A System for Keyword Search on Probability XML Data

Weidong Yang\textsuperscript{1}, Hao Zhu\textsuperscript{1}, Zheng Zheng\textsuperscript{1}, Huirong Chen\textsuperscript{2}, Lei Wang\textsuperscript{2}

\textsuperscript{1}Computer School, Fudan University, Shanghai, China
\textsuperscript{2}Commercial Aircraft Corporation of China, Ltd, Shanghai, China

Abstract—Many probabilistic XML data models have been proposed to store XML data with uncertainty information, and based on them the issues such as structured querying are extensively studied. As an alternative to structured querying, keyword search in probabilistic XML data needs to be concerned. In this paper we addressed the issue of keyword search on probabilistic XML data. The probabilistic XML data is viewed as a labeled tree, and a concept of Minimum Meaningful Fragment (MMF) is defined as the searching result. A MMF is a minimum subtree of the probabilistic XML data which has a positive probability of containing all keywords. To sort the MMFs a novel scoring function mainly considering the degree of uncertainty information is presented. We propose a system to compute top-k searching results efficiently based on the scoring function. The experiments shows the efficiency for our system.

Keywords: Probabilistic XML Data, Keyword Search

1. Introduction

Recently, there is a growing interest in researching XML data with uncertainty. In the last few years, a plethora of probabilistic XML data models have been proposed \cite{4,7,1,2,5,3}, and most of them are modeled in trees. B. Kimelfeld et al. performed an elaborate survey of them in \cite{6}. They also presented a flexible model called p-documents in \cite{5}, trying to cover all existing ones. To the best of our knowledge, there is no research about keyword search in probabilistic XML data before this. However, it is very natural to employ keyword search over probabilistic XML data. As a typical scenario, integrating heterogeneous XML data not only generates a mass of uncertainty in the result, but also lets the schemas go out of control. As an effective information discovery technique keyword search is very suitable to such a case \cite{9}.

On the other hand, many effective approaches have been proposed to do keyword search on XML data, and the most popular ones of them are the SLCA (Smallest Lowest Common Ancestor) method \cite{8}. In SLCA method the XML document is viewed as a rooted, labeled, unordered tree and a searching result is defined as a subtree of it that: (1) the labels of whose nodes contain all the keywords, (2) none of its subtree satisfies the first condition except itself. The root of such a subtree is called a SLCA node. Similarly, ELCA method is trying to find a set of ELCA nodes which is a superset of all SLCA nodes. There is a naive way to find all ELCA nodes, and this will help us to understand the concept of an ELCA node: first, retrieve all SLCA nodes and remove all the subtrees rooted in them from the tree; second, repeat the first step until there is no SLCA node can be found.

To evaluate queries on probabilistic data, a well known model adopted is the possible world model in which each possible world of the original data is a piece of deterministic data with its existence probability. The query will be executed upon each possible world and some deterministic results can be obtained. Afterwards all the same results are clustered into one with their corresponding possible world’s probabilities being summed up. At last, a group of separate results are obtained and each is attached with a value to indicate its probability of existence. The same model is adopted by us. The searching object is defined as a Probabilistic XML Tree which is a family from p-documents (denoted as PrXML\textsuperscript{ind,\textit{max}} in \cite{5}). In a naive way, we generate all the possible pieces of the tree and calculate the probabilities, then evaluate the keyword search on them and retrieve the ELCA nodes. Each ELCA node is considered as a searching result, and the probabilities of the same result are added up. Since the number of possible worlds is exponentially large, we provide another efficient approach to retrieve all the ELCA nodes and get their probabilities. Also, we prove the correctness of the approach.

The remainder of this paper is organized as follows. Section 2 gives some preliminary definitions. Section 3 presents some basic formulas for calculating the uncertainty information. In Section 4, we propose the ranking model for results Afterwards, the algorithms of finding top-k results and system are given in Section 5. Experimental results are exhibited in Section 6.

2. Preliminaries

We define the searching object as a tree structure called Probabilistic XML Tree. The formal definition of a probabilistic XML tree is as follows.

Definition 1. (Probabilistic XML Tree) A probabilistic XML tree (PXT) \textit{p} is an 8-tuple \textit{p} = (\textit{O}, \textit{D}, \textit{E}, \textit{root}, \textit{L}, \lambda, \sigma, \omega), in which:
Definition 2. (Possible Document of Probabilistic XML Tree) For a PXT $p = (O, D, E, root, L, \lambda, \sigma, \omega)$, a subtree $m = (O, D, E, root, L, \lambda, \sigma, \omega) \subseteq (O, D, E, root, L, \lambda, \sigma, \omega)$ is a possible document of tree $p$ if (1) the root node of $m$ is root; (2) for any distributional node $d$ in $m$ that $\omega(d) = \text{max}$, $d$ can have at most one child node in $m$. A corresponding possible world can be easily built by removing all the distributional nodes in $m$ and connecting the ordinary nodes directly.

Figure 1 (b) illustrates three of the possible document of the PXT in Figure 1 (a). The calculation for each distributional node is described as follows.

Given a distributional node $d_i$, $C$ is the set of its child nodes. Suppose we choose a subset from $C$ as $C'$ and $C'$ is the set of $d_i$'s children that are not in $C'$. If $\omega(d_i) = \text{ind}$, then apparently the probability of choosing the subset $C'$ is $\prod_{v \in C'} \sigma(v) \prod_{v \notin C'} (1 - \sigma(v))$. Otherwise if $\omega(d_i) = \text{max}$, there are only two cases to choose child nodes. Case one is to choose any child, and the second case is choosing none. If choose $c_i$ from $C$, then the probability is simply $\sigma(c_i)$ itself. And when choose none of the children, the probability will be $1 - \sum_{v \in C} \sigma(v)$.

Let $pw(p)$ be the set of all the possible documents of $p$ and $g$ is any one in it. We use $Pr(g)$ to denote the probability of $g$, then it can be easily proved that $\sum_{g \in pw(p)} Pr(g) = 1$.

Definition 3. (Minimum Meaningful Fragment) A minimum meaningful fragment (MMF) of searching $W$ in $p$ is a subtree $m = (O, D, E, root, L, \lambda, \sigma, \omega)$ of $p$ which satisfies: (1) the labels of nodes in $K \subseteq D$ contain all keywords in $W$. (2) $\forall n_i, n_j \in K, lca(n_i, n_j) \in O$ or $lca(n_i, n_j) \in D, \omega(lca(n_i, n_j)) = \text{ind}$. (3) no subtree of $m$ is also a MMF except $m$ itself.

We use MMF as searching results. It is for two reasons: First, we do believe the structure is the most important property of XML keyword searching results needs to be kept...
introduced as follows. For a node set ones while some are ind or only lca node. Second, next, PXT which still keeps the root of nodes from 1 pdt of of N node in contains all the nodes which don’t have any descendant (denoted as \( pds(N) \)) of \( N \) node. Hence, \( Pr(n_j) = \sigma(n_j) \times Pr(n_i) \) through the Bayes’ Theorem. Since \( n_j \) only exists when \( n_i \) surely does, \( Pr(n_i|n_j) = 1 \). Hence, \( Pr(n_i) = \sigma(n_j) \times Pr(n_i) \). Furthermore, in terms of this formula we can calculate the global probability of any node by multiplying the local probabilities of all nodes on the path from the node to the PXT’s root. For instance, in Figure 1 it’s easy to find that \( Pr(n_{11}) = 0.9 \times 0.5 \times 0.8 = 0.36 \) and \( Pr(n_{23}) = 0.4 \times 0.5 = 0.2 \).

Definition 4. (Probability Dependency Tree and Set) For a certain PXT \( p = (O, D, E, root, L, \lambda, \sigma, \omega) \) and any node set \( N \subseteq O \) and \( N \neq \emptyset \), another PXT \( p' = (O', D', E', root', L', \lambda', \sigma', \omega') \) can be formed from \( p \) by keeping the original root (let \( root' = root \)) and using all nodes from \( descendants(N) \) as leaf-nodes. The function \( descendants(N) \) returns a node set \( N' \subseteq N \), and \( N' \) contains all the nodes which don’t have any descendant node in \( N \). \( p' \) is called the Probability Dependency Tree of \( N \) in \( p \), and \( O' \cup D' \) is called the Probability Dependency Set of \( N \) in \( p \). Moreover, they are denoted as \( pdt(N) \) and \( pds(N) \) respectively. For example, in Figure 1 \( pdt(\{n_3, n_5, n_6, n_8, n_{11}, n_{13}\}) \) is a part of the original PXT which still keeps the root \( n_1 \) and uses nodes in \( \{n_6, n_{11}, n_{13}\} \) as leaves. Then, \( pds(\{n_3, n_5, n_6, n_8, n_{11}, n_{13}\}) \) is the set of sequential nodes from \( n_1 \) to \( n_{13} \).

Given any node set \( N \), to compute \( Pr(\cap_{n_i \in N} n_i) \) and \( Pr(\cup_{n_i \in N} n_i) \) we conclude the possibilities of the distributions between nodes into three cases. First, \( \forall n_i, n_j \in N, n_i \neq n_j \) and \( n_j \neq n_i \): \( lca(\{n_i, n_j\}) \) is a max distributional node. Second, \( \forall n_i, n_j \in N, n_i \neq n_j \) and \( n_j \neq n_i \): \( lca(\{n_i, n_j\}) \) is an ordinary node or an ind distributional node. Third, other situations. In other words, for all nodes in \( pdt(N) \), either only max distributions exist between siblings, or only ind distributions do, or some distributions are mnx ones while some are ind ones. Two integrated formulas are mentioned as follows. For a node set \( N \) with \( n \) random nodes in a PXT, we have:

\[
Pr(\cap_{n_i \in N} n_i) = \prod_{n_i \in pds(N)} \sigma(n_i) \quad \text{only ind}
\]

(1)

and

\[
Pr(\cup_{n_i \in N} n_i) = \sum_{n_i \in N} \prod_{n_j \in pds(n_i)} \sigma(n_j) \quad \text{only mnx}
\]

(2)
in which

\[
F = \sum_{k=1}^{n} \sum_{I \subseteq \{1, \ldots, n\}, |I|=k} Pr(\cap_{j \in I} n_j)
\]

is a formula of the Inclusion-Exclusion Principle.

4. Ranking

In this section, we investigate the extra factor needed to be considered when ranking the results of searching probabilistic XML data comparing to the ranking models in conventional XML keyword search approaches.

Before discussing how to score a MMF, let us review the ranking models in conventional XML keyword search researches. For convenience, we name a node whose label contains some keyword as a keyword node. When searching results are regarded as XML fragments with various kinds of structures, most of the former researches employed simple approaches considering intuitive factors of these structures. Commonly accepted factors are: result size (amount of all nodes), number of keyword nodes, number of distinct keywords, and the compactness (number of keyword nodes divided by result size). When applying keyword search techniques to retrieve meaningful information from probabilistic XML data, an extra factor needs to be imposed on the scoring function to reflect the uncertainty degree in results. Since the factor is proposed from a totally different aspect, the conventional ranking factors are orthogonal to it.

For a MMF \( r \) not all its random documents are meaningful to users. We define a concept of Meaningful Random Document as follows.

Definition 5. (Meaningful Random Document) For a PXT \( p \) and a keyword set \( W \), one of \( p's \) MMF is \( r \). Then, a random document of \( r \) can be generated in two steps. First, for any distributional node \( d_i \) in \( r \) whose ancestor nodes are all ordinary ones, either remove \( d_i \) and all its descendant nodes from \( r \), or (1) choose any number of \( d_i \)'s children and connect the subtrees rooted in them to \( d_i \)'s parent if \( \omega(d_i) = \text{ind} \), (2) choose one child and connect the subtree rooted in it to \( d_i \)'s parent if \( \omega(d_i) = \text{mnx} \). Second, implement the first step repeatedly until there is no distributional node. Finally, a random document of MMF is
called a meaningful random document (MRD) if the labels of nodes contains all keywords in $W$.

We use $mrd(r)$ to denote the set of meaningful random documents of $r$. For any $rd \in mrd(r)$, suppose the probability of $rd$ is $RP(rd, r)$ and $D$ is the set of distributional nodes in $r$ considered when generating $rd$. Then, apparently

$$RP(rd, r) = \prod_{d_i \in D} P(d_i, rd),$$

in which: if $\omega(d_i) = ind$ and none of $d_i$’s child is chosen,

$$P(d_i, rd) = \prod_{n_j \in children(d_i)} (1 - \sigma(n_j));$$

else if $\omega(d_i) = ind$ and nodes in $C \subseteq children(d_i)$ are chosen,

$$P(d_i, rd) = \prod_{n_j \in C} \sigma(n_j) \times \prod_{n_k \in children(d_i), n_k \notin C} (1 - \sigma(n_k));$$

else if $\omega(d_i) = max$ and none of $d_i$’s child is chosen,

$$P(d_i, rd) = 1 - \sum_{n_j \in children(d_i)} \sigma(n_j);$$

else if $\omega(d_i) = max$ and $d_i$’s child $c_j$ are chosen,

$$P(d_i, rd) = \sigma(c_j).$$

Formally, for a MMF $r$ in which $rt$ is the root, we define the degree of uncertainty in $r$ as the Uncertainty Score of $r$ denoted as $US(r)$, and

$$US(r) = Pr(rt) \times \sum_{rd \in mrd(r)} RP(rd, r) \tag{3}$$

This is actually the extra factor reflecting the uncertainty degree of searching results, and $Pr(rt)$ is taken into account because the existence of $rt$ is the precondition of calculating the uncertainty information of $r$. As mentioned the scoring functions of other intuitive factors are orthogonal to $US(r)$, thus we define a composite function for other factors as $OS(r)$. Finally, we have the ultimate scoring function $score(r)$ of a MMF $r$ as follows.

$$score(r) = US(r)^{\alpha} \times OS(r)^{\beta} \tag{4}$$

5. Retrieving Top-k Results

5.1 Indistinguishable Set and Score Bounds

As mentioned in Section 2, our ultimate purpose is: given a PXT $p$, a keyword set $W$, and a positive integer $k$ given by users, finding $k$ MMFs with the largest ranking scores. When calculating the ranking score for a MMF $r$ ($US(r) \times OS(r)$), $OS(r)$ (the number of keywords in $r$) is quite easy to obtain while $US(r)$ is not. Actually, to figure out the uncertainty score for each MMF is a nontrivial task. Therefore, the key problem here is: how to obtain top-$k$ results with computing as less uncertainty scores as possible. We propose a simple yet effective ranking algorithm as follows: when retrieving a MMF, lower and upper bounds of its score are calculated, which costs negligible time; at the same time an Indistinguishable Set of MMFs is maintained; after the results-finding process the top-$k$ results are affirmatively in the set, then the real scores of the MMFs in this set are calculated and top-$k$ results are obtained through any popular sorting algorithm.

Definition 6. (Indistinguishable Set) For any MMF $r$, suppose $ub(r)$ and $lb(r)$ are the upper-bound and lower-bound functions of $r$ respectively (which means $lb(r) \leq score(r) \leq ub(r)$). Given a certain positive integer $k$ and a MMF set $R$ which satisfies $|R| > k$, we can sort the MMFs in $R$ by the upper bounds in descending order and get a list $ub(l)(R)$, also we can sort them by the lower bounds in descending order and get a list $lb(l)(R)$, then $R$ is called a $k$-indistinguishable-set ($k$-i-set) if: the upper bound of the last MMF in $ub(l)(R)$ is greater than or equal to the lower bound of the $k$th MMF in $lb(l)(R)$. This kind of set is called an indistinguishable set because in a top-$k$ search results cannot be distinguished from it yet. The detail of the procedure is presented in Section 5.2.

Definition 7. (Tightest Meaningful Tree) Let $K$ be the set of all keyword nodes in the MMF $r$, then a PXT is called the Tightest Meaningful Tree generated from $r$ if it uses $lca(K)$ as root and the nodes in $descendants(K)$ as leaves. The tightest meaningful tree of $r$ is denoted as $tmt(r)$.

Upper Bound 1. Suppose $rt'$ is the root of $tmt(r)$, then $Pr(rt')$ is an upper bound of $US(r)$. Since $rt'$ is an ancestor of any keyword node (or itself could be one), $US(r) = Pr(E) = Pr(E|rt') \times Pr(r') \leq Pr(r')$.

Upper Bound 2. $Pr(\bigcup_{n_i \in children(rt')} n_i)$ is used as the second upper bound, in which $children(rt')$ is the child-node set of $rt'$ in $tmt(r)$.

Upper Bound 3. Using the same proof from upper bound 2, a smaller upper bound can be deduced. We use a function $ancestors(K)$ to get a keyword-node set $K' \subseteq K$, and $K'$ contains all the nodes in $K$ which don’t have any ancestors in $K$. Obviously the nodes in $ancestors(K)$ separates all the keyword nodes below (only $rt$ could be an exception). Consequently, we can have the upper bound 3 as the middle one of following inequation:

$$US(r) \leq Pr(\bigcup_{n_i \in ancestors(K)} n_i) \leq Pr(\bigcup_{n_i \in children(rt')} n_i) \tag{5}$$

Next three lower bounds of $US(r)$ are introduced. As we will see the lower bounds are set more casually.

Lower Bound 1. Suppose a random set $R \subseteq K$, $R$ contains all keywords and $|R| = 1$ as the number of keywords. Thus, we have $Pr(\bigcap_{n_i \in R} n_i) \leq US(r)$.

Lower Bound 2. For any keyword in the $R$ of lower bound 1, apparently a higher place of it in the tree often brings a higher probability. So, for each keyword-node set
of a keyword we choose the one which has a shortest path to the root. Therefore a set \( R' \) is obtained, and we have
\[
Pr(\bigcap_{i \in R'} n_i) \leq US(r).
\]

**Lower Bound 3.** Actually we can get a lower bound more casually. If \( E(N) \) is used as the DNF logic formula which denotes “the probability \( N \) containing all keywords”. Then, obviously for any \( N \subseteq K \) we have \( Pr(E(N)) \leq US(r) \), and based on \( N \) a subpart of the MMF can be generated and afterwards a new uncertainty score can be figured out. Apparently, it is significant to choose a small number of \( N \) and get a close value to \( US(r) \), however there is one thing should be noticed that whatever a keyword-node set is chosen, it must contain all the keywords.

### 5.2 System Overview

Here, we proposed the overview architecture for our algorithms.

The system gets the keyword query from users, and perform query algorithm on the target PXT.

Firstly, the system retrieved all MMFs satisfying the query. As a keyword searching result MMF is defined as a subtree of the PXT, to retrieve a MMF is equal to finding its root. Hence, For the target PXT, we use the procedure \texttt{get\_m\_roots} to denote finding the roots of all MMFs that satisfying the query.

Then, for these candidate MMFs, we are filtering out some candidates which are definitely not in the top-k results using Score Bounds introduced in the previous section. The system use the procedure \texttt{update\_iset} in the indistinguishable set and eliminate these candidates from the set.

Finally, the system calculates the actual ranking scores for candidates in the indistinguishable set and output the top-k results to the users.

These procedures are illustrated in details in the following section.

### 5.3 System in details

The procedure \texttt{update\_iset}(r, R) illustrates the process of updating and maintaining the indistinguishable set.

To implement the results-finding algorithm, for a PXT \( p \) as the searching object we code all the nodes in it with Dewey Code and build an inverted list for all terms in the labels of all the ordinary nodes. For any term (input keyword), we can find a list storing the Dewey codes as the occurrence positions in the tree in pre-order. For any node \( n_i \) in \( p \), a function \( pre(n_i) \) is defined to get the sequence number of \( n_i \) in pre-order. Thus \( pre(n_i) < pre(n_j) \) means \( n_i \) is at a position top-left to \( n_j \) in the tree. Furthermore, two mapping tables are also built to store the types of a distributional nodes and the local probabilities.

```plaintext
procedure update_iset(r, R)
Input: k, i-set \( R \), ubl\((R)\), lbl\((R)\), MMF r
Output: \emptyset

1: if \(|R| < k\) then
2: insert r into R;
3: insert r into ubl\((R)\), lbl\((R)\) at the right places;
4: else
5: \( l_k = \) the kth item in \( \text{lbl}((R)) \);
6: if \( \text{ubl}(r) \geq \text{lbl}(l_k) \) then
7: insert r into R;
8: insert r into ubl\((R)\), lbl\((R)\) at the right places;
9: \( l_k = \) the kth item in \( \text{lbl}((R)) \);
10: for each \( r' \) in \( \text{ubl}(R) \) (from tail to head) do
11: if \( \text{ubl}(r') < \text{lbl}(l_k) \) then
12: remove \( r' \) from \( R \), ubl\((R)\), and lbl\((R)\);
13: else
14: break;
```

As a keyword searching result MMF is defined as a subtree of the PXT, to retrieve a MMF is equal to finding its root. For a PXT \( p \) and a set of keywords \( W = \{w_1, w_2, \ldots, w_l\} \), \( K_i \) is the set of keyword nodes whose labels contain \( w_i \) in \( p \). We use \( m\_\text{roots}(K_1, K_2, \ldots, K_t) \) to denote the procedure of finding the roots of all MMFs of searching all the keywords in \( p \), and if only the keywords \( \{w_1, w_2, \ldots, w_{l-1}\} \) are considered then the corresponding roots are \( m\_\text{roots}(K_1, K_2, \ldots, K_t) \). We use \( K' \) to denote the list of the roots, and the nodes in \( K' \) are regarded as keyword nodes whose labels contain a new keyword \( w' \). Then the MMF roots of searching \( \{w', w_t\} \) in \( p \) are \( m\_\text{roots}(K', K_t) \). It can be easily proved that:

\[
m\_\text{roots}(K_1, K_2, \ldots, K_t) = m\_\text{roots}(K', K_t)
\]

Apparently, through conducting the formula recursively we can turn the problem into finding the MMF roots for two keywords \( m\_\text{roots}(K_1, K_2) \). Suppose \( n_i \) is a certain node in \( K_i \), \( \forall n_j \in K_j \), \( pre(n_j) < pre(n_i) \), and \( \omega(lca\{n_i, n_j\}) \neq \text{max} \), we call the one with the largest \( pre(n_j) \) as the left neighbor of \( n_i \) in \( K_j \). Similarly, \( \forall n_j \in K_j \), \( pre(n_j) > pre(n_i) \), and \( \omega(lca\{n_i, n_j\}) \neq \text{max} \), we call the one with the smallest \( pre(n_j) \) as the right neighbor of \( n_i \) in \( K_j \). Moreover, the left and right neighbors of \( n_i \) in \( K_j \) are denoted as \( \text{ln}(n_i, K_j) \) and \( \text{rn}(n_i, K_j) \) respectively. Then,
the procedure of getting the MMF roots for two keywords \( w_1 \) and \( w_2 \) is as follows.

The approach of finding all MMFs is similar to the algorithm proposed in [8]. The only difference is the definition of two neighbor nodes. Finally, the algorithm of finding top-\( k \) MMFs is given.

```
procedure get_m_roots(K_i, K_j)
Input: K_i, K_j, and R = ∅
Output: R
1: mr = the root of the PXT;
2: for each node \( n_i \) in \( K_i \)
3: if lca(\{\( n_i, ln(n_i, K_i) \}) < lca(\{\( n_i, rn(n_i, K_i) \})
4: l = lca(\{\( n_i, rn(n_i, K_i) \});
5: else
6: l = lca(\{\( n_i, ln(n_i, K_i) \});
7: if pre(mr) ⊆ pre(l)
8: if mr ⊈ l
9: add mr in R;
10: mr = l;
11: return R ∪ \{mr\}
```

Algorithm Finding top-\( k \) MMFs
Input: the PXT \( p \), keyword set \( W = \{w_1, w_2, \ldots, w_t\} \)
Output: top-\( k \) MMFs with the largest scores

```
1: R = ∅; // i-set
2: when considering the last two keyword sets;
3: for each MMF \( r \) retrieved
4: update_iset(r, R);
5: compute the ranking score for each MMF in \( R \);
6: sort \( R \) according to scores;
7: return top-\( k \) MMFs in \( R \) with largest scores;
```

Fig. 3: Average number of keyword nodes in a MMF

6. Experiments

The main purposes of our experiments include: (1) to show the efficiency of calculating the scores (2) to estimate the efficiency of the algorithm retrieving top-\( k \) MMFs provided in Section 5. The hardware environment of the experiments is a laptop with a 2.1GHZ Duo-CPU and 2G RAM running Windows XP. All programs are developed in Java 6.0. We added the uncertainty information into the encrypted TreeBank data set (document size 82M, containing 2437667 nodes, max-depth 36, and average-depth 7.9) to build a PXT as the searching object. A certain amount of distributional nodes with random types are inserted into the original XML tree, afterwards any child node of them are given a random real number in (0, 1]. We call the generated PXT as the P-TreeBank data set, and a vocabulary containing frequent terms is built upon it. To get more exact results, some keywords are randomly chosen from the vocabulary and then utilized to search the P-TreeBank. For a certain number of keywords, the process is conducted for many times, and then average values are figured out as the final experimental results.

6.1 Calculating ranking scores

As we can see, when computing the ranking score for any MMF it is set to be conducted automatically each time a new MMF is generated or a MMF is modified. It means at any time we are considering a tightest meaningful tree. Figure 3 illustrates the average number of keyword nodes in MMF for each certain number of keywords. For a MMF \( r \), suppose \( K \) is the set of keyword nodes in \( r \), then apparently in the worst case calculating \( US(r) \) will cost \( O(|K| \times 2^{|K|}) \). Although the computation complexity could fall to \( O(|K|) \) through employing the strategies, we cannot calculate the score with the brute approach when \( |K| \) is really large. Actually it is found that the time cost becomes unbearable when \( |K| > 27 \). From the semantics of a MMF, we can see that such condition is really hard to fulfill except some extreme cases. For example all the keyword nodes are siblings and share an ind distribution. Figure 4 shows that the time cost of calculating the uncertainty score will rise when there are more keyword nodes and more distributional nodes in the MMF. The cost doesn’t grow in geometric progression as in a first thought, because many ancestor-descendant relations exist between the distributional nodes, and in this case much less random documents are generated.

6.2 Retrieving top-\( k \) MMFs

To estimate the efficiency of the top-\( k \) results finding algorithm, 5 integers are selected as \( k \) (10, 20, 30, 40, and 50), and for each \( k \) the size of indistinguishable set is counted. Due to the difficulty of calculating ranking scores, the smaller the i-set size is, the higher efficiency we will get. Figure 5 shows the results when different bounds are used. “u-bound 2 & l-bound 2” means utilizing the upper bound 2 and lower bound 2 defined in Section 5, “u-bound 3 & l-bound 3” indicates using upper bound 3 and lower bound 3. The average amount of all the MMFs is 12834, thus we
can see the dramatic abatements of the MMFs needs to be considered to retrieve top-$k$ results when upper bound 3 and lower bound 3. Also, from their definitions it’s easy to find that these four bounds all have negligible costs.

### 6.3 Acknowledgments

This work is supported by Airplane Research Project(MJ-Y-2011-39), Shanghai High-Tech Project(11-43), Chinese Polar Project(CHINARE2012-04-07).

### References


