Optimal Structure Weighted Retrieval

Andrew Trotman
Department of Computer Science
University of Otago
Dunedin, New Zealand
andrew@cs.otago.ac.nz

Abstract *Improving ranking functions for structured information retrieval has received much attention since the inception of XML. Weighting document structures is one method providing significant improvement – but how good can these improvements be?*

Optimal structure weighted retrieval occurs when each query is processed using the optimal set of weights for that query. Optimal retrieval for a set of queries occurs when a set of weights optimized for that set of queries is used. Measuring mean average precision for each of these will give a performance upper bound for document structure weighted retrieval.

In this investigation a near optimal set of weights is learned for TREC WSJ collection topics 101-200 using a genetic algorithm. Weights are learned for vector space inner product, naïve probability and BM25 ranking functions and a performance upper bound is calculated.

The upper bound using a different set of weights for each query, gives mean average precision improvements of about 15% for BM25 and naïve probability; about 30% for inner product. This suggests structure weighting might be useful for relevance feedback. Optimal weights for the set of queries shows improvements of about 5% for naïve probability and inner product, but of only about 1% for BM25; suggesting this technique is not as effective for ad hoc retrieval.

Keywords Information Retrieval.

1. Introduction

As markup languages such as SGML [13] and XML [3] became more popular, many data providers switched to offering their data in these formats. The two big information retrieval collections, TREC [11] and INEX [5], are both available in XML.

This availability of structured documents raises questions about relevance ranking – how, exactly, can the document structure be used to improve whole document ranking? One approach is to weight term occurrences based on where in the document they lie. A term found in an abstract is perhaps of greater significance than the same term found in the body text of the same document, fulfilling the principle of summarization. This was the suggestion of Fuller *et al.* [7].

In structure-weighted retrieval each document structure (or tagged element) is given a weight. Each occurrence of each search term is weighted according to the structure in which it occurs. Document term frequencies are then replaced by linear weighted term frequencies based on term structure occurrences. In this way whole documents are ranked using the structural information present in the documents.

There are several ways to choose the weights. Proposals include trial and error [28], simulated annealing [2], genetic algorithms [26], and asking the user to supply them as part of the query [6].

Regardless of how the weights are chosen, there remains a question central to all ranking functions: performance. Ordinarily this is easily measured. Any standard test collection is obtained, the mean average precision (MAP) is computed over all the queries in the collection and compared to that of other ranking functions. The highest performing function is considered "best" for that collection. A test such as the *t*-test is often used to show the significance of this difference.

With structure weighting, exactly the same approach is used, only the ranking function doesn't change, only the structure weights change. In effect, the process is to optimize an existing function to a given set of structures. But how good is this optimization in isolation? What is the expected performance gain using just this approach? How does it compare to using a different ranking function?

Imagine there is an oracle that knows, in advance and for every query, the optimal set of weights to use. In optimal structure weighted retrieval, a query is received from a user, given to the oracle that returns the weights, those weights are loaded into the retrieval engine and the query processed. In essence, the oracle ensures every query is answered optimally.

In this investigation the performance gain of structure weighted retrieval is measured in exactly this way. The TREC Wall Street Journal collection and TREC topics 101-200 are used. The oracle is

Proceedings of the 9th Australasian Document Computing Symposium, Melbourne, Australia, December 13, 2004.

simulated by a genetic algorithm [12], which learns, for each topic, a near optimal weight set. Experiments were conducted using vector space inner product [20], naïve probability [10] and the BM25 [19] ranking functions. An approximation to the performance upper bound is found.

Results for unweighted retrieval show BM25 outperforms naïve probability, which outperforms inner product. Using the oracle, the mean gain is approximately 15% for BM25 and naïve probability, and 30% for inner product (which does not change this order). When the oracle was tasked to learn a single set of weights to apply to all queries, the improvements were much smaller. This suggests document structure weighting might be better suited to relevance feedback than to *ad hoc* retrieval.

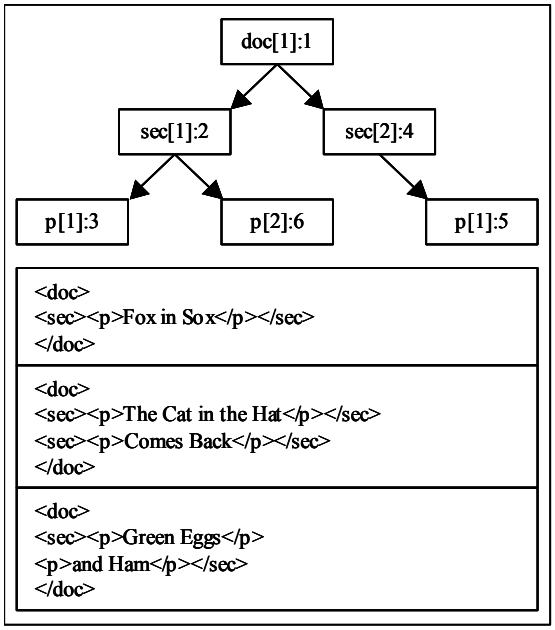

Figure 1: Three documents and the corpus tree for those documents. Instances are shown in square braces, e.g., p[2] represents the second paragraph. Identifiers, issued on an as encountered basis, are shown after the colon.

2. Structured information retrieval

In an inverted file information retrieval system there is one dictionary, and each dictionary term points to one inverted list of postings. The postings are usually represented as $\{<d_1, f_1><d_2, f_2>, \ldots, <d_n, f_n>\}$ where d_n is a document number and f_n is the frequency of the given term in the given document.

For structured information retrieval it is also necessary to know where in the document terms are found, so the postings must be annotated with this. Exactly how the postings are annotated does not matter, so long as it is possible to know, for each posting, in which structure that posting is found.

Several encoding techniques have been suggested. Each different tag in the DTD could have a separate inverted list [21; 22]. The path to the posting could be stored as a path directly in the indexes [4; 25]. Or a tree representing the structure of the collection could be built, labeled, and used.

To build the tree, first each document tree is built; then these are superimposed to form a single tree. This corpus tree includes every path through every document, but is unlikely to match the structure of any one document. Three documents and the corpus tree they form are shown in Figure 1.

Nodes in the tree can be encoded in several ways. If the tree is considered to be a *k*-way virtual tree (where some nodes may not exist) and each node is labeled with an identifier reflecting this, paths upwards through the tree can be computed directly from the identifiers. These identifiers can then be stored directly in the postings [16; 24].

Alternatively, each node in a tree of *M* nodes can be given an ordinal identifier from 1 to *M*. Then for each posting a bitstring of length *M* is constructed with a 1 bit for every node above the given postings and a 0 for all others. These bitstrings are then stored with each posting [17].

The corpus tree can be built dynamically during single pass indexing [15; 26]. As each new path in each new document is encountered it is added to the corpus tree and given a unique ordinal identifier. These identifiers are then stored with each posting, $\{<d_1, p_1, f_1><d_2, p_2, f_2,>, \ldots, <d_n, p_n, f_n>\}$ where p_n is the corpus tree node identifier.

With each of these approaches it's possible to determine which terms occur where in which document and how many times. For the experiments herein it doesn't matter how the indexes are stored so long as that information is available.

3. Ranking

3.1. HTML

Tag weighting schemes for HTML have received much attention. The document weight is computed from not only the term occurrences, but from the tags in which the term occurs.

One approach [14] is to give each tag a weight. Then for ranking, compute two values: term frequency within the document, tf_{id}, and the product of the tag weights for each tag in which the term is found. These are multiplied to give a weighting for the term in the document. This approach averages the tag weights over the parts of the document covered by the term. A document containing a term twice in <TITLE> and once in <B> is weighted identically to a document containing the same term once in <TITLE> and twice in <B>.

To overcome this averaging, each occurrence of each term can be multiplied by an element weight [2; 18]. Term frequency tf_{id} for a given term, *i*, in a given document, *d*, is replaced by a structure weighted term frequency, ctf_{id}, computed as

$$ctf_{id} = \sum_{p=1}^{n} \left(C_p \times tf_{ipd} \right) \quad (1)$$

where p is a given element, C_p is a weight for that element, and tf_{ipd} is the number of occurrences of term i in structure p of document d.

Different approaches vary in which elements (and other heuristics) are used for ranking. Some, for example, include the number of incoming web page links.

3.2. SGML and XML

There are two approaches to searching SGML and XML, retrieval of elements and retrieval of whole documents. The annual INEX workshop focuses on element retrieval for which there are a wide number of approaches [5]. This investigation focuses on whole document retrieval (as seen in digital academic libraries), for which element retrieval techniques are not appropriate.

For whole document retrieval, Kotsakis [15] used the weighted term frequency approach from equation (1) with vector space ranking and Trotman [26] did the same for naïve probabilistic and BM25 ranking.

3.3. Structured ranking equations

3.3.1. Vector space inner product

The vector space inner product weight, w_{dq}, is computed as the inner product of the document vector, w_{id}, and the query vector, w_{iq}. The structure-weighted variant is

$$w_{dq} = \sum_{i=1}^{t} \left(w_{id} \times w_{iq} \right) \tag{2}$$

where

$$w_{iq} = tf_{iq} \times IIDF_i \tag{3}$$

and tf_{iq} is the number of occurrences of term i in query q and

$$w_{id} = ctf_{id} \times IIDF_i \tag{4}$$

and

$$IIDF_i = \log_2 \frac{N+1}{n_i} \tag{5}$$

where N is the number of documents in the collection and n_i is the number of occurrences of term i in the collection.

3.3.2. Naïve probability model

The weight of the document with respect to a query, w_{dq}, is given by

$$w_{dq} = \sum_{i=1}^{t} \left(\left(C + PIDF_i \right) \times \left(L + \left(1 - L \right) \times \frac{ctf_{id}}{m_d} \right) \right) \tag{6}$$

where

$$PIDF = \log_2 \frac{N - n_i + 1}{n_i} \tag{7}$$

where $C = 1.0$, $L = 0.3$ and m_d is the term frequency of the most frequent term in document d.

3.3.3. Okapi BM25

For Okapi BM25 ranking,

$$w_{dq} = \left(\sum_{i=1}^{t} BIDF \times \frac{(k_1 + 1) \times ctf_{id}}{K + ctf_{id}} \times \frac{(k_3 + 1) \times tf_{iq}}{k_3 + tf_{iq}} \right) \tag{8}$$

where

$$BIDF = \log \frac{N - n_i + 0.5}{n_i + 0.5} \tag{9}$$

and

$$K = k_1 \times \left(\left(1 - b \right) + \frac{b \times T_d}{T_{av}} \right) \tag{10}$$

where $k_1 = 1.2$, $k_3 = 7$, $b = 0.75$, T_d is the number of terms in the document, and T_{av} is the average document length measured in terms (the same unit of measure as T_d).

If all weights are set equal to 1, no structure is preferred over any other and unstructured retrieval results. If a structure weight is set to 0, that structure will not be included in ranking. Other choices of weights set the relative importance of each structure to the others. There remains, however, the problem of choosing the weights.

4. Choosing document structure weights

One way to choose weights is to ask the user to supply them as part of the query. Graphic query languages supporting this have already been proposed [1], as have text based query languages [6]. Although these languages allow the user to customize the weights for the each query, it is not obvious how to choose good weights. Tests on HTML documents demonstrate a decrease in precision for almost all points of recall when a human subject is asked to do this [18].

Manual approaches to choosing collection-wide weights have also been tried. By setting all weights equal and varying one weight it is possible to measure the influence of each element in isolation. A set of weights for all the elements is then chosen based on how each element performs in isolation [28]. This technique, however, is unable to identify co-dependence between element.

To get an optimal set of weights a fully automated machine learning approach is needed. Gradient-based

optimization has successfully been used [18]. Recently genetic algorithms have also been show to produce good weights for HTML [14] and for XML [26]. These investigations have shown that a set of weights learned on one set of topics is effective for improving another.

4.1. The genetic algorithms approach

Genetic algorithms [12] have been shown to be a robust optimization mechanism in many fields [8]. In information retrieval they have been used to build indexes [9], for relevance feedback [27], and for choosing document structure weights.

Each node in the corpus tree is given a unique ordinal node identifier as it is created. Consequently, once indexing has completed, the nodes will be labeled 1 to *M*. Weights for each node can, therefore, be represented as an array of length *M*, one array position for each weight.

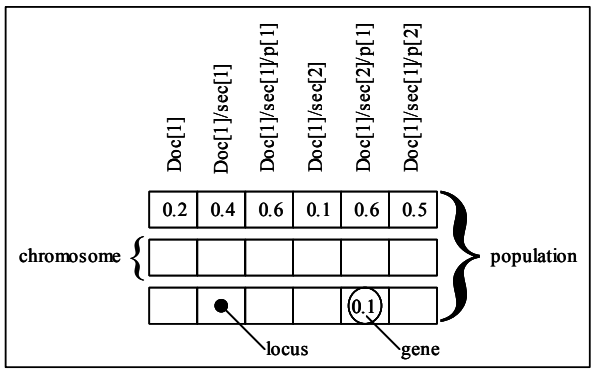

Figure 2: The chromosomal encoding of a set of weights forms a genetic algorithms individual. A set of individuals is a population.

The array is analogous to an individual chromosome in a genetic algorithm simulation. The array positions are analogous to loci and the values in the array to genes. A population is a set of arrays. This analogy is shown in Figure 2.

The genetic operations are a direct manipulation of these arrays.

For reproduction an individual is selected with fitness proportionate selection.

In mutation, an individual is chosen using fitness proportionate selection. From that individual a random locus is chosen. A random gene is then selected and placed at that locus.

In one-point crossover, two individuals are chosen using fitness proportionate selection. A random locus is chosen. Two new individuals are then created with the genes after the locus switched.

The simulation is run until a satisfactory result is found or for a fixed number of generations.

5. Experimental design

Two experiments were conducted; both used the Wall Street Journal collection (1987-1992) from TREC disks 1 and 2. Topics 101 to 200 were used, with topics having fewer than 5 judgements being discarded (121, 175, 178 and 181). Topics were converted into queries by extracting terms from the description field and stopping common words. Stemming was not used.

The collection was indexed with a structured information retrieval system using a corpus tree and with postings annotated with pointers into the tree.

A population of 50 individuals was chosen at random. The probability of mutation was 0.2, of crossover was 0.5, and of reproduction was 0.3 (other ratios were not tested). Genes took a value between 0 and 1. The initial population was seeded with an individual with weights all 1; the equivalent of unweighted retrieval. Each simulation ran for 25 generations. The experiments were conducted 50 times to reduce the chance of error.

The purpose of these experiments is to compute an upper bound for structure weighted retrieval (in isolation of other factors). This is done by deliberately over fitting to a training set and then reporting the result as the optimum – it is the best that can be seen on the training set (in this case the entire set), and so can reasonably be reported as optimal. Although it is possible the optimal set of weights will not be found in one run of the genetic algorithm (it might become stuck at a local maximum), repeating the experiment 50 times decreases this inherent error. As no proof of optimality of the weighs is given, the results herein must be considered near-optimal and not optimal.

5.1. Experiment 1: The oracle

Each time a query is issued, the oracle is consulted for the optimal set of structure weights. These weights are then loaded into the retrieval engine and the query resolved. Such an oracle cannot exist, but the effect of having one can be simulated.

In experiment 1, weights are learned for each topic individually. From the 50 runs, weights from the one run showing the largest improvement in average precision are chosen as the best weights. These weights can reasonably be expected to be near optimal.

Using the near optimal weights with each query is an approximation of the oracle. For each query, the near optimal weights are loaded into the retrieval engine and the query resolved against the document collection. The average precision for this query is near the optimal average precision possible for this query, using this technique.

This experiment was conducted three times, once for each of inner product, naïve probability and BM25 (equations (2), (6) and (8) respectively).

5.2. Experiment 2: One answer fits all

If the oracle could be consulted only once, it would be to find the set of weights that would maximize MAP

over all queries. This is simulated in experiment 2 where one set of weights is learned using all queries.

This experiment was conducted three times, once for each of inner product, naïve probability and BM25 (equations (2), (6) and (8) respectively).

6. Results

The results of experiment 1 are compared to those of experiment 2 and to unweighted retrieval to get an approximation to how good document structured retrieval can be.

6.1. Experiment 1

Table 1 presents the results from Experiment 1. Using the oracle results in mean average prevision improvements varying between less than one percent and several hundred percent. The mean improvement is calculated as the sum of improvements divided by the number of queries. This is not the improvement in the mean average precision, which is presented in Table 2.

No queries showed a decrease in precision. In Figure 3 the improvements are plotted from greatest to least improvement and for each ranking function. Only a small number of queries receive large improvements, most receive small improvements. Average precision improvements of greater than 5% were seen in 78% of queries using inner product, 57% using naïve probability and 67% using BM25.

Without using structure weighting, BM25 outperforms naïve probability, which outperforms inner product. When the oracle is used with each of the three functions, they still perform in the same order; BM25 is better than naïve probability is better than inner product.

Unexpectedly, different queries showed different improvements using different ranking functions. In Figure 4 the topics are shown sorted in order of most to least improvement for inner product. The lines representing BM25 and naïve probability show improvement spikes. From visual inspection, the queries showing large improvements appear to be different for each ranking function.

The differences in Figure 4 can be quantized using the Pearson correlation. Pearson values near 1 show a positive correlation; those near -1 show a negative correlation, while values close to 0 show no correlation. From Table 3; there is no cross function correlation in how much each topic is improved.

Method	Max	Mean	Min
I. Prod	1081.88%	71.83%	0.77%
N. Prob	356.75%	28.39%	0.04%
BM25	848.18%	35.04%	0.04%

Table 1: Average precision improvements seen in experiment 1.

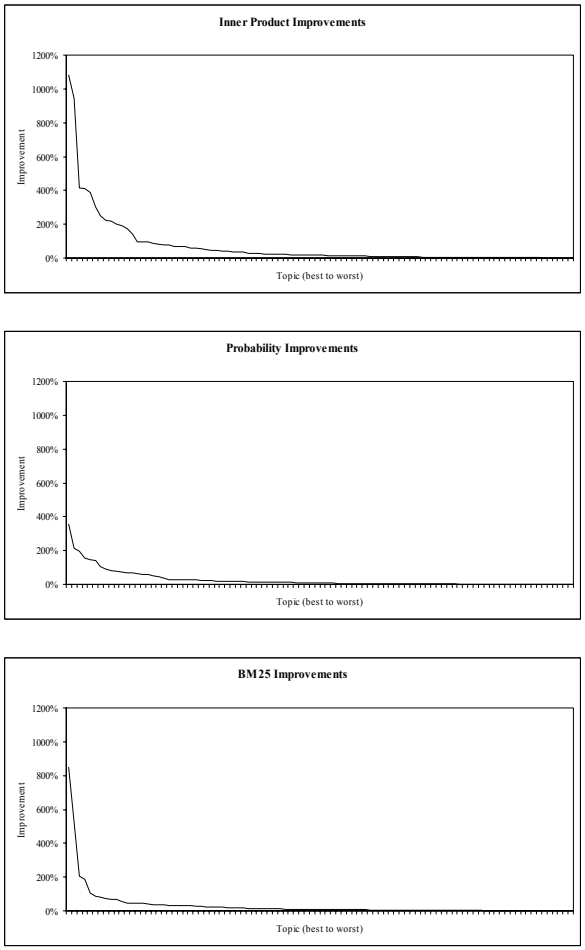

Figure 3: Improvements in precision vary greatly, but are mostly small.

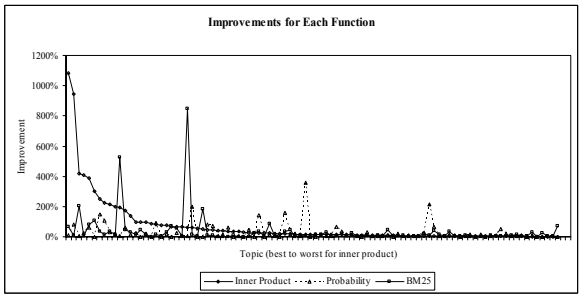

Figure 4: Different topics show different improvements with different ranking functions.

Method	I. Prod	N. Prob	BM25
Unweighted	0.15674	0.17804	0.24178
Experiment 1			
The Oracle	0.20329	0.20560	0.27750
Improvement	29.70%	15.48%	14.77%
P (*t*-test)	0.00	0.00	0.00
Experiment 2			
One Answer Fits All	0.16399	0.18991	0.24332
Improvement	4.63%	6.67%	0.64%
P (*t*-test)	0.00	0.00	0.04

Table 2: Mean average precision of experiments 1 & 2. Significance (P) is computed with a 2-tailed *t*-test.

Exp 1	I. Prod	N. Prob	BM25
I. Prod	1.00	0.09	0.14
N. Prob	0.09	1.00	-0.07
BM25	0.14	-0.07	1.00

Table 3: Pearson correlation showing improvements seen in one function do not correlate to those seen in another when weights are learned for each topic.

Using this result, its possible to ask a new question. What if there existed a super-oracle who knew the best function to use and the best set of weights to use with that function? In this case the MAP increases to 0.29046, this is a 20% improvement on unweighted BM25, a just under 4.7% improvement on the oracle result for BM25.

6.2. Experiment 2

Results for experiment 2 are shown in Table 2. When the oracle returns a single set of weights suitable for all queries, the improvement is substantially less than when weights are returned for each topic. This is exactly as expected, one general purpose set of weights can not be expected to perform as well as a set of special purpose weights tailored to each query.

Improvements for inner product and naïve probability are about 5%. Those for BM25 show a less than 1% improvement. Even so, BM25 outperforms naïve probability, which outperforms inner product. The same order as when no weighting was used.

Exp 2	I. Prod	N. Prob	BM25
I. Prod	1.00	-0.08	-0.13
N. Prob	-0.08	1.00	0.05
BM25	-0.13	0.05	1.00

Table 4: Pearson correlation showing improvements seen in one function do not correlate to those seen in another when one set of weights is used for all topics.

Exp 1 / Exp 2	Pearson
I. Prod	0.58
N. Prob	0.87
BM25	0.01

Table 5: Pearson correlation computed between experiment 1 and experiment 2. Those topics showing improvements are correlated across experiments for inner product and naïve probability, but not for BM25.

The Pearson correlation is shown in Table 4 – again there is no cross function correlation between how much each topic is improved. When the results from experiment 1 and experiment 2 are correlated, a strong correlation is seen for naïve probability, less so for inner product, and none for BM25, as shown in Table 5. BM25 most likely shows no correlation because the improvements in experiment 2 are themselves not significant.

BM25 showed close to no improvement across these queries. This may be because BM25 was developed for this collection. The tuning parameters, k_1, k_3 and b, are used to customize the function to a given set of documents, and are already optimized for TREC.

7. Discussion and future work

The documents in the TREC WSJ collection are marked up using about 20 unique tags. The documents are short, and the markup is sparse. The queries used in these experiments are derived from topic descriptions, they, too, are short. These two factors may account for why the near optimal improvements are small.

Should all the query terms occur in only one element (in the case of WSJ, the TEXT element), each document receives a constant weight resulting in a linear scaling of the ranking score; which is order preserving. Should only a few relevant documents additionally contain terms elsewhere (e.g. the title or HL element), only the order of those documents will be adjusted relative to the others; only a small change in MAP will be observed.

This could be verified by using longer queries. By increasing the number of terms in the query the chance of hitting terms outside the TEXT element will increase so the number of documents changing order will increase and the MAP will reflect this change.

Alternatively, a collection of documents with longer elements could be used. The INEX collection [5] contains long documents broken into title abstract, sections and subsections and may be appropriate. It does, however, contain 192 unique tags, many of which are unlikely to be useful for ranking (e.g. citation identifiers). Such tags would be removed, the content being preserved, before such experiments are conducted.

The TREC binary relevance judgments may also be the source of the small MAP improvement. If the ranking function has successfully identified the relevant documents, and ordered those at the top of the results list, any amount of re-ordering to place "more" relevant documents first will not be observed. This could be verified using a document collection that has non-binary relevance judgments (such as the cystic fibrosis collection [23])

The difference seen between experiment 2 and experiment 1 is substantial. When one set of weights is learned for a set of topics the improvement is small, when weights are learned for each query the improvement is larger. This suggests the latter technique might be useful, if it could be utilised in a retrieval engine. One way to improve MAP given a set of known relevant documents is relevance feedback. Such improvements are expected to be smaller than those observed herein as in relevance feedback only a subset of relevant documents are identified so fewer documents are available for learning; over-fitting to this subset may also occur.

8. Conclusions

An oracle that can return the optimal set of document structure weights to use for structure weighted information retrieval is theorized. Using the oracle gives the performance upper bound for this technique. For the TREC WSJ collection, an approximation to the oracle is constructed and tested and an approximation to the upper bound is computed.

When the oracle was not used, BM25 was shown to outperform naïve probability, which outperformed inner product. When the oracle was consulted for each query, mean average precision showed an improvement of 15% for naïve probability and BM25, and of 30% for inner product. Even so, using the oracle did not change the order of the functions; BM25 was still better then naïve probability which is still better than inner product.

Only small improvements are seen in most queries, while very few queries show large improvements. The queries that showed large improvements were different for each ranking function.

When the oracle was consulted once for a single set of weights to use for every query, smaller improvements were seen. Inner product and naïve probability showed a 5% improvement whereas BM25 showed a 0.6% improvement. Again, the order of goodness of the functions did not change. BM25 was better than naïve probability which was better than inner product.

These results suggest document structure weighted retrieval is better suited to relevance feedback than to *ad hoc* retrieval.

References

[1] Baeza-Yates, R., Navarro, G., & Vegas, J. (1998). A model and a visual query language for structured text. In *Proceedings of the String Processing and Information Retrieval: A South American Symposium*, (pp. 7-13).

[2] Boyan, J., Freitag, D., & Joachims, T. (1996). A machine learning architecture for optimizing web search engines. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*.

[3] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (2003). Extensible markup language (XML) 1.1 W3C proposed recommendation. The World Wide Web Consortium. Available: http://www.w3.org/TR/2003/PR-xml11-20031105/.

[4] Fuhr, N., & Gövert, N. (2002). Index compression vs. Retrieval time of inverted files for XML documents. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management*.

[5] Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (2002). INEX: Initiative for the evaluation of XML retrieval. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*.

[6] Fuhr, N., & Großjohann, K. (2000). XIRQL an extension of XQL for information retrieval. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*.

[7] Fuller, M., Mackie, E., Sacks-Davis, R., & Wilkinson, R. (1993). Structured answers for a large structured document collection. In *Proceedings of the 16th ACM SIGIR Conference on Information Retrieval*, (pp. 204-213).

[8] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*: Addison-Wesley.

[9] Gordon, M. (1988). Probabilistic and genetic algorithms in document retrieval. *Communications of the ACM,* 31(10), 1208-1218.

[10] Harman, D. (1992). Ranking algorithms. In W. B. Frakes & R. Baeza-Yates (Eds.), *Information retrieval: Data structures and algorithms* (pp. 363-392). Englewood Cliffs, New Jersey, USA: Prentice Hall.

[11] Harman, D. (1993). Overview of the first TREC conference. In *Proceedings of the 16th ACM SIGIR Conference on Information Retrieval*, (pp. 36-47).

[12] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

[13] ISO8879:1986. (1986). *Information processing - text and office systems - standard generalised markup language (SGML).*

[14] Kim, Y.-H., Kim, S., Eom, J.-H., & Zhang, B.-T. (2000). SCAI experiments on TREC-9. In *Proceedings of the 9th Text REtrieval Conference (TREC-9)*, (pp. 392-399).

[15] Kotsakis, E. (2002). Structured information retrieval in XML documents. In *Proceedings of the ACM Symposium on Applied Computing*, (pp. 663-667).

[16] Lee, Y. K., Yoo, S.-J., Yoon, K., & Berra, P. B. (1996). Index structures for structured documents. In *Proceedings of the 1st ACM International Conference on Digital Libraries*, (pp. 91-99).

[17] Meuss, H., & Strohmaier, C. (1999). Improving index structures for structured document retrieval. In *Proceedings of the 21st Annual Colloquium on IR Research (IRSG'99)*.

[18] Rapela, J. (2001). Automatically combining ranking heuristics for HTML documents. In *Proceedings of the 3rd International Workshop on Web Information and Data Management*, (pp. 61-67).

[19] Robertson, S. E., Walker, S., Beaulieu, M. M., Gatford, M., & Payne, A. (1995). Okapi at TREC-4. In *Proceedings of the 4th Text REtrieval Conference (TREC-4)*, (pp. 73-96).

[20] Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM,* 18(11), 613-620.

[21] Schlieder, T., & Meuss, H. (2000). Result ranking for structured queries against XML documents. In *Proceedings of the DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*.

[22] Schlieder, T., & Meuss, H. (2002). Querying and ranking XML documents. *Journal of the American Society for Information Science and Technology,* 53(6), 489-503.

[23] Shaw, W. M., Wood, J. B., Wood, R. E., & Tibbo, H. R. (1991). The cystic fibrosis database: Content and research opportunities. *Library and Information Science Research,* 13, 347-366.

[24] Shin, D., Jang, H., & Jin, H. (1998). BUS: An effective indexing and retrieval scheme in structured documents. In *Proceedings of the 3rd ACM International Conference on Digital libraries*, (pp. 235-243).

[25] Thom, J. A., Zobel, J., & Grima, B. (1995). *Design of indexes for structured documents* (CITRI/TR-95- 8). Melbourne, Australia: Department of Computer Science, RMIT.

[26] Trotman, A. (2005). Choosing document structure weights. *Information Processing & Management,* 41(2), 243-264.

[27] Vrajitoru, D. (2000). Large population or many generations for genetic algorithms? Implications in information retrieval. In F. Crestani & G. Pasi (Eds.), *Soft computing in information retrieval. Techniques and applications* (pp. 199-222): Physica-Verlag.

[28] Wilkinson, R. (1994). Effective retrieval of structured documents. In *Proceedings of the 17th ACM SIGIR Conference on Information Retrieval*, (pp. 311-317).