Hierarchical Indexing and Flexible Element Retrieval for Structured Document

Hang Cui[1*], Ji-Rong Wen[2], Tat-Seng Chua[1]

[1] Department of Computer Science,
School of Computing,
National University of Singapore, Singapore
cuihang@comp.nus.edu.sg
chuats@comp.nus.edu.sg

[2] Microsoft Research Asia
No.49 Zhichun Road Haidian District
Beijing, P.R.China
jrwen@microsoft.com

Abstract. As more and more structured documents, such as the SGML or XML documents, become available on the Web, there is a growing demand to develop effective structured document retrieval which exploits both content and hierarchical structure of documents and return document elements with appropriate granularity. Previous work on partial retrieval of structured document has limited applications due to the requirement of structured queries and restriction that the document structure cannot be traversed according to queries. In this paper, we put forward a method for flexible element retrieval which can retrieve relevant document elements with arbitrary granularity against natural language queries. The proposed techniques constitute a novel hierarchical index propagation and pruning mechanism and an algorithm of ranking document elements based on the hierarchical index. The experimental results show that our method significantly outperforms other existing methods. Our method also shows robustness to the long-standing problems of text length normalization and threshold setting in structured document retrieval.

1 Introduction

Traditional information retrieval treats document as the smallest retrieval unit, but in many scenarios a user may actually require part of the document with higher precision and finer granularity. Suppose a user who studies history of military operations would like to find out "what military aircrafts were used in Desert Storm". He or she may retrieve articles named *Military Aircrafts* and *Gulf War* as two of the top-ranked results, both of which contain only a part of relevant content. The user then has to scan each (usually very long) document to look for relevant information. This is a time-consuming process which hinders the effectiveness of information retrieval. Such an information overload is very common in typical Web searching applications.

[*] This work was performed when the author was a visiting student at Microsoft Research Asia.

Today, with the widely use of XML, there is an increasing demand to develop better techniques for structured document retrieval. XML provides a standard and effective way for the author to explicitly express the structure of a document. For example, our corpus from the Encarta website (http://encarta.msn.com) can be considered as a set of content-oriented XML documents. A typical structured document is represented as a collection of nodes such as sections, subsections, and paragraphs, as shown in Figure 1. We call each node as an element in the rest of this paper. The node representing the whole document, known as the root, is also considered as an element such that all nodes in the entire document tree are treated equally. The leaf nodes are made up of paragraphs. All upper-level nodes are ancestors of paragraphs, with their contents formed by those of paragraphs.

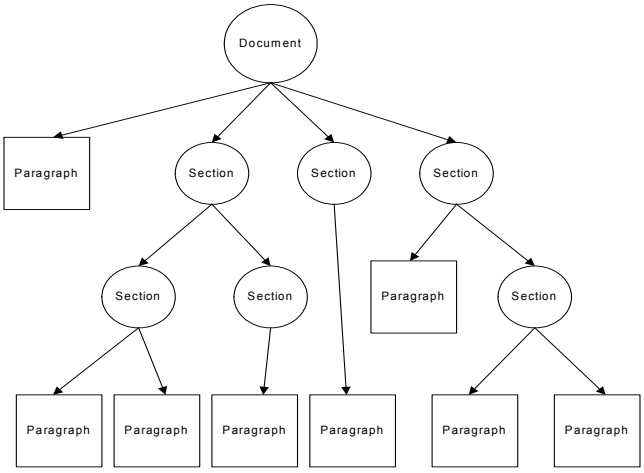

Figure 1. Document Structure in Encarta

A document, especially a long one, usually covers multiple aspects of a central topic. The elements in a document can be viewed as a concept tree, i.e., the upper element represents a broader concept which covers all the concepts beneath it. Document retrieval can only be partially called *information* retrieval unless the elements expressing the appropriate level of concept can be precisely retrieved. An effective retrieval system should provide this capability without imposing too much burden on users.

In this paper, we propose a method of retrieving relevant document elements, exploiting both structural information and the statistics of term distributions in structured documents. The main thrust of this solution is to allow the retrieval of relevant document elements with arbitrary granularity using keyword-based queries. We call it a flexible element retrieval strategy. Our solution is mainly made up of two parts – a novel hierarchical index propagation and pruning mechanism and an algorithm for selecting suitable document elements based on the hierarchical index.

Comparing to existing works, we put much emphasis on the indexing phase. Applying specific indices to retrieving data with structural information has long been studied in the areas of database and IR, such as indexing semi-structured data for

XML documents retrieval [12] and bottom-up indexing schemes for structured documents retrieval [11]. Previous approaches assign index terms to only the leaf nodes [10] [12] [14] [15] or fixed-length passages [3] [8] [9]. The main drawback of such a kind of indexing mechanism is that the flat index does not match the hierarchical structure of documents. It discards semantic relationships among the elements. The inconsistency between the structures of documents and indices prevents users from obtaining composite elements, thus results in many discrete passages, which leads to the tough work of assembling the resulting excerpts of text to users [15].

The essential problem for indexing structured document is that, in order to get elements at arbitrary levels, the weights for various elements against a given query must be comparable. [4] has a similar purpose in indexing and retrieving hypertext medical handbook in which related materials are represented as linked cards. In their method, the weight of a card *E* is determined by the TF-IDF values of all the query terms in *E* plus the average TF-IDF weights of all immediate-descendant elements of *E*. Card weights are propagated recursively from the leaf elements to the root element. This is one of the first works to index document elements by combining the content and structure information. But this method may not be practical to index a large amount of structured documents mainly due to two reasons. First, in the hypertext medical handbook model, every element has its own content and the contents of its descendant elements are only viewed as supplements to its own content. However, in the case of general structured documents, such as the XML documents, an intermediate element usually does not have its own content and it is totally made of the contents of its descendant elements. If the weight propagation technique in [4] is directly employed, the weight of a composite element without its own content will always be ranked lower than that of its descendents, because its weight is the average value of the summed weights of all its descendents. Consequently, the leaf elements will always be retrieved as best matches. Second, the propagation mechanism in [4] does not perform any pre-processing and thus the same index terms may be distributed in multiple elements of one document, which is very costly in terms of both storage space and computation time, especially when handling a large amount of documents. Moreover, the author did not give any quantitative evaluation of the proposed method. Thus it is hard to judge its effect in a real application scenario.

We approach the goal of flexible element retrieval by a hierarchical indexing mechanism, which is not only able to index the leaf nodes but also intermediate nodes, i.e. section and document nodes. Basically, we use a propagation and pruning mechanism to select index terms. From bottom up, terms that can "exactly" describe the inherent concept of an element are propagated to it while terms with too broad or too narrow meanings are pruned. Index pruning is employed to ensure that an index term appearing in an element would not appear in any of its descendent elements thus the content overlap in the text is avoided in the index. This saves much storage space and retrieval time. Moreover, this hierarchical indexing mechanism produces an index structure that is identical to the document structure. Hence we can perform document element retrieval on the index space directly. Figure 2 illustrates the process of index propagation and pruning. Assuming that we have a document named "China" with a section "History" and this section contains subsections such as "Tang dynasty", "Ming dynasty" and "Qing dynasty", etc. Then for the section "History", only terms

like “history” and “dynasty” are good index terms, while for the whole document, only the term “China” is the best choice.

Based on the hierarchical index, we also propose a flexible element retrieval algorithm to rank candidate elements against queries so that suitable document elements that precisely meet user’s information needs can be returned.

We conducted a series of experiments to evaluate the performance of our method in terms of precision and recall at element level. The results show that our method significantly outperforms the compared method and is less sensitive to threshold setting than the traditional passage retrieval methods.

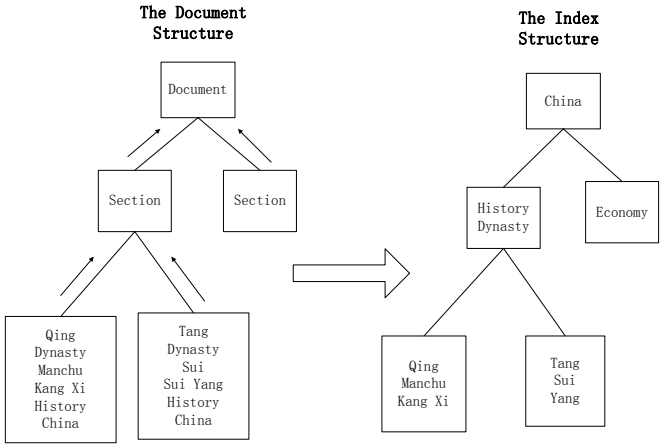

Figure 2. Index propagation and pruning mechanism

2 Related Work

In recent years, many structured document retrieval techniques have been developed. In traditional IR community, due to the absence of explicit structural information, documents are treated as a sequence of fixed-length [3] or pre-defined [15] portions of text, which are considered as passages or paragraphs. Passage retrieval [3] [8] [9] [15] [16] is one of the early techniques aiming to retrieve and return more compact and shorter answers at passage level to the user. A passage retrieval method usually indexes the documents at passage or paragraph level, and applies the variants of TFIDF measure to rank passages, while [13] is an exception, which suggests using Hidden Markov Model to retrieve both documents and passages. More recently, researchers start to address the problem of mixing content and structure in retrieval models [2]. [12] suggests a model containing a number of useful operators that can achieve relatively high efficiency.

Another group of methods, mainly developed by the database community, concentrate on retrieving specific fields of semi-structured or XML data by indexing structures and strictly defined query languages [1] [7]. In the case of XML query languages, these methods require the user to specify structured queries. However, without the knowledge of the document structure, it would be very hard for the users to

formulate meaningful queries. Moreover, only the data elements whose structures exactly match the specified query structure can be retrieved.

We found there was a lack of an appropriate method that balances the trade-off between the full utilization of document structure and the convenience of common users. Some researchers attempt to address this problem. [14] explores the use of inference network to represent elements of a document at different levels so that all elements can be treated equally. However it still has difficulty in properly ranking various elements with the existence of content overlaps. [6] proposes a new way to index a bibliography repository with a hierarchical structure. Focused retrieval method of locating document components that contain relevant information is introduced in [10]. [5] describes a new query language introducing some information retrieval features, such as weighting to XML documents retrieval.

3. Hierarchical Indexing of Structured Documents

In this section, we describe the details of our hierarchical indexing strategy. For each document, we automatically establish a hierarchical index with the same structure as that of the document. Index terms are distributed across all nodes in the document tree. The basic idea of assigning an index term to an element node is that the term should characterize the concept of this element and differentiate it from the others. Thus, a rule of thumb for selecting good index terms is that the term should appear frequently and be distributed evenly in the text of an element and, its rank is high compared to its peer terms.

3.1 Term Weighting for Elements

By taking advantage of the hierarchical structure of the documents, the distribution of a term in an element can be measured by investigating the term's appearances in the descendant elements of this element. It is noted here that we consider only immediate-descendant elements of the element because we believe that the topic of an element is best supported by the elements that it owns directly. If a term is distributed evenly in a composite element's immediate-descendant elements, this term would be a good candidate index term for this element.

We introduce the concept of entropy here as a criterion to measure the distribution of a term in an element. Here we distinguish between two types of elements – the intermediate elements and leaf elements which are paragraphs. For an intermediate element, we compute the weight of a term by combining the term's intra frequency in this element and the term's distribution in its immediate-descendent elements. That is the weight of term t_i in an arbitrary composite element E_j can be defined as:

$$Weight(t_i, E_j) = \ln(1 + tf(t_i, E_j)) \times I(t_i, E_j) \quad \textbf{(1)}$$

where $tf(t_i, E_j)$ denotes the frequency of term t_i in the element E_j. $I(t_i, E_j)$ is the entropy measure, i.e. the distribution of the term t_i in element E_j and is defined as:

$$I(t_i, E_j) = \frac{-\sum_{Sub_k \in E_j} tf(t_i, sub_k) \times \ln \frac{tf(t_i, sub_k)}{tf(t_i, E_j)}}{-\sum_{Sub_k \in E_j} \frac{tf(t_i, E_j)}{N(sub)} \times \ln \frac{1}{N(sub)}}$$

$$= \frac{-\sum_{Sub_k \in E_j} tf(t_i, sub_k) \times \ln \frac{tf(t_i, sub_k)}{tf(t_i, E_j)}}{-tf(t_i, E_j) \times \ln \frac{1}{N(sub)}} \quad (2)$$

where sub_k stands for the k^{th} immediate-descendant element of E_j and $N(sub)$ the number of such descendant elements.

In Equation 2, it is worthwhile to notice that term frequency varies greatly in different elements due to the great variance of text lengths. Entropy measure may encounter the same length normalization problem as in other document or passage retrieval methods. [3] [8] [9] [6] [14] [15] addressed the normalization of text length but were limited to the factor of term frequency. We compute the theoretic maximum entropy $-tf(t_i, E_j) \times \ln \frac{1}{N(sub)}$ and use this as normalization factor. It hypothesizes that all appearances of this term in a specific element are exactly equal in each of its immediate-descendant elements. The proportion of this value is used as the distribution measure of a term. It counters the negative effect of varying text lengths to some extent.

Leaf elements of paragraphs are "atomic" elements, which have no children elements, thus we simply employ the traditional TFIDF measure to compute the weight of terms in a single paragraph. A term's weight in a paragraph is defined as:

$$Weight(t_i, P_j) = \ln(tf(t_i, P_j)) \times \ln \frac{N}{n_i} \quad (3)$$

$Weight(t_i, P_j)$ represents the weight of term t_i in paragraph P_j. $tf(t_i, P_j)$ is the term frequency of t_i in the paragraph. N denotes the total number of documents in the corpus and n_i the number of documents containing t_i.

Term weights are further normalized to be comparable in different elements. Term weights obtained by Equations 1 and 3 are divided by the maximum weight of all terms in the same element so that all terms' weights fall into the range of between 0 and 1.

3.2 Propagation and Pruning of Index Terms

Recall that a term in an element whose weight is relatively high should be selected as the index term for this element. Specifically, the selection of index terms is realized by the propagation and pruning process. In the previous section, we derive the weights for each term in an arbitrary element. A term is propagated to an upper element if its weight exceeds a certain threshold, and meanwhile this term is pruned from these descendant elements since it may stand for a more general concept. This process is done recursively from bottom up until all the nodes in the tree are assigned

proper index terms without duplications in the same branch of the index tree. Obviously, the threshold controlling the term selection should be dynamically adjusted according to the statistics of all the terms' weights in a specific element. More precisely, a term is chosen as an index term for an element if and only if its weight is above the average value plus the standard deviation of all terms' weights in this element. Our indexing propagation and pruning mechanism can be described as follows:

Algorithm 1 – terms selection (index terms propagation and pruning)

1. For each leaf element, i.e. paragraph, calculate all terms' weights for paragraphs according to Equation (3).
2. For each composite element E_j at the next upper level, calculate the terms' weights using formula (1) by measuring these terms' occurrences in this element and the distributions in the immediate-descendant elements of E_j.
3. For term t_i, if $Weight(t_i, E_j) \geq average(E_j) + std\_dev(E_j)$, then term t_i is selected as an index term of the element E_j and all the descendent elements of E_j would eliminate t_i from their index term lists. This process is called the index term propagation and pruning. Here $average(E_j)$ denotes the arithmetic average of all terms' weights in element E_j and $std\_dev(E_j)$ the standard deviation of these weights.
4. Recursively perform step 2 onwards until the root node (i.e., the document) is reached.

This indexing solution makes full use of the internal structural information of documents. Since all terms are compared to each other at the same level and a theoretic maximum entropy value is used as the normalization factor, the negative effect of varying lengths of text in elements at different levels is minimized. Our experimental results are able to testify this. In addition, an index term of an element need not necessarily appear in all sub-elements of this element due to the nature of the measurement of the term weight. Thus more representative index terms other than just a few words in titles can be found.

4. Flexible Element Retrieval and Result Browsing

In this section, we describe the flexible element retrieval algorithm which is used to select suitable document elements. With the help of hierarchical index, the main task of the retrieval phase is online searching and ranking of candidate elements.

4.1 Path Ranking and Retrieval Process

For each document, we use a path ranking algorithm to calculate relevance values of all candidate elements against a query. A path for an element is defined as the branch containing all the ancestor elements of this element (including itself) in the document tree. According to our hierarchical indexing mechanism, an element does not share

any index terms with its ancestors. Thus we say that an element is completely represented by all index terms of the elements along its path. Conversely, a path can be expressed as the element at the lowest level in the path. Therefore, the element ranking problem can be transformed to a path ranking problem, that is, to find those element paths with high relevance values to the query.

The relevance value for a path against a given query is defined as:

$$\mathrm{Re}levance(Path_p) = \sum_{i=1}^{Q} Weight(t_i, Path_p) \times \ln\frac{N}{n_i} \qquad (4)$$

$\ln\frac{N}{n_i}$ is the IDF value of query term t_i and is used here as the query term's weight. Q stands for the number of query terms in a query. $Weight(t_i, Path_p)$ is defined as the weight of the query term t_i for path $Path_p$. We define that a term's weight for a path is its weight for the element that containing this term along the path, as is defined by Equations (1) and (3).

Given a new query, we use traditional document retrieval methods to get a list of relevant documents first in order to narrow down the search space. Then when the user selects one of the relevant documents, the system searches for all candidate elements of this document and ranks their paths according to Equation (4). The most relevant elements are sorted and displayed with the structural context to the user. The overall process is described as below:

Algorithm 2 – Path ranking

1. Find all elements that contain at least one query term.
2. Get paths for all candidate elements and merge the paths, that is, merge two paths into one if one is a part of the other.
3. Assign the weights of the query terms for elements to their paths respectively.
4. Rank these paths according to Equation (4).
5. Return the elements corresponding to the ranked paths with the ranks satisfying the pre-defined threshold in a descending order.

A long-standing problem in structured document retrieval is how to select proper elements which best satisfy the user's query needs. Usual method to solve this problem is to set a fixed threshold and the elements with ranks above this threshold are returned as the results [15]. However due to the variation in text length, the proper threshold varies with documents and queries. We use the average of all retrieved elements' ranks as the dynamic threshold. The experiments show that a more accurate element retrieval can be attained based on this dynamic threshold.

4.2 Result Browsing

Flexible information retrieval may return larger or smaller granularity results than what the user needs. Therefore a good user interface for browsing the results in the original tree structure context is crucial for improving users' query process. Figure 3 shows a snapshot of the interface of our flexible element retrieval system with a given query "Qing dynasty".

In Figure 3, we can see that total of sixteen elements are returned for the document named "China", among which there are sections and paragraphs. The top element is a section with the title "The Manchu Qing Dynasty" that is dedicated for describing the Qing Dynasty in the history of China. This section is under the 7th section of this document, whose title is "History". From the left browsing pane, we can see clearly each section or paragraph's position in the document. In comparison, when we click the first article "Qing Dynasty", we get the whole document since the entire document is rooted on this topic. In summary, the flexible retrieval system returns the most appropriate document elements to users according to their queries.

5. Evaluation

In this section, we evaluate the performance of our proposed flexible element retrieval method and investigate the effects of threshold settings on element retrieval. The experiments are conducted on the Encarta corpus, which contains 41,942 well structured XML documents. The query set is made up of 10 queries, which can be best answered by only a part of the relevant documents. The 10 queries used in this experiment are listed in Figure 4.

For comparison purpose, we implemented a passage retrieval system, TFIDF Para. This system uses only pre-defined paragraphs in Encarta documents as passages while ignoring other structural information. A term's weight in a paragraph is defined by the conventional TFIDF measure [15], which is the same as Equation (3). The relevance measure between a given query and a specific paragraph is the cosine similarity between their term vectors.

1. History of China
2. Qing Dynasty
3. Atomic bomb in American history
4. Ford Motors in World War II
5. What is the impact of Newton on calculus?
6. What is the attitude of Microsoft to World Wide Web?
7. What is the influence of Lincoln in American history?
8. Fleet Street in London
9. Military aircrafts used in Desert Storm
10 .What missiles can nuclear submarines carry?

Figure 4. Queries for element retrieval evaluation

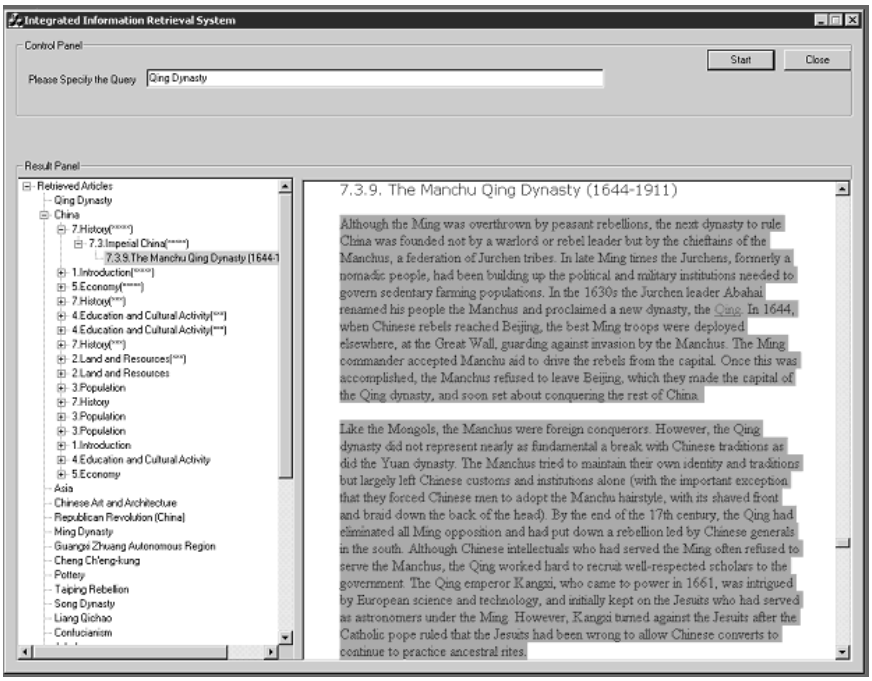

Figure 3. The interface of the Flexible Element Retrieval System

5.1 Performance Evaluation of Element Retrieval

Previous work on passage retrieval or structured document retrieval focuses their evaluations mainly on the impact of passage level evidence on retrieving the whole documents [3] [9] [15]. Some of them gave out several examples of extracted components in a selected document given a specific query [15]. But none of them conducted special experiments dedicated to the evaluation of the effectiveness of element retrieval. [14] intends to implement such experiments but they lack appropriate test collection. We conducted a series of experiments in order to testify if our flexible element retrieval method can find elements with proper granularity against the users' queries.

Relevance judgments are made by human assessors. For each query, the assessors first select a document that is considered as most relevant. Then the relevant elements in that document against this query are judged and selected by the assessors without the knowledge of the targeting systems. Besides precision and recall, we also employ F-Value to be an integrated measure for performance evaluation.

$$F-Value=\frac{2}{1/recall+1/precision} \quad (5)$$

When deciding what fractions of the retrieved elements should be returned to the users as the answers, we use both fixed thresholds from 0.1 to 0.9 at the increment of 0.1 plus 0.95 and two dynamic thresholds. One such dynamic threshold is the average of the rank values of all retrieved elements for a query (Avg), and the other is Avg

plus the standard deviation of these values (Std_Dev). The results obtained by these two methods with various thresholds are illustrated in Tables 1, 2 and 3 for precision, recall and F-value respectively. For the flexible element retrieval method, we test its performance on two different sets of index. Each composite element, say a document or a section, has a title, which is a good indicator for its content. In order to get more convincing results, we build the first set of index without using the titles. Experiments indicate that most of the title terms can be re-constructed by our indexing mechanism. In the second set of index, we add the title for a document or a section to every paragraph below it as index terms. For TFIDF Para system, the index utilizes the titles as is done in the second set of index.

Table 1. Comparison of precision

Threshold	TFIDF Para	Flexible Retrieval (with titles)	Flexible Retrieval (without titles)	improvement (with titles) over TFIDF	improvement (without titles) over TFIDF
0.1	0.3549	0.5263	0.5059	48.30%	42.55%
0.2	0.3948	0.5318	0.5107	34.70%	29.36%
0.3	0.4374	0.5361	0.5338	22.57%	22.04%
0.4	0.5096	0.5361	0.5478	5.20%	7.50%
0.5	0.5158	0.5854	0.5800	13.49%	12.45%
0.6	0.5801	0.5902	0.6159	1.74%	6.17%
0.7	0.6482	0.6864	0.6478	5.89%	-0.06%
0.8	0.6487	0.7521	0.7521	15.94%	15.94%
0.9	0.6333	0.8212	0.7855	29.67%	24.03%
0.95	0.6167	0.7917	0.7839	28.38%	27.11%
Avg	0.4045	0.7665	0.6115	89.49%	51.17%
Avg+Sdev	0.5457	0.7790	0.6667	42.75%	22.17%

Table 2. Comparison of recall

Threshold	TFIDF Para	Flexible Retrieval (with titles)	Flexible Retrieval (without titles)	improvement (with titles) over TFIDF	improvement (without titles) over TFIDF
0.1	0.9350	1.0000	0.8667	6.95%	-7.30%
0.2	0.9350	1.0000	0.8667	6.95%	-7.30%
0.3	0.9350	1.0000	0.8667	6.95%	-7.30%
0.4	0.9021	1.0000	0.8667	10.85%	-3.92%
0.5	0.7309	1.0000	0.8500	36.82%	16.29%
0.6	0.5964	0.9500	0.8417	59.29%	41.13%
0.7	0.5571	0.9333	0.7333	67.53%	31.63%
0.8	0.3999	0.8121	0.6583	103.08%	64.62%
0.9	0.2407	0.7793	0.5833	223.76%	142.33%
0.95	0.2401	0.6377	0.5527	165.60%	130.20%
Avg	0.7456	0.9417	0.6800	26.30%	-8.80%
Avg+Sdev	0.5670	0.5839	0.5756	2.98%	1.52%

Table 3. Comparison of F-Values

Threshold	TFIDF Para	Flexible Retrieval (with titles)	Flexible Retrieval (without titles)	improvement (with titles) over TFIDF	improvement (without titles) over TFIDF
0.1	0.5145	0.6896	0.6389	34.04%	24.17%
0.2	0.5552	0.6943	0.6427	25.07%	15.76%
0.3	0.5960	0.6980	0.6607	17.12%	10.85%
0.4	0.6513	0.6980	0.6713	7.17%	3.07%
0.5	0.6048	0.7385	0.6895	22.11%	14.01%
0.6	0.5881	0.7281	0.7113	23.79%	20.94%
0.7	0.5992	0.7910	0.6879	32.01%	14.80%
0.8	0.4948	0.7809	0.7021	57.84%	41.90%
0.9	0.3488	0.7997	0.6695	129.26%	91.92%
0.95	0.3456	0.7064	0.6483	104.38%	87.57%
Avg	0.5245	0.8451	0.6439	61.14%	22.78%
Avg+Sdev	0.5561	0.6675	0.6178	20.02%	11.09%

From the above tables, we can see clearly that with the various threshold settings our flexible element retrieval method has a significant improvement in retrieval performance, especially measured by precision and F-Value, over the method of applying TFIDF measure to paragraph level directly. With respect to F-Value, the average improvement is 56.02% involving titles, and 40.89% without considering titles. In both cases of adding title terms into index terms and not dealing with title terms, the precision of the flexible element retrieval system is much better than the TFIDF Para system with the average improvement of 48.83% and 41.67% respectively. We attribute the drastic augment in precision to the high quality index terms selected by our index propagation and pruning algorithm. In addition, the flexible element retrieval method can return elements with various granularities which may be paragraphs, sections or even the whole documents depending on the specification of queries. In contrast, previous passage retrieval methods return only fixed-level passages. However, there is slight decrease in recall for some threshold settings when using the index set without adding the title terms. This is caused by our index term selection threshold, which is somehow too tight such that some proper terms are missed because their distributions in text do not meet the selection threshold. But we deem that the decreased recall can be compensated by our interface which allows users to browse in the document structure freely.

Previous leaf nodes indexing methods make an element available against a query only if the element contains a part of the query, i.e., a relevant composite element can be retrieved with all of its descendant elements if and only if each of the descendants contains at least one query term. This is not the case in many documents so a lot of relative paragraphs containing no query terms are missed in the TFIDF Para system's results. On the other hand, TFIDF Para system introduces much noise into the final result by adding some paragraphs which do not cover the meaning of the user query but do contain some query terms. In comparison, with the index propagation and pruning mechanism, the index with the tree structure in our system can make sure of a

relatively better concept matching. To a composite element, say a section, the appropriate index terms would be propagated to it even if only a part of its descendant elements contain these terms. This index structure ensures the integrity of the resulting elements.

5.2 Threshold Setting

Threshold setting is very crucial for structured document retrieval to get a set of desirable resulting elements. In previous works, the thresholds are usually fixed [15]. In our experiments, we find that using a single threshold cannot make the system always perform well for different queries since the documents vary greatly in structure and length. We explore the use of dynamic thresholds instead of fixed threshold in our experiments.

In order to see how various thresholds affect the retrieval performance, we plot the F-Values obtained with various thresholds in Figure 5. The Figure shows that the curve generated by our method, especially the curve representing the results obtained without using title terms, is much flatter than that obtained by the TFIDF Para method. The performance of the TFIDF Para method varies greatly with the changing of threshold. The highest F-Value obtained by TFIDF Para is 0.6513(at the threshold of 0.4), which is 177.96% greater than the lowest value of 0.2343 (at no threshold). In comparison, with the use of title terms, the maximum (at threshold Avg) and the minimum (at threshold Avg+SDev) F-Values of our method vary only 26.61%; and in the case of without considering title terms, the variation is only 15.13%. This indicates the fact that our method is less sensitive to threshold setting. We attribute the robustness to that our method takes full advantage of the document structure to mix the statistics of term occurrences and distributions in weighting terms.

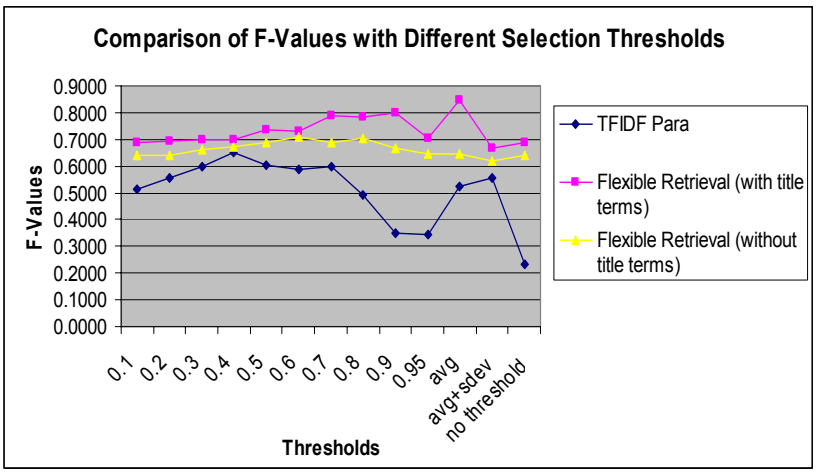

Figure 5. Comparison of F-Values with different thresholds

Moreover, from Figure 5, it is interesting to note that the dynamic thresholds, such as Avg and Avg+SDev, can produce desirable results. When using the index set with title terms, the F-Value of the flexible retrieval system achieves the best performance when using the threshold avg. Our method using the index set without title terms can also get very good result with the threshold Avg, which is slightly less (9.47%) than the best one. But due to the sensitivity to the threshold setting, TFIDF Para system cannot be improved when using dynamic threshold. This testifies that dynamic threshold is a good alternative for threshold setting for our system since in most cases we cannot use one threshold to ensure the best performance for all documents and queries.

6. Conclusion

Passage retrieval based on structural information in documents has long been suggested as effective ways to retrieve elements of a document with finer granularity. In this paper, we proposed a new hierarchical index propagation and pruning mechanism for structured documents and realize a flexible element retrieval system based on this index structure. An index term is propagated to an upper level element in the tree structure if it represents a more general concept, which is judged by comparing its statistical information with other peer terms' weights in that element. Index terms are distributed across the whole document tree and each element has a list of index terms which can best represent the concept of that element. The flexible element retrieval method is dedicated to providing users with the most appropriate elements at any level. We conducted experiments to evaluate our method in terms of precision and recall in element level. Experimental results showed that our method significantly outperformed the method of applying TFIDF measure to only the paragraph level. It was also found that our method was not sensitive to threshold setting compared to other passage retrieval methods. Moreover, we observed that dynamic threshold is a better solution for the threshold setting for element retrieval.

Acknowledgement

The authors would like to express their sincere thanks to Dr. Wei-Ying Ma for his valuable comments and suggestions to improve this paper.

References

[1] Abiteboul S., Quass D., McHugh J., Widom J. and Wiener J., 1996, The Lorel Query Language for Semi-structured Data, Department of Computer Science. Stanford University, California, USA, 1996.

[2] Baeza-Yates, R., Navarro, G., 1996, Integrating contents and structure in text retrieval, ACM SIGMOD Record, 25(1):67-79, March 1996.
[3] Callan, J., 1994, Passage-level evidence in document retrieval. In Proceedings of the 17 Annual ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, 1994, Pages 302-310.
[4] Frisse, M, 1988, Searching for Information in a hypertext medical handbook, Comm. of ACM, 31(7), July 1988, Pages 263-271.
[5] Fuhr, N., Grobjohann, K., 2001, XIRQL: a query language for information retrieval in XML documents, In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA, September 2001, Pages 172-180.
[6] Geffet, M., Feitelson, D., 2001, Hierarchical indexing and document matching in BoW, In Proceedings of JCDL'01, Roanoke, Virginia, USA, 2001, pages 259-267.
[7] Goldman, R., Shivakumar, N., Venkatasubramanian, S. and Garcia-Molina, H., Proximity search in databases, In Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases, New York, USA, August 1998, Pages 26-37.
[8] Kaszkiel, M., Zobel J. and Sacks-Davis R., 1999, Efficient passage ranking for document databases, ACM Transactions on Information Systems, Vol. 17, No. 4, October 1999, Pages 406-439.
[9] Kaszkiel, M., Zobel, J., 1997, Passage retrieval revisited, In Proceedings of the 20th Annual ACM SIGIR International Conference on Research and Development in Information Retrieval, 1997, Philadelphia, PA, USA, Pages 178-185.
[10] Kazai, G., Lalmas, M., and Rölleke, T., 2001, Aggregated Representation for the Focussed Retrieval of Structured Documents, SIGIR 2001 Workshop, Mathematical/Formal Methods in IR, New Orleans, 2001.
[11] Lee, Y., Yoo, S. Yoon, K. and Berra, P., 1996, Index structures for structured documents, In Proc. of the First ACM International Conf. on Digital Libraries, pp. 91-99, 1996, Bethesda, Maryland.
[12] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., and Widom, J., 1997, Lore: a database management System for semistructured data, SIGMOD Record, 26(3), September 1997, Pages 54-66.
[13] Mittendorf, E., and Schauble, P., 1994, Document and Passage Retrieval Based on Hidden Markov Models, In Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, July, 1994, Pages 318-327.
[14] Myaeng, S., Jang, D., Kim, M. and Zhoo Z., 1998, A flexible model for retrieval of SGML documents, In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, Australia, 1998, Pages 138-145.
[15] Salton, G., Allan, J. and Singhall, A., 1996, Automatic Text Decomposition and Structuring, Information Processing and Management. 32(2), Pages 127-138.
[16] Wilkinson, R., 1994, Effective retrieval of structured document, In Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, 1994, Pages 311-317.