GPX - Gardens Point XML IR at INEX 2006

Shlomo Geva

Faculty of Information Technology
Queensland University of Technology
Queensland 4001 Australia
s.geva@qut.edu.au

Abstract. The INEX 2006 evaluation was based on the Wikipedia collection in XML format. It consisted of several tasks that required different approaches to element selection. In this paper we describe the approach that we adopted in an attempt to satisfy the requirements of all the tasks, Thorough, Focused, Relevant in Context, and Best in Context. We have used the same underlying system to approach all tasks. The retrieval strategy is based on the construction of a collection sub-tree, consisting of all nodes that contain one or more of the search terms. Nodes containing search terms were then assigned a score using the GPX ranking scheme which incorporates TF-IDF or BM25 variants, but extends them. Scores are recursively propagated to ancestors in the document XML tree, and finally all scoring XML elements are ranked. We present results that demonstrate that the approach is versatile and produces consistently good performance. We also provide empirical analysis of the GPX ranking scheme and compare its performance against a baseline TF-IDF and a BM25 scoring scheme..

Keywords: XML IR Information Retrieval GPX INEX Evaluation.

1 Introduction

The INEX 2006 Ad-hoc track consisted of 4 tasks, namely Thorough, Focused, All in Context and Best in Context retrieval. These tasks are described elsewhere in the proceedings. We have used the 2005 GPX search engine algorithms with some minor modifications [1,3]. The software was ported in 2006 from C# and MS-Access to Java and the Apache Derby relational database. This was done to achieve speedup in searching, but the basic system remained almost unchanged. We implemented extended support for more complex queries [2], lifting some of the limitations of NEXI. GPX thus represents an evolving system that started in 2004 and its evolution documented in the annual workshop proceedings. The reader is invited to read the 2004 and 2005 descriptions of GPX for more detail [1,3]. In this paper we describe the approaches we took to the various tasks in 2006 and discuss the results. Finally, we analyze the performance of the GPX scoring scheme by comparing it against TF-IDF and BM25 [4]. In the following sections we have also attempted to provide a comprehensive enough description of GPX so that it can be reproduced on the back of any XML index system that supports the retrieval of XPath inverted lists.

N. Fuhr, M. Lalmas, and A. Trotman (Eds.): INEX 2006, LNCS 4518, pp. 137–150, 2007.

2 The GPX Search Engine

For the sake of completeness we provide a brief description of GPX. The search engine is based on XPath inverted lists. For each term in the collection we maintain an inverted list of XPath specifications. This includes the file name, the absolute XPath identifying a specific XML element, and the term position within the element. The actual data structure is designed for efficient storage and retrieval of the inverted lists which are considerably less concise by comparison with basic text retrieval inverted lists.

2.1 Inverted List Representation

We have chosen to implement GPX using a relational database as backbone architecture; however, the system is in fact based on a traditional inverted list which was extended to support XML IR. The choice is motivated by the extensive off the shelf functionality of a DBMS and ease of programming of all I/O operations. We do not however use any of the expensive recovery and concurrency mechanisms that the DBMS supports by using minimal footprint embedded mode executables. The software that we used is the Apache Derby[1] open source DBMS, freely available under Apache License, Version 2.0. Derby's footprint is small -- about 2 megabytes for the base engine and embedded JDBC driver.

In principle, before optimising the database schema, a suitable inverted list for our purposes consists of a single table with the following structure:

Term-Context = { **Term**, File-Name, XPath, Position }

This structure is sufficient to allow us, given a term, to retrieve all contexts in which the term appears. The Position column allows us to support phrase searches or proximity operators. The collection contains approximately 140 million postings hence each byte in a posting contributes 140MB to the size of the inverted list (ignoring other overheads). It is obvious that this structure is exceedingly redundant and the representation of postings in the list can become very expensive. We describe several ways by which we kept the inverted lists index size under control.

There are approximately 2 million unique terms in the collection. It is not necessary to store the Term column in the table because it could be stored in an auxiliary table, recording the Term, Start position, and End position in the inverted list table. However, this makes it more difficult to manage a dynamic collection where insertions and deletions are allowed. We have chosen not to adopt this approach. However, we do map a term to Term-ID. The Term-ID is only a 4 byte integer and an auxiliary table provides the mapping of Term to Term-ID. Terms usually exceed 4 bytes, particularly since there is always an overhead of several bytes if variable length strings are used. Similarly, file names are very long and so a simple normalization is performed to map a File-Name to a File-ID. With about 660,000 files in the collection 4 bytes are sufficient. The Position column can be safely stored on 2 bytes since text nodes do not exceed 64KB.

[1] http://db.apache.org/derby/

The representation of the XPath is more problematic. The tag names can be rather long, and XPath expressions can contain numerous nodes and be very long. We could have encoded tag names - there are less than 256 meaningful tags so one byte could suffice to represent a tag. We have chosen not to do so because that would render the lists unreadable without decoding and this was inconvenient. With XPath lengths varying from 10 bytes to over 300 in this collection, the overhead of storing the explicit XPath is significant even with coding. We observe that each unique XPath is repeated in the inverted list for each term in the same node, and the XPaths themselves are repeated in many files. For instance, almost every article in the collection has a node /article[1]/body[1]/p[1]. There is significant redundancy here already, but furthermore, many paths which are not identical share a common sub-path. This suggests a compression scheme like LZW might be effective. We considered this to be unnecessary, particularly given the processing overheads and so we have adopted the following simple yet effective compression scheme.

Consider the XPath:

/article[1]/bdy[1]/sec[5]/p[3]

This could be represented by two expressions, a Tag-set and an Index-set:

Tag-set: article/bdy/sec/p

Index-Set: 1/1/5/3

The original XPath can be reconstructed from the tag-set and the index-set. It turns out that there are over 48,000 unique tag-sets, and about 500,000 unique index-sets in the collection. We assign to each tag set and each index-set a hash code and create auxiliary database tables mapping the hash-codes to the corresponding tag-set and index-set entries. These hash tables are small enough to be held in memory and so run-time decoding is efficient. We have used Java's inbuilt hash code function. It is only a 32 bit code, but given the number of elements the risk of hash collisions is minimal and we take it.

Finally, in order to implement BM25 it is necessary to record node sizes. So an XPath table is maintained where the XPath is represented by a hash code computed from the concatenation of the FileName and the XPath columns. We use a 63 bit MD5 hash code because there are about 25 million distinct nodes in the collection and the probability of a collision is too high with Java's 32 bit inbuilt hash function. We use 63 bits although MD5 provides a 64bit code since Java's *Long* integers are limited to the constant 0x7FFFFFFFFFFFFFFF.

Finally, the database schema consists of the following tables:

Term-Context ={ **Term-ID**, File-ID, XPath-Tag-ID, XPath-IDX-ID, Position }
Terms = { **Term**, Term-ID }
Files = { **File-Name**, File-ID }
TagSet = { **XPath-Tag-ID**, Tag-Set }
IndexSet = { **XPath-IDX-ID**, Index-Set }
XPathSize = { **XPath-ID**, Node-Size }

The size of the database is 15GB and this represents an overhead of about 3:1 over the source documents (uncompressed). This is quite acceptable with current disk costs and capacities even if much more efficient representations are possible.

Some performance figures of this database are as follows. The time it takes to parse and load the 659,388 files on a 3GHz PC with 2 GB RAM is 9 hours. When the search engine is started, the Files, TagSet, IndexSet, and XPathSize tables are loaded into RAM – this takes about 15 seconds. The only required I/O operations during a search are then on the Terms table and the Term-Context inverted list table. The average time to evaluate a topic is 7.2 seconds, but a few topics take more than 30 seconds to evaluate on account of having more terms and longer inverted lists that correspond to common terms.

2.2 The GPX Ranking Scheme

Retrieval is performed by processing the NEXI expression and interpreting the query constraints to combine the inverted lists. In the simple case of a CO query we simply compute scores for all elements that contain at least one of the search terms. Several steps are followed in evaluating a query and these are described in the following sub-sections.

2.2.1 Calculation of Text Nodes Score

Equation 1: Calculation of element relevance score from its content

$$L = K^{n-1} \sum_{i=1}^{n} \frac{t_i}{f_i} \tag{1}$$

Here n is the count of unique query terms contained within the element, and K is a small integer (we used K=5). The term K^{n-1} scales up the score of elements having multiple distinct query terms. This heuristic of rewarding the appearance of multiple distinct terms can conversely be viewed as taking more strongly into account the absence of query terms in a document. Here it is done by rewarding elements that do contain more distinct query terms. The system is not sensitive to the value of K as demonstrated in the results section and a value of $k=5$ is adequate. The summation is performed over all n terms that are found within the element where t_i is the frequency of the i^{th} query term in the element and f_i is the frequency of the i^{th} query term in the collection. Similar results are obtained if we use the *TF-IDF* to compute the sum, but it does not lead to significantly different results in our experience. We describe the results of experiments with *TF-IDF* and with *BM25* in a later section. At INEX 2006 we used the term inverse collection frequency and refer to it as *TF-ICF* (as distinct from other *TF-IDF* variants.)

Finally, phrases are weighted more heavily than individual terms (phrase weight are multiplied by 10) and nodes that contain query terms that are preceded by a minus sign (undesirable) are not returned at all.

2.2.2 The GPX NEXI Interpretation

The GPX search engine supports an extended set of functionalities which are a superset of NEXI. These are described in more detail by Geva et al in [3]. Here we limit the discussion to the details which are relevant to the INEX 2006 evaluation.

The evaluation of a NEXI expression always starts by converting the query expression from postfix to infix for sequential evaluation. For example, consider the query –

```
//article[about(.,Albert Einstein)]/body[about(.//figure,Copenhagen)  OR
  about(.//section,Bohr)]
```

The query is converted to the following stack oriented evaluation specification –

1) PUSH(//article[about(.,Albert Einstein)])
2) PUSH(//article/body[about(.//figure,Copenhagen)])
3) PUSH(//article/body[about(.//section,Bohr)]
4) PUSH(OR(POP, POP))
5) PUSH(SUPPORT(POP,POP))

This set of operations is evaluated by using an inverted lists stack. The first 3 steps evaluate the 3 distinct filters that appear in the NEXI expression. Each list of elements satisfies its respective *about* clause. In step 4 the top two lists are ORed and the resulting list pushed back onto the stack. This effectively evaluates the OR operator on the body element. In the final step the SUPPORT operator is applied to take into account the filter on the article node which "supports" the selection of the body node on account of content which is not necessarily contained in the body. The implementation of OR, AND, and SUPPORT is now explained.

The OR operator computes the union of two inverted lists, X and Y. The call to OR(X,Y) returns a new list. Elements in the lists identify XML result elements by file-id, full XPath expression, and relevance score. The OR operator performs a set union whereby elements that appear in both lists are merged and their scores added together. Other elements that appear in either list keep their original scores.

The AND operator computes the intersection of two inverted lists, X and Y. In GPX this operator is not necessarily a strict set intersection but can be loosely interpreted in one of three ways. The default option is to simply implement it as OR(X,Y). However, in some queries the user really means AND in a strict sense; therefore, a second option is to implement it as a strict set intersection - only XML elements that appear in both X and Y are kept, and their scores are added together. This option is too restrictive because sometimes the lists contain overlapping elements and then the relationship with respect to AND is unclear. By insisting on a strict match many relevant results are lost. The third implementation keeps overlapping nodes, combines the scores, but keeps only the largest node (oldest common ancestor). In the experiments that we report in the next section, we used the first (default) option. This seems to work quite well in most instances, and works better on average.

The SUPPORT operator does not have an equivalent set operator and is specific to our interpretation of NEXI. In NEXI, we refer to support elements and target elements. Target elements are those elements that appear at the tail of the NEXI expression with a filter, while support elements are internal elements with filters that appear along the path to the target elements. The SUPPORT operator takes a list of nodes in X that provide support to the selection of nodes from list Y. For instance, when we look for paragraphs about Americium in articles with abstracts about the

Periodic Table, the target elements are paragraphs about Americium, and paragraphs are supported by abstracts about the Periodic Table. Both the support and target elements must have a common ancestor within the document tree. In the case of the Wikipedia this is the article element. The supporting abstract must appear in the same article as the supported paragraphs. The support operator identifies for each result element in Y, all the support elements in X, and combines the scores. It is important to note that all the elements in Y are returned, regardless of support. However, elements with support have an increased score.

2.2.3 The GPX Derivation of a Full Recall Base

Having computed the scores of all elements in the collection which contain query terms directly as text within the XML node, we must proceed to consider the scores of elements on account of their relevant descendents. The scores of retrieved elements are now recursively propagated upwards in the document XML tree according to the following scheme.

Equation 2: Calculation of a Branch Element Relevance Score

$$R = L_{0+} D(n) \sum_{i=1}^{n} L_i \tag{2}$$

Where:

L_0 = the score of the current node (from Equation 1), zero by default
n = the number of children elements
$D(n)$ = N1 if n = 1
= N2 Otherwise
L_i = the relevance score of the i^{th} child element

The introduction of the term L_0 was necessary when moving from the IEEE articles collection to the Wikipedia since text appeared only in leaf nodes in the IEEE collection, but many Wikipedia nodes contained both direct text and descendents with text. The value of the decay factor D depends on the number of relevant children that the branch has. If the branch has one relevant child then the decay constant is smaller. Generally we have 0<=N1<=N2<1. A branch with only one relevant child will be ranked lower than its child. The decay factor N2 may be chosen large enough so that a branch with several relevant children will be ranked higher than its descendants. Thus, a section with a single relevant paragraph would be judged less relevant than the paragraph itself, but a section with several relevant paragraphs might be ranked higher than any of its descendent paragraphs.

It is attractive to consider the use of node size directly in score propagation. As we progress upwards through the tree, node specificity tends to decrease, but coverage (recall) can increase when multiple descendents are combined in an ancestor node. Node size can provide some direct information in relation to precision, but we were unable to discover a robust way to incorporate the node size into Equation 2.

Finally, the cost of propagating the scores can be very high. Many of the terms that appear in the topics are rather common. This leads to a very high computational load. In order to reduce the total time it took to generate the results we have imposed two limitations. All terms that occur with a frequency greater than 100,000 in the

collection were treated as stop-words and ignored. This proved to be of little consequence with the topic set on hand. Of course one can imagine situations when this would be a costly decision. This reduced the processing time of the entire set of topics by 50% with very minimal degradation in MAep values (less than 1%). The second limitation that we imposed was more severe. It reduced the processing time by more than 7 fold. The limitation was placed on the number of nodes that were taken forward from the first phase (Equation 1) towards generation of the full recall base. We have taken the 3000 highest scoring nodes at most. In some instances hundreds of thousands of scoring elements were dropped. This meant that incomplete information was used in propagating scores upwards in the document tree. As it turns out this did make a significant difference to precision and recall. This performance cost is discussed later in the experimental section. It must be noted that both limitations can be lifted at the cost of increased processing time which can be reduced by other means without sacrificing precision and recall – for instance, by storing inverted lists for word-grams instead of single terms. With extremely common terms this could reduce the list lengths by several orders of magnitude by reducing the time required to complete the I/O and set operations over the lists.

After the scores of all nodes are computed GPX proceeds to add the score of the Article node in each document to the score of each node in the article (including the article node itself). This heuristic correction is intended to generally push up the scores of elements that appear in documents that are more relevant. Empirically we were able to establish that better results may be obtained in this manner.

Finally, we note that GPX is based on a simple variation of TFIDF. Robertson provides a comprehensive discussion of various theoretical arguments for IDF [5], but in the end IDF is still a heuristic approach, and so is GPX.

3 Experimental Results

In this section we present and discuss the results that were obtained at INEX 2006. We also present the results of an empirical sensitivity analysis of various parameters of the GPX search engine, performed with the Wikipedia collection.

3.1 Thorough Retrieval

All the QUT runs were generated with GPX search engine, starting with thorough retrieval. We have experimented with several settings of the decay factor, with strict and loose interpretation of NEXI expressions, and with various query expansion techniques. The official results of the Thorough Retrieval task are reproduced in Figure 1. The solid line is the GPX submission that was ranked 3rd, with a MAep value of 0.0699. This is just 0.001 below the top submission, but qualitatively there seems to be a difference - the precision of the GPX run is slightly lower at low recall levels, but the overall recall (area under the curve) is higher. This run was obtained with N1=0.11 and N2=0.31 in determining D(n) in Equation 2. Both GPX runs were CO runs. The COS runs did not perform as well. This is the reverse of what was observed with the IEEE CS collection that was used in earlier evaluations. We attribute the difference to the apparent lack of semantic tagging in the Wikipedia.

The solid dash-dot line corresponds to the GPX run which was ranked 9th with MAep value of 0.0620 and was obtained with the values of the decay parameter N1=0.31 and N2=0.71. Qualitatively it is similar to the other runs in the top 10.

The solid dotted line is an unofficial run that corresponds to exactly the same system setting as our best official run, except that we kept 30,000 elements rather than 3,000 in producing the full recall base through score propagation (equation 2). The MAep increased by 10% to 0.077. This is a very significant improvement and it quantifies the cost of more efficient retrieval. Instead of 15 minutes, it took 111 minutes to generate a complete run submission of 125 topics.

Another difference between the two GPX runs is that the better performing run (by MAep) was produced by adding to the CO title element a support filter. The filter was placed over the article name. In this manner, elements that appeared in articles whose name was *about* the same keywords as the title received a boosted score. This was simply achieved by adding the filter to the title. For example,

//article[about(.,X Y Z)] → //article[about(./name,(X Y Z)]//*[about(.,X Y Z)]

It should be noted that in GPX the meaning of the "//*" path specification is "this node or any descendents" rather than "any descendent". The modified expression is evaluated by GPX in the usual manner and supported nodes receive an additional score from the support element – if found in the same article. The heuristic is obvious – if the search terms appear in the Wikipedia article name itself then it is more probable that the article is relevant.

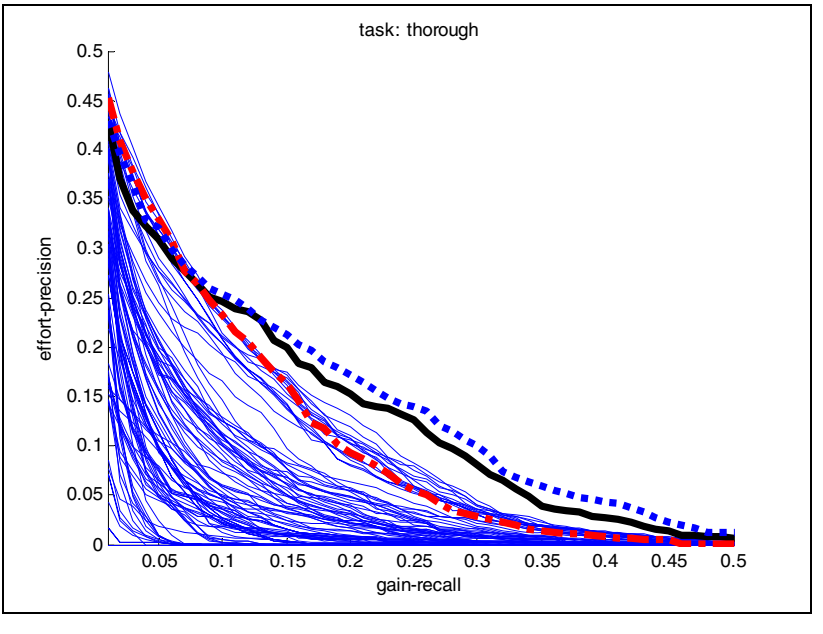

Fig. 1. GPX Thorough Retrieval

3.2 Focused Retrieval

Focused Retrieval starts with the thorough results recall base. The highest scoring elements on a path are selected by keeping only elements that have a higher score than any of their descendents or ancestors. Figures 2 and 3 depict the official plots, with the solid heavy line corresponding to the best official GPX submission. Again, the unofficial dotted line was later produced by running GPX with the same setting as the best official run, but taking the top 30,000 elements rather than the top 3,000 in generating the thorough recall base from which the focused run was produced. The performance difference is again quite large. There is a clear incentive to write a more efficient implementation of GPX that will keep processing low while providing a significant performance improvement.

3.3 Best in Context

We tested a trivial approach here – we simply kept the highest scoring element in each document appearing in the recall base. This simple approach seems to have produced good results with the BEPD metric. The GPX submission was ranked 3rd with the setting A=0.01. This, according to the official BEP metric documentation, means that the system was comparatively successful in pinpointing the BEP. Low A values favor runs that return elements that are very close to the BEP.

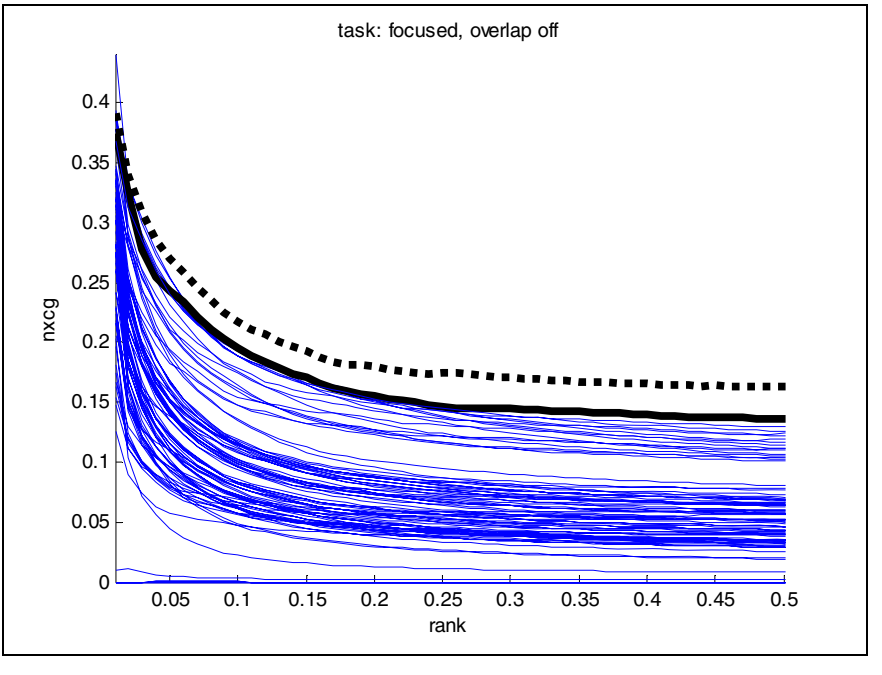

Fig. 2. GPX Focused Retrieval, with overlap OFF

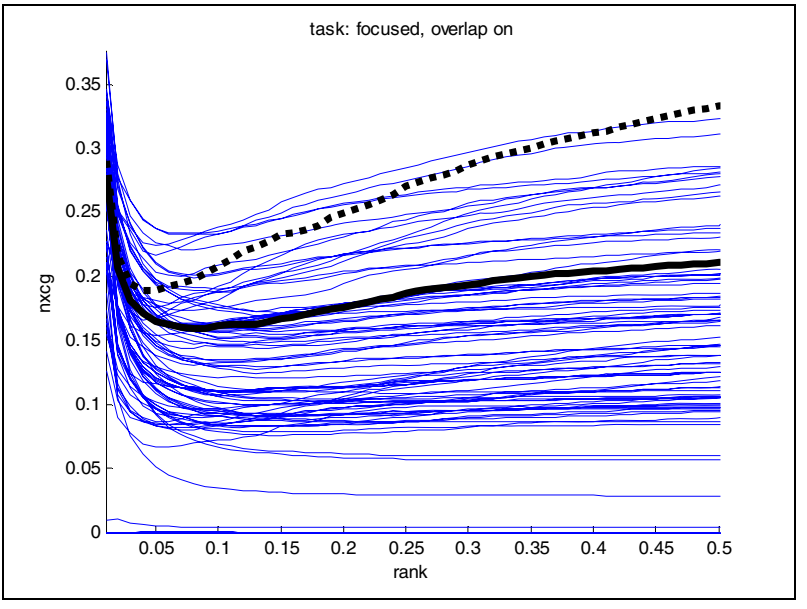

Fig. 3. GPX Focused Retrieval, with overlap ON

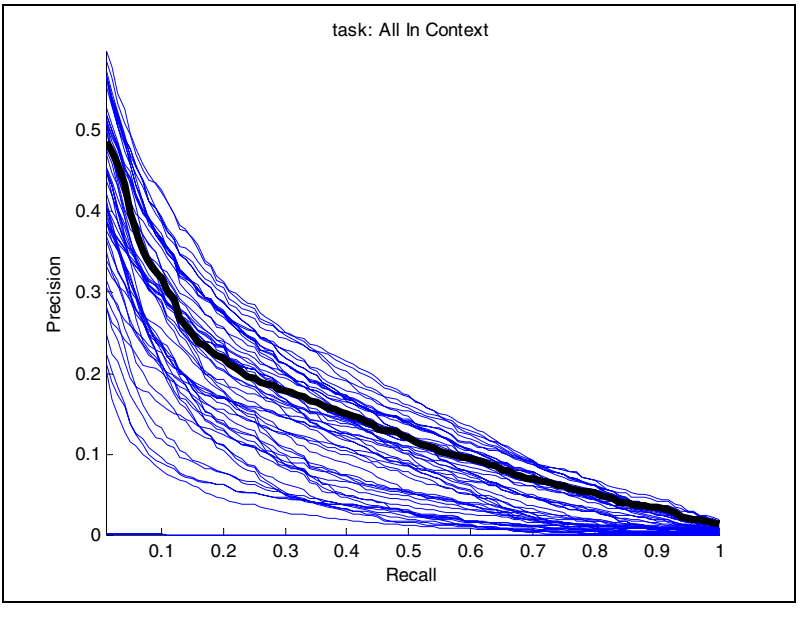

Fig. 4. GPX All in Context Retrieval

3.4 All in Context

The objective of the task was to balance article retrieval and element retrieval. Whole articles are first ranked in descending order of relevance and within each article a set of non-overlapping most focused elements are grouped. We have used the focused results, which were overlap free already, but grouped the elements within articles and sorted the articles by score. The results are reasonable but performance suffers because the focused recall base is not ideal for this task. The focused recall base retains the most relevant elements in the collection, but *out of context*. This means that high recall within article is not assured. Nodes that have a lower score can drop out of the top 1500 elements of the Focused run. This works against All in Context retrieval where the F-Score of an article demands high recall within each article. In addition to this, since from the outset we have only considered the top 3000 elements in generating the thorough recall base, even the thorough recall base was incomplete. This means that elements that might otherwise appear in highly ranked articles were never retained. Figure 4 depicts the official results with the GPX best run depicted as a heavy solid line.

3.5 Empirical Evaluation of GPX Scoring

In this section we present the results of empirical evaluation of the GPX scoring strategy. We study the effect of each of the components in Equation 1 and Equation 2. In order to evaluate the sensitivity of the evaluation to the score propagation constant D(n) in Equation 2, we have fixed all other parameters, and used N1=N2=N, and varied the value of N in small steps from zero to one. The results are summarized in Table 1.

Table 1. The impact of choice D(n)

D(n)	Thorough MAep	Focussed nxCG@50 overlap OFF	Focussed nxCG@50 overlap ON	All in Context MAep
0	0.017	0.262	0.219	0.082
0.1	0.039	0.280	0.231	0.117
0.2	0.049	0.289	0.232	0.125
0.3	0.056	0.296	0.237	0.131
0.4	0.062	0.298	0.233	0.135
0.5	0.066	0.287	0.234	0.142
0.6	0.068	0.268	0.235	0.148
0.7	0.070	0.248	0.230	0.153
0.8	0.069	0.235	0.223	0.156
0.9	0.069	0.218	0.211	0.157
1.0	0.068	0.154	0.158	0.100

The best performance is achieved at different D(n) values for the different tasks. In the Thorough and All in Context higher values (0.8) produce better results since higher D(n) values lead to exhaustive selection within article. In the Focused task lower values (0.3) produce better results since precision is favoured by lower D(n) values. This is also confirmed in our official run results. Importantly however, the performance is not extremely sensitive to the selected value and there is no catastrophic degradation of performance for some values.

In order to ascertain that the heuristic motivation to Equation 1 is indeed sound, we have conducted experiments where we evaluated the individual components in isolation. Figures 5 and 6 depict the performance of runs when the score calculation is based on the number of unique query terms alone (K^{n-1}), on a TFIDF variation alone (TFICF or BM25), and on the combination of the two as in Equation 1. It is clear that when both components are used the performance is much improved (dotted line). Furthermore, there is no advantage to using BM25 over the simpler TFICF variation. This is not surprising since most text elements are small and BM25 is indeed expected to make little difference for small documents [4]. We have used BM25 with the default values K1=2 and b=0.75, but other values produced similar results.

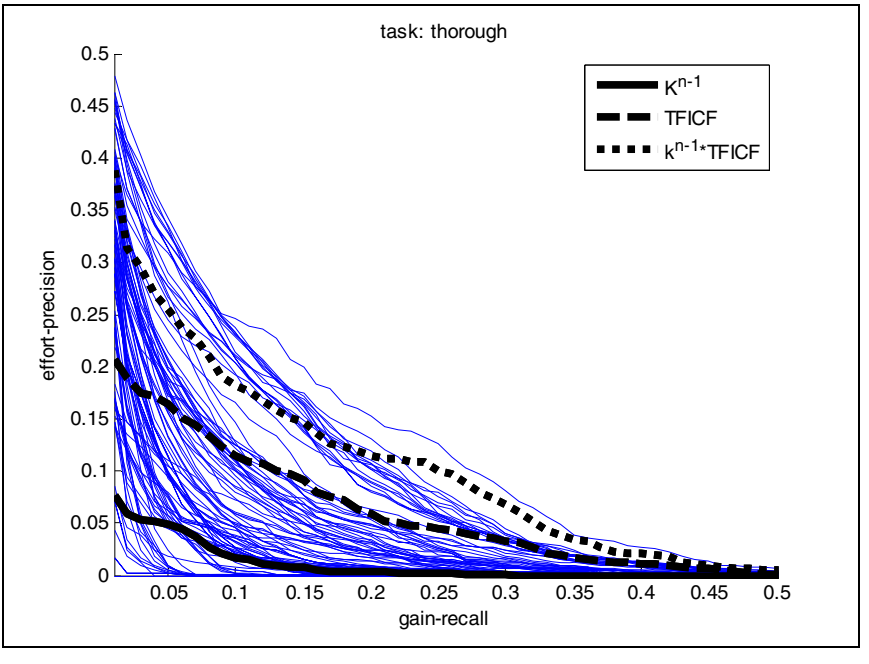

Fig. 5. GPX with TFICF

Finally, we tested several variations for the value of K in Equation 2. While holding all other parameters constant we have varied the value of K from 1 to 50. Figure 7 depicts the results. There is an improvement as K values increase to about 5 and then for values of between 5 and 50 there is no further improvement and the

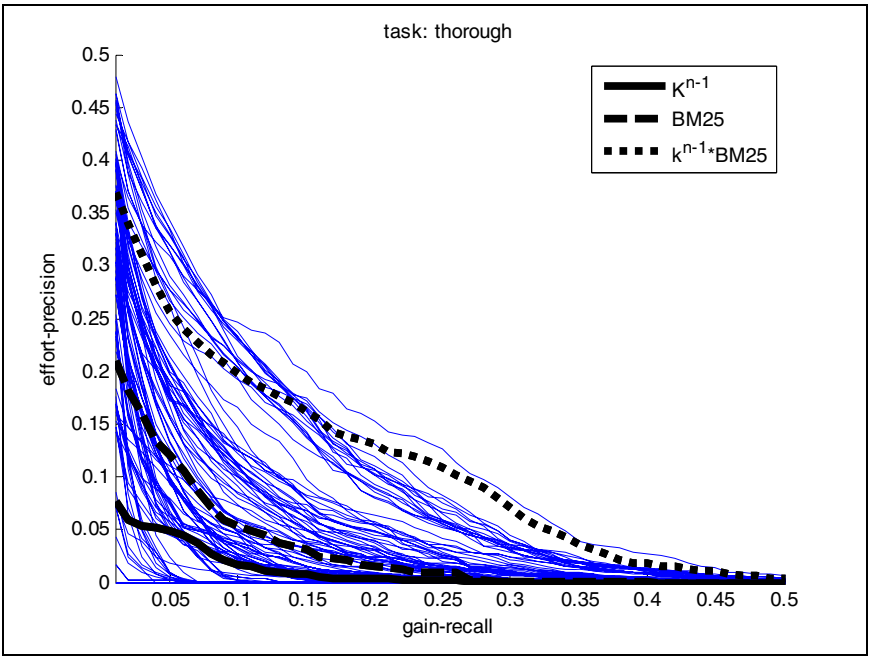

Fig. 6. GPX with BM25

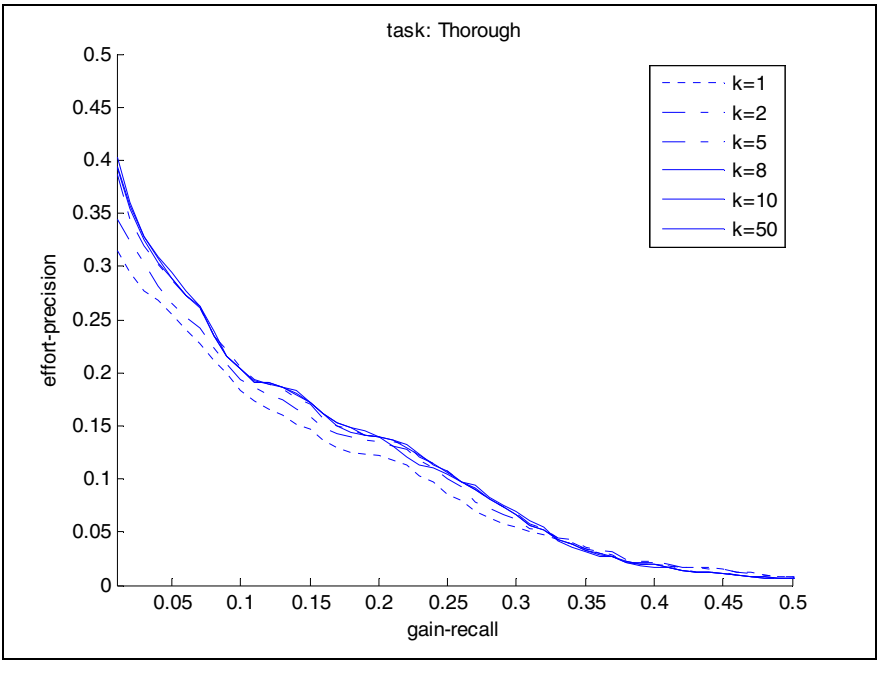

Fig. 7. GPX with varying K values

performance is stabilized. This can be understood as follows. Equation 1 heavily rewards elements that contain more distinct terms, through K^{n-1}. The TFIDF component in Equation 1 moderates that score. Once the value of K is large enough the moderation that is contributed by the TFIDF component is no longer sufficient to moderate the rank order of elements with a very different number of distinct query terms.

4 Conclusions

GPX performed rather well on the Wikipedia in most tasks. This result demonstrates that the method is quite robust since GPX was designed and implemented with the IEEE collection, but evaluated with the Wikipedia. The best relative performance was achieved in the Thorough, Focused (overlap off) , and BEP tasks. The performance in the All in Context task and Focused (overlap on) was not quite as good, but respectable nevertheless. Future work will focus on ranking strategies that take node size and structure into account in an explicit manner, to try and capture the intuitively appealing F-Score calculation which was used in the evaluation of the All in Context task. More work is also required on improving search efficiency. List processing is extensive and the current implementation is CPU bound rather than I/O bound. A response time of 7 seconds per topic is inadequate for implementing a high throughput online system. We are unable to compare the efficiency of our system with that of other systems at this stage because the INEX evaluation does not formally support a systematic comparison of this aspect.

References

1. Geva, S.: GPX - Gardens Point XML Information Retrieval INEX 2004. In: Fuhr, N., Lalmas, M., Malik, S., Szlavik, Z. (eds.) Advances in XML Information Retrieval. Third International Workshop of the Initiative for the Evaluation of XML. LNCS, pp. 211–223. Springer, Heidelberg (2005)
2. Geva, S.: GPX - Gardens Point XML IR at INEX 2005, INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) Advances in XML Information Retrieval. Fourth International Workshop of the Initiative for the Evaluation of XML. LNCS, pp. 240–253. Springer, Heidelberg (2006)
3. Geva, S., Tannier, X., Hassler, M.: XOR - XML Oriented Retrieval Language, SIGIR 2006, Workshop on XML Element Retrieval Methodology, Proceedings online at: http:/www.cs.otago.ac.nz/sigirmw/Proceedings.pdf
4. Robertson, S.E., Sparck Jones, K.: Simple, proven approaches to text retrieval, University of Cambridge Technical Report UCAM-CL-TR-356, ISSN 1476-2986, December 1994, last updated February 2006. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-356.pdf
5. Robertson, S.: Understanding Inverse Document Frequency: On theoretical arguments for IDF. Journal of Documentation 60(5), 503–520 (2004)