A Filter for Structured Document Retrieval[*]

Christian Strohmaier Holger Meuss
Center for Information and Language Processing
University of Munich
{strohmai,meuss}@cis.uni-muenchen.de

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 architectures 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 handed from the index structure to the query evaluation algorithm drastically by selecting only occurrences appearing in the right context, and thus decreases computational effort in query evaluation.

With the notion of "selectivity" we introduce a measure for the added value of the filtering technique. Based on this notion, new techniques are proposed in order to reduce space requirements for the additional information necessary for the filtering process. One technique utilizes varying patterns of labels in the document database for compressing the information. The other technique uses grammars describing the document structure for finding labels with similar properties. Out of a pair of those labels, only information about one has to be included 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 ([ISO86]) or the future WWW language XML ([W3C98b]), sophisticated and efficient indexing techniques for structured documents become more and more important. A recent W3C workshop dedicated to XML query languages ([W3C98a]) illustrates the urgent need for efficient query mechanisms on structured documents.

Indexing and filtering techniques are crucial for the efficiency of Database Systems (DBS) and Information Retrieval (IR) systems. With an appropriate index or filter 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 ([BKSS90]).

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 keyrole in this task. Some data has a natural topology (like geographical data), whereas for other data the index structure defines a topology. One of the problems of an efficient index

[*]This work was partially funded by Siemens AG.

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 [BYN96] and [Loe94] for surveys.) The IR view derives from conceiving a document as a sequence of words, whereas the DBS aspect interprets the tree structure of a document as nesting of information containers. This hybrid role 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: [MAG$^+$97]). Most formalisms simply adopted techniques and definitions for index structures well established in IR, e.g. inverted files, or occurrences as offsets in a text file. But that neglects the outstanding peculiarity of structured documents, namely of having 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 document 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.

In addition, we provide mathematical tools based on the notion of "selectivity" that help the database administrator to decide which structural information should be integrated and thus find an optimal solution for the space/speed trade-off.

This work is organized as follows. The next section describes the field of structured document retrieval by discussing its general task and peculiarities and reviewing some formalisms known in the literature. Section 3 introduces 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 and how selectivity can be approximated efficiently. Section 5 introduces new techniques to reduce the size of the context filter. Section 6 shows for an example database how the techniques proposed here behave for a real-world document database. Section 7 concludes with some directions for future work.

2 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. Our work refers to the models reviewed in [BYN96] and [Loe94], and in addition to [Meu98], many of the new XML query languages [W3C98a], and the Lore system ([MAG+97]).

All these models represent structured documents as labeled, directed graphs (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 storing the textual 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. In order to structure the text, every node is assigned a label, e.g. `chapter` or `author`.

Structured document retrieval systems provide formalisms to query a document database, to evaluate the query and present the answer to the user. The query may specify textual and structural aspects of the documents. Query evaluation is supported by various index structures, but most of the considered models provide at least two classes of 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, because our proposed filter technique is applicable analogously to both of them. Therefore we will use the word "search term" to refer to both terms and labels in the following.

We will use the following abstract view upon the various query evaluation systems mentioned in the beginning of this section: during query evaluation, for every search term in the query an index structure is called by the query evaluation algorithm. This index structure is given a search term and returns a set of occurrences of the search term in the document database. The actual representation of these occurrences may differ from system to system; some systems represent occurrences as offsets in text files, others as paths in the document structure. We make no distinction on this point and conceive index structures as a mapping from search terms to occurrence sets, independent from the actual representation of an occurrence. These occurrences are manipulated and combined by the query evaluation algorithm. Note that the size of the returned occurrence sets determines the efficiency of the query evaluation.

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 `chapter`s whose `title` reads *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 ([ST94]), Overlapped Lists([CCB95a, CCB95b]), Proximal Nodes ([NBY97, Nav95]). 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 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.)

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

Its syntax tree is depicted in Figure 1.

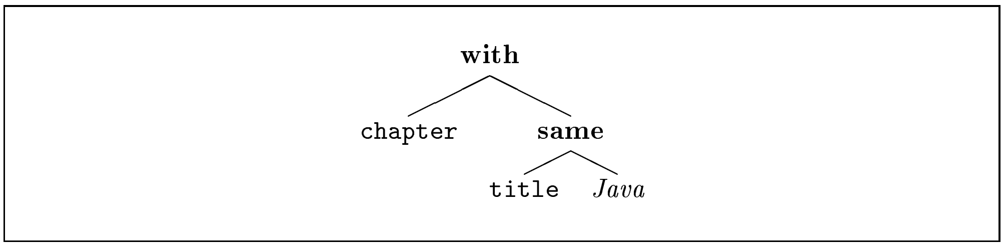


Figure 1: Syntax Tree of the Proximal Nodes Query

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** operator. 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 **with** node and finally returned as answer.

Bottom-up path-based query evaluation guided by the structural tree: This class consists of the Indexed Tree Matching formalism ([Meu98, MS98]), that is an extension of the Tree Matching formalism ([Kil92]). 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.

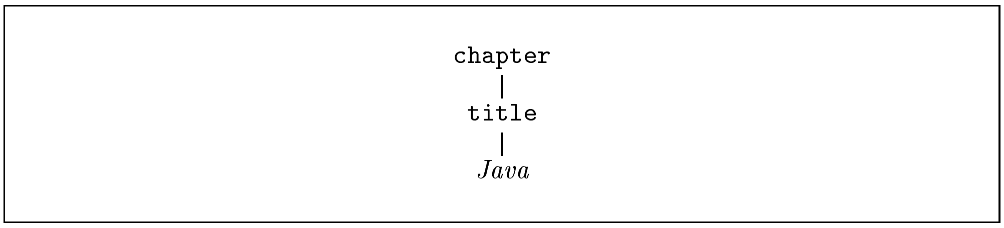


Figure 2: Indexed Tree Matching Query

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 Tree Matching.

Flexible query evaluation guided by the syntax tree: This class consists of the Lore system ([MW97, MWA$^+$98]), a formalism for querying graph-structured data: Its sophisticated query evaluation mechanism supports various evaluation strategies (bottom-up, top-down, hybrid). A query plan is optimized at run-time based on statistical information about the document database called DataGuides ([GW97]). Four index structures are maintained to support the various query evaluation techniques used. Every index structure maps search terms (i.e. 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

Disregarding irrelevant parts of the document database is a main strategy for query evaluation algorithms of database and IR systems. The earlier and more effortless this is possible, the more efficient an algorithm can perform. Index structures support query evaluation in this task, since they avoid scanning the complete database in order to find objects. This section describes a method to enrich index structures with additional contextual information in a way that the result sets of calls to the index structure can be drastically reduced with a computationally cheap test. Occurrences not complying with the contextual conditions prescribed by the query are filtered out. The query evaluation algorithm will only treat the reduced occurrence sets and will thus perform better.

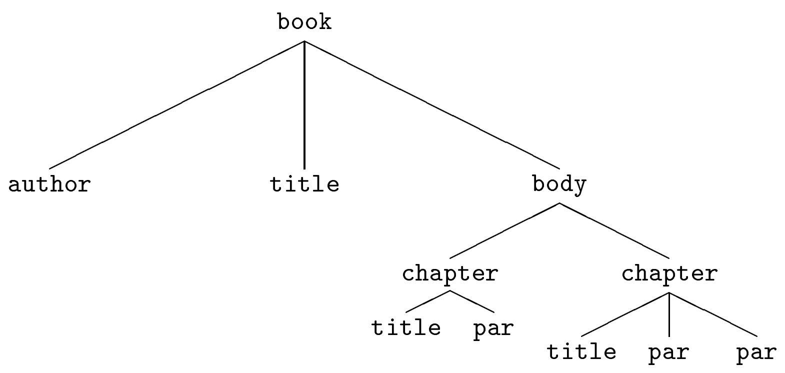
In the following we use the term *linear context* of an occurrence to denote the set of all ancestor labels containing that occurrence in the document tree.

The proposed modification to the host architecture consists of three components:

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. This is done at the time of index generation. The contextual 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 3 illustrates this. It is possible to integrate the additional context information into all index architectures used by the systems mentioned in Section 2.

Query Preprocessing: The query is analyzed in order to find the linear context for every search term, i.e. for every search term a set of those labels is computed that “include” the search term as contexts. 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.

Document database:



Bit-mask: $\begin{array}{|c|c|c|c|c|c|}\hline \texttt{book} & \texttt{author} & \texttt{title} & \texttt{body} & \texttt{chapter} & \texttt{par}\\\hline\end{array}$

The search term *Java* occurs in the `title` of the `book` (occurrence $o1$) and in the `title` of the second `chapter` (occurrence $o2$). The label `title` has three occurrences: One as the `title` of the `book` (occurrence $o3$), two as `title`s of `chapter`s (occurrences $o4$ and $o5$). A part of the enriched index mapping is:

$$\begin{aligned} \mathit{Java} \mapsto \{o1 + & \begin{array}{|c|c|c|c|c|c|}\hline 1&0&1&0&0&0\\\hline\end{array},\\ o2 + & \begin{array}{|c|c|c|c|c|c|}\hline 1&0&1&1&1&0\\\hline\end{array}\} \end{aligned}$$

$$\begin{aligned} \texttt{title} \mapsto \{o3 + & \begin{array}{|c|c|c|c|c|c|}\hline 1&0&0&0&0&0\\\hline\end{array},\\ o4 + & \begin{array}{|c|c|c|c|c|c|}\hline 1&0&0&1&1&0\\\hline\end{array},\\ o5 + & \begin{array}{|c|c|c|c|c|c|}\hline 1&0&0&1&1&0\\\hline\end{array}\} \end{aligned}$$

Figure 3: Linear contexts of occurrences

Since this preprocessing step depends on the size of the query only and can be done very efficiently, it increases the overall time complexity of query evaluation with a neglectable addend dependent on the query size only.

For Query 1, the preprocessing computes the linear context {`chapter`,`title`} for the search term *Java*, and the linear context {`chapter`} for the search term `title`. These two linear contexts are encoded in the bit strings $\begin{array}{|c|c|c|c|c|c|}\hline 0&0&1&0&1&0\\\hline\end{array}$ and $\begin{array}{|c|c|c|c|c|c|}\hline 0&0&0&0&1&0\\\hline\end{array}$ in the same way as the occurrences in the document database in Figure 3.

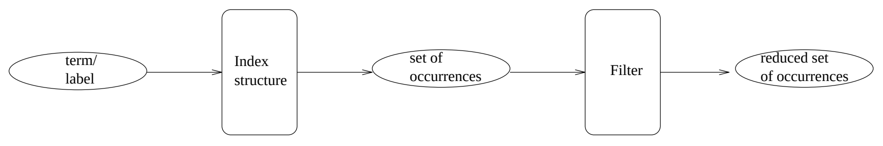


Figure 4: Integration of the context filter

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 (see also Figure 4): The linear context of every occurrence is compared to the linear context of the respective search term 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.

This filtering of occurrences is realized by an efficient bit comparison, an ANDNOT[1] operation between the respective search term'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, since then all labels specified in the linear context of the search term do in fact occur in the linear context of the occurrence. Otherwise the occurrence is dropped. This can be implemented very efficiently, because the used operations are hardware-oriented.

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 mentioned in Section 2.

Consider for example Query 1 and the document database depicted in Figure 3. The 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 we do not interfere with the host query evaluation mechanism in any way, since the filtering procedure does not depend on the point of time the index calls are made. The important thing is that every result of an index call runs through the filter before it is passed back to the host algorithm.

4 Selectivity

In this section we will define a measure for the efficiency of the context filter. The "selectivity" of a label measures the number of occurrences in the document database that can be rejected by the context filter. The higher the selectivity of a label is, the more occurrences can be filtered out, if that label is specified as a context in the query. In addition we will show how the selectivity of labels can be approximated with reasonable computational effort. As we will see in the next section, a selectivity analysis of labels can also be used in order to minimize the space requirements for the occurrence bit strings stored in the index structure.

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

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

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.

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

[1]ANDNOT executes a bitwise AND on the first and the negated second operator.

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

Obviously, the smaller the 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 is illustrated 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. This informal reasoning will have its formal reflection in Theorem 4.1.

Next 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 $\bar{o}$.[2] The exact definition of what a term actually is, together with its sophisticated, but tideous accompanying problems is left to the host formalism. We describe the relation "occurrence o is in the scope of the label l" by $o \in l$. The proposition $o \in DB$ (with the reserved symbol DB) shall be true for all occurrences o in the document database database. This symbol DB can be conceived as a unique label of an artificial root node that is an ancestor of all nodes in the document 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:*

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

The selectivity can take values in the interval $[0,1]$. If $\varsigma(l) = 0$, then l is a label covering all term occurrences of the document. If the document collection has a root element with label l_r, then $\varsigma(l_r) = 0$, e.g. the `XML` tag framing every XML document.
If $\varsigma(l) = 1$, then l is a label covering no term occurrences at all, e.g. a bachelor tag (in the XML terminology).

The formula shows that the computation of selectivity is a high computational effort. Therefore we provide with the notion of coverage a means to approximate the selectivity:

[2] We conceive terms as equivalence classes of term occurrences, i.e. a term is the set $\bar{o}$ containing exactly the occurrences of the term. If o is an occurrence of term $\bar{o}$, we write $o \in \bar{o}$.

Coverage

When neglecting the effects of "differing vocabularies" in labels, we can approximate the selectivity of a label by its size, i.e. by the number of occurrences in the document database under that label. This is captured by the notion of coverage. The relation between coverage and selectivity will be stated in Theorem 4.1.

Definition 4.2 *The* coverage $\gamma(l)$ *of a label l is computed with the formula:*

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

The next definition captures the notion of label independent terms, i.e. terms, whose relative number of occurrences is equal for all labels. This notion provides a basis for a formal reasoning on the orthogonal influences of label inter-dependence and coverage.

Definition 4.3 *A term $\bar{o}$ is called* label independent*, if*

$$\frac{\left|\{o \mid o \in \bar{o} \wedge o \in l\}\right|}{\left|\{o \mid o \in l\}\right|} = \frac{\left|\{o \mid o \in \bar{o} \wedge o \in DB\}\right|}{\left|\{o \mid o \in DB\}\right|}$$

holds for all labels l.

The following theorem shows that coverage is a good approximation for selectivity if we neglect the effects of label dependent terms. Obviously this assumption does not hold in a real-world document database, but the results in our case study (see Section 6) and their discussion show that coverage is a realistic approximation for selectivity.

Theorem 4.1 *Let $\hat{\varsigma}(l)$ be an estimation for the selectivity $\varsigma(l)$ of a label l computed with*

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

If all terms are label dependent, this approximation is exactly the selectivity of a label:

$$\hat{\varsigma}(l) = \varsigma(l)$$

Proof: Let all terms be label independent. Then we can define the size of a term as

$$\text{size}(\bar{o}) = \frac{\left|\{o \mid o \in \bar{o} \wedge o \in DB\}\right|}{\left|\{o \mid o \in DB\}\right|}.$$

With this definition, the following equations hold:

$$\begin{aligned}
\varsigma(l) &= \frac{1}{n}\sum_{j=1}^{n} \frac{\left|\{o \mid o \in \bar{o}_j \wedge o \notin l\}\right|}{\left|\{o \mid o \in \bar{o}_j \wedge o \in DB\}\right|} \\
&= 1 - \frac{1}{n}\sum_{j=1}^{n} \frac{\left|\{o \mid o \in \bar{o}_j \wedge o \in l\}\right|}{\left|\{o \mid o \in \bar{o}_j \wedge o \in DB\}\right|} \\
&= 1 - \frac{1}{n}\sum_{j=1}^{n} \frac{\text{size}(\bar{o}_j) \cdot \left|\{o \mid o \in l\}\right|}{\left|\{o \mid o \in \bar{o}_j \wedge o \in DB\}\right|} \\
&= 1 - \frac{1}{n}\sum_{j=1}^{n} \frac{\left|\{o \mid o \in \bar{o}_j \wedge o \in DB\}\right| \cdot \left|\{o \mid o \in l\}\right|}{\left|\{o \mid o \in DB\}\right| \cdot \left|\{o \mid o \in \bar{o}_j \wedge o \in DB\}\right|}
\end{aligned}$$

$$
\begin{aligned}
&= 1 - \frac{1}{n}\sum_{j=1}^{n} \frac{\left|\{o \mid o \in l\}\right|}{\left|\{o \mid o \in DB\}\right|} \\
&= 1 - \frac{1}{n} \cdot n \cdot \frac{\left|\{o \mid o \in l\}\right|}{\left|\{o \mid o \in DB\}\right|} \\
&= 1 - \gamma(l)
\end{aligned}
$$

For the the second equality we used the fact, that $\{o \mid o \in \bar{o}_j\}$ form a partition for the set $\{o \mid o \in DB\}$ of all occurrences. The next equation follows from equal distribution of terms and the rest is pure arithmetics. □

5 Reduction of Space Requirements

The additional information associated with every occurrence is stored in a bit string, whose length depends on the number of different labels in the database. The objective of this section is to present techniques that reduce the length of these bit strings. We propose a dynamic method that utilizes the varying coverage of labels in order to shorten the bit strings, and two static methods, that provide the database administrator with means to decide which labels can be excluded from the occurrence bit strings.

5.1 Dynamic Bit String Compression

The space reduction technique we present in this section is based on three observations. After describing these observations and their causality, we will discuss two methods of compressing the bit strings in order to achieve a smaller total length of the occurrence bit strings. All examples given in the following discussion on the observations and their background refer to the example document database introduced in Section 3.

Peculiarities of the Bit String Distribution
Observation 1: The actual number of different bit strings generated by encoding linear contexts of occurrences is much smaller than the number of bit strings of a fixed size.

The reasons for observation 1 are:

Grammatical: In most cases, a grammar restricts occurrence bit strings to valid paths. For instance, a possible grammar for our example database could define `author` and `title` as atomic elements (i.e. leaves), consisting of flat text only. Obviously an occurrence can not be in the context of two different atomic elements at the same time.

Semantical: Even if no grammar exists or the grammar is not strict enough, the inherent semantics of the document description will restrict occurrence bit strings to a given set of patterns. An `author` element for instance will normally have no `date` sub-element.

Empirical: Some meaningful and grammatically correct occurrence bit strings fail to appear in a specific database merely by chance. The example database could for instance contain `books` whose `chapters` have different `authors`, represented by `author` nodes for `chapters`. If, by chance, only `books` by single `authors` are entered into the database, no occurrence will be in the context of `author` and `chapter` at the same time.

In order to make observation 1 more formal, we have to use some definitions: Let L be the set of different labels in the database and let B_L be the set of bit strings with the fixed length $|L|$. It follows that $|B_L| = 2^{|L|}$. Let $\beta : \{o \mid o \in DB\} \rightarrow B_L$ be the function that computes for every occurrence the corresponding bit string. We define the image of that function as follows, $\mathrm{Im}(DB) := \{\beta(o) \mid o \in DB\}$. Observation 1 means that $|\mathrm{Im}(DB)| \ll |B_L|$.

Observation 2: The bit strings are label dependent: A few bit strings are generated very often, some are generated rarer and the rest never (the latter corresponds to observation 1).

Again, we will formulate this observation more formally: the function β applied to all occurrences in the database can be considered as a generator for a distribution of bit strings; we denote that distribution $D_{\beta(DB)}$. The concentration of a distribution can be measured, e.g. with the measure κ of Lorenz-Münzner ([Fer85]), describing a distribution with respect to its conformity. Observation 2 states that $\kappa(D_{\beta(DB)})$ is nearly 1, i.e. there is a tendency towards few bit strings occurring with a high frequency.

The reasons for observation 2 are:

word frequency The bigger the coverage of a label on leaf level is, the more frequently the corresponding bit strings appear. Typically, the label `par` has got a big coverage, while the label `author` has got a low coverage. The bit string encoding the linear context $\{\texttt{paragraph}, \texttt{chapter}, \texttt{body}, \texttt{book}\}$ appears much more frequently than the bit strings encoding the linear contexts $\{\texttt{author}, \texttt{book}\}$ or $\{\texttt{author}, \texttt{chapter}, \texttt{body}, \texttt{book}\}$.

label frequency Some labels appear in one context more frequently than in another, hence the corresponding bit strings appear more frequently. For example each `book` in the database has only one `title`, but several `chapters` with a `title` each.

Observation 3: The order of bit positions in the context filter is arbitrary. This means that at time of database creation the administrator is free to decide which position encodes which label.

General Outline for Bit String Compression

From observation 1 we conclude that there must be a compressed presentation of the occurrence bit strings. From observation 2 we conclude that a representation of occurrence bit strings with dynamical length can help us to reduce the space requirements for the context filter. Observation 3 will help us to encode the occurrences. The general approach is to use an alternative name space C, whose elements correspond via a translation function τ to the actual bit strings describing occurrences. Let C be the alternative name space, and $\tau : B_L \rightarrow C$ an injective function. Then τ is the compression and $\tau^{-1} : \tau(C) \rightarrow B_L$ the decompression function. The idea is to construct C and τ in a way that the total storage space for all occurrences is reduced, i.e.

$$\sum_{o \in DB} \text{storage\_space}(\beta(o)) \gg \sum_{o \in DB} \text{storage\_space}(\tau(\beta(o))).$$

An occurrence o is now enriched with a bit string $\tau(\beta(o)) = c \in C$ representing the actual occurrence bit string $\beta(o)$. In the filtering process itself, the compressed bit string c has to be translated into the occurrence bit string $\tau^{-1}(c) = \beta(o)$, and then the filtering process continues as described before.

We formulate the following requirements for a compression function τ:

- Decompression, i.e. computation of τ^{-1}, can be done in constant time.
- Total storage space for compressed bit strings is smaller than original size.
- The compression technique copes with dynamic databases, i.e. a growing $\mathrm{Im}(DB)$ does not enforce a reorganization of the whole index structure.

Front Compression
Let $|b|$ denote the length of a bit string b. Observation 3 gives us the freedom to choose an ordering of labels for bit string representation. We are looking for an alignment of the bit positions so that β tends to generate regular bit patterns. Remember, that the lower the coverage of a label, the lower the probability that the corresponding bit position contains a "1". If we arrange the bit positions by ascending coverage, the bit strings will show the tendency to start with a long run of "0"s. Hence we can conclude that every bit string starts with a run of $0 \leq k \leq |L|$ "0"s (k maximal). There are $|L| + 1$ possibilities for k, representing $|L|+1$ possibilities for the first "1". These possibilities can be encoded as bit strings of length $\lceil \log_2(|L| + 1) \rceil$. The rest of the occurrence bit string is headed by a "1" and followed by a body b' of arbitrary many bits (not more than $|L|$). The length $|b'|$ of the body can be derived from the number k of leading "0"s by the formula $|b'| = |L| - k - 1$. If $k = |L|$, i.e. $|b'| = -1$, then the occurrence bit string consists of "0"s only.[3]

$\tau(b)$ for a bit string b is defined in the following way: A prefix of fixed length $\lceil \log_2(|L|+1) \rceil$ encodes the number k of leading "0"s in b. Then the body (excluding the first "1" after the prefix of b) is appended to this prefix, resulting in $\tau(b)$. An example of this encoding can be found in Section 6.

The decompression step can be realized by the hardware-oriented SHIFT operation.[4] Hence the decompression step (computation of $\tau^{-1}(b)$) can be done in constant time. Every bit string $b \in B$ can be represented with this compression technique, i.e. an index reorganization is not enforced by growing $\mathrm{Im}(DB)$.

Similarly to the discussion of front compression, the observation that the strings also show the tendency to end with a long run of "1"s, suggests the idea to reapply this technique, a suffix compression. But in the next sections we present other techniques for avoiding unnecessary information caused by labels with a high coverage.

5.2 Label Elimination based on Absolute Selectivity

The selectivity of a label is an exact measure for the benefits of the context filter. Labels with a low selectivity have little benefits and can thus be excluded. The database administrator may chose a threshold, e.g. 85%, and exclude all labels with a selectivity lower than this threshold. Labels not incorporated in the index structure receive no special treatment from the context filter.

5.3 Grammar-based Label Elimination

As an extension to the considerations before, that elaborated on superfluous labels based on absolute selectivity, we will now generalize this to a concept based on

[3]An occurrence bit string consisting of "0"s may only occur in combination with one of the other space reduction techniques.

[4]In a concrete realization of this optimization we have to take into account hardware definitions like Byte length or word length. These strict boundaries have a negative influence on the actual benefit of front compression, but the good experimental results in Section 6 suggest that front compression will perform well even with this handicap.

relative selectivity. This requires that the general structure of the documents in the database is at least partially known and described by a grammar.

In most document databases this grammar exists; in the case of SGML or XML it is usually provided in the form of a DTD. If this is case, we can use the hierarchic information provided in the grammar together with knowledge of the selectivity of labels in order to locate labels that can be neglected in the occurrence bit string. Consider the following motivating example:

The grammar of a `book` describes a `chapter` as a sequence of one `title` and arbitrary many `paragraphs`. In addition we know, that a `paragraph` cannot occur outside a `chapter`. From a selectivity analysis we can infer that all `chapters` of a `book` have almost the same content as the `paragraphs`, i.e. $\varsigma(\texttt{chapter}) \approx \varsigma(\texttt{paragraph})$. This means that almost every word occurring in a `chapter` occurs also in a `paragraph`. The few remaining occurrences in a `chapter` reflect exactly the contribution of the `title`. From this follows that almost all occurrences inside `chapters` are also inside `paragraphs`. If a query specifies an occurrence to be inside a `paragraph` we can filter out occurrences that are not inside `chapters` instead, while taking into account only a small increase of recall noise. This increase consists of the occurrences inside of `titles`. This means that we can drop information about `paragraphs` in the index structure. In the phase of query preprocessing, `paragraph` contexts in the query are changed to `chapter` contexts for the filtering technique. Note that the increase of recall noise only affects the occurrence sets that are handed back to the query evaluation algorithm and has absolutely no influence on the quality of the overall answer.

This example leads to the definition of relative selectivity:

Definition 5.1 *Let l_1 and l_2 be labels.*

- *We call l_1* fixed under *l_2, if it occurs under a given context l_2 only and $\varsigma(l_2) < 1$.*
- *The* relative selectivity *$\varsigma(l_1, l_2)$ of a label l_1 fixed under l_2 is defined as*

$$\varsigma(l_1, l_2) = 1 - \frac{1 - \varsigma(l_1)}{1 - \varsigma(l_2)}$$

- *We say that l_1 is c-dependent on l_2 iff $\varsigma(l_1, l_2) < c$.*

The relative selectivity $\varsigma(l_1, l_2)$ takes values of the interval $[0, 1]$ and reflects the independence of l_1 relative to l_2. $\varsigma(l_1, l_2) = 0$, iff $\varsigma(l_1) = \varsigma(l_2)$, i.e. l_1 and l_2 contain absolutely the same portions of text. $\varsigma(l_1, l_2)$ becomes bigger, the more $\varsigma(l_1)$ and $\varsigma(l_2)$ differ. This means a decreasing conformity of vocabulary and coverage of l_1 and l_2 and thus an increasing relative selectivity of l_1 under l_2.

The relative selectivity turns out to be a generalization of the (absolute) selectivity defined in Section 4: $\varsigma(l) = \varsigma(l, DB)$ if we use (slightly imprecisely) the symbol DB for a label covering the complete document collection, since $\varsigma(DB) = 0$ holds.

As in the case of selectivity, labels do not carry enough information, if their relative selectivity is too low, and can be neglected in the occurrence bit strings. Furthermore, analogously to selectivity, the relative selectivity of a label can be approximated by the (relative) coverage: $\gamma(l_1, l_2) := \frac{\gamma(l_1)}{\gamma(l_2)}$. We can approximate $\varsigma(l)$ with $1 - \gamma(l_1, l_2)$, since under the assumption of label independence $\varsigma(l_1, l_2) = 1 - \gamma(l_1, l_2)$ holds.

If labels are disregarded due to their low relative selectivity, the changes for the filter components are as follows: If for a given threshold c a label l_1 is c-dependent on l_2, then l_1 is not represented in the bit strings attached to every occurrence. In query preprocessing, contexts l_1 are replaced by l_2, i.e. an occurrence specified in the query to appear inside context l_1 is filtered using context l_2. The filtering process itself does not change further.

5.4 Overview and Combination of the Space Reduction Techniques

We have presented so far three techniques for reducing the space requirements: Bit string compression and selectivity- and grammar-based label elimination. Another technique not mentioned here is the removal of labels that are very unlike to be used in queries, for example layout descriptions. This section gives a synopsis of the context filter if all these techniques are used.

Before the database with its index structure is actually created the administrator has to decide which labels are suppressed in the occurrence bit strings. This can be done by choosing a threshold[5] c and then using the automatic techniques based on absolute and relative selectivity. The result is a set of labels carrying enough information to be integrated into the index structure. For labels suppressed due to grammar-based reduction, a mapping f is established, that provides a means for mapping linear contexts to their encoding bit strings.

f is defined as

$$f(l) \quad = \quad \begin{cases} f(l') & \text{if } l \text{ is } c\text{-dependent on } l' \\ \textbf{no contribution} & \text{if } l \text{ is removed due to absolute selectivity} \\ \mathit{bit\_position}(l) & \text{otherwise} \end{cases}$$

The function *bit_position* returns the position of the label in the bit string. This position is determined by the order imposed by the front compression technique.

Of course in an implementation some modifications are suggestive:

- dissolve the recursion with the help of a table
- $f(l)$ does not return the *bit_position* but the corresponding bit-mask, where only the bit at position *bit_position*(l) is set
- $f(l)$ does not return **no contribution** if necessary but a bit string that consists of "0"s only

With these modifications the bit-mask for a linear context $\{l_1, \ldots, l_n\}$ can easily be computed hardware-orientated: $\mathit{bit\_mask} = f(l_1)$ OR $\ldots$ OR $f(l_n)$

A coverage analysis used to approximate absolute and relative selectivity is used to define the order of labels in the bit strings. The (fixed) length of the prefixes can be computed from the number of remaining labels.

Now the database can be created, with every inserted document being given to a procedure updating the index structure. The index structure has the architecture as described by the host system, with the addition that every occurrence is enriched by a front-compressed bit string encoding the contextual information.

In query evaluation, the query is preprocessed in order to determine the linear contexts for all search terms. Labels removed due to absolute selectivity are neglected in the linear contexts. Labels removed due to relative selectivity are mapped

[5]The two techniques may use two different thresholds.

to the labels they are dependent on. The linear contexts are now represented as a bit string.

For every index access with a search term a set of enriched front-compressed bit strings is returned. These bit strings are unfolded using SHIFT operations with the value of the prefixes as parameters. For every occurrence this can be done in constant time.

Then the occurrence bit strings are ANDNOT compared with the linear context bit strings. Only the matching occurrences are handed back to the algorithm.

6 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 XML 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 ([Str97, Rei98]).

The document database consists of 1620 text documents of 5.3 Mbyte in total. The documents contain more than 390000 term occurrences and are structured by 34 labels (see Appendix A for the document statistics).

Approximation of Selectivity by Coverage
We consider example queries that search for two different terms (*Asthma* and *Software*) in six different contexts (`name`, `keyword`, `publication`, `research`, `project` and the unique root label `unico`). 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 (third) columns states the absolute number of the occurrences of the term *Asthma* (*Software*) in the context of the respective label and its fraction among all occurrences of *Asthma* (*Software*). As can be seen in the last row, the (unmodified) index structure returns 26 occurrences for the search term *Asthma* and 51 occurrences for *Software*. The fractions of *Asthma* and *Software* occurrences under a given label can be seen as samples to measure the quality of our selectivity approximation through the coverage.

The result shows that the coverage gives some approximate value for the reduction of the occurrence sets. With the discussion about the influence of vocabulary

	Coverage	Occurrences of *Asthma*		Occurrences of *Software*	
		absolute	relative	absolute	relative
`name`	3.90%	2	7.69%	3	5.88%
`keyword`	4.86%	6	15.38%	10	19.61%
`publication`	22.80%	4	6.67%	3	5.88%
`research`	39.59%	14	53.85%	28	54.90%
`project`	82.02%	18	69.23%	34	66.67%
`unico`	100.00%	26	100.00%	51	100.00%

Table 1: Coverage and actual occurrences

inter-dependence upon selectivity, the significant difference between the coverage of `keyword` and the high percentage of *Asthma* and *Software* occurrences under this label can be interpreted as follows: The label `keyword` is used for marking lists of keywords. Obviously *Asthma* and *Software* are good candidates for keywords, since they are nouns with a strong semantical content. Therefore they have a higher proportion of occurrences in this text region than other terms. This difference reflects the orthogonal influence of coverage and vocabulary inter-dependence upon the selectivity. A similar effect can be observed on the label `publication`.

Reduction Based on Absolute Selectivity

With the reduction techniques based on absolute selectivity as described in Section 5.2 we could find six labels (`unico`, `project`, `p`, `research`, `publication` and `institution`) that could be eliminated (threshold $c = 0.85$ for absolute selectivity, i.e. $c' = 0.15$ for coverage) due to their low selectivity (i.e. high coverage; cf. Table 4).

Grammar-based Reduction

The 34 labels contain only 14 fixed labels, i.e. labels appearing in one context only. (The reason for this is that the documents evolved as an overlay between layout marked documents and semantically marked documents.) We will approximate the relative selectivity $\varsigma(l_1, l_2)$ with the relative coverage $\gamma(l_1, l_2) = \frac{\gamma(l_1)}{\gamma(l_2)}$.

The values of $\gamma(l_1, l_2)$ for labels l_1 fixed under l_2 are listed in Table 2.

Label l_1	Context label l_2	$\gamma(l_1, l_2)$
`tr`	`table`	96.85%
`td`	`tr`	87.85%
`project`	`unico`	82.02%
`dd`	`dl`	77.86%
`publication`	`project`	27.80%
`dt`	`dl`	22.02%
`institution`	`unico`	17.98%
`contact`	`institution`	11.07%
`equipment`	`institution`	9.13%
`institute`	`project`	4.87%
`child`	`institution`	1.92%
`duration`	`project`	1.32%
`parent`	`project`	0.42%
`internet`	`institution`	0.11%

Table 2: Relative coverage $\gamma(l_1, l_2)$ of fixed labels l_1

With a threshold $c = 0.85$ for relative selectivity, i.e. $c' = 0.15$ for relative coverage, we find 7 c-dependent labels, namely `institution`, `dt`, `publication`, `dd`, `project`, `td`, `tr`. 3 of them have already been eliminated before, but with the notion of relative selectivity we gained a benefit of another four bits that could be avoided in the bit string. Note that these labels can be used in queries, but information about them has not to be stored in the bit strings associated with the occurrences. Nonetheless, the new filtering techniques also can be applied to these labels, as elaborated in Section 5.3.

This means that the notion of relative selectivity gives a benefit of another four bits that could be avoided in the bit string. Both methods combined result in the reduction of a bit string of length 34 (for 34 labels) down to length 24, i.e. 3 Bytes. The function used for mapping labels to surrounding labels they are dependent on consists of four pairs only: $\{\texttt{dd} \mapsto \texttt{dl}, \texttt{dt} \mapsto \texttt{dl}, \texttt{tr} \mapsto \texttt{table}, \texttt{td} \mapsto \texttt{table}\}$. The labels `project`, `publication` and `institution` are removed due to absolute selectivity and are therefore not mapped to another label.

Bit String Compression
We will discuss the benefits of bit string compression again on the two example occurrence sets for the search terms *Asthma* and *Software*. There are 26 occurrences of *Asthma* and 51 occurrences of *Software* in the document database. For 24 remaining labels this sums up to a total length of all concatenated occurrence bit strings of 624 or 1224 bits, respectively.

If we use front compression, the prefix has always length 5, in order to represent a sequence of $0 \leq k \leq 24$ leading "0"s. For the encoding of labels in occurrence bit strings we used the order of Table 4. The linear context $\{$`equipment`, `institution`, `li`, `ul`, `unico`$\}$ is represented as occurrence bit string `000000000000010000011000` and encoded in the compressed bit string `01101 0000011000`. In Table 3 we divided the occurrences with respect to their contexts and stated for every class of occurrences the number of leading "0"s in the occurrence bit string o, the length $|\tau| = |\tau(\beta(o))|$ of the encoded occurrence bit string, the number of occurrences of the respective search terms under that context and the total length of the concatenated name bit strings.
The total length of the concatenated, compressed bit strings for all occurrences of *Asthma* (resp. *Software*) is 174 or 510, respectively. This means that the bit strings are reduced down to 30% (resp. 40%) of the size using 24 bit encoding. Together with the reduction techniques based on selectivity the size of storing all occurrence bit strings for *Asthma* (resp. *Software*) was reduced from 884 (1734) bits down to 20% (30%) of their original size.
Every occurrence of *Asthma* (*Software*) needs now 6.7 bits (10 bits) on average for storing the compressed bit string. But this great improvement still has to be verified with other experimental data.

Term $\bar{o}$	Linear context	"0"s	$\|\tau\|$	# occ.	Total
Asthma	`p, research, project, unico`	24	5	10	50
	`p, publication, project, unico`	24	5	4	20
	`p, research, institution, unico`	24	5	4	20
	`keyword, institution, unico`	18	10	4	40
	`keyword, project, unico`	18	10	2	20
	`name, project, unico`	16	12	2	24
Software	`p, research, project, unico`	24	5	16	80
	`p, publication, project, unico`	24	5	3	15
	`p, research, institution, unico`	24	5	4	20
	`li, ul, research, institution, unico`	19	9	1	9
	`keyword, institution, unico`	18	10	4	40
	`keyword, project, unico`	18	10	6	60
	`institute, project, unico`	17	11	2	22
	`name, project, unico`	16	12	3	36
	`li, ul, cooperation, institution, unico`	15	13	1	13
	`contact, institution, unico`	14	14	1	14
	`li, ul, equipment, institution, unico`	13	15	2	30
	`p, equipment, institution, unico`	13	15	1	15
	`li, ul, ol, research, project, unico`	8	20	4	80
	`li, ol, research, institution, unico`	8	20	2	40
	`b, li, ol, research, institution, unico`	3	25	1	25

Table 3: Effects of front compression on storage space

7 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 architectures for structured document retrieval with little effort. It also presented with the notion of "selectivity" a measure to quantify the added value of the context filter.

Based on this measure techniques that reduce the size of the context filter were introduced. One technique was based on the idea to exploit patterns in the occurrence bit strings in order to compress these bit strings to shorter ones. The other methods used selectivity and relative selectivity of labels in order to drop labels carrying not enough information.

Besides an integration of the filter with its space optimizing techniques into existing systems and evaluation of the resulting prototypes, we are planning a deeper investigation of the following topics:

- What is the benefit of extending the concept of coverage and selectivity to structural elements, i.e. to count the number of elements governed by a label in addition to the number of words?
- An analysis of users' query needs and habits could lead to a strong reduction of the needed labels. Only labels that are used in queries frequently are needed in the index structure.
- Sometimes attributes of nodes behave in a way similar to labels, i.e. they have a nested hierarchy and inherit values over that hierarchy (e.g. in computational linguistics). [FGR98] and [FMB98] treat attributes like this. If these attributes behave similar to labels, the introduced techniques can be used for attributes as well.
- How can we use the insights gained by investigating the selectivity in order to find new means to automatically classify vocabulary based on selectivity?

Acknowledgments

We would like to thank Jochen Sauter for proofreading a preliminary version of this work.

References

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. R-tree. an efficient and robust access method for points and rectangles. *SIGMOD Record*, 19(2):322–331, June 1990.

[BYN96] R. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *SIGMOD Record*, 25(1):67–79, 1996.

[CCB95a] 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.

[CCB95b] 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.

[Fer85] Franz Ferschel. *Deskriptive Statistik.* Physika-Verlag Würzburg Wien, 1985.

[FGR98] N. Fuhr, N. Gövert, and T. Röllecke. DOLORES: A system for logic-based retrieval of multimedia objects. In *Proc. ACM SIGIR '98*, 1998.

[FMB98] F. Fourel, P. Mulhem, and M.-F. Bruandet. A generic framework for structured document access. In *Proc. DEXA'98*, 1998.

[GW97] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97*, pages 436–445, 1997.

[ISO86] ISO. *Information Processing - Text and Office Systems - Standard General MarkUp Language (SGML).* ISO8879, 1986.

[Kil92] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases.* PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.

[Loe94] A. Loeffen. Text databases: A survey of text models and systems. *SIGMOD Record*, 23(1):97–106, March 1994.

[MAG$^+$97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3), 1997.

[Meu98] H. Meuss. Indexed tree matching with complete answer representations. In *Proc. Fourth Workshop on Principles of Digital Document Processing (PODDP'98)*, 1998.

[MS98] H. Meuss and K. Schulz. Complete answer aggregates for structured document retrieval. Technical Report 98-112, CIS, University of Munich, 1998.

[MW97] J. McHugh and J. Widom. Query optimization for semistructured data. Technical report, Stanford University, Computer Science Department, 1997.

[MWA$^+$98] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajamaran. Indexing semistructured data. Technical report, Stanford University, Computer Science Department, 1998.

[Nav95] G. Navarro. A language for queries on structure and contents of textual databases. Master's thesis, Dept. of Computer Science, University of Chile, 1995.

[NBY97] 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.

[Rei98] R. Reiner. Ein Blackboard-System zur Informationsextraktion aus semistrukturierten Daten für die UNICO-Datenbank. Master's thesis, University of Munich, 1998.

[ST94] A. Salminen and F. W. Tompa. PAT expressions: an algebra for text search. *Acta Linguistica Hungarica*, 41(1-4):277–306, 1994.

[Str97] C. Strohmaier. UNICO, eine OMNIS-Datenbank zur Hochschulkooperation mit der Industrie. Master's thesis, University of Munich, 1997.

[W3C98a] W3C. QL'98 - the query languages workshop, December 1998. `http://www.w3.org/TandS/QL/QL98`.

[W3C98b] 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

The database contains **1620 documents** with **390696 occurrences** altogether. They are structured by **34 Labels**. The following table states how many occurrences every label contains (absolutely and relatively). The labels are translated from German into English.

Label	Coverage	# occ.
`internet`	0.02%	75
`u`	0.03%	119
`sup`	0.04%	148
`b`	0.05%	177
`sub`	0.11%	449
`parent`	0.34%	1346
`child`	0.35%	1349
`key`	0.42%	1642
`ol`	0.54%	2092
`td`	0.63%	2448
`tr`	0.72%	2795
`table`	0.74%	2886
`duration`	1.08%	4226
`created`	1.26%	4926
`acquisition`	1.26%	4926
`equipment`	1.64%	6415
`contact`	1.99%	7775
`dt`	2.99%	11688
`cooperation`	3.00%	11728
`name`	3.90%	15227
`institute`	4.00%	15614
`keyword`	4.86%	18999
`ul`	5.39%	21048
`li`	5.84%	22825
`i`	6.23%	24348
`dd`	10.58%	41324
`person`	13.48%	52673
`dl`	13.58%	53072
`institution`	17.98%	70239
`publication`	22.80%	89089
`research`	39.59%	154685
`p`	60.92%	238004
`project`	82.02%	320457
`unico`	100.00%	390696

Table 4: Document database statistics