ABSTRACT

XML data is normally queried by rigorous structured query languages, e.g., XPath, XQuery, etc. In recent years keyword search has become more and more popular because it provides a more user-friendly way to explore data. Keyword search on XML data has also been a hot research issue recently. So far none of the existing XML keyword search methods has considered range queries. In this paper we point out that supporting range queries in XML keyword search is beneficial and non-trivial to the user, especially in the case of querying business semi-structured data, where numerals (like stock price, product quantity, market share percentage, etc.) could be the main part of the data. Actually existing XML keyword search methods do not support range queries at two levels: keyword query syntax level and keyword search method level. To support range queries in XML keyword search: (1) we enrich the current XML keyword query syntax to let the user make range specification; (2) we then extend existing XML keyword search methods by proposing a new index to support both range match and point match. The new index is transparent to existing XML keyword search methods. It can seamlessly work with them and well support range queries in XML keyword search.

1. INTRODUCTION

XML is a de facto standard of information representation and exchange over the Internet, which is widely adopted to represent business information, scientific data, etc. Normally, XML data can be queried by rigorous structured query languages, e.g., XPath or XQuery. Before a user can retrieve information from the XML data, the user is required to learn the complex query language and to be familiar with the schema of the XML data. For example, an XML data tree in Figure 2 describes the book information of an online book store. Each book contains information like title, author, book id, price, etc. To find some books written by Winston, one possible query in XQuery is shown in Figure 1(a).

In contrast, keyword search, which is the major form of retrieval method in information retrieval systems (like Google, Bing, etc.), can free users from learning complex query language and data schema before they issue a query. Figure 1(b) shows the counterpart of Figure 1(a) in keyword query, which enables novice to explore the XML data without pre-knowledge of the query language or the data.

Keyword search on XML has been a hot research issue recently. Since keyword query only specifies some keywords, the main challenge of XML keyword search is to define the matching semantics, i.e., what should be returned as query results. Existing XML keyword search methods, such as LCA [10], SLCA [16], ELCA [2], etc., are all based on Lowest Common Ancestor (LCA), which returns some minimal subtrees containing all query keywords as query results. Efficient retrieval algorithms and the corresponding indexes are also built to support XML keyword search.

FOR $b$ IN document("bookstore.xml")//book
LET $a := $b//author "Winston book"
WHERE contains($a, "Winston")
RETURN $b
(a) XQuery
(b) Keyword Search

Figure 1: XQuery vs. Keyword Search

As keyword search becomes more and more popular, more advanced features are added to keyword search. For example, Google has added Advanced Search feature to its search engine [13], where range query is one of the important features.

However, none of the existing XML keyword search methods support range queries. In XML keyword search, sometimes a user may want to search for something within a specific price range. Existing keyword search methods cannot meet such a need. Given a keyword query, all existing works will exactly match all query keywords to the XML data. They are not able to capture the range specification in a range query. Besides, existing keyword search methods, which are designed to do exact match of user’s keywords to find query results, cannot support range match directly. In this paper, we will propose a solution to support range queries in XML keyword search. First let us look at a motivation example.

EXAMPLE 1. For the XML data tree in Figure 2, if a user wants to find all the books written by Neil Winston, she may want to issue a query $Q = \text{"Winston book"}$ to search for such books. Existing works, such as LCA [10], SLCA [16], ELCA [2], can work pretty well to server such a user. E.g., for SLCA, the query results are two subtrees: one is rooted at book:0.0.1 and another one is rooted at...
book:0.1.1. These two subtrees are actually two books written by Neil Winston which are exactly what the user wants.

However, as a common case in real world, a user may also want to search for some books which are both written by Neil Winston and lower than a specific price, say $15. In this case, the user may issue a query \( Q = \text{“Winston book price less than 15”} \) or \( Q = \text{“Winston book price<15”} \), etc. Unfortunately, user issuing the query will finally feel disappointed because no existing work on XML keyword search can support such a query with range specification. Such limitation forbids novice users to explore the database further to get their desired results (even though the results do exist).

As we can see, supporting range queries in XML keyword search is obviously beneficial and non-trivial to the user, especially in the case of querying business semi-structured data, where numerals (like stock price, product quantity, market share percentage, etc.) could be the main part of the data.

To handle user’s keyword query, existing works, such as LCA, SLCA, ELCA, all will match each keyword in user’s keyword query to a specific node in the XML data. Therefore, they are not able to support range specification in a keyword query. Actually existing works on XML keyword search do not support range query at two levels:

- keyword query syntax (upper level);
- keyword search method (lower level).

Therefore, to extend the current techniques of XML keyword search to support range queries, we need to tackle the following two tasks: (1) to enrich the keyword query syntax to make it possible to express user’s range specification; (2) to extend the current keyword search methods to evaluate keywords queries with range specification.

In this paper, we will try to tackle the above two tasks to support range queries in XML keyword search. Our major contributions in this paper include:

1. We address the needs for a way to make range specification in XML keyword search. We enrich the current XML keyword query syntax to support range specification in an XML keyword query.
2. We analyze the existing keyword search methods and find out a simple yet practical way to extend the current keyword search techniques to support range queries. We propose a new index, called Point & Range Hybrid-entrance Index, to support such an extension. The index is transparent to the existing keyword search methods and can work with them seamlessly.

The rest of the paper proceeds as follows. Related works are given in Sec. 2. We present preliminaries in Sec. 3. Enriching XML keyword query syntax is in Sec. 4. Section 5 discusses how to extend existing keyword search methods to support range queries. Section 6 presents the algorithms. Experimental evaluation is in Sec. 7 and we conclude in Sec. 8.

2. RELATED WORK

The first part of XML keyword search is the definition of matching semantics. LCA (lowest common ancestor) semantics is first proposed and studied in [10, 2] to find XML nodes, each of which contains all query keywords within its subtree. For a given query \( Q = \{k_1, ..., k_n\} \) and an XML document \( D \), \( L_i \) denotes the inverted list of \( k_i \). Then the LCAs of \( Q \) on \( D \) are defined as \( LCA(Q) = \{v | v = lca(m_1, ..., m_n), v_i \in L_i(1 \leq i \leq n)\} \). Subsequently, SLCA (smallest LCA [15]) is proposed, which is indeed a subset of \( LCA(Q) \), of which no LCA in the subset is the ancestor of any other LCA. Another widely adopted semantics is ELCA [2], which is also a subset of \( LCA(Q) \), is defined as follows: a node \( v \) is an ELCA node of \( Q \) if the subtree \( T_v \) rooted at \( v \) contains at least one occurrence of all query keywords, after excluding the occurrences of the keywords in each subtree \( T_{v'} \) rooted at \( v' \)'s descendant node \( v' \) and already contains all of the query keywords. Recently, structural consistency [4] is proposed to further constrain LCA such that there is no query result having an ancestor-descendant relationship at the schema level with any other query results.

The second part of XML keyword search is the proposals of efficient result retrieval methods based on a certain matching semantics: [16] for computing SLCA nodes and [2] for computing ELCA nodes. Moreover, improving user experience is studied in various ways [9, 5, 1, 8]. In particular, Sun et al. [11] generalize SLCA to support keyword search involving combinations of AND and OR boolean operators. XSeek [1] generates the return nodes which can be explicitly inferred by keyword match pattern and the concept of entities in XML data. [8] further proposed an axiomatic way to decide whether a result is relevant to a keyword query in term of the monotonicity and consistency properties w.r.t. the XML data and query. [3] proposed Valuable LCA (VLCA) by eliminating redundant LCAs that should not contribute to the answer, but also retrieves the false negatives filtered out wrongly by...
SLCA. [6] proposed a keyword search method that models semi-structured and other forms of data as graphs. [9] studied how to differentiate the search results of an XML keyword query, aiming to save user efforts in investigating and comparing potentially large results. [11] proposed a statistical way to identify the search target candidates. [12] studied the problem of finding the nearest node containing a specific keyword to a given node. [13] proposed a ranking approach for keyword queries based on an extension of the concepts of data dependencies and mutual information.

However, none of the existing works on XML keyword search consider extending the XML keyword search methods to support range queries. This motivates us to find out a solution to support range queries in XML keyword search to offer a better search experience to the user.

3. PRELIMINARIES

We model XML data as a rooted, labeled and ordered tree. Each node of the tree corresponds to an element of the XML data and has a tag name. To accelerate the keyword query processing, all existing works adopt the Dewey labeling scheme [15]. As shown in Figure 2, for a node n, its Dewey label consists of a sequence of components that implicitly contain all ancestor nodes on the path from the document root to n. E.g., from book:0.0.1.0, it is easy to find the label of its parent is 0.0.

**Definition 1. Attribute** The parent of a leaf node in the XML data is called an attribute.

**Definition 2. Value** The child of an attribute is called the value of the attribute.

Attributes with different tag names are called with different types. For example, in Figure 2 title:0.0.1.0 is an attribute because it is the parent of a leaf node; Pippi:0.0.1.0.0 is a child of the attribute title:0.0.1.0, so it is a value of title.

Next, we will present how we extend the current techniques on XML keyword search to support range queries. As discussed in Section [11] the main problems we need to solve are 1) to enrich the XML keyword query syntax, 2) to alter the XML keyword search method. These two tasks are tackled in the following two sections.

4. ENRICHING XML KEYWORD QUERY SYNTAX

As discussed in Section [11] it is beneficial and non-trivial to the user if we can let the user make range specification in their XML keyword queries. However, for existing XML keyword search methods, they all try to match each keyword in the query to a specified node in the XML data. In other words, all the terms in a query are supposed to matched something appearing in the XML data. Therefore, from the perspective of keyword query syntax, there is no way to make range specification in a traditional XML keyword query. So to support range queries in XML keyword search, we need to enrich the current keyword query syntax.

4.1 Current XML Keyword Query Syntax

Before we enrich the keyword query syntax, let us have a quick look at the current syntax for XML keyword query. Existing XML keyword search methods adopt the following syntax: a query consists of multiple terms concatenated by space(s) between them, where each term is a nonempty string without space. Such a syntax adopted by existing keyword search methods can be represented in the following formal grammar:

\[ T \to \text{a nonempty string without space} \]

\[ Q \to T \mid QT \]

where \( T \) is a term and \( Q \) is a keyword query: \( \to \) means “defined as” and the vertical bar \( | \) means or.

As the base case, a query \( Q \) can be formed by a single term \( T \). Recursively, it can also be formed by a query concatenated with another term by space (“\( QT \)”).

Given a keyword query \( Q \), we need to parse \( Q \) in order to get all the terms \( T \). This can be done by simply splitting \( Q \) by space and then clearing all redundant spaces.

4.2 Range-supported XML Keyword Query Syntax

Here we will try to enrich the current XML keyword query syntax to support range specification. To support range queries, only one kind of term is not enough, because some terms are supposed to exactly match the data in the XML while some terms are specifying the range of the data. Therefore, we propose two new types of terms: range term \( T_r \) and normal term \( T_n \).

Normal term \( T_n \) works exactly the same as a traditional term in Section [4.1] which is supposed to match a specified node in the XML data. Range term \( T_r \) is used to specify data range, and it will not be used to match a specified node in the XML data.

Here we need to define the pattern of a range term to differentiate itself from a normal term. With range term, user can specify a particular attribute value to fall into a required range. For example, user may have the following possible range specification: “price:10-15”, “price:<20”, “pages>200”, etc. As some symbols (like “<”, “>” etc.) could be common in a normal term, so we use some less common symbols (like “:”, “;” etc.) here because these special symbols will be reserved as a pattern to identify a range term. Therefore we can define a new keyword query syntax:

\[ T \to \text{a nonempty string without space} \]

\[ Q \to T \mid QT \]

where \( T \) is a term and \( Q \) is a keyword query: \( \to \) means “defined as” and the vertical bar \( | \) means or.

As the base case, a query \( Q \) can be formed by a single term \( T \). Recursively, it can also be formed by a query concatenated with another term by space (“\( QT \)”).

Given a keyword query \( Q \), we need to parse \( Q \) in order to get all the terms \( T \). This can be done by simply splitting \( Q \) by space and then clearing all redundant spaces.

For example, if a user wants to search for some books where the number of pages is ranging from 200 to 300, she can issue a query

“\( Q \) \( \to \) \( T_r \) \( \mid QT_n \) \( \mid QT_r \) \]

where \( T_r \) is a range term and \( T_n \) is a normal term.

As the base case, a query \( Q \) can be formed by either a normal term \( T_n \) or a range term \( T_r \). It can also be formed by a mix of both of them (“\( QT_n \)” and “\( QT_r \)”.

As we can see from the syntax definition of the range term \( T_r \), it can support range specification, i.e. less than (or equal), larger than (or equal) and range from...to...

For example, (1) if a user wants to search for some books which are both written by Winston and less than 15 dollars, she can simply issue a query

“Winston book price :< 15”

(2) if a user wants to search for some books where the number of pages is ranging from 200 to 300, she can issue a query

“book pages : 200 \(-\) 300”

Actually we can further extend the syntax of range term \( T_r \) to consider more possible range term patterns, such that we can make
the syntax more relaxed and capture more variants of range specification of the user. But note that, all the user query keywords will have a priority to be identified as a range term rather than a normal term. Therefore, when we define range term pattern, we should avoid some common symbols which may frequently appear in a normal term. E.g., we are using “:” rather than “<” here.

Now given a keyword query, we need to parse the query and extract both range terms and normal terms. This can be done in the following way:

- split the keyword query by space and remove all redundant space(s);
- use regular expression to identify range terms (5 possible patterns in the above syntax);
- the remaining terms will be normal terms.

So far we have enriched the XML keyword query syntax to support range specification. Next we will talk about how to extend the current XML keyword search techniques to support range queries, where a new index supporting both range match and point match will be proposed.

5. EXTENDING XML KEYWORD SEARCH METHOD

With the enriched XML keyword query syntax, now we can capture user’s range specification in the keyword query. But to return what user wants, we need to alter the existing XML keyword search methods to support range queries. In this section we will analyze the existing XML keyword search methods. Thereafter, we find out a simple yet practical way to extend the existing XML keyword search methods to support range queries by proposing a new index.

5.1 Anatomy of Existing XML Keyword Search Methods

Now we will analyze the existing XML keyword search methods, trying to find out a way to support range queries by extending the current methods.

Existing XML keyword search methods [2][16][3][7][11], no matter LCA [2], SLCA [16][3] or ELCA [2], their search methods are all based on inverted list index.

For each keyword appearing in the XML data, there is an inverted list built for it, which is a list of XML nodes containing this keyword. For example, Figure 4 shows the inverted lists for two keywords, namely “Winston” and “book”.

However, there could be a lot of different keywords appearing in the XML data. So to efficiently retrieve an inverted list for a specific keyword, say “Winston”, we need to organize the inverted lists in an efficient way. Some works [3] adopt Hash Map to organize the inverted lists, while some works [2][16] adopt B+ tree to organize the inverted lists. Figure 3 shows the inverted lists organized with B+ tree for the XML data in Figure 2.

With the index for inverted lists, given a keyword query, existing keyword search methods will do the query processing in two steps:

Step 1: Retrieving the related inverted lists.

As discussed in Section 2, existing XML keyword search methods will extract each term by simply splitting the keyword query by space. Then for each term extracted, they will retrieve the corresponding inverted list from the index.

For example, for a query “Winston book” issued on the XML data in Figure 2, the two terms extracted are “Winston” and “book”. For each of the two terms, the corresponding inverted list will be retrieved from the index in Figure 3. The two inverted lists retrieved are shown in Figure 4. Each inverted list contains a list of nodes containing the corresponding keyword. Each node in the inverted list may contain different information depending on different methods. Here we store the Dewey label for each node.

Step 2: Calculating query results based on the retrieved inverted lists.

Based on the inverted lists retrieved in Step 1, most of the existing works focus on how to efficiently/effectively calculate the query results. Take LCA as an example, it will pick one node from each inverted list, and then it will calculate the Lowest Common Ancestor of these nodes as a candidate query result.

For example, based on the inverted lists in Figure 4, we can pick one node from Winston list, say 0.0.1.1.0, and one node from book list, say 0.0.1. Then the lowest common ancestor of these two nodes is 0.0.1, which will be a candidate query result. The lowest common ancestor of two nodes is actually calculated by taking the longest prefix of their Dewey labels. All existing works so far, such as SLCA, ELCA, etc., are LCA-based. They propose methods in order to get rid of meaninglessness results and to efficiently calculate the results. We refer readers to Section 2 for more detail.

As we can see from the above analysis, Step 1 of existing XML keyword search methods do not consider range queries, because each term in the keyword query needs to exactly match one entry in the index in Figure 3. So in order to support range queries, we
need to alter the traditional index for XML keyword search. For Step 2, which is also the main focus of existing works, can remain unchanged after adopting a new index we propose in the next section.

5.2 Extending Keyword Search Method to Support Range Query

In this section, we will propose a new index for Step 1 (recall Section 5.1) of existing keyword search methods. After adopting our new index, Step 2 of current works can remain the same and work seamlessly with our new index, as we will show later.

Recall Section 4, we enrich the XML keyword query syntax, where we have normal term(s) and range term(s) in a keyword query. For normal terms, traditional index in Figure 3 can support it very well because each normal term will match a specific entry in the B+ tree index. We call such an exact match in an index as point match.

However, for range terms, such as "price:20-40", we are not able to match the term to any entry of the index. In Figure 3, even we search the position of keyword "20" (it will be between "10" and "220" in the index) and the position of "40" (it will be between "35" and "43BH2" in the index), we can find that between these two positions, there are a lot of attribute values which are not price. Even worse, the index is sorted by alphabetical order rather than numerical order. E.g., "220" is in the position of front of "27", while "27" should be in front of "220" because it is smaller. So a new index is needed. We call such a range match in an index as range match.

Therefore, we propose a new index, called Point & Range Hybrid-entrance Index, to support both point match and range match in an index.

Figure 5 shows the new index we propose to cater the needs for both point match and range match. The index has two entrances, namely range match entrance and point match entrance.

When we retrieve inverted lists for range terms, we proceed to range match entrance; when we retrieve inverted lists for normal terms, we go to point match entrance.

Now let us look at how the index can support both range match and point match.

Range Match Entrance

On the left-hand side of Figure 5 for attributes supporting range match, inverted lists for such attribute values are grouped together according to their attribute types, and each group is organized by an independent B+ tree. For example, inverted lists for price values are all grouped together and organized by a B+ tree.

On the top of the B+ trees, there is a hash map, which maps a specific attribute name to the proper B+ tree index. For example, given an attribute name "price", the hash map will map it to the B+ tree for price attribute.

As we can see, values of the same attribute type are now in the same B+ tree. So doing range match, say "price:20-40", is just to locate the position of the borders of the range and to get all the elements in between.

The B+ trees for each attribute type can correctly and efficiently support range match because:

1. Only attribute values of the same type are organized in the same B+ tree, so it is possible to store the values as numerical data (if needed) and sort them in numerical order. E.g., price value can now be stored as numerical data and sorted in numerical order. This cannot be done in a traditional XML keyword search index.

2. Nodes containing the same value but with different attributes type are now stored in different places, which makes range match for a specific attribute type more efficient. E.g., there are some pages nodes as well as some price nodes containing keyword "35". Now they are stored in different places. Therefore, when we do range match for price nodes, price nodes will not mix with pages nodes, which will make the retrieval more efficient.

Point Match Entrance

As we can see, the B+ tree on the right-hand side of Figure 5 is almost the same as the traditional XML keyword search index (shown in Figure 3 except the case that: for the nodes which support range match and already appear on the left-hand side of Figure 5, only pointers pointing to them will appear under the point match entrance. E.g., for the keyword "35", its data part in the B+ tree under point match entrance is actually a set of two pointers pointing to its possible position.

So the point match entrance actually works exactly the same as the traditional XML keyword search index.

Keyword Query Processing Using New Index

Now we will talk about how Step 2 of existing keyword search methods (recall Section 5.1) can work seamlessly with our new index. As discussed in Section 5, we have normal term(s) and range term(s) in the enriched XML keyword query syntax. For each normal term in a keyword query, we can retrieve an inverted list by using the point match entrance of our new index. For each range term, we can proceed to the range match entrance. With the hash map, we can get the B+ tree index for a specific attribute type. Then we can get the nodes in the required range from the B+ tree.

![Figure 5: Point & Range Hybrid-entrance Index for XML Keyword Query Processing](image-url)
6. ALGORITHMS

6.1 Index Construction

We will parse the XML tree to construct the Point & Range Hybrid-entrance Index. The index can be built by scanning the XML data. Algorithm 1 shows the main procedure of building the index.

The input is the XML tree and the output will be the Point & Range Hybrid-entrance Index. We will traverse the XML tree (line 3). For each node, we get its Dewey label and the keywords it contains (line 4-5). For each keyword (line 6) contained by the node, if the node is not a leaf node (line 7), it will be inserted to point match entrance (line 17-18). This is because values are all in the leaf node in an XML tree. Note that the dewey label of the node will be added to the end of the existing inverted list (line 18). If the node is a leaf node, then we will get its attribute type (line 8) to see whether it needs to support range match (line 9). If it does not need to support range match, the node will be added to the point match entrance (line 15-16). If it needs to support range match, we will add a new B+ tree to range match entrance for that attribute type if there is not any such B+ tree yet (line 10-11). After that, we just need to get the proper B+ tree for that attribute type (line 12), and insert the node into the inverted list (line 13), where it will return a pointer pointing to the inverted list. Then we need to insert the pointer to point match entrance (line 14). After finish the parsing, we can return the index (line 21).

For example, for a query “Winston book price:<15” issued on the XML data in Figure 2, there are two normal terms extracted, namely “Winston” and “book”, and one range term extracted, namely “price:<15”. For “Winston” and “book”, we can retrieve two inverted lists using the point match entrance in Figure 5. For “price:<15”, we can proceed to the range match entrance. With the help of the hash map, we can find the B+ tree for price, where we can also easily find the nodes within the specified range. The three inverted lists retrieved are shown in Figure 6.

Algorithm 1: buildIndex(XML)

- **Input**: XML data tree XML
- **Output**: Point & Range Hybrid-entrance Index
  1. Leaf pointEntrance;
  2. Hashmap <Type,Btree> rangeEntrance;
  3. foreach node n in XML do
     4. dewey = getDeweyLabel(n);
     5. keywords = getContents(n);
     6. foreach keyword k ∈ keywords do
        7. if isLeafNode(n) = true then
           8. type = getAttributeType(n);
           9. if supportRangeTerm(type) = true then
              10. if rangeEntrance.contains(type) = false then
                  11. rangeEntrance.add(type, new Btree);
                  12. tree = rangeEntrance.get(type);
                  13. pointer = tree.insert(k, tree.get(k) ∪ dewey);
                  14. pointEntrance.insert(k, pointEntrance.get(k) ∪ pointer);
                 15. else
                    16. pointEntrance.insert(k, pointEntrance.get(k) ∪ dewey);
                17. else
                    18. pointEntrance.insert(k, pointEntrance.get(k) ∪ dewey);
          19. index.rangeEntrance = pointEntrance;
          20. index.rangeEntrance = rangeEntrance;
     21. return index;

Algorithm 2: search(Q, index)

- **Input**: user’s keyword query Q, Point & Range Hybrid-entrance Index
- **Output**: query results result
  1. terms = splitBySpace(Q);
  2. foreach term t ∈ terms do
     3. if isRangeTerm(t) = true then
        4. rangeTerm.add(t);
     5. else
        6. normalTerm.add(t);
     7. foreach normal term t ∈ normalTerm do
        8. list = index.pointEntrance.get(t);
        9. invertedLists.add(list);
     10. foreach range term t ∈ rangeTerm do
        11. list = index.rangeEntrance.get(t);
        12. invertedLists.add(list);
     13. result = LCA(invertedLists); //can also be any other keyword search method
     14. return result;

6.2 Query Processing

As discussed in Section 5.2, existing keyword search methods work based on the inverted lists of the terms extracted from the query. The main part of existing keyword search methods can work seamlessly with our new index. Algorithm 2 shows the procedure to do keyword query processing with our new index.

The input is the user’s query and the Point & Range Hybrid-entrance Index. The output will be the query results as any other keyword search methods. The keyword query will be split by space (line 1). Then we will check each term (line 2), if it is a range term according to our syntax in Section 4, it will be added to range term list (line 3-4). Otherwise it will be added to normal term list (line 5-6). Then for each normal term (line 7), we extract the inverted list from the point match entrance of the index (line 8) and store it for later use (line 9). For each range term (line 10), we extract the inverted list from the range match entrance of the index (line 11) and store it (line 12). With the inverted lists ready, we can pass
them to any existing keyword search methods to process it and get the results (line 13).

7. EXPERIMENTS

We have conducted some experiments to verify the performance of our approach to support range queries in XML keyword search. We mainly focus on two aspects: how well our **Point & Range Entrance Index** supports (1) keyword queries **without** range specification and (2) keyword queries **with** range specification.

Since our new index is transparent to existing keyword search methods and it can work with them seamlessly to support range queries, here we integrate our index with SLCA [16], which is one of the most efficient ones so far. For expository convenience, we refer to our approach as **Point&Range Search**. In the following experiments, we will compare **Point&Range Search** with the original XML keyword search method SLCA [19].

![Figure 7: Average Processing Time for Random Queries without Range Specification](image)

All experiments are conducted on a 2.83GHz Core 2 Quad machine with 3GB RAM running 32-bit windows 7. All codes are implemented in Java. The dataset we used is IMDB\[1\] , where around 200,000 movies of recent years are selected in our dataset. Each movie contains information like title, rating, year, director, cast, etc.

**Queries without Range Specification.** First of all, we compare the efficiency of **Point&Range Search** and SLCA on keyword queries without range specification. We generate some random queries with different number of keywords, where there are 100 queries for each category. The two approaches all return the same query results because their only difference is the index. Figure 7 shows average processing time of the two approaches. As we can see, the processing time of the two approaches are almost identical. This is because, as discussed in Section 5.2, the **Point Match Entrance** of our index can support queries without range specification almost the same way as the traditional index.

**Queries with Range Specification.** Secondly, we also test the efficiency of **Point&Range Search** and SLCA on keyword queries with range specification. We test ten sample queries with range specification. SLCA cannot capture the range specification and subsequently return empty result for all ten sample queries. In contrast, **Point&Range Search** captures the range specification and return the results according SLCA matching semantics. Figure 8 shows the processing time for the ten sample queries. SLCA takes little time to process because it terminates when it cannot find an inverted list for the range term in the keyword queries.

![Figure 8: Processing Time for Queries with Range Specification](image)

8. CONCLUSIONS

In this paper, we address the needs for supporting range queries in XML keyword search. We point out that existing keyword search methods do not support range queries at two levels: keyword query syntax level and keyword search method level. We propose a solution to extend existing keyword search methods to support range queries. First we enrich the current keyword query syntax to support range specification. Second we extend the current keyword search techniques to support range queries by proposing a new index, which can support range match and point match. The index can work seamlessly with existing keyword search methods.

9. REFERENCES


