The Effect of Structured Queries and Selective Indexing on XML Retrieval

Börkur Sigurbjörnsson[1] and Jaap Kamps[1,2]

[1] ISLA, Faculty of Science, University of Amsterdam
[2] Archives and Information Studies, Faculty of Humanities, University of Amsterdam

Abstract. We describe the University of Amsterdam's participation in the INEX 2005 ad hoc track, covering the *Thorough*, *Focused*, and *Fetch-Browse* tasks and their structured ($+S$) counterparts. Our research questions for this round of INEX were threefold. Our first and main research question was to investigate the contribution of structural constraints to improved retrieval performance. Our main results were that the two types of structural constraints have different effects. Constraining the target of result elements gives improvements in terms of early precision. Constraining the context of result elements improves mean average precision. Our second research question was to experiment with selective indexing strategies based on either the length of elements, the tag-name of elements considered relevant in earlier INEX years, or simply by indexing all sections or articles. Our experiments show that disregarding 80–90% of the total number of elements does not decrease retrieval performance. Third, we considered the automatic creation of structured queries using blind feedback. Here, our results are inconclusive, mainly due to few queries used and lack of comparison to traditional blind feedback.

1 Introduction

In this paper we describe the University of Amsterdam's participation in the INEX 2005 adhoc track. The three research questions we addressed in this year's round of INEX were to explore the contribution of structured constraints, to try to make our system more efficient by reducing the size of our index, and to construct structured queries as a form of query expansion. These research questions built on experiences obtained in our previous INEX participations.

In previous years we created our runs based on an index of all overlapping XML elements [10,11]. Our main technical objective this year was to experiment with different methods of creating a more selective index. The aim was to create a more efficient retrieval system without sacrificing retrieval effectiveness. We measured the effect of several different index reduction schemes. In the *Focused* task we looked at two simple non-overlapping indexes: section index and article index. As a baseline we used a run from our overlapping element index, where overlap was removed in a list based manner. Both the section run and the article run performed considerably worse than the baseline. In the *Thorough* task we

looked at two reductions of our full overlapping element index: one based on element length and another based on past relevance assessments. Our main finding was that even with a 80–90% reduction in the number of indexing units, we do not see a reduction in retrieval effectiveness.

In our experiments with structured queries in previous years, we found that structural constraints lead to improvements in early precision. This year we wanted to explore whether different types of structural constraints contribute differently to this gain. We measured the effect of different aspects of structural constraints. The *CO+S* tasks provide an excellent framework for these experiments. We compared four runs using different structural aspects. First of all, we had a baseline where no structural constraints were used. We had two runs which used a single aspect of the structured queries: either by restricting the type of target element, or by restricting the context of target element. Finally, we had a run which used both aspects of structured queries. Our main finding is that the target constraints are useful for improving early precision. On the other hand, the context constraints are more useful for mean average precision.

Some of the topics in the *CO+S* task were only formulated using a content only query, but had no content and structure formulation. For these queries we attempted to create a structured query automatically using blind relevance feedback. As with so many blind feedback experiments, our results are inconclusive, partly due to a limited number of queries used.

Finally, we also looked at the the new *FetchBrowse* task, for which we considered the simple clustering of our *Focused* runs. That is, for the *FetchBrowse* task we re-ordered our *Focused* runs such that results from the same document appear at consecutive positions in the ranking. Our results for this task are inconclusive, mainly because it is not clear how to evaluate this task.

The remainder of the paper is organized as follows. In Section 2 we introduce our retrieval framework: indexing, query processing, and retrieval. We describe our runs in Section 3. In Section 4 we present and discuss our results. We review related work in Section 5. Finally, we provide some more discussion and conclusions in Section 6.

2 Retrieval System

2.1 Indexing

For effective and efficient XML retrieval indexing plays an important role. Any element can, in theory, be retrieved. It has been shown, however, that not all elements are equally likely to be appreciated as satisfactory answers to an information need [4]. In particular, retrieval of the very many, very small elements is not likely to be rewarded by users. Furthermore, users (and hence metrics) may be willing to partially reward near misses. This prompted us to investigate whether we could reduce our indexing size, both in terms of retrievable units and storage size, without harming our retrieval effectiveness.

Element Indexes For retrieving elements we built four indexes.

- *Overlapping element index* We built the "traditional" overlapping element index in the same way as we have done in the previous years (see further [10,11]).
- *Length based index*: Very short elements are not likely to be regarded as relevant. We analyzed the average length of elements bearing different tag-names. We then indexed only element types having an average length above a certain threshold. For INEX 2005 we set the threshold to be 25 terms. The term count was applied before stop-words were removed. The choice of threshold value was not based on rigorous empirical analysis; hence, thresholding is a subject for future work.
- *Qrel based index*: Elements with certain tag-names are more likely than others to be regarded as relevant. We analyzed the INEX 2003 and 2004 assessments and looked at which elements were assessed more frequently than other. We indexed only elements that had appeared relatively frequently in previous assessment sets (i.e., they should constitute at least 2% of the total assessments). As a result, we indexed only 8 element types: `article`, `bdy`, `sec`, `ss1`, `ss2`, `p`, `ip1`, and `fig`.
- *Section index*: Retrieval of non-overlapping elements is a hot topic in XML retrieval. We wanted to investigate how simple one can make non-overlapping retrieval. We built an index based on non-overlapping passages, where the passage boundaries are determined by the structure. We decided to go for a simple solution. We indexed only section elements (`<sec>`). We believed that this simple strategy would be effective, despite (or perhaps even due to) the fact that the sections do not provide a full coverage of the collection.

Article Indexes For retrieving articles we built two indexes.

- *Article index*: the "normal" article index
- *Fielded index*: An article index containing both content and a selection of fields. This index was used for processing context restrictions for the structured queries. The fields were chosen based on INEX 2003 and 2004 structured queries. For our INEX 2005 experiments we used: `abs`, `fm//au`, `fm//atl`, `kwd`, `st`, `bb//au`, `bb//atl`, and `ip1`. Those fields were the ones most frequently used in the INEX 2003 and INEX 2004 content-and-structure queries.

For all indexes, stop-words were removed, but no morphological normalization such as stemming was applied. Table 1 shows some statistics of the different indexes.

2.2 Query Pre–Processing

Recall that the fielded article index only contains the most common query constraints. More precisely, our system handles two types of constraints: target constraints and context constraints. The context constraints we support are a

Table 1. Properties of the the different indexes. *Unit* stands for the number of retrievable units. *Storage* stands for the size occupied in physical storage. *Query time* stands for the time needed to retrieve 200 retrieval units from the index for each of the INEX 2003-2005 CO topics. All retrieval times are relative to the maximum retrieval time.

Index	Units	Storage	Query time
Element index	10,629,617	1.9G	1.00
Length based	1,502,277	1.3G	0.81
Qrel based	1,581,031	1.1G	0.66
Sections	96,600	223M	0.14
Articles	16,819	204M	0.08
Query fields	16,819	275M	–

kind of 'meta-data' constraints on the article. We support 8 types of context constraints. In terms of end-to-end usage, one can think of this as an advanced query interface where the user can add query terms to the following meta-data fields:

- *Abstract*: Article's abstract (`abs`).
- *Article author*: Authors of the article (`fm//au`).
- *Article title*: Title of the article (`fm//atl`).
- *Article keywords*: Keywords manually assigned to the article (`kwd`).
- *Section title*: Section title (`st`).
- *Referenced author*: Author name in the bibliography (`bb//au`).
- *Referenced title*: Article title in the bibliography (`bb//atl`).
- *Initial paragraph*: First paragraph of a section (`ip1`).

We pre-processed the `<castitle>` queries such that they matched our indexing scheme. We processed the `<castitle>` constraints in different ways, depending on their format. *First*, for the the `<castitle>` constraints that match our fielded article index, we only need to rewrite the query such that it fits our indexing scheme. For example, the query:

```
//article[about(.//abs, ipv6)]//sec[about(., ipv6 deployment) or
about(., ipv6 support)]
```

becomes

```
abs:ipv6 ipv6 deployment ipv6 support.
```

Second, for the constraints that only partly match our indexing scheme, we need to do additional processing, i.e.,

```
//*[about(.//au, moldovan) and about(., semantic networks)]
```

becomes

```
fm//au:moldovan bb//au:moldovan semantic networks,
```

Table 2. Frequency of different context constraints in the castitle queries. *Query freq.* refers to how many queries contained the constraint. *Total freq.* refers to how often the constraint was used in total. *Match* refers to how the constraint matches our indexing scheme.

Context constraint	Query freq.	Total freq.	Match
`//*//au`	1	1	partial
`//*//p`	1	1	
`//sec//fig`	1	1	
`//article//atl`	1	2	partial
`//article//abs`	3	3	full
`//article//kwd`	1	1	full
`//article//bdy`	3	4	
`//article//sec`	1	1	
`//article//bb`	1	1	
`//article//bdy//sec`	1	1	
`//article//sec//p`	2	2	

since our index makes a distinction between article authors and referenced authors. *Third*, for `<castitle>` constraints that do not fit our index, we simply extract the query terms. I.e.,

```
//article[about(.//bdy, synthesizers) and about(.//bdy, music)]
```

becomes

```
synthesizers music.
```

For the 28 INEX 2005 `<castitle>` queries, 11 had context constraints that did not match our scheme and 2 had context constraints that did partially match our scheme. Table 2 shows the frequency of different context constraints in the query set. Out of 11 types of constraints we only support 4. Of the 7 constraint types that we do not support 5 have the element names `p`, `sec`, and `bdy` as their deepest context. We believe that the usefulness of these constraints is limited since almost all text is contained within such a context. The remaining two constraint types (using `bb` and `fig`) it might have been useful to be able to handle.

2.3 Automatically Generating Structured Queries

For queries without a `<castitle>`, we added structured query fields using pseudo relevance feedback on the fielded article index [9]. We calculated the top 20 feedback terms and we added up to n fielded terms where n is the length of the original query. For example, the query

```
computer assisted composing music notes midi
```

becomes

```
bb//atl:music bb//atl:musical st:music ip1:musical ip1:music
fm//au:university computer assisted composing music notes midi.
```

2.4 Retrieval

For all our runs we used a multinomial language model [3]. We use the same mixture model implementation as we used in INEX 2004 [11]. We assume query terms to be independent, and rank elements according to:

$$P(e|q) \propto P(e) \cdot \prod_{i=1}^{k} P(t_i|e), \tag{1}$$

where q is a query made out of the terms $t_1, \ldots, t_k$. We estimate the element language model by taking a linear interpolation of three language models:

$$P(t_i|e) = \lambda_e \cdot P_{mle}(t_i|e) + \lambda_d \cdot P_{mle}(t_i|d) + (1 - \lambda_e - \lambda_d) \cdot P_{mle}(t_i), \tag{2}$$

where $P_{mle}(\cdot|e)$ is a language model for element e; $P_{mle}(\cdot|d)$ is a language model for document d; and $P_{mle}(\cdot)$ is a language model of the collection. The parameters λ_e and λ_d are interpolation factors (smoothing parameters). Finally, we assign a prior probability to an element e relative to its length in the following manner:

$$P(e) = \frac{|e|}{\sum_e |e|}, \tag{3}$$

where $|e|$ is the size of an element e. For a more thorough description of our retrieval approach we refer to [11].

We handled the structured queries slightly differently. For each structured query, e.g.

```
//*[about(.//au, moldovan) and about(., semantic networks)]
```

we have a *fielded version*, e.g.

```
fm//au:moldovan bb//au:moldovan semantic networks,
```

and a *content-only* version, e.g.

```
moldovan semantic networks.
```

We used the *fielded version* to create an article run using the fielded article index. We used the *content-only* version to create an element run using an element index. We created a new element run by re-ranking the existing element run using the scores from the article run, i.e., each element is assigned the score of its containing article. Finally, we combined the two element runs using the combination method combSUM [2].

3 Runs

Our retrieval model has two smoothing parameters, λ_e for the element model and λ_d for the document or article model, and the remaining weight $(1-\lambda_e-\lambda_d)$ will be put on the collection model. These parameters determine the amount of

smoothing and optimal values, especially in the case of XML retrieval, depend on the statistics provided by the index. We are using widely different indexes, varying from an index containing all individual elements or subtrees to indexes containing only the article or section elements, it is non-trivial to compare these settings over different indexes. Hence, we typically fix the parameters at values corresponding to traditional adhoc document retrieval, with 0.85 of the weight on the collection model. The exception here are the *Focused* element index runs, where we put more weight on the element and document model based on pre-submission experiments.

3.1 Content-Only Runs

CO.Focused In our focused task we experimented with two different ways of choosing focused elements to retrieve: first, based on the hierarchical segmentation of the collection, and second, based on a linear segmentation of the collection. We also wanted to compare these two approaches with a non-focused baseline, namely a document retrieval system. We submitted three runs:

- *F-Articles* (UAmsCOFocArticle) A baseline run created using our article index. We used a $\lambda = 0.15$ and a normal length prior.
- *F-Elements* (UAmsCOFocElements) A run created using a mixture model of the overlapping element index and the article index. We set $\lambda_e = 0.4$ and $\lambda_d = 0.4$. No length prior was used for this run. Overlap was removed in a list-based fashion, i.e., we traversed the list from the most relevant to the least relevant and threw out elements overlapping with an element appearing previously in the list.
- *F-Sections* (UAmsCOFocSections) A run created using a mixture model of the section index and the article index. We set $\lambda_e = 0.05$ and $\lambda_d = 0.1$. A normal length prior was used.

CO.Thorough The main research question was to see if we could get away with indexing only a relatively small number of elements. In our runs we compared three element indexes. The "normal" element index, the qrel-based element selection and the length-based element selection. We submitted three runs:

- *T-Elements* (UAmsCOTElementIndex) A run using a mixture model of the full element index and the article index. We set $\lambda_e = 0.05$, $\lambda_d = 0.1$, and used a normal length prior.
- *T-Qrel* (UAmsCOTQrelbasedIndex) A run using a mixture model of the qrel-based element index and the article index. We set $\lambda_e = 0.05$, $\lambda_d = 0.1$, and used a normal length prior.
- *T-Length* (UAmsCOTLengthbasedIndex) A run using a mixture model of the length-based element index and the article index. We set $\lambda_e = 0.05$, $\lambda_d = 0.1$, and used a normal length prior.

CO.FetchBrowse For the fetch and browse we mirrored the focused task submissions, but clustered the results so that elements within the same article appear together in the ranked list.

- *FB-Articles* (UAmsCOFBArticle) This run was exactly the same as the article run we submitted for the focused task.
- *FB-Elements* (UAmsCOFBElements) We took the focused element run and reordered the results in such a way that elements from the same document are clustered together. The document clusters are ordered by the highest scoring element within each document. We returned a maximum of 10 most relevant elements from each article.
- *FB-Sections* (UAmsCOFBSections) We took the focused section run and reordered the result set in such a way that the elements from the same document are clustered together. The document clusters are ordered by the highest scoring section within each document.

3.2 Content-Only with Structure Runs

For the CO+S task we experimented with three ways of using structural constraints.

Target-only For queries that have a CAS title we only return elements which satisfy the target constraint of the CAS title. For queries asking for sections, we accepted the equivalent tags as listed in the topic development guidelines. NB! We used the terms in the title field of the queries because we want a direct comparison to CO runs. Retrieval was performed using a mixture model using the overlapping element index and the normal document index.

Context-only We retrieved elements as described in Section 2.4.

Target and Context We retrieved elements as described in Section 2.4. Additionally we filtered out elements that did not match the target constraint.

+S.Focused

- *F-Target* (UAmsCOpSFocStrictTarget) A run created using a mixture model of the overlapping element index and the article index. We set $\lambda_e = 0.4$ and $\lambda_d = 0.4$. No length prior was used for this run. Target restriction was implemented for queries that had one. Overlap was removed in a list-based fashion
- *F-Context* (UAmsCOpSFocConstr) We applied the context-only approach, described above, on the focused CO element run (UAmsCOFocElements).
- *F-ContTarg* (UAmsCOpSFocConstrStrTarg) We applied the context-only approach on the strict on target run (UAmsCOpSFocStrictTarget).

+S.Thorough

- *T-Target* (UAmsCOpSTStrictTarget) A run created using a mixture model of the overlapping element index and the article index. We set $\lambda_e = 0.05$ and $\lambda_d = 0.1$. We applied a normal length prior. Target constraints were respected for queries that had one.
- *T-Context* (UAmsCOpSTConstr) We applied the context-only approach, described above, on the thorough CO element run (UAmsCOTElementIndex).
- *T-ContTarg* (UAmsCOpSTConstrStrTarg) We applied the context-only approach on the strict on target run (UAmsCOpSTStrictTarget).

+S.FetchBrowse

- *FB-Target* (UAmsCOpSFBStrictTarget) We reordered the focused strict on target run (UAmsCOpSFocStrictTarget) such that results from the same article were clustered together. Only the 10 most relevant elements were considered for each article.
- *FB-Context* (UAmsCOpSFBConstr) We reordered the focused run using constraints (UAmsCOpSFocConstr) such that results from the same article are clustered together. Only the 10 most relevant elements were considered for each article.
- *FB-ContTarg* (UAmsCOpSFBConstrStrTarg) We reordered the focused run using constraints and strict targets (UAmsCOpSFocConstrStrTarg) such that results from the same article are clustered together. Only the 10 most relevant elements were considered for each article.

4 Results

In this section we will present and discuss our results. The results are based on version 7 of the INEX 2005 assessments. The results were generated using version 1.0.3 of EvalJ. We will report our results using a limited number of metrics, compared to the plethora of metrics available as part of EvalJ. We will use two MAP-like metrics: MAnxCG@1500 (called MAnxCG from now on), and ep/gr (MAep). Three early-precision metrics will be used: nxCG@5, nxCG@10, and R-measure. Our assumption is that these metrics provide a representative subset of all the available metrics. We will report results using the generalized quantization. The INEX organizers have labeled the nxCG measures as the "the official" measures for user-oriented tasks and ep/gr measures as "the official" for system-oriented tasks.

We will report results in 4 subsections. First, we will look at index reduction experiments. Next, we will look at effects of manually adding structural constrains. Then, we will look at the effects of automatically created structural constraints. Finally, we will present our results for the FetchBrowse task.

Table 3. *CO.Focused* runs, using *generalized* quantization and overlap *on*.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
F-Elements	0.262	–	0.068	–	0.202	–	0.194	–	0.155	–
F-Sections	0.200	-24%	0.062	-8.8%	0.174	-14%	0.150	-23%	0.211	36%
F-Article	0.096	-63%	0.046	-32%	0.178	-12%	0.165	-15%	0.189	22%

Table 4. *CO.Thorough* runs, using *generalized* quantization and overlap *off*.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
T-Elements	0.301	–	0.082	–	0.266	–	0.257	–	0.262	–
T-Length	0.294	-2.3%	0.083	1.2%	0.268	0.8%	0.256	-0.4%	0.265	1.1%
T-Qrel	0.294	-2.3%	0.086	4.9%	0.280	5.3%	0.267	3.9%	0.269	2.7%

4.1 Index Reduction

For the INEX reduction we will look at two tasks: CO.Focused and CO. Thorough. For the CO.Focused task we compare a general element index to two reduced indices, section index and article index. For the CO.Thorough task we compare the same general element index to two reduced indices, length-based reduction and qrel-based reduction.

Results for the CO.Focused task can be seen in Table 3. We see that the full element retrieval approach improves over both index reduction methods, except in the case of R-measure.

Results for the CO.Thorough task can be seen in Table 4. There is very little difference between the three runs. This means that reducing the indexed elements from 10.6M elements to circa 1.5M (14-15%) did not affect the effectiveness of the retrieval. Table 1 shows, however, that the reduced indexes lead to improved efficiency.

4.2 Structural Constraints

In this section we will analyze the effect of adding structural constraints to queries. We will distinguish between two types of constraints: target-constraints and context-constraints. Target constraints restrict the target of the results to be of certain tag-type, e.g. "give me only sections". Context-constraints restrict the environment in which the result elements live, e.g. "give me results from articles that are authored by Moldovan".

We will report results for the CO(+S).Focused task. We will look at results for 4 runs: our 3 official CO+S.Focused runs and our CO.Focused baseline run (F-Elements). In this sub-section we will only look at the 19 (assessed) CO+S topics that had a structured version. The remaining 10 (assessed) CO+S topics will be discussed in the next sub-section.

Table 5. *CO(+S).Focused* runs, using *generalized* quantization and overlap *on*. Here, we evaluate only over the 19 queries having a `<castitle>`.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
F-Elements	0.289	–	0.074	–	0.219	–	0.211	–	0.139	–
F-Context	0.326	13%	0.086	16%	0.234	7%	0.213	1%	0.166	19%
F-Target	0.226	-22%	0.077	4%	0.253	16%	0.246	17%	0.204	47%
F-ContTarg	0.241	-17%	0.092	24%	0.260	19%	0.246	17%	0.228	64%

Table 6. *CO(+S).Focused* runs, using *generalized* quantization and overlap *on*. Here we evaluate only over the 10 queries that do *not* have a `<castitle>`.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
F-Elements	0.213	–	0.057	–	0.171	–	0.163	–	0.185	–
F-Context	0.230	8.0%	0.046	-19%	0.161	-5.8%	0.166	1.8%	0.155	-16%

Table 5 shows the evaluation results. First let's look at the effect of context-constraints. The context-constraint run (F-Context) performs considerably better than our CO baseline (F-Elements). The improvement is, however, negligible for the nxCG@5 and nxCG@10. Next let's look at the effect of target-constraints. We see that in terms of MAnxCG the performance of our target-constraint run (F-Target) is considerably worse than our CO baseline. In terms of nxCG@5, nxCG@10, and R-measure, there is a considerable performance improvement when we constrain the target. By using both context and target constraints we gain back some of the MAP score lost by enforcing the target constraint, while maintaining our early-precision improvement.

4.3 Automatic Structural Constraints

In this section we will look at the effects of automatically generating structural constraints. We could, in principle, have generated automatic structured queries for all the CO queries. However, in practice, we only did it for the topics that did not have a `<castitile>` representation. Hence the comparison will only be done over the limited number of topics. Since we only create context-constraints we will only compare our baseline CO (F-Elements) run and our F-Context run. The F-Target is in this case equivalent to F-Elements and F-ContTarg is equivalent to F-Context.

Table 6 shows the results of the evaluation. As with so many blind-feedback experiments the results are mixed. We will not analyze these results further here. It remains as future work to evaluate the structured blind-feedback over a greater number of queries, and compare it with normal content-based blind-feedback.

Table 7. *CO.FetchBrowse* runs, using *generalized* quantization and overlap *on*.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
FB-Elements	0.263	–	0.066	–	0.162	–	0.171	–	0.160	–
FB-Sections	0.207	-21%	0.052	-21%	0.130	-20%	0.119	-30%	0.194	21%
FB-Article	0.096	-63%	0.046	-30%	0.178	10%	0.165	-3.5%	0.189	18%

Table 8. *CO.Focused* and *CO.FetchBrowse* runs, using *generalized* quantization and overlap *on*.

	MAP-like precision				Early precision					
	MAnxCG		ep/gr (MAep)		nxCG@5		nxCG@10		R-measure	
F-Elements	0.262	–	0.068	–	0.202	–	0.194	–	0.155	–
FB-Elements	0.263	0.4%	0.066	-2.9%	0.162	-20%	0.171	-12%	0.160	3.2%
F-Sections	0.200	–	0.062	–	0.174	–	0.150	–	0.211	–
FB-Sections	0.207	3.5%	0.052	-16%	0.130	-25%	0.119	-21%	0.194	-8.1%

Table 9. *CO.Focused* and *CO.FetchBrowse* runs, using *generalized* quantization.

	MAP		P@5		P@10		R-prec	
FB-Articles	0.489	0%	0.690	0%	0.648	0%	0.481	0%
FB-Elements	0.441	-10%	0.655	-5.1%	0.635	-2.0%	0.465	-3.3%
FB-Sections	0.455	-7.0%	0.648	-6.1%	0.607	-6.3%	0.483	0.4%

4.4 FetchBrowse

Here we discuss our FetchBrowse results. We will evaluate the task in with respect to two different aspects. First, since our FetchBrowse runs are simply reordering of our Focused runs, we evaluate the FetchBrowse using the same metrics as the Focused task. Second, since the FetchBrowse is an extension of a document retrieval task, we will massage our runs into document retrieval runs and evaluate using `trec_eval`.

Table 7 shows the results of evaluating the FetchBrowse task as a Focused task. The results are quite similar to the results for the Focused task. That is, the element run outperforms the section and article runs, except for R-measure.

Let us take a closer look at the difference between our Focused and our FetchBrowse runs. Table 8 shows the difference between the two run-types. The results are quite similar except perhaps for nxCG@5 and nxCG@10. That is, the reordering we did to transform the Focused runs into FetchBrowse runs did not change our results much, when we look at the metrics for Focused.

Let us now look at the document retrieval quality of our FetchBrowse runs. Since the 'Fetch' part of the task refers to plain old document retrieval we evaluate it using the standard document retrieval metrics that come with the `trec_eval` package. We transform our runs and assessments to TREC format. An article is considered relevant in the TREC sense if it contains a relevant element. We use thus a rather lenient quantization. Results of our evaluation can be seen

in Table 9. We see that the two element-based runs are worse than the article run. In terms of mean average precision, FB-Elements is even significantly worse than FB-Articles (at .95 significance level).

5 Related Work

Here we will discuss some related work. The main goal with this section is to locate our work within the INEX community.

Language-Models for XML Retrieval A number of alternative language modeling approaches for XML retrieval have been used in INEX. Mihajlović et al. [7] use a standard multinomial language model [3] including a number of advanced features such as phrase modeling. Especially in the context of the relevance feedback task, they experimented with a range of priors, such as a length prior, an XML tag name prior, and a journal prior. Ogilvie and Callan [8] take a quite different approach. First, standard language for all text nodes are estimated. Second, language models for all elements are constructed by, bottom-up, repeatedly calculating a mixture language model of all child nodes.

Selective Indexing Various types of selective indexing schemes have been used in INEX. Gövert et predefined list of tag names. The list is compiled after careful analysis of tag name semantics. Mass et al. [6] and Clarke et al. [1] used existing relevance assessments to define the appropriate units for their index. Index reduction based on eliminating the very many very short elements has been used by many teams at INEX.

Structured constraints In this paper we have looked at the effectiveness of strict interpretation of target constraints and compared it to a baseline where target constraints are ignored. There is quite some room in between the two approaches. Liu et. al. [5] propose a few relaxations of target constraints, both based on path similarity and content similarity.

Structured Feedback Automatic generation of structured queries has been proposed previously by Mihajlović et al. [7]. They use true relevance feedback to expand queries with journal information and target constraints.

6 Conclusions

In this year's INEX we had three main research questions. We wanted to explore the effect of different types of structured constraints on retrieval effectiveness. We also wanted to see if we could use selective indexing to make our system more efficient without loosing retrieval effectiveness. And, third, we consider the automatic construction of structured queries using blind relevance feedback.

We showed that context-constraints and target-constraints have different effect on retrieval performance. The context constraints are helpful for improving

average precision. Interpreting the target constraints in a strict manner does hurt average precision, but do give considerable improvement if we look at early precision.

For the Focused task, we compared two selective indexes to our full element index: section index and article index. Retrieving sections and articles is more efficient than retrieving from the full element index. The effectiveness of section and article retrieval is, however, inferior to retrieval from the full element index.

For the Thorough task, we experimented with two different pruning of the full overlapping element index, using element length and past qrels as pruning criteria. Neither of the pruning strategies lead to a considerably lower average performance. Both pruning strategies were, however, more efficient than the full overlapping element index.

For the FetchBrowse task, it is difficult to draw any final conclusions since it is not clear how this task should be evaluated. For the document ranking part of the FetchBrowse task, ranking documents based on their own retrieval score outperformed the document retrieval based on the highest scoring of either elements or sections.

The results of our automatic generation of structured queries are inconclusive. Further experiments are needed to verify its (in)effectiveness.

Acknowledgments

This research was supported by the Netherlands Organization for Scientific Research (NWO) under project numbers 017.001.190, 220-80-001, 264-70-050, 354-20-005, 612-13-001, 612.000.106, 612.000.207, 612.066.302, 612.069.006, 640.001.-501, and 640.002.501.

References

1. C. L. Clarke and P. L. Tilker. MultiText experiments for INEX 2004. In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*, volume 3493 of *LNCS*, pages 85–87. Springer-Verlag GmbH, 2005.
2. E. A. Fox and J. A. Shaw. Combination of multiple searches. In *The Second Text REtrieval Conference (TREC-2)*, pages 243–252, 1994.
3. D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, 2001.
4. J. Kamps, M. de Rijke, and B. Sigurbjörnsson. The importance of length normalization for XML retrieval. *Information Retrieval*, 8:631–654, 2005.
5. S. Liu, R. Shahinian, and W. Chu. Vague content and structure (VCAS) retrieval over document-centric XML collections. In A. Doan, F. Neven, R. McCann, and G. J. Bex, editors, *Proceedings of the Eighth International Workshop on the Web and Databases (WebDB 2005)*, pages 79–84, 2005.
6. Y. Mass and M. Mandelbrod. Retrieving the most relevant XML components. In N. Fuhr, M. Lalmas, and S. Malik, editors, *INEX 2003 Workshop Proceedings*, pages 53–58, 2004.

7. V. Mihajlović, G. Ramírez, A. P. de Vries, D. Hiemstra, and H. E. Blok. TIJAH at INEX 2004 modeling phrases and relevance feedback. In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*, volume 3493 of *LNCS*, pages 276–291. Springer-Verlag GmbH, 2005.
8. P. Ogilvie and J. Callan. Hierarchical language models for XML component retrieval. In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*, volume 3493 of *LNCS*, pages 224–237. Springer-Verlag GmbH, 2005.
9. J. Ponte. Language models for relevance feedback. In W. Croft, editor, *Advances in Information Retrieval*, chapter 3, pages 73–96. Kluwer Academic Publishers, Boston, 2000.
10. B. Sigurbjörnsson, J. Kamps, and M. de Rijke. An Element-Based Approch to XML Retrieval. In *INEX 2003 Workshop Proceedings*, pages 19–26, 2004.
11. B. Sigurbjörnsson, J. Kamps, and M. de Rijke. Mixture models, overlap, and structural hints in XML element retrieval. In N. Fuhr, M. Lalmas, S. Malik, and Z. Szlávik, editors, *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004*, volume 3493 of *LNCS*, pages 196–210. Springer-Verlag GmbH, 2005.