of them do not realize the problems caused by refinement of result granularity. Therefore, the problems stated in Section 1 will appear in their search systems. In addition, there also exist in them some other defects, which are pointed out in the following sections.

3 Data Model and Query Semantics

3.1 XML Data Model

An XML document can be regarded as a labeled tree. The leaf node is the data value, and the inner node the element. An XML document contains nested elements, and each element has its own attributes, values or subelements. To facilitate the expression, attributes can be regarded as subelements, too. In the XML tree, a value node is represented with a pane, and an element node with a circle.

Definition 1. XML Data Graph $DG = (N, E, \text{ROOT}, \text{Label}_{NE}, \text{Value}_{NV}, \text{Order})$. $N = NE \cup NV$. N is the set of nodes, in which NE is the set of element nodes, and NV is the set of value nodes. A node n has its unique global number $\text{id}(n)$. $E \subseteq N \times N$. E is the set of containment edges. $\text{ROOT} \in NE$. ROOT is the unique root node. Function $\text{Label}_{NE}(n)$ returns the tag of element node n , and function $\text{Value}_{NV}(n)$ returns the value of value node n . The child nodes of an element node are in order. Function $\text{Order}(n)$ returns a number that represents the relative position of node n among its siblings.

Definition 2. XML Schema Graph $SG = (NE, E, \text{ROOT}, \text{Label}_{NE}, \text{Cardinality}_E)$. NE is the set of element nodes. $E \subseteq NE \times NE$. E is the set of containment edges. $\text{ROOT} \in NE$. ROOT is the unique root node. Function $\text{Label}_{NE}(n)$ returns the tag of element node n , and function $\text{Cardinality}_E(e)$ returns the cardinality of containment edge e . Suppose that, in the XML schema graph, e connects node n_1 with node n_2 , and n_1 is the parent node of n_2 . Function $\text{Cardinality}_E(e)$ calculates the scope of times n_2 may appear as the child nodes of some specific node n_1 in the corresponding XML data graph. $\text{Cardinality}_E(e)$ is represented as $c_1:c_2$, in which c_1 is the minimum times of appearance, while c_2 the maximum times. If $c_1 = c_2 = m$, $c_1:c_2$ can be simplified as m . Please note that the difference between the XML schema graph and the ordinary DTD graph lies in that the former exactly records the cardinalities of containment edges in a specific XML document, whereas the cardinalities in the latter may just represent approximate scopes, sometimes even far different from the real situation of the document.

In the XML schema graph, if the cardinality of containment edge e is 1 or 0:1, e is called the edge of low frequency, otherwise the edge of high frequency. Figure 2 is the text of an XML document, and Figure 3 is the XML data graph of the document. We can see that only the leaf nodes, which are the value nodes, contain the really useful information. Figure 4 is the XML schema graph of the document, in which the edges of high frequency are represented with the thick lines. Moreover, the cardinality

```

<dblp>
  <conference>
    <name>SIGMOD</name>
    <year>
      <conf year>2003</conf year>
      <paper pages=4-15>
        <title>Querying Structured Text in an XML Database</title>
        <author>Shurug Al-Khalifa</author>
      </paper>
      <paper pages=16-27>
        <title>XRANK: Ranked Keyword Search over XML Documents</title>
        <author>Lin Guo</author>
      </paper>
      ...
    </year>
  </year>
  <conference>
    ...
  </conference>
</dblp>

```

Fig. 2. XML Document (DBLP)

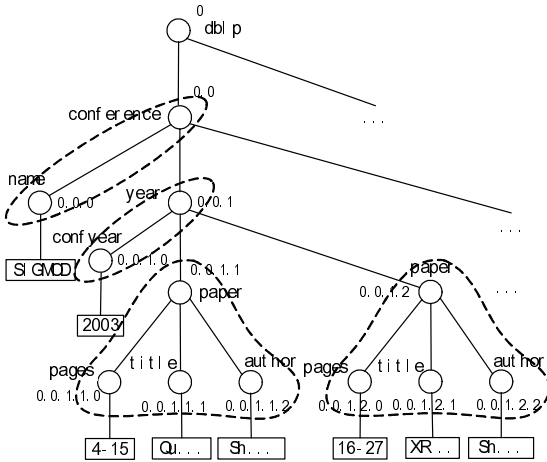


Fig. 3. Data Graph (DBLP)

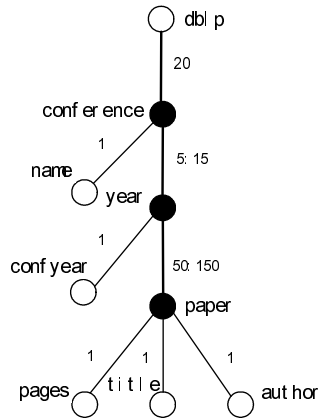


Fig. 4. Schema Graph (DBLP)

of each containment edge is marked beside the edge. For instance, the cardinality of the edge between node *conference* and node *year* is 5:15, which means that for any conference, it has been held at least for five years and at most for fifteen years.

3.2 Result Definition of Keyword Search

Keyword search Q is the keywords submitted by the user. Suppose that $Q = \{w_1, w_2, \dots, w_k\}$. XKSearch defines the result set of keyword search as all the SLCAs. The

so-called SLCA node is the element node in the XML tree and satisfies the following two conditions: (1) the subtree rooted at the node contains w_1, w_2, \dots, w_k ; (2) there does not exist any subtree of the subtree mentioned in (1), which also contains w_1, w_2, \dots, w_k .

XKSearch defines the result set as all the SLCA's, so it may lose some results. For example, in the XML tree of Figure 5, suppose that k_1 and k_2 are the keywords submitted by the user. Through observation, it is not difficult to find out that the search results should be the two XML fragments shown in Figure 5, whose root nodes are node 1 and node 4 respectively. However, in XKSearch, because node 1 is the ancestor node of node 4, node 1 will, according to ILE algorithm, be removed by the procedure *removeAncestor()*. Obviously, such removal is not appropriate. The root cause of this problem just lies in that the definition of the result set made in XKSearch has its deficiency.

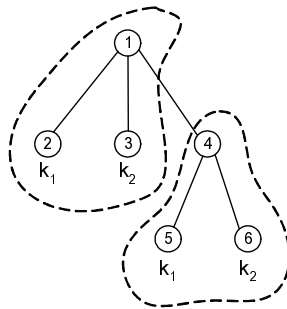


Fig. 5. SLCA & XLCA

To solve the above-mentioned problem, XLCA (eXclusive Lowest Common Ancestor) is proposed on the basis of SLCA and defined as follows: first of all, for an XML data graph T , find out all the SLCA's of T ; these SLCA's are just XLCAs; second, remove from T the fragments rooted at these SLCA nodes, and then find out again all the SLCA's of T , which are left out by ILE algorithm but revealed after the removal; these SLCA's are also XLCAs; the second step repeats until there is no SLCA in T . It can be seen that all the XLCAs are exclusive, i.e. non-overlapped. From the definition, SLCA must be XLCA, while XLCA is not necessarily SLCA.

Suppose that $xlca(T)$ is the set of all the XLCAs in an XML data graph T , and that $slca(T)$ is the set of all the SLCA's. We define the result set of XML keyword search as all the XLCAs, that is, $xlca(T)$. For T , $slca(T) \subseteq xlca(T)$. We can see that the definition of the result set made in XKSearch is not complete, because it cannot find out $xlca(T) - slca(T)$. For example, in Figure 5, XKSearch cannot find out the fragment rooted at node 1, but we can because the XLCA node set just includes node 1 and node 4. Note that from Section 4 MIU is regarded as the granularity of indexing and searching. Therefore, since then, XLCA is composed of MIUs, rather than elements.

4 Minimal Information Unit

4.1 Definition of MIU

Keyword search aims at searching for the information of the entities associated with the keywords. Entity should be the finest granularity of the search result, otherwise the information returned to the user may be semantically incomplete. The search result should also contain the necessary context information, which requires returning to the user the entities closely related to those directly containing the keywords. Meanwhile, the entities that have no or weak relationship to those directly containing the keywords should also be filtered out as much as possible. Therefore, the first step in keyword search is to identify all the entities in the search target.

An XML document is a set of entities. The element in the document represents the entity in the real world or its attribute. For example, in Figure 3, the *paper* element refers to the paper entity, and its *title* subelement serves as the title attribute of the paper entity. Each entity can have one or more attributes. Attributes can be simple or composite. A specific entity is just an instance of certain entity type. In the XML document, it means that there exist elements of the same type but with different contents. Besides, there exists certain relationship of containment between entities. Similarly, in the XML document, it means that there exists nesting between elements. For example, *dblp* is the set of several conferences. Since a conference is held in different years, it can be further divided by year. Then, in a specified year, the conference contains many papers.

Definition 3. Minimal Information Unit (MIU) of DG is an XML fragment in DG. Its root node refers to the entity it represents, and all the descendant nodes refer to the attributes of the entity. This fragment as a whole makes up an information unit with relatively complete semantics. Similarly, we can also define the Minimal Information Unit of SG. To distinguish between them, we name MIU of DG *dMIU*, and MIU of SG *sMIU*. The former represents the specific entity, while the latter refers to the entity type. Usually, there are several *dMIU*s corresponding to one *sMIU*.

The introduction of MIU is conducive to solving the problems caused by refinement of result granularity. First, after the recognition of *dMIU*s, the search result can be composed of *dMIU*s representing entities, rather than arbitrary elements. This makes its components semantically complete. Second, only after the recognition can the entities in the context be identified and returned to the user. In this way the search result can contain the necessary context information. Third, it is not until the recognition of *dMIU*s that we can filter out the entities unrelated to those directly containing the keywords in order to further reduce the irrelevant information.

XSearch solves the problem of invalid results by means of judging the relationship between two elements. XSearch holds that different elements with the same tag represent different entities of the same type, and so concludes that they are semantically unrelated. This kind of judging method is so severe that it probably judges some valid results to be invalid. Take the XML tree in Figure 6 for example. Suppose the user submits the keywords “Prentice Hall Database”, for he wants to search for the

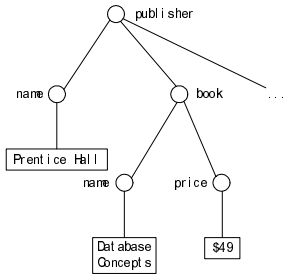


Fig. 6. A Sample XML Tree

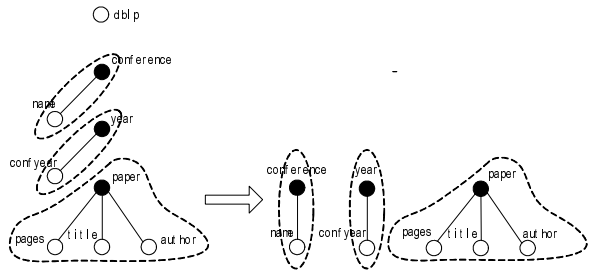


Fig. 7. Partitioning of Schema Graph

Information about books on Database published by Prentice Hall. However, because “Prentice Hall” and “Database” are both in the elements with the same tag “name”, XSearch will hold that “Prentice Hall” and “Database” belong to different entities of the same type, and so they are semantically unrelated. Therefore, it will not return the result to the user. Yet, it is not the fact. After partitioning an XML document into MIUs, the different elements with the same tag must belong to different dMIUs. So what can be solved by XSearch can also be achieved by our method. What’s more, our method has its own advantage. We do not think that there is no relationship between different dMIUs. Instead, in Section 5, we give a mechanism to judge the quality of the result according to its classification.

4.2 Partitioning of MIU

Some nodes in the XML tree have no real meanings, and usually have no attributes. The nodes of this kind only serve for connection, like node *dblp* in Figure 3, so we call them connection nodes, or dumb nodes.

The algorithm of partitioning SG into sMIUs is described as follows: In SG, through removing all the edges of high frequency and then removing all the isolated nodes (i.e. connection nodes), all the subgraphs acquired are sMIUs of this SG. Take the SG in Figure 4 for example, the partitioning is demonstrated in Figure 7.

To partition the XML document into dMIUs, we first carry out the preprocessing, i.e. scanning over its XML DG. After the acquisition of its corresponding XML SG, we partition SG into sMIUs according to the above-mentioned algorithm. Then, we can partition DG into dMIUs according to the acquired sMIUs.

4.3 Tuning of MIU

The edges of high frequency can be further divided into the weak edges of high frequency and the strong edges of high frequency. The former are the edges of $c_2 \leq \alpha$, in which α is a tunable parameter, while the latter are the edges of $c_2 > \alpha$.

In some situations, because of practical needs, we can tune the partition of SG to gain sMIUs with the coarser granularity. After fixing the value of α , we can regard all the weak edges of high frequency as the edges of low frequency, and repartition SG.

4.4 Indexing of MIU

Dewey encoding is a numbering technique widely used in the context of general knowledge classification. The common used numbering method in XML keyword search is as follows: (1) the dewey number of the root node is 0; (2) suppose that the dewey number of some node is x , then the dewey number of its first child node is $x.0$, the second is $x.1$, and so on. In Figure 3, each node of the XML tree is numbered according to dewey encoding. The length of a dewey number is defined as the number of components it contains, like $|0.0.1|=3$. The length is just the depth of the corresponding node in the XML tree. We can also know whose value is bigger between two dewey numbers by comparing the first component at first; if the same, then the second, and so on, like $0.0.1.1.0 < 0.1.1.1$. Furthermore, the number of the root node can be specified as the document number as well, thus supporting the global numbering of the document set. Note that what we index is the root nodes of dMIUs, so we can greatly reduce the lengths of the dewey numbers as well as the quantity of them.

5 Search Algorithms Based on MIU

5.1 Classification of Search Results and System Indexes

According to the result quality in terms of semantics, the results of keyword search $Q = \{w_1, w_2, \dots, w_k\}$ can fall into two categories: the optimal results and the common results. The optimal result means that w_1, w_2, \dots, w_k appear in the same entity, i.e. in the same MIU. The common result means that w_1, w_2, \dots, w_k appear in several entities, i.e. in several MIUs. The common results can be further divided into the linear common results and the non-linear common results. The linear common result means that w_1, w_2, \dots, w_k appear in several entities which have close context relationship with each other. In other words, w_1, w_2, \dots, w_k appear in several MIUs which have ancestor-descendant relationship. They appear on the same path originating from the root. And the non-linear common result means that w_1, w_2, \dots, w_k appear in several entities which have loose relationship. In other words, w_1, w_2, \dots, w_k appear in several MIUs and no path originating from the root can contain all of these MIUs.

The known researches all regard element as the finest granularity. Consequently, it is difficult, from the angle of semantics, to classify the search results. The existing researches usually acquire at first all the results and then sort them using information retrieval techniques. By contrast, this paper takes MIU as the finest granularity. The search results are divided into three categories: the optimal, linear common and non-linear common results. Obviously, from the angle of semantics, the optimal results are superior to the common ones, and, in most situation, the linear common results are superior to the non-linear common ones. Therefore, we can calculate successively the results of the three categories. If the result output of the search system is based on *top-k*, the search may finish earlier when the optimal result set (or the linear common result set) is searched out. Furthermore, from section 5.2, we can see that the time and space cost is quite low in searching for the optimal results and the linear common ones.

To support keyword search, we design efficient index structures. First of all, with regard to the XML document to be searched, we establish an inverted index of MIU

numbers in the external storage. For each keyword w in the document, there is a list of inverted index items, $miuIL[w]$, which contains all the dewey numbers of the MIUs where w appears directly. Besides, the MIU numbers in the list are arranged at first in the descending order according to length, and then, for those with the same length, they are arranged in the ascending order according to value. For the XML document in Figure 2, $miuIL(SIGMOD)=\{0.0, \dots\}$, $miuIL(2003)=\{0.0.1, \dots\}$, $miuIL(Querying)=\{0.0.1.1, \dots\}$, $miuIL(XRANK)=\{0.0.1.2, \dots\}$, ... Second, we establish an inverted index of PATH numbers. For each keyword w , there is a list of inverted index items, $pathIL[w]$, which contains all the numbers of the paths where w appears. The PATH numbers in the list are arranged in the ascending order according to value. For the XML document in Figure 2, $pathIL(SIGMOD)=\{P1, P2, \dots\}$, $pathIL(2003)=\{P1, P2, \dots\}$, $pathIL(Querying)=\{P1, \dots\}$, $pathIL(XRANK)=\{P2, \dots\}$, ...

5.2 Searching for Optimal Results and Linear Common Results

The optimal result is an MIU which contains w_1, w_2, \dots, w_k . Suppose that the $miuIL$ lists corresponding to w_1, w_2, \dots, w_k are respectively $miuIL[w_1], miuIL[w_2], \dots, miuIL[w_k]$. Algorithm 5.1 can calculate the optimal results through merging $miuIL[w_1], miuIL[w_2], \dots, miuIL[w_k]$. It optimizes the process of merging as well. The search for the results is carried out successively from the low level to high level. Thus, if different are the lengths of the MIU numbers that the current k pointers point, the MIU numbers on lower levels can be directly skipped. In this way, we only need to compare their lengths, rather than the specific redundant MIU numbers.

Algorithm 5.1 computeOptimalResults

```

while (miuIL[w1] <> ∅ & ... & miuIL[wk] <> ∅) {
  minlength = min{|top(miuIL[w1]|), ..., |top(miuIL[wk]|)}
  for (i=1; i<=k; i++)
    while (|top(miuIL[wi]|) > minlength) {
      remove(top(miuIL[wi]));
      if (miuIL[wi] = ∅) exit;}
  rearrange the order of miuIL[w1], ..., miuIL[wk]
  to make the value of top(miuIL[w1]) the smallest
  temp = remove(top(miuIL[w1]));
  for (i=2; i<=k; i++)
    if (top(miuIL[wi]) == top(miuIL[w1])) remove(miuIL[wi])
    else break;
  if (i==k+1) output(temp);
}

```

Fig. 8. Searching for the Optimal Results

The linear common result is composed of several MIUs, and each MIU contains at least one keyword. Moreover, there exists ancestor-descendant relationship among these MIUs, that is to say, they appear on the same path originating from the root. Suppose that the $pathIL$ lists corresponding to w_1, w_2, \dots, w_k are respectively $pathIL[w_1], pathIL[w_2], \dots, pathIL[w_k]$. We can calculate the linear common results through merging $pathIL[w_1], pathIL[w_2], \dots, pathIL[w_k]$. The algorithm is similar to

that of searching for the optimal results. With limited space, the detailed description of the algorithm is omitted here.

5.3 Searching for Non-linear Common Results

Algorithm 5.2 presents the algorithm of evaluating the non-linear common result set, which is called BUP (Bottom-up Pass) algorithm. The basic idea of BUP algorithm is as follows: the search process starts from the lowest level of the XML tree, bottoming up one level after another; during the process, the information on the keywords is passed upwards along the path level by level; once some MIU node contains all the keywords (some of them may be passed here from the lower levels), the corresponding XML fragment will be regarded as a search result.

This algorithm uses a list, *miuList*, to store the relevant information on the MIU nodes in the current level which contain the keywords or whose subtrees contain the keywords. The steps of the algorithm are described below. For the current level l , (1) check the *miuList* to see if there exist the MIU nodes containing all the keywords. If there do exist, remove the nodes from the *miuList*. Then check the removed nodes. If they do not belong to the optimal or linear common results, output the XML fragments corresponding to them; 2) after checking level l , pass the information on the keywords of the remaining nodes in the *miuList* to their parent nodes (level $l-1$), and replace all the child nodes with their parent nodes. If there exist several child nodes with the same parent node, merge the subtrees together and meanwhile record in the parent node the position of each child node containing keywords; 3) read the numbers of the MIU nodes in level $l-1$ from the *miuL* list of each keyword, and merge them with the *miuList*. Note that, in step 2, we record the accurate information on the positions of the keywords, so the XML fragment returned is not necessarily the whole subtree which probably contains much irrelevant information. In step 3, if there does not exist the keyword w_i in level $l-1$ or in its upper levels, further check if w_i exists in the *miuList*. If there is none, either, then the algorithm can finish earlier.

In algorithm 5.2 l is the level currently under review. *miuList.KS* represents the set of keywords appearing in *miuList*. *miuList[i].KS* represents the set of keywords appearing in the subtree which takes *miuList[i]* as its root. Function *prefix()* returns the prefix of the dewey number with the designated length. *merge(miuStart, miuEnd-1)* is to replace the nodes in the *miuList* from *miuStart* to *miuEnd-1* with their parent node. Simultaneously, the information on the positions of these child nodes is recorded to filter out irrelevant information later. Function *miuLLevel($w_i, level$)* returns the set of the MIU numbers in *miuL[w_i]*, each of which has the same level *level*.

BUP algorithm will be further demonstrated with the example in Figure 5. First, review the lowest level, i.e. the third level to examine every node where k_1 or k_2 appears. Because there is no node which contains all the keywords, every node where k_1 or k_2 appears on the third level passes upwards to its own parent node the information on the keywords. Thus, node 5 passes the keyword k_1 to node 4, and node 6 passes the keyword k_2 to node 4. And when it comes to the nodes on the second level, it is found that node 4 contains all the keywords, and thus the corresponding XML fragment is output. Likewise, node 2 and node 3 pass k_1 and k_2 respectively to node 1, and when it comes to the nodes on the first level, it is found that node 1 contains all the keywords, and thus the corresponding XML fragment is output.

Algorithm 5.2 BUP Algorithm

```

miuList=∅;
level=docdepth+1;
while (level>0) {
  //check for results
  if (miuList<>∅) {
    for (i=0;i<size(miuList);i++) {
      if (miuList[i].KS contains all keyword) {
        if ((miuList[i] not in OptimalResultSet)
          & (miuList[i] not in LinearCommonResultSet))
          output(miuList[i])
          delete(miuList[i])
      }
    }
    //pass keyword information to their parents
    if ((miuList<>∅) & (level>1)){
      miuStart=0;
      while (miuStart<size(miuList)) {
        miuEnd=miuStart+1;
        while ((miuEnd<size(miuList))
          & (prefix(miuList[miuStart], level-1)==prefix(miuList[miuEnd], level-1)))
          miuEnd++;
        merge(miuStart, miuEnd-1)
      }
    }
    //read keyword information of the upper level
    level--;
    if (level>0) insert miuILLevel( $w_1$ , level), ..., miuILLevel( $w_k$ , level) into miuList
    for (i=0; i<k; i++) {
      if ((miuILLevel( $w_i$ , level)=∅) & ... & (miuILLevel( $w_i$ , 1)=∅) & !(miuList.KS contains  $w_i$ ))
        exit;
    }
  }
}

```

Fig. 9. Searching for the Non-linear Common Results

Now we compare our work with XRANK. First, DIL algorithm proposed in XRANK searches for the results from left to right with the sequence of the results having no regularity in terms of semantics. By contrast, BUP algorithm presented in this paper adopts the bottom-up search strategy. Intuitively speaking, the deeper results contain the richer context information, so they are usually the better results from the angle of semantics. The bottom-up search strategy can guarantee that the deeper results will be produced first, that is to say, the results with richer context information will be produced first. Therefore, in terms of semantics, the sequence of the results produced by BUP algorithm is obviously better than that of DIL algorithm. Thus, in the situation where the response time is strictly demanded, the results produced by BUP algorithm can be directly output. Second, for DIL algorithm, even if, at some moment in the process of search, it is predictable that the index items left will not produce results any more, it is not until all the index items are scanned that the results can be output. Whereas, BUP algorithm carries out a predictable check whenever it finishes its search on one level; if it is predicted that no new result will be produced, the search can finish earlier. In addition, the stack-based DIL algorithm searches from left to right, and so will involve large quantities of push and pop operations on the stack when the keywords appear randomly in the XML tree. However, the above-mentioned operations do not exist in BUP algorithm with its adopting the bottom-up search strategy, and it only needs to neglect the several components in the rear part of the dewey numbers when passing upwards the information on the keywords.

6 Experimental Evaluation

We experimentally evaluate the techniques presented in this paper. First, we investigate the space savings due to the MIU partition. Second, we present some evidence that our search results are of high quality from the angle of semantics. Finally, we evaluate the performance of our search algorithms.

6.1 Experimental Setup

For our experiments, we use both the DBLP and XMark data sets. DBLP, the popular computer science bibliography database, is widely used in XML benchmarking. In the version we use, there are almost 100,000 records, totaling about 80MB of data. We filter out the references and other information only related to the DBLP website and group first by journal/conference name, then by year. We also generate an 100MB XMark data set. We choose to experiment with the DBLP and XMark data sets for they represent real-life and synthetic data sets, respectively. The experiments have been carried out on a PC with a 3.0GHz Pentium IV processor and 1GB of RAM.

6.2 Space Savings

To the best of our knowledge, the existing works all take element as the finest granularity. However, we take MIU. Table 1 gives the space requirements of the two approaches. As shown, the element approach incurs a significant space overhead for both DBLP and XMark. It is because the indexing scope of the element approach covers all the elements. By contrast, the MIU approach requires less space because its indexing scope covers all the MIUs, and one MIU just contains several elements. In our experiments, the average number of elements in one MIU is 7.8. Furthermore, the index item in the MIU approach has a shorter length because the dewey number of the MIU is just that of the root node of the MIU.

Table 1. Space Requirements of the Different Approaches

	DBLP		XMark	
	Inv. List	PATH List	Inv. List	PATH List
For element (XRANK)	136MB	N/A	205MB	N/A
For element (XKSearch)	177MB	N/A	258MB	N/A
For MIU	78MB	21MB	132MB	26MB

6.3 Result Quality

Each search result produced by our system is composed of MIUs rather than elements. Thus, the components of the search result have semantically complete information. Moreover, the existing works do not take the context information into consideration and their search mechanism cannot obtain the necessary context information automatically. By contrast, in our search system, it is easy to obtain such information.

We choose four typical keyword searches for the experiments in this and the next sections. Q1={Ullman}, Q2={keyword, search}, Q3={SIGMOD, XML}, Q4={XML,

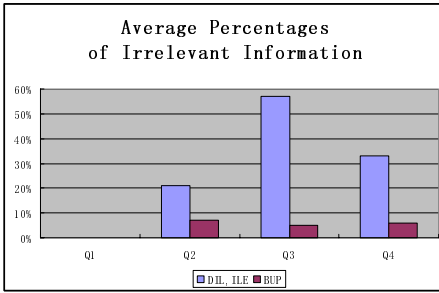


Fig. 10. Percentages of Irrelevant Information

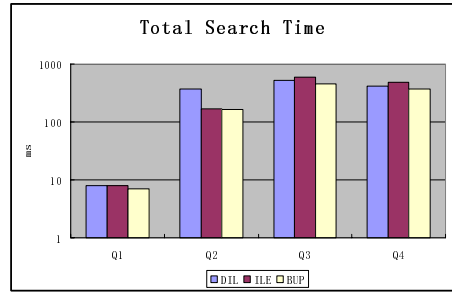


Fig. 11. Total Search Time

relational}. First, we compare the percentages that irrelevant information occupies in the search results acquired through DIL algorithm, ILE algorithm and our search algorithms respectively, which is shown in Figure 10. In terms of refinement of result granularity, the existing researches can only reach the level of element without further filtering out the irrelevant information in the elements. Our search algorithms outweigh DIL algorithm and ILE algorithm in this aspect, especially in the situation the keywords appear across levels, which means that the keywords submitted by the user do not appear in the same level, when a great deal of irrelevant information may appear in the lower levels. From the figure, we can find out that the percentages that irrelevant information occupies in our search results remains steady and in a comparatively lower level, while those in the results of DIL algorithm and ILE algorithm are in a higher level and vary a lot.

6.4 Search Performance

We now evaluate the search performance of the different algorithms. Figure 11 shows the total search time of DIL algorithm, ILE algorithm and BUP algorithm respectively. From the figure, we can find out that the total search time taken in BUP algorithm is the least, while DIL a little more, and ILE the most. In the experiments, we also validate that when keywords have significantly different frequencies, the search speed of ILE algorithm is faster than DIL algorithm and BUP algorithms, which is determined by its algorithm scheme. However, for ILE algorithm, it is just the performance in the special situations of this kind. Usually, ILE algorithm has no such predominance, and just as stated in Section 3, it cannot guarantee that all the possible results will be found out.

7 Conclusion

In this paper, with the focus on effective keyword search in XML documents, we first analyze the problems caused by the refinement of result granularity during XML keyword search, and then give the description of how to partition an XML document into XML fragments with the granularity of Minimal Information Unit (MIU). By regarding MIU as the granularity of indexing and searching, we design efficient index structures and the corresponding search algorithms. Through the sufficient

experimental evaluation, we demonstrate that our index structures and query evaluation techniques do provide significant space saving and performance gains. And our search results are semantically complete with necessary context information remaining and irrelevant information filtered out.

References

1. Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *Proc. of SIGMOD*, 2003.
2. Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword Proximity Search on XML Graphs. In *Proc. of ICDE*, 2003.
3. Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSearch: A Semantic Search Engine for XML. In *Proc. of VLDB*, 2003.
4. Yu Xu and Yannis Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proc. of SIGMOD*, 2005.
5. Daniela Florescu, Donald Kossmann, and Ioana Manolescu. Integrating Keyword Search into XML Query Processing. In *Proc. of IJCTN*, 2000.
6. Norbert Fuhr and Kai Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proc. of SIGIR*, 2001.
7. Anja Theobald and Gerhard Weikum. The Index-based XXL Search Engine for Querying XML Data with Relevance Ranking. In *Proc. of ICEDT*, 2002.
8. Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proc. of VLDB*, 2002.
9. Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proc. of ICDE*, 2002.
10. Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proc. of ICDE*, 2002.
11. Shurug Al-Khalifa, Cong Yu, and H. V. Jagadish. Querying Structured Text in an XML Database. In *Proc. of SIGMOD*, 2003.
12. Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In *Proc. of VLDB*, 2005.
13. World Wide Web Consortium. <http://www.w3.org>
14. DBLP XML Records. <http://dblp.uni-trier.de/xml/dblp.xml.gz>