# Improving Index Structures for Structured Document Retrieval

Holger Meuss and Christian M. Strohmaier

Center for Information and Language Processing, University of Munich

Germany

**Abstract**

Structured document retrieval has established itself as a new research area in the overlap between Database Systems and Information Retrieval. This work proposes a filtering technique, that can be added to already existing index structures of many structured document retrieval systems. This new technique takes the contextual structure information of query and document database into account and reduces the occurrence sets returned by the original index structure drastically. This improves the performance of query evaluation.

A measure is introduced that allows to quantify the added value of the proposed index structure. Based on this measure a heuristic is presented that allows to include only valuable context information in the index structure.

## 1 Introduction

With the growing importance of Information Retrieval in the presence of a vast amount of structured documents in formalisms like SGML ([6]) or the future WWW language XML ([18]), sophisticated and efficient indexing techniques for structured documents become more and more important. In general, index structures are crucial for the efficiency of Database Systems (DBS) and Information Retrieval (IR) systems. With an appropriate index structure irrelevant parts of the database can be disregarded in the search. Very sophisticated index structures have been proposed in the research in DBS and IR, some of them dedicated to a special class of data only, e.g. geographical data ([2]).

Index structures in DBS try to support access to data by organizing it in an appropriate way. The notion of ordering the data plays a key-role in this task. Some data has a natural topology (like geographical data), for other data the index structure defines a topology. One of the problems of an efficient index structure is to map this (usually multidimensional) topology onto the linear layout of the storage medium.

So far, index structures in IR are confronted with the one-dimensional form of the problem only: They implement a mapping from terms (i.e. words) in a set of documents to occurrences (i.e. offsets in the files storing the documents). The mapping problem becomes trivial, since text is seen in traditional IR as a linear medium.

In the last years, several formalisms for structured document retrieval have been proposed, a field that combines aspects of DBS and IR. (See [1] and [8] for surveys.) This raises the question of appropriate index structures for structured document retrieval. Only few formalisms did pay attention to this question (e.g. the Lore system: [10]). Most formalisms simply adopted techniques and definitions from index structures well established in IR, e.g. inverted files, or occurrences as offsets in a text file. But this approach neglects the special property of structured documents, that they have no linear topology. Structured documents are usually conceived as trees and have thus a super-linear topology.

This work postulates a marriage between indexing techniques from the field of DBS and IR, in the same way as structured document retrieval is conceived as a marriage between DBS and IR. It proposes as a first step an integration of additional structural information into the index structure. This can be seen as the invention of DBS indexing techniques for IR related systems, since now the text topology is taken into account in a nontrivial way.

The additional structural information to be integrated into the index structure consists of linear contexts associated with occurrences. With the help of these linear contexts and a simple automatic query preprocessing a filter can be applied to the results of index calls, thus reducing the size of the occurrence sets. With these reduced sets, the query evaluation algorithms will perform more efficiently.

The proposed filter can be integrated into many formalisms for structured document retrieval. Examples of the necessary changes for different systems are given. The benefits may vary from system to system and from application to application, but are likely to speed up query evaluation remarkably in overall.

This work is organized as follows. The next section introduces some formalisms and query evaluation techniques for structured document retrieval known to the literature. They are described with respect to aspects concerning the integration of the new index structure. Section 3 describes the context filter and how it is used in query evaluation. The following section shows how the added value of the context filter can be quantified by using the notion of “selectivity of labels”, how the selectivity can be approximated efficiently, and how this quantification can be used to reduce the size of the context filter. Section 5 shows for an example database how one of our approximations of selectivity is related with the actual reduction of the sets returned by the index structure in a real-world setting. Section 6 concludes with some directions for future work.

## 2 Models for Structured Document Retrieval

A lot of models for structured document databases have been proposed in the recent years. This section will introduce their features as far as they are relevant for the proposed filter. For a more comprehensive review of these models, see [1] and [8].

All these models have in common that they represent structured documents as directed graphs of nodes (in most cases trees). The leaf nodes contain the actual textual content of the documents. This is implemented by associating these nodes with regions in a file stor-

ing the textual (as opposed to structural) content of the document. Structural containment is represented by edges in the graphs. Nodes containing other nodes (e.g. chapters containing paragraphs) are associated with the union of all text regions of their children. Query evaluation is supported by various index structures, but most of the considered models provide at least two index structures:

- Text index: This index structure implements a mapping from search terms to occurrences in the document database. Every structured document retrieval model implements a text index.
- Structure index: This index structure implements a mapping from labels of structural elements to occurrences in the document database. This mapping is implement by most models.

For the proposed filter the distinction between these two index mappings is not relevant. Therefore we use the the term "search item" for every object (term or label) that can be input to an index structure.

For the description of the various formalisms we will refer to the following example query, cited here informally in natural language:

**Query 1:** "Give me all `title`s of `chapter`s reading *Java*."

The structured document retrieval models we consider here are distinguished by their query evaluation strategy and can be divided into three classes:

**Bottom-up query evaluation guided by the syntax tree:** This class comprises most of the models known to the literature, e.g. PAT Expressions ([16]), Overlapped Lists([3, 4]), Proximal Nodes ([14, 13]). These models have in common, that query evaluation is guided by the syntax tree of the query, i.e. an operator in the syntax tree is evaluated after all its children are evaluated. The resulting set of occurrences is then attached to that operator, and query execution can climb up further in the syntax tree. The leaves are evaluated by a text and or a structure index, respectively. In these models an occurrence is a region in a file containing the flat text of the documents. This region is specified by two offset values.

Query 1 is expressed in the Proximal Nodes syntax in the following way. (Queries in the two other mentioned formalisms look similar.)

(`title` **same** *Java*) **in** `chapter`

Its syntax tree is depicted in Figure 1.

Query 1 is evaluated in the following way: A set with all elements with the content *Java* is produced by an index call, as well as a set with all `title` elements. The two occurrence sets are attached to the respective leaves. Then they can be compared with a simple intersection operation being the operational equivalent for the **same**

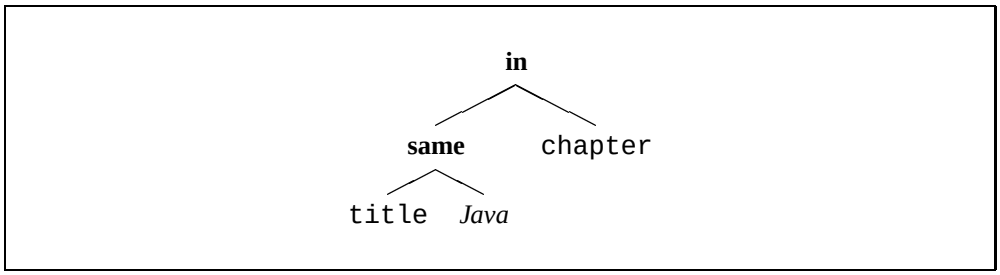


Figure 1: Syntax Tree of the Proximal Nodes Query

construct. The resulting set is attached to the **same** node. Next comes evaluation of the `chapter` leaf, finding (with the help of the index structure) all `chapter` nodes in the document database, and in the end, the sets attached to the `chapter` and to the **same** nodes are combined to the result set, that is attached to the **in** node and finally returned as answer.

**Bottom-up path-based query evaluation guided by the structural tree:**

This class consists of the Indexed DAG Matching formalism ([11, 12]), that is an extension of the Tree Matching formalism ([7]). Query evaluation is guided by the structural tree that reflects the search pattern, as in the example tree depicted in Figure 2 being the graphical formulation of Query 1.

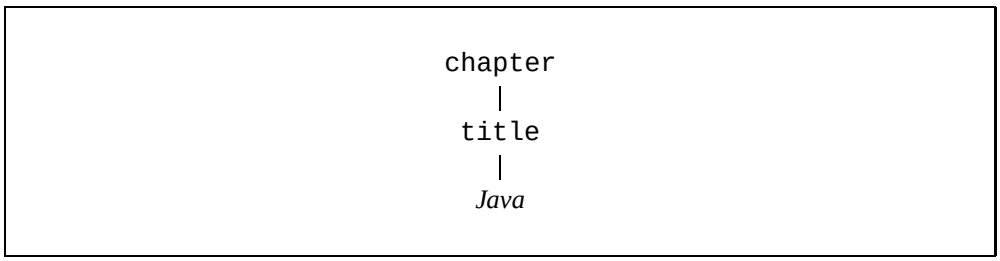


Figure 2: Indexed DAG Matching Query

For query evaluation the search terms in the leaves of the tree are evaluated using an index that maps the search terms to their occurrences. An occurrence is the path in the document database leading to the leaf containing the search term. The set of all occurrence paths is thereafter combined in a data structure derived from the query. Traversal of the (syntax or structural) tree in single steps is avoided by manipulating paths instead of single nodes. For the same reason, there is no need for a structure index in Indexed DAG Matching.

**Flexible query evaluation guided by the syntax tree:** This class consists of the Lore system ([9, 10]), a formalism for querying graph-structured data: Its sophisticated query evaluation mechanism supports various evaluation strategies (bottom-up, top-down, hybrid). The optimal query plan is chosen at run-time based on statistical information about the document database called DataGuides ([5]). Four index structures are maintained to support the various query evaluation techniques used. Every in-

dex structure maps search items (i.e. search terms, relational or path expressions) to occurrences. An occurrence is a node in the database.

Apart from a path index that maps node/path pairs to nodes reachable from the input node via the path, all these index structures can be reasonably enriched with contextual information. For the path index the additional storage of contextual information produces no better results.

# 3 The Context Filter

Efficiency of query evaluation algorithms depends heavily on the possibility to disregard irrelevant parts of the document database in the search. The earlier this is possible, the more efficient an algorithm can perform. This section proposes a method to enrich index structures with additional contextual information in a way that the result sets of calls to the index structure will be smaller. Occurrences not complying with the contextual conditions prescribed by the query are filtered out.

An index for text databases implements a mapping from search items to occurrences. Depending on the underlying text and computation model, the meaning of occurrence may vary: Most formalisms conceive occurrences as offsets in the file storing the flat content of the documents. In Indexed DAG Matching ([11, 12]) an occurrence is defined as a path in the document database, that ends in a leaf containing the search term. In Lore's ([9, 10]) index structures an occurrence is a node containing the respective search term.

The proposed modification consists of three components:

- As a preparation, the index structure is enriched with additional contextual information.
- In query evaluation, the query is preprocessed to find the linear contexts of search items.
- Then the result sets of calls to the index structure are filtered using the contexts generated in the previous step and the contextual information attached to the occurrences. This is depicted in Figure 3.

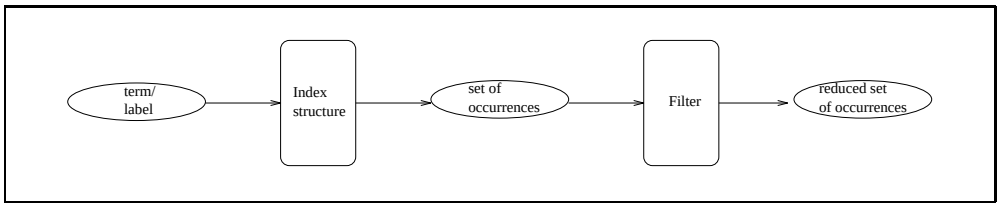


Figure 3: Integration of the Context Index

**Enriching the Occurrences:** In order to filter the set of occurrences, the index structure has to be equipped with the linear contexts of the occurrences. A linear context of an occurrence is just the set of all its ancestors in the document database. This information is stored in the condensed form of a bit string attached to every occurrence. All bit strings are of a fixed size, and every position in the bit strings corresponds to a label in the document database. A "1" indicates that the occurrence is in a region associated with that label, whereas a "0" indicates that this is not the case. The example in Figure 4 illustrates this. It is possible to integrate the additional context information in all index structures used by the various systems reviewed in Section 2.

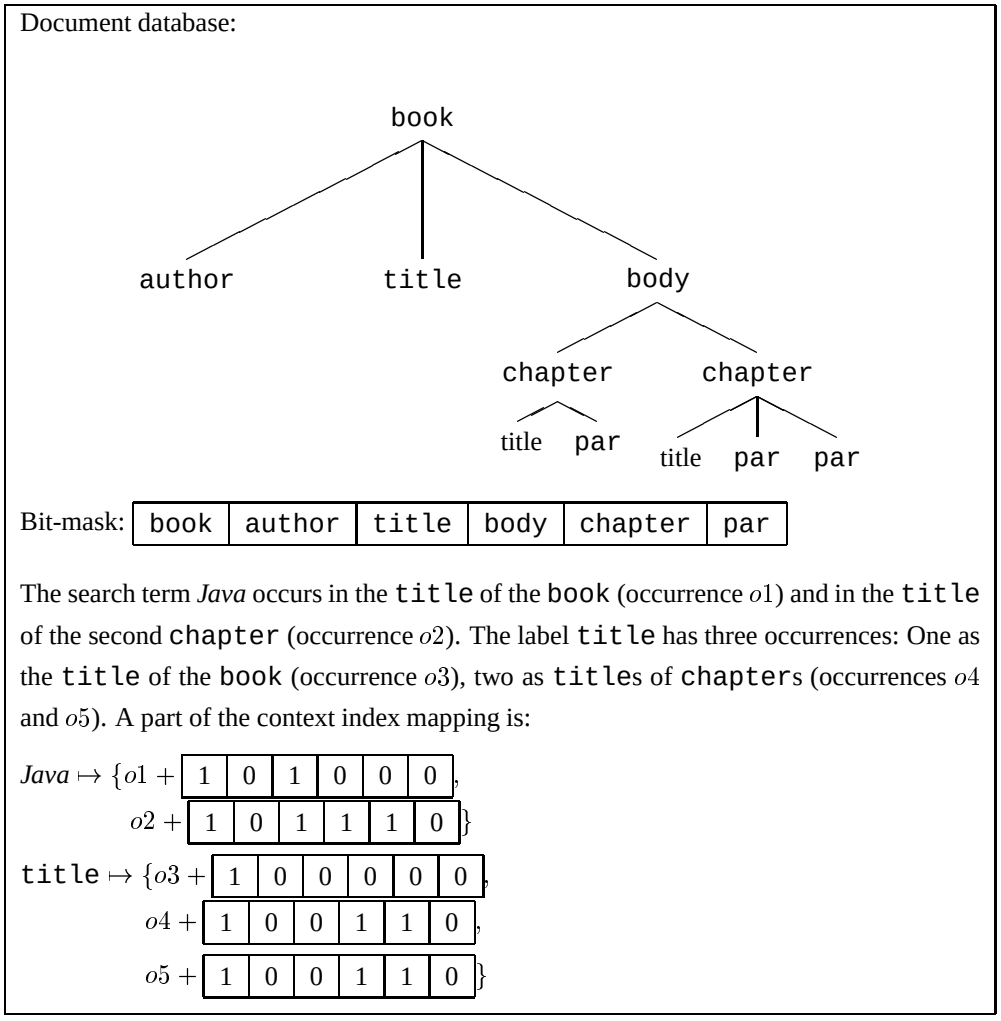


Figure 4: Linear Contexts of Occurrences

**Query Preprocessing:** The query is analyzed in order to find the linear context for every search item. The algorithm itself depends on the query syntax, and is therefore not elaborated here. These contexts are now encoded in the same way in bit strings as the contextual information attached to the occurrences.

For Query 1, the preprocessing computes the linear context {`chapter`,`title`} for the search item *architecture*, and the linear context {`chapter`} for the search item `title` (not used in Indexed DAG Matching). These contexts are encoded in the bit strings

| 0 | 0 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|

and

| 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|

.

**Filtering the Index Result Sets:** We suppose that the index structure returns sets of enriched occurrences, i.e. occurrences together with their linear contexts. Every call to an index structure in query evaluation is now processed in the following way: The linear context of every occurrence is compared to the linear context of the respective search item in the query that was computed in the preprocessing step. Occurrences not complying with the contextual conditions are filtered out and only the complying nodes are passed back to the query evaluation algorithm. Illustrated by an algorithm:

```
FORALL occurrences occ DO {
        FORALL labels lab DO {
                IF (queryRequiresContext(lab) ∧
                    ¬ contextPresent(occ,lab))
                THEN { drop(occ); }
        }
}
```

The algorithm is realized by an efficient bit comparison. It is just an ANDNOT operation between the respective search item's linear context and the occurrences' linear contexts. If this results in 0 (the bit string consisting of 0's only), the respective occurrence is kept, otherwise it is dropped. This can be implemented very efficiently, because the used operations are hardware-oriented. If a hardware does not support the ANDNOT operation, a technical variant is applicable: storing the complementary bit mask of every occurence and using the AND operation.

The reduced set of occurrences is now processed in the same way as the full set of occurrences would have been without the context filter. Again, integration of this filter into the respective query evaluation mechanism is possible in all the models reviewed for Section 2.

Consider Query 1 and the document database depicted in Figure 4. The index calls made during query evaluation depend on the formalism: Some formalisms call a structural index for finding occurrences of the structural elements `chapter` and `title`, while every formalism calls the text index for finding occurrences of the search term *structure*. The (maximally) three calls to the index structure have the following results:

- There are two `chapter` elements.
- There are three `title` elements ($o3$,$o4$ and $o5$).
- There are two occurrences of the search term *Java* ($o1$ and $o2$).

When examining the bit strings of the enriched occurrences returned by each call to the index, the filter can detect very fast that occurrence $o1$ of the search term *Java* is not inside a *chapter*. From the three *title* occurrences one can be removed as well.

Note that this filter treatment did not depend on the point of time or order in the algorithm the index calls were made. The important thing is, that every result of an index call runs through the filter before it is passed back to the algorithm.

It can be seen that the number of occurrences that has to be treated by the algorithm is reduced notably. Out of six occurrences two could be eliminated. This percentage will increase with the size of the document database in question, since there is a bigger variety in label occurrences.

# 4 Reduction of Space Requirements

The additional information associated with every occurrence is stored in a bit string, whose length depends on the number of labels in the database. In order to reduce the length of these bit strings, in this section we take a closer look at the labels, and decide which are worth to be stored in a filter and which are not. We propose two methods for scoring the labels. They are based on:

- the notion of "selectivity", i.e. a measure for the filtering power of each label in the context filter and
- manual, application-specific judgement of the labels.

The decision which labels are disregarded in the context filter has to be taken at the time of database definition (i.e. index creation).

In the this section we only discuss the context filter for the text index, but a transfer to the structure index easily can be done.

## 4.1 Selectivity

This section motivates and introduces the notion of selectivity informally, before defining it in a mathematical way, and then elaborating on efficient approximative computations.

We define the selectivity $\varsigma(l)$ of a label $l$ as the average proportion between "recall noise" and "total recall".

$$\varsigma(l) = \frac{recall\_noise(l)}{total\_recall}$$

Total recall is the number of all occurrences returned by an index call, and recall noise is the number of those occurrences outside the scope of the label $l$.

The lower the selectivity of a label, the lower the benefit of integrating that label into our context filter. Note that most classes of structured documents have a unique root label (e.g. the `HTML` tag in web pages), which covers every term occurrence. The selectivity of that label is exactly 0 and the benefit for our filter, too.

In order to reduce the size of the bit string stored with every occurrence, only those labels are incorporated, whose selectivity is greater than a given threshold. We defer the determination of that threshold to further experimental work.

As elaborated in the following, the selectivity of a label is influenced by the

- coverage of that label, and
- inter-dependence of that label with others.

Obviously, the smaller text regions are that are covered by a given label, the more selective that label is. And also, the more independent labels are from each other, i.e. the more each label tends to have its own vocabulary, the less selective they are. Inter-dependence of labels is mainly caused by element nesting or redundancy in natural language. The causality between selectivity and inter-dependence becomes clear in the following example.

Regard a Franco-phone IR system of local conference papers in French with additional English abstracts marked with `rés_angl`. Querying English terms will produce a recall set of occurrences which are almost inevitably covered by the label `rés_angl`, because just a few terms occur in both languages, like *sale*. In the terms defined above this means a low recall noise of the label `rés_angl`, resulting in a low selectivity of that label. Querying the French abstracts (`rés_franç`) with French terms is different, since the text index call returns occurences which are covered by `rés_franç` as well as occurences which are not. Therefore a filtering step is useful for that label. Notice that French and English abstracts have roughly the same size but different selectivity. Hence coverage and inter-dependence are orthogonal influences on the selectivity.

Now we will present the exact definition of selectivity. The basic idea in it is to consider every potential recall set for every label. The fraction between the recall noise and total recall is computed for every search term. The unweighted average of all these fractions is the selectivity of that label.

We distinguish term occurrences $o$ and terms $\bar{o}$.[1] There is a mapping from term occurrences to their equivalence class, $o \mapsto \bar{o}$. We describe the relation "occurrence $o$ is in the scope of the label $l$" by $o \in l$.

If we write $DB$ instead of $l$, the range is the whole database, e.g. $\{o \mid o \in DB\}$ is the set of all term occurrences in the database.

**Definition 4.1** *Let $T = \{\bar{o_1}, \ldots, \bar{o_n}\}$ be the set of different terms in the database. The* selectivity $\varsigma(l)$ *of a label $l$ is computed with the formula:*

$$\varsigma(l) = \frac{1}{n} \sum_{j=1}^{n} \frac{\left|\{o \mid o \mapsto \bar{o}_j \wedge o \notin l\}\right|}{\left|\{o \mid o \mapsto \bar{o}_j \wedge o \in DB\}\right|}$$

Two problems are encountered, if the actual selectivity is computed on the basis of that formula:

[1] We conceive terms as equivalence classes of term occurrences.

- Obviously it cannot be computed efficiently.
- Insert and delete operations on the database may alter the exact results.

Hence we present two estimations in the following.

#### *4.1.1 Random Sample of Terms*

Every addend in the computation of $\varsigma(l)$ corresponds to the contribution of one term $\bar{o}$ to the selectivity of the label $l$. Under the assumptions that the selectivity of all terms in $T$ is normally distributed, the selectivity can be estimated with a random sample.

**Definition 4.2** *Let $S \subset T$ be a random sample of $T$ with a default size $k = |S|$, $k \ll n$. The estimated selectivity $\hat{\varsigma}(l)$ of a label $l$ is computed with the formula:*

$$\hat{\varsigma}(l) = \frac{1}{k} \sum_{j=1}^{k} \frac{\left|\{o \mid o \mapsto \bar{o}_{i_j} \wedge o \notin l\}\right|}{\left|\{o \mid o \mapsto \bar{o}_{i_j} \wedge o \in DB\}\right|}$$

This is an enormous reduction of the complexity of computation, because the size of $k$ is under control, it is independent of the size of the database.

#### *4.1.2 Coverage*

With a further simplifying assumption, the effort of estimating the selectivity of all labels can be reduced to a complexity linear in the number of term occurrences in the database. Assume that all terms are equally distributed over the documents and their elements, i.e. the influence of label inter-dependences is neglected, only the coverage is considered.

**Definition 4.3** *The* coverage $\gamma(l)$ *of a label $l$ is computed with the formula:*

$$\gamma(l) = \frac{\left|\{o \mid o \in l\}\right|}{\left|\{o \mid o \in DB\}\right|}$$

**Definition 4.4** *An estimation $\hat{\varsigma}'(l)$ for the selectivity of a label $l$ easily can be derived from its coverage:*

$$\hat{\varsigma}'(l) = 1 - \gamma(l)$$

If the effort for computing the coverage is still judged too high, it could be approximated by only considering a random sample of documents.

### 4.2 Application Specific Review of the Labels

One can observe that some labels tend to be specified very often and some tend to be specified never in a query. Figuring out which labels are important for users can be done by user questioning or logging of user queries.

# 5 A Case Study

This section presents a case study on the benefits of the context filter and its quantification by the notion of selectivity, approximated by the notion of coverage. The document database in question is a set of automatically generated SGML documents. These documents were generated from HTML input describing university departments in Germany. The tool used for the information extraction was developed in the UNICO project ([17, 15]).

The document database consists of 319 short documents of 1 Mbyte in total. The documents contain more than 70000 term occurrences and are structured by 34 elements (see Appendix A for the document statistics).

**Approximation of Selectivity by Coverage**

We consider example queries that search for two different terms (*asthma* and *filter*) in four different contexts (`unico`, `schlagwort`, `ausstattung` and `publikation`; for a translation and interpretation of the element names see Appendix A). Table 1 compares the occurrences of the two search terms with the coverage values of the labels. The first column reflects the coverage of the respective label, while the second (fourth) column states the absolute number of the occurrences of the term *asthma* (*filter*) in the context of the respective label and its fraction among all occurrences of *asthma* (*filter*). As can be seen in the first row, the (unmodified) index structure returns 15 occurrences for the search term *asthma* and 18 occurrences for *filter*.

The result shows that the coverage gives some approximate value for the reduction of the occurrence sets. We want to point out that query evaluation is drastically accelerated by reducing the occurrence sets in most cases to 10% of their original size.

| | Coverage | Occurrences of *asthma* | | Occurrences of *filter* | |
|---|---|---|---|---|---|
| | | absolute | relative | absolute | relative |
| `unico` | 100% | 15 | 100% | 18 | 100% |
| `schlagwort` | 7.76% | 6 | 40% | 3 | 16.67% |
| `ausstattung` | 8.65% | 0 | 0% | 1 | 5.56% |
| `publikation` | 1.57% | 1 | 6.67% | 1 | 5.56% |

Table 1: Coverage and Actual Occurrences

**Space Reduction**

From the statistics in Appendix A we can conclude that all documents have the toplevel element `unico`, since it has a coverage of 100%. It is trivial to point out that this element can be neglected in the bit string. If we assume a threshold of 15% for the coverage (i.e. selctivity of 85%), beyond which labels are not represented in the bit string, we can drop another 5 labels. A manual analysis of the elements and the underlying DTD by their semantic value can reduce the number of labels to be represented in the bit string down to 15.

This means that each occurrence is attached with a bit string of length 15, what is a reasonable space increase for the index structure, compared to the time savings. For a size of 20 Bytes per index entry, this means a 10% increase in the size.

# 6 Conclusion

This work presented a filtering technique for structured document retrieval that takes contextual information into account. It showed that this filtering technique can be integrated into many existing systems for structured document retrieval with little effort. It also presented a measure to quantify the added value of the context index. This measure enables a heuristic to reduce the size of the additional contextual information to be stored in the index structure by maintaining only valuable information.

It is planned to integrate this filter into existing systems and evaluate the resulting prototypes. Other plans include a more close look upon the co-occurrences of labels in queries and documents and how to exploit this information to further reduce the size of the context index. Information inherent in grammars describing the documents' structure could also be used. Another promising direction is the extension of the concept of coverage to structural elements, i.e. to count the number of elements governed by a label instead of the number of words.

# References

[1] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *SIGMOD Record*, 25(1):67–79, 1996.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. R-tree. an efficient and robust access method for points and rectangles. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2):322–331, June 1990.

[3] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, 1995.

[4] C. L. A. Clarke, G. V. Cormack, and F. J. Burkowski. Schema-independent retrieval from heterogenous structured text. In *Proc. Fourth Annual Symposium on Document Analysis and Information Retrieval*, pages 279–290, 1995.

[5] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97*, pages 436–445, 1997.

[6] ISO. *Information Processing - Text and Office Systems - Standard General MarkUp Language (SGML)*. ISO8879, 1986.

[7] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.

[8] A. Loeffen. Text databases: A survey of text models and systems. *SIGMOD Record*, 23(1):97–106, March 1994.

[9] J. McHugh and J. Widom. Query optimization for semistructured data. Technical report, Stanford University, Computer Science Department, 1997.

[10] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajamaran. Indexing semistructured data. Technical report, Stanford University, Computer Science Department, 1998.

[11] H. Meuss. Indexed tree matching with complete answer representations. In *Proc. Fourth Workshop on Principles of Digital Document Processing (PODDP'98)*, 1998.

[12] H. Meuss and K. Schulz. DAG matching techniques for Information Retrieval on structured documents. Technical Report 98-112, CIS, University of Munich, 1998.

[13] G. Navarro. A language for queries on structure and contents of textual databases. Master's thesis, Dept. of Computer Science, University of Chile, 1995.

[14] G. Navarro and R. Baeza-Yates. Proximal Nodes: A model to query document databases by contents and structure. *ACM Transactions on Information Systems*, 15(4):400–435, 1997.

[15] R. Reiner. Ein Blackboard-System zur Informationsextraktion aus semistrukturierten Daten für die UNICO-Datenbank. Master's thesis, University of Munich, 1998.

[16] A. Salminen and F. W. Tompa. PAT expressions: an algebra for text search. *Acta Linguistica Hungarica*, 41(1-4):277–306, 1994.

[17] C. Strohmaier. UNICO, eine OMNIS-Datenbank zur Hochschulkooperation mit der Industrie. Master's thesis, University of Munich, 1997.

[18] World Wide Web Consortium: Extensible Markup Language (XML) 1.0. W3C Recommendation, February 1998. `http://http://www.w3.org/TR/REC-xml`.

# A The Example Database

```
319 Documents

Total number of occurrences:   73794

 Element  sup:                0.01% ~   10
 Element  mutter:             0.04% ~   26
 Element  sub:                0.04% ~   28
 Element  u:                  0.09% ~   63
 Element  netzadresse:        0.09% ~   68
 Element  dauer:              0.10% ~   76
 Element  b:                  0.20% ~   148
 Element  institut:           0.40% ~   297
 Element  schluessel:         0.43% ~   319
 Element  erstellt:           1.30% ~   957
 Element  erfasst:            1.30% ~   957
 Element  dt:                 1.52% ~   1124
 Element  publikation:        1.57% ~   1160
 Element  ol:                 1.70% ~   1254
 Element  kind:               1.79% ~   1323
 Element  td:                 2.31% ~   1707
 Element  tr:                 2.71% ~   1997
 Element  table:              2.81% ~   2075
 Element  name:               4.83% ~   3561
 Element  kooperation:        7.53% ~   5558
 Element  projekt:            7.76% ~   5723
 Element  schlagwort:         7.76% ~   5729
 Element  ausstattung:        8.65% ~   6381
 Element  i:                  9.25% ~   6828
 Element  kontakt:            10.39% ~  7667
 Element  dd:                 11.82% ~  8722
 Element  person:             12.87% ~  9499
 Element  dl:                 13.34% ~  9846
 Element  ul:                 18.29% ~  13496
 Element  li:                 19.77% ~  14587
 Element  p:                  34.70% ~  25606
 Element  forschung:          40.95% ~  30216
 Element  institution:        92.24% ~  68071
 Element  unico:              100.00% ~ 73794

Total number of elements: 34
```

The documents and markup are in German, therefore we translate the some element names:

| | |
|---|---|
| `unico`: | root element for all documents |
| `institut`: | institute |
| `publikation`: | list of publications |
| `schlagwort`: | keywords |
| `ausstattung`: | equipment |
| `forschung`: | research |