Efficient Algorithms for Skyline Top-K Keyword Queries on XML Streams*

Lingli Li , Hongzhi Wang, Jianzhong Li, and Hong Gao

Harbin Institute of Technology
lwsbrr@gmail.com, {wangzh,lijzh,honggao}@hit.edu.cn

Abstract. Keywords are suitable for query XML streams without schema information. In current forms of keywords search on XML streams, rank functions do not always represent users' intentions. This paper addresses this problem in another aspect. In this paper, the skyline top-K keyword queries, a novel kind of keyword queries on XML streams, are presented. For such queries, skyline is used to choose results on XML streams without considering the complicated factors influencing the relevance to queries. With skyline query processing techniques, algorithms are presented to process skyline top-K keyword queries on XML streams efficiently. Extensive experiments are performed to verify the effectiveness and efficiency of the algorithms presented.

Keywords: XML streams, keyword search, top-K, skyline.

1 Introduction

In some application, XML data is accessible only in streaming form, which is termed **XML Stream.** Querying XML streams with keywords without schema information is in great demand. The keyword search on XML streams is that given a set of keywords Q, the set of all the XML fragments in the XML streams with each of them containing all the keywords is retrieved. We call a node in XML fragment a *keyword match* if its value contains any keyword in Q.

Since in XML streams, infinite XML fragments will come and there are too many XML fragments satisfying the keyword search, to find the most relevant results and to limit the number of results are both required during the processing of keyword queries. How to identify the most relevant results for users is still an open problem. There are many techniques proposed to identify and rank the relevance of query results on XML data[1-4]. But sometimes, results are incomparable and it is hard to find a rank function of relevance. In some cases even though users submit the same query, different results may be required. In such cases, to find a definite rank function to rank the relevance of results is improper. Without rank functions, a problem comes how to identify the top-K results the most relevant to the requirements of users.

* Support by the Key Program of the National Natural Science Foundation of China under Grant No.60533110; the National Grand Fundamental Research 973 Program of China under Grant No.2006CB303000; the National Natural Science Foundation of China under Grant No.60773068 and No.60773063.

X. Zhou et al. (Eds.): DASFAA 2009, LNCS 5463, pp. 283–287, 2009.

Inspired by [5] , skyline query is a good tool to effectively solve this problem. For the applying skyline on keyword search, the distances between keyword matches are used as properties in the skyline. We propose a new kind of queries, loose skyline top-K keyword queries on XML streams. An efficient algorithm is presented to process such queries on XML streams.

The rest of this paper is organized as follows. In Section 2, we present the definition and processing algorithm of loose skyline top-K keyword queries on XML streams. In Section 3 we evaluate the performance of these algorithms by experiments. Section 4 concludes the whole paper.

2 Loose Skyline top-K Query on XML Streams

In this section, skyline is applied to improve the result selection for keyword queries on XML streams.

In an XML tree, the distance between nodes n and e is the number of edges on the path of n-e, denoted as $d(n, e)$. If d is unreachable to e, then $d(n, e) = \infty$.

Definition 2.1(**Keyword Match**). For a keyword set $W= \{K_1, \ldots, K_N\}$ and a leaf node u in a XML segment, if the value of u contains some keyword $K_i \in W$, node u is said to match keyword K_i and u is a keyword match of K_i, denoted by $K_i \subseteq u$.

Definition 2.2 (**Smallest Lowest Common Ancestor (SLCA)**). Given a keyword query Q on XML data, an XML node i in the XML data is called SLCA if it contains all the keywords in Q in its subtree, and none of its descendants does.

Definition 2.3 (**Query Result**). Given a keyword query Q, an XML segment t on the XML data is called a *query result* if it satisfies (i) the root of t is a SLCA of Q and (ii) each leaf of t contains at least one keyword not contained by other leaves of t.

For any two leaves a and b in a query result T and their minimum common ancestor s, a smaller sum $d(a, s)+d(s, b)$ implies larger relevance. By the definition of distance, the smaller the distance between two leaves is, the larger the relevance between them is. For every keyword K_i in a keyword query W, the set $A_i=\{a|a$ is a leave of a query result T of W and a contains $K_i\}$. The *relevance* of keyword K_i and keyword K_j in query result T is $\min_{a_i \in A_i, a_j \in A_j} d(a_i, a_j)$, denoted by $R_T(i, j)$.

For a keyword query $Q =\{K_1, \ldots, K_N\}$, $P_Q=\{(i, j)|K_i \in Q, K_j \in Q \text{ and } i<j\}$. All pairs in P_Q are sorted in the order of two dimensions. The order of a pair (i, j) in P_Q is k_{ij}. A result T can be considered as a vector $(R_1(T), \ldots, R_{|P_Q|}(T))$, where $R_{k_{ij}}(T) = R_T(i, j)$.

Definition 2.4 (**Keyword Dominate**). For a keyword query $Q =\{K_1, \ldots, K_N\}$ on XML streams, a result T dominates T', denoted by $T \prec T'$, if $\forall$ k $(1 \le k \le |P_Q|)$, $R_k(T) \le R_k(T')$.

From the definition of relevance, for two results of a keyword query Q, T_1 and T_2, T_1 dominating T_2 means that T_1 is more relevant than T_2. For all results of a keyword query Q, the most relevant results must be those dominated by none of other result. Therefore, we have the definition of *skyline point* of a keyword query Q.

Definition 2.5 (**Skyline Point**). Given a keyword query $Q= \{K_1, K_2, \ldots, K_N\}$ on XML streams, the query result T is called the skyline point of the received XML fragments, if T is not dominated by any other query result.

With the consideration that more results than skyline points for a keyword query may be required, inspired by [7], *loose skyline top-K keyword query* is defined.

Definition 2.6 (**Loose Skyline top-K Keyword Query**). A loose skyline top-K keyword query(LSK query in brief) is a query Q= $\{W, K\}$, where $W=\{K_1,K_2\ldots, K_N\}$ is a keyword query, K is a constraint number. The results of Q on XML streams are a subset of the query results D of keyword query W, denoted as LS_D satisfying i) $\forall u \in LS_D$ is not dominated by any query result in D/LS_D ii) $|LS_D|=K$.

To process the LSK queries on XML streams, we propose efficient algorithms. The basic idea of the algorithms is that for a LSK query $Q=\{W, K\}$, a set of query results of W containing the results of Q is maintained as *intermediate results* and such set is updated when new query results of W are generated.

The intermediate results should contain the results of a LSK query $Q=\{W, K\}$ and be as small as possible for the convenience of processing. At first, the *skyline layer* as the unit of *intermediate result* is defined.

Definition 2.7 (**Skyline Layer**). For a keyword query $Q=\{K_1,K_2,\ldots,K_N\}$ and a set of query results M on XML streams, a subset of M is called *skyline layer i*($i\geq 1$), denoted as L_i if it satisfies: i) $\forall a,b\in L_i$, a is not dominated by b, b is not dominated by a; ii) $\forall a\in L_i$, if $i\geq 2$, $\exists b\in L_{i-1}$, $b \prec a$; if $i=1$, $\forall b\in L_j$ ($j\geq 1$) a is not dominated by b; iii) $\forall a\in L_i$, if $L_{i+1}\neq\emptyset$, $\exists b\in L_{i+1}$, $a \prec b$

By this way, M can be divided into skyline layers from L_1 to L_n($n\geq 1$). By using this definition, we present the *garlic rule* to select the *intermediate results* denoted as L, of a LSK query $Q=\{W, K\}$ from the result set M of keyword query W.

Garlic Rule: Initially, $L=\emptyset$. For the query result set M of Q, L_1 is added to L at first. If $|L|$ is larger than K or all query results have been processed, the selection is finished. Otherwise the next skyline layer L_2 is chosen to be added to L. Such steps are processed until a layer L_t is added to L and following constraints are satisfied: i)$|L_1\cup L_2\cup\ldots\cup L_t| \geq K$; ii)if $t>1$, $|L_1\cup L_2\cup\ldots\cup L_{t-1}| < K$.

With the intermediate result set L selected by garlic rule, the results of Q can be easily generated by randomly selecting some nodes from the layer L_t. That is, if $t=1$ and $|L_1|>K$, K results are selected randomly from L_1; otherwise, $K-|L_1\cup L_2\cup\ldots\cup L_{t-1}|$ results are randomly selected from L_t as L'_t and the results of Q is $L_1\cup L_2\cup\ldots\cup L_{t-1}\cup L'_t$. Such selection rule is called **the lowest layer rule**. Therefore, for a LSK query Q on XML streams, the result of Q can be easily generated from the intermediate results. The update algorithm for intermediate result is presented in *Algorithm1*. In this algorithm, a new result m is compared with intermediate results in L from the lowest layer L_t to the higher layers until an element u in some layer L_i dominates m or none of elements in the top layer dominates m. If m is dominated by some element in L_i, m should be in lower layer than L_i. If L_i is the lowest layer and $|L|\geq K$, m is discarded. Otherwise, m is inserted to L_{i+1} and the elements in L_{i+1} dominated by m are inserted to the lower layer L_{i+2}. Such process is continued until element set E is to be inserted to the lowest layer L_t. Once E is inserted to L_t, the element set E' with each element dominated by some element in E should be pushed to lower layer than L_t. If $|L|+1-|E'|\geq K$, E' is discarded; otherwise, a new layer is created as the lowest layer with elements in E'. If none of elements in the top layer dominates m, it means that none of elements in L dominates m. m should be inserted to L_1 and the elements dominated by m are processed similarly as above steps. In order to accelerate the pushing elements to lower levels, for each elements e in $L_i\subseteq L$, pointers to the elements in L_{i+1} dominated by e is maintained.

Algorithm 1
Input: $Q=\{W=\{K_1,K_2,\ldots K_N\},K\}$, intermediate result set L, new query result m
Isdominated=false
for i=t to 1 **do**
  for each $r \in L_i$ **do**
  if r dominates m **then**
    Isdominated=true; Break;
  if Isdominated=true **then**
    l=i+1; Break;
if Isdominated=false **then**
  l=1
if l=t+1 **then**
  if |L|<K **then**
    add a new layer L_{t+1}={m} to L
  else
    add m to L_l
    E={u|u$\in L_l \wedge$ m $\prec$u}
    while $l \neq$t **do**
      $L_l=L_l$-E
      E={u|u$\in L_{l+1} \wedge \exists$v$\in$E,v $\prec$u }
      $L_t=L_t$-E
    if |L|-|E|<K **then**
    add a new laver L_{t+1}=E to L

Algorithm 2
Input: $Q=\{W,K\}$
Output: LS // the results of query Q
BEGIN(n)
  n.state=00…0
  Stk.push(n)
endfunction
TEXT(n)
  for each $K_i \in$ W **do**
    if $K_i \subseteq$n.v **then**
      n.state= n.state OR 2^i
      P[m][i].insert(n.id)
        //m is the parent of node n
endfunction
END(n)
//the descendants of node n contain all keywords
  if n.state =2^N-1 **then**
    Generate query result set M with root n
    for each result m in M **do**
      Update_intermediate(Q, L, m)
    Select LS from L with the lowest layer rule
  else
    Stk.top().state= Stk.top().state OR n.state
  Stk.pop()
endfunction

With intermediate result update algorithm just discussed, LSK query processing algorithm on XML streams is presented. To represent the structural relationship among nodes in XML streams, they are encoded by Dewey Code[6]. The state of each active node n is represented by an N-bits binary signature s_N. If any descendant of n contains keyword K_i, the ith position of s_N is set to 1; otherwise, it is set to 0.

In our algorithm, a stack is used to store the coding and states of all active nodes in the incoming sequence. An array P is used to store the keyword lists of all the non-leaf active nodes. For each non-leaf node i, the entry $P[i][j]$ stores all the descendants of node i containing keyword K_j. L stores current intermediate result set and LS stores the current results for the query.

The details of the algorithms are shown in *Algorithm2*. Node n is initialized by coding n with Dewey Code and pushing node n into the stack when opening tag of n is encountered (in Begin()). When the closing tag of an element n is received (in End()), the temporal results with n as root are generated by every keyword list K_i of node n stored in $P[n][i]$. The intermediate result L is updated with elements in the result $P[n][1]\times P[n][2]\times\ldots\times P[n][N]$ and current results are generated from L with the *lowest layer rule*. The details of the algorithms are shown in *Algorithm2*.

3 Experiments

We have implemented all the algorithms we proposed with reading XML files once from the disk to simulate XML streams. Two datasets, XMark and DBLP, are used. For each dataset, we selected a set of queries in order to test the efficiencies on different queries, we used both frequent keyword sets and rare keyword sets.

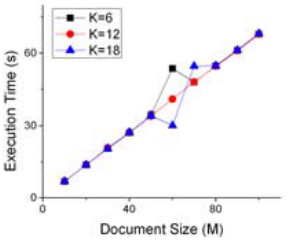

Fig. 1. Run time VS Doc. Size

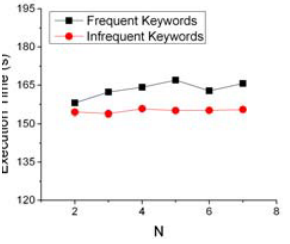

Fig. 2. Run time VS N

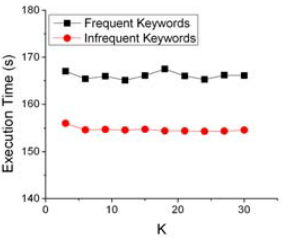

Fig. 3. Run time VS K

Scalability Experiments. In order to test the scalability, we generate XMark files with different parameters from 0.1 to 1.0 and the number of result *K* from 6 to 18. The experimental results are shown in Fig.1. From the results, the execution time is nearly linear to the number of elements.

Changing N. From Fig.2, it can be seen that both rare keyword sets and frequent keyword sets are insensitive to the value of *N*. That is because that in most cases the SLCA nodes has a small number of keyword matches.

Changing K. From Fig.3 we observe that, neither frequent nor infrequent query set are sensitive to the value of *K*. That is because in most cases, only a small number of query results need to be inserted to *L* instead of all of them.

4 Conclusion

With the broader application of XML data streams, top-k keyword search has become an important query on XML streams. Considering the different demands under the same keyword set query, in this paper, we propose the loose skyline top-K keyword query on XML streams. And we also propose effective algorithms for such kind of queries. Both analysis and experiments show that the techniques in this paper can obtain the search results efficiently and effectively with good scalability.

References

1. Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword Proximity Search on XML Graphs. In: ICDE (2003)
2. Barg, M., Wong, R.K.: Structural proximity searching for large collections of semi-structured data. In: CIKM (2001)
3. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: A Semantic Search Engine for XML. In: VLDB (2003)
4. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked Keyword Search over XML Documents. In: SIGMOD (2003)
5. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proc. 17th Int. Conf. on Data Engineering (2001)
6. Tatarinov, I., Viglas, S., Beyer, K.S., et al.: Storing and Querying Ordered XmlUsing a Relational Database System. In: SIGMOD (2002)
7. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic Skylines on Uncertain Data. In: VLDB (2007)