Content and Structure in Indexing and Ranking XML

Felix Weigel[*] Holger Meuss[†] Klaus U. Schulz[*] François Bry[‡]

[*]Centre for Information and Language Processing
University of Munich (LMU)
Oettingenstr. 67, D-80538 Munich
{weigel,schulz}@cis.uni-muenchen.de

[†]European Southern Observatory
Headquarter Garching
K.-Schwarzschild-Str. 2, D-85748 Garching
hmeuss@eso.org

[‡]Department of Computer Science
University of Munich (LMU)
Oettingenstr. 67, D-80538 Munich
bry@informatik.uni-muenchen.de

ABSTRACT

Rooted in electronic publishing, XML is now widely used for modelling and storing structured text documents. Especially in the WWW, retrieval of XML documents is most useful in combination with a relevance-based ranking of the query result. Index structures with ranking support are therefore needed for fast access to relevant parts of large document collections. This paper proposes a classification scheme for both XML ranking models and index structures, allowing to determine which index suits which ranking model. An analysis reveals that ranking parameters related to both the content and structure of the data are poorly supported by most known XML indices. The *IR-CADG* index, owing to its tight integration of content and structure, supports various XML ranking models in a very efficient retrieval process. Experiments show that it outperforms separate content/structure indexing by more than two orders of magnitude for large corpora of several hundred MB.

1. INTRODUCTION

As the *Extensible Markup Language (XML)* is omnipresent in the World Wide Web (WWW) and many other computing applications, persistent storage and retrieval of large collections of XML documents becomes increasingly important. Of special interest in a web context are retrieval models for XML that allow for approximate matching and ranked results: As proved by their success in classic Information Retrieval (IR) and the WWW, ranked retrieval models provide access to retrieval results that is tailored to human needs. Applications range from web search engines, web databases hosting XML documents and digital libraries to automated agents locating web services, to name just a few.

The main contribution of this paper is threefold. First, a generic classification scheme, the *PTN hierarchy*, is introduced which describes ranking models in terms of their dependency on three building blocks of an XML document (*Path*, *Term*, and *Node*). Since the same vocabulary is used to specify which document properties a given XML index can store, the *PTN* hierarchy makes it easy to relate the needs of a ranking model to the capabilities of an index.

In a second step, we apply the *PTN* classification to five well-known ranking models and different index structures in order to identify promising combinations for efficient ranked XML retrieval. Several models for ranking XML data w.r.t. structured queries have been proposed [3, 15, 11, 10, 13, 12]. Most of them are based on term frequencies, adapting classic flat-text IR to structured documents and queries. The *PTN* analysis reveals that a particular feature of the more sophisticated ranking models, which requires tight integration of content and structure in the index, is not directly supported by most XML index structures. Focussing on the structure of the documents, they index the textual content separately, such that ranking parameters depending on both content and structure cannot be precomputed and stored in the index, but must be collected on-line during query evaluation. Furthermore, expensive joins needed for recombining path and term occurrences slow down retrieval.

These observations motivate the adaptation of a combined index for content and structure, the *Content-Aware DataGuide (CADG)* [14], to the selected ranking models. The resulting *IR-CADG* (for *Integrated Ranking CADG*) supports even the most elaborate ranking models and also inherits the algorithmic advantages of its ancestors, CADG and original *DataGuide*[1] [4]. Unlike the flat table structures proposed in [15, 3, 13], DataGuide-based indices represent identical document paths collectively in main memory, which avoids many disk operations during path matching. Compared to the DataGuide and similar indices [12], the CADG and IR-CADG further reduce the number of disk accesses by means of (1) a *content-aware path matching* procedure which completely excludes from retrieval all parts of the document collection where the query terms do not occur, and (2) a *materialized content/structure join.* The second property enables the IR-CADG to store also those ranking parameters which the DataGuide cannot handle.

This paper considers tree-shaped XML documents, i.e. links are ignored. Since the investigated ranking models all employ different query formalisms, no specific query language is considered. Instead, we use quite informally common terms like *tree query*, *query path* (restricted to XPath's `child` and `descendant` axes and possibly the $*$ node test), and *label path.* The latter denotes a sequence of `child` steps

Seventh International Workshop on the Web and Databases (WebDB 2004)
June 17-18, 2004, Paris, France

[1]The term *DataGuide* also subsumes the *1-Index* [7] here, which is equivalent to the DataGuide for tree databases.

including no $*$ node tests. We do not consider queries involving order conditions like XPath's `preceding` axis.

Since all examined XML ranking models are based on the *tf·idf* model for flat documents [8] known from IR, we sketch its basic ranking technique here. In order to compute relevance values for documents in response to a given query, a term (keyword) t in document d is assigned a relevance score $\varrho_{td} = \mathit{tf}_{td} \cdot \mathit{idf}_t$ depending on two *ranking parameters*. The term frequency tf_{td} denotes the number of occurrences of term t in document d. Usually *tf* is normalized by the maximal occurrence count $\mathit{maxfreq}_d$ of any term in document d. The inverted document frequency $\mathit{idf}_t = \frac{D}{\mathit{df}_t}$ is calculated by dividing the total number D of documents by the number df_t of documents containing term t. Intuitively, the *idf* gives terms occuring in fewer documents a higher relevance, privileging them over other terms in the same query.

The remainder of this paper is organized as follows. The next section introduces the *PTN* hierarchy used for classifying XML ranking models (Section 2.1) and index structures (Section 2.2). Section 3 then briefly reviews the CADG as it is used for exact retrieval. Section 4 explains how to turn the CADG into an *Integrated Ranking CADG (IR-CADG)* adapted to different ranking models. The adaptation is described in detail for the *XPRES* model, which is also shortly reviewed, whereas other models are only sketched by analogy, where applicable. The purpose of this section is to examine which special requirements are imposed by the individual ranking models and how they can be satisfied by the IR-CADG. The paper concludes with observations from experimental evaluations and remarks on future work.

2. CLASSIFICATION OF XML RANKING AND INDEXING MODELS

Most ranking models for flat or structured documents are *frequency-based* in that the number of occurrences of a given token (e.g. a term or label path) is used to compute the relevance scores.[2] While the models examined in this paper differ with respect to their choice of frequency parameters, they all rely on index structures for storing precomputed frequencies. Tailored to the specific needs of the different ranking models, these data structures often ignore state-of-the-art XML indexing. On the other hand, most efficient XML index structures known from the literature were designed for exact retrieval and therefore need to be adapted to a given ranking model first.

To find out which indices are apt for structured document ranking, one must describe the parameters used in different ranking models and then check if a given index structure can provide time- and space-efficient access to them. The following framework uses the three building blocks *Path (P)*, *Term (T)*, and *Node (N)* to characterize the various ranking parameters (see Section 2.1) as well as the type of information a given index can store (see Section 2.2). *PTN* analysis reveals that while the ubiquitous combination *TN* from flat retrieval is well supported by most known XML indices, the structure-specific levels P and PT are often neglegted.

[2]Models allowing for approximate matches that are not frequency-based, e.g. [10], can be supported by the CADG directly and are therefore not considered here. Since the *s-term model* [11] generalizes the notion of terms in a way that affects the definitions of all basic parameters like *tf* and *idf*, it is also excluded from the following discussion.

2.1 Functional Signatures for Ranking Models

As described above, documents or document nodes are ranked w.r.t. to a query consisting of terms and, in the case of XML documents, paths restricting the term occurrences to be considered. The two query dimensions structure and content are represented by the building blocks P and T, respectively, whereas N stands for the (parts of) documents to be retrieved and ranked. Note that in the ranking context, an information item on the P level is a query path, which needs to be mapped to one or more matching label paths during query evaluation.

Ranking parameters may in principle depend on any combination of P, T and N (called a *PTN level*), although in practice not all constellations are observed. Figure 1 depicts the *PTN* level hierarchy as a Directed Acyclic Graph (DAG), where higher levels are of a finer granularity than lower levels. The set of *PTN* levels covered by a given ranking model forms its *functional signature* σ, summarizing the parameters involved in computing relevance scores according to that model. More precisely, when viewing the different ranking parameters as functions producing frequency-based values, the functional signature of a model specifies their domains, i.e. which input arguments are needed to determine the frequency values. For instance, the signature of the *tf·idf* model in classical IR is $\sigma_{\mathit{tf}\cdot\mathit{idf}} = \{TN, T, N\}$, since *tf* is on TN level, *idf* on T level, and *maxfreq* on N level.

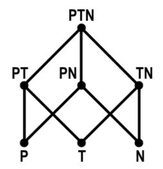

Figure 1: The *PTN* Hierarchy

Table 1 gives a synopsis of the five *tf·idf*-based ranking models examined in Section 4. Only the TN and N levels appear in all functional signatures. The two models XIRQL and XXL merely replace the notion of documents in *idf* with documents nodes, leaving it on the T level. By contrast, XPRES and BUS define the *idf* in terms of path and term information, namely as the proportion of document nodes matching a specific query path (P level) to the subset of those which additionally contain occurrences of a given term (PT level). We refer to this path/term-specific definition of *idf* as *structured idf* in the sequel, as opposed to the *flat idf* used by the other models. As an example, consider an appointment database with, say, a single document describing a meeting with "June Smith", whose name appears in a `name` node. Suppose there are numerous other documents, however, where the string "June" occurs in a `timeDate` node. When searching for `name`s containing "June", the flat *idf* used for ranking matching occurrences is much smaller than the structured *idf* because all occurrences of "June" as dates count, too. Hence using flat *idf* the term "June", though highly selective as a name, is mistakenly regarded as little informative and neglected during relevance computation.

model	*PTN*	*PT*	*PN*	*TN*	*P*	*T*	*N*
tf·idf				×		×	×
XIRQL				×		×	×
XXL				×		×	×
BUS		×		×	×		×
XPRES	×	×		×	×		×

Table 1: Functional Signatures

2.2 Storage Signatures for Index Structures

PTN signatures also serve to characterize indices for structured documents. Every index may store information on one or more *PTN* levels, both for ranking or retrieval purposes.

A *storage signature* assembles the levels covered by the data structures of the index, just like the functional signature lists the levels of all ranking parameters. For instance, a simple *node list* mapping document nodes to their maximal term frequencies represents the N level needed by *tf·idf*'s *maxfreq* parameter. The *document/term matrix* known from traditional IR covers the TN level since each field in the matrix holds information for a particular term and document or document node. Note that storing T or N level information in the matrix entails significant redundancy, each term being combined with all documents and vice versa. In behalf of space efficiency we restrict storage signatures to those PTN levels causing no needless redundancy, and do not consider T and N as part of the matrix storage signature. By contrast, an *inverted node list* (inverted file) mapping terms to the set of document nodes where they occur is capable of storing TN as well as T information without redundancy. As a non-first normal form (NF^2) variant of the document/term matrix, it contains only one entry for a given term, avoiding duplication of term-specific information. Thus T information like the flat *idf* can be attached to the appropriate entry, in addition to its TN information (*tf*). Only the N level is not supported without redundancy: since a given document node may contain more than one term, that node's data would appear in multiple term entries.

In the case of flat IR, the inverted node list can still hold all data needed for computing the ranking scores. In particular no additional data structure is required for representing the N level if the *tf* values in the list are already normalized by *maxfreq*. This precomputation eliminates the redundancy which results from storing *maxfreq* explicitly. The situation is different for XML ranking models, however, where a term occurrence in a given node implicitly contributes to the term frequency of that node's ancestors. Since their maximal term frequency may differ from the one of the lowest containing node, *tf* and *maxfreq* values must be separately available, and hence stored explicitly in the inverted node list (TN level) and in a node list (N level), respectively.

index	*PTN*	*PT*	*PN*	*TN*	*P*	*T*	*N*
node list							×
d/t matrix				×			
inv. node list				×		×	
DataGuide			×		×		
CADG	×	×		×	×	×	

Table 2: Storage Signatures

The index structures discussed so far ignore path information altogether. As a consequence, document nodes retrieved from the index must be matched against a given query path one by one. For document collections of a reasonable size, this entails numerous disk operations. With path indices [4, 7] like the *DataGuide* all document nodes with a particular label path are retrieved in a main-memory index lookup, followed by a single disk access. The DataGuide consists of two data structures. A memory-resident graph comprising all distinct label paths from the document collection serves as a structural summary for path matching. For tree-shaped documents, every index node is reached by exactly one label path and can therefore store P information. A *path table* maps a given index node to the set of all document nodes reached by that label path (these *annotation* sets need to be kept on disk). To retrieve all document nodes matching a given query path, one searches the index graph for matching label paths and then simply looks up their annotations in the path table all at once. In this sense, the DataGuide graph materializes the aforementioned mapping on the P level from query paths to label paths (see Section 2.1). The path table can store PN information (which is not used in the models, though). Analogously to the inverted node list, it allows non-redundant P information only when in NF^2. For tree documents, even the N level is supported without redundancy since no document node appears in two distinct label path entries. Yet when processing pure content queries, node-specific data could only be obtained in a full scan of all nodes sets in the path table, which is infeasible. Hence N information typically resides in a separate node list. The resulting storage signature $\sigma_{\mathrm{DG}} = \{PN, P\}$ also applies to similar index structures like the Signature File Hierarchy [1], T-Index [7], or BUS index (see Section 4.2).

To handle document content, the DataGuide is used together with an inverted node list. The structural and textual parts of a query are evaluated separately using the DataGuide and the inverted node list, respectively. Then the resulting sets of document nodes from both lookups are combined in a *content/structure join*, i.e. an intersection of both sets in the easiest case. As shown in [14], this involves the fetching and manipulation of possibly large node sets with many false hits. Another drawback of this approach is the lack of PTN and PT support needed for ranking models with structured *idf*. The number of document nodes with a specific path and term, being distributed over the two tables, cannot be stored physically. Instead it must be reconstructed during the content/structure join by counting the remaining document nodes. Experiments show that a materialized combination of content and structure also dramatically increases the retrieval efficiency (see Section 5 and [14]). The CADG combines these two aspects for efficient ranking and retrieval.

3. COMBINING CONTENT AND STRUCTURE INDEXING WITH THE *CADG*

The *Content-Aware DataGuide (CADG)* [14] enhances the original DataGuide in two respects to speed up retrieval. First, it combines content and structure information in a single *Content/Annotation Table (CA Table)*, thus avoiding the expensive content/structure join at query time. The CA Table, which replaces both the DataGuide's path table and inverted node list, maps a given index node (representing a label path) and term to the set of document nodes with that label path where that term occurs. Index node and term together make up the primary key to support pure structure, pure content and combined content/structure queries.

Second, a term-driven path matching procedure permits to prune all parts of the index tree which are irrelevant to a given set of query terms. This *content-aware navigation* not only reduces the number of index nodes to be visited during path matching, but also prevents any false positives from being fetched from disk. For a given set of query terms, fast main-memory operations check (1) whether the index node being visited references at least one document node where these terms occur and (2) whether any descendant of that index node does. The first check, called *containment test*, avoids table look-ups for matching label paths which do not lead to the right term occurrences. The second check, referred to as *government test*, saves the futile exhaustive search of entire index subtrees representing those parts of the document collection where no query terms occur.

Content-aware navigation as well as the use of a CA Table are means to combine content and structure tightly in the CADG, such that both can be matched simultaneously during retrieval. A single light-weight table look-up at the beginning of evaluation makes the CADG content-aware w.r.t. to the given query terms. During path matching, each index node matching a prefix of the current query path must pass the government test before path matching continues in its subtree. Annotations for an index node matching the entire query path are only fetched if the containment test succeeds for that index node. The fetching is content-aware in the sense that among all document nodes with a given matching label path, only those are retrieved which also contain the right terms. This combined path/term look-up is possible since the CA Table materializes the content/structure join, which is no longer performed at query time.

Although introduced for exact retrieval in [14], the CADG can be enhanced to support ranked retrieval as an *Integrated Ranking CADG (IR-CADG)* (see Section 4). In terms of the *PTN* hierarchy, the CA Table not only accomodates the *PTN* and *PT* levels without redundancy, but also *TN* and *T* for tree documents, such that $\sigma_{\text{IR}} = \{PTN, PT, TN, P, T\}$. Term/node-specific information such as *tf* is distributed over multiple entries in the CA Table, one for each label path leading to an occurrence of that term (*PTN* level). However, since in a document tree every node is reached by only one label path, all entries for the same term are disjoint and no data is duplicated. During retrieval, the CADG accesses only the *TN* information for matching label paths. Analogously, the *T* level document frequency *df* needed for flat *idf* is stored on *PT* level, i.e. distributed over multiple disjoint label path entries. For a given term, the resulting path-specific *df* values for all its entries are summed up to produce the flat *df* (and *idf*). Since $N \notin \sigma_{\text{IR}}$, the IR-CADG stores *maxfreq* data in a separate node list (see Section 2.2).

Another indexing approach committed to tight integration of content and structure is the *IndexFabric* [2] which is based on the DataGuide, too, and has the same storage signature as the IR-CADG. While equipped with a sophisticated layered storage architecture, it lacks, however, support for content-aware navigation and ranking.

4. SUPPORTING XML RANKING MODELS WITH THE *IR-CADG*

This section describes how to adapt the CADG to the different XML ranking models, turning it into an *Integrated Ranking CADG (IR-CADG)*. We will concentrate on *XPRES* [15] as a reference model, explain its specific requirements in more detail, and subsequently illustrate how to support it using the IR-CADG. The other models are reviewed in a more cursory way, using *PTN* terminology (see Section 2).

4.1 *XPRES*

Preliminaries. In XPRES, query paths (*roles*) correspond to XPath expressions with `child` and `descendant` steps only and no qualifiers. As a shorthand let $e \in s_k$ iff document node (*element*) e matches the role s_k. Analogously, $i \in s_k$ iff index node i matches the role s_k. For a label path p, $e \in p$ holds iff e is reached by p in the document tree. $t_j \in e$ means that the term t_j occurs in e or any of its descendants. $e' \lhd e$ means that either $e' = e$ or e' is contained in e.

Ranking Model. In XPRES [15], a query is a set of pairs (t_j, s_k) where t_j is a term and s_k a role. The role s_k specifies that t_j shall occur in an element $e' \lhd e$ with $e \in s_k$. The relevance $\varrho(q, e)$ of an element e w.r.t. a query q is the sum $\sum_{(t_j,s_k)\in q} \varrho((t_j, s_k), e)$ of relevance values $\varrho((t_j, s_k), e)$ for the individual term/role pairs (t_j, s_k). (This sum can also be weighted by the user or automatically with weights ω_{t_j,s_k} for term/role pairs.) Not only full XML documents, but any element of a document can be a ranked query result. This permits to retrieve and rank arbitrary *logical documents* [11] differing from the actual physical documents in the corpus.

XPRES uses the following formula for measuring the relevance $\varrho((t_j, s_k), e)$ of an element e w.r.t. a pair (t_j, s_k):

$$\varrho((t_j, s_k), e) = (C + \mathit{ief}(t_j, s_k)) \cdot \left(K + (1-K) \cdot \frac{f((t_j, s_k), e)}{\mathit{maxfreq}_e} \right)$$

The relevance formula reveals that the XPRES model extends *tf·idf* to structured document retrieval. It contains the following ranking parameters (C, K are constant):

$$\begin{aligned}
\mathit{ief}(t_j, s_k) &= \log\left(\frac{N_{s_k} - n_{t_j,s_k}}{n_{t_j,s_k}}\right) \\
N_{s_k} &= |\{e \mid e \in s_k\}| \\
n_{t_j,s_k} &= |\{e \mid e \in s_k \wedge t_j \in e\}| \\
f((t_j, s_k), e) &= \sum_{e' \lhd e, e' \in s_k} \mathit{freq}(t_j, e') \\
\mathit{freq}(t_j, e) &= \text{number of } t_j\text{'s occurrences in any } e' \lhd e \\
\mathit{maxfreq}_e &= \max_{t_j} \left(\sum_{e' \lhd e} \mathit{freq}(t_j, e') \right)
\end{aligned}$$

Proposition. Let $p_1, \ldots, p_r$ be the set of label paths matching the query path s_k (obtained from DataGuide matching, see Section 2.2). Then $f((t_j, s_k), e) = \sum_{l=1}^{r} f((t_j, p_l), e)$, $N_{s_k} = \sum_{l=1}^{r} N_{p_l}$, and $n_{t_j,s_k} = \sum_{l=1}^{r} n_{t_j,p_l}$.

IR-CADG for XPRES. Apart from f and *ief*, all ranking parameters listed above are computed and stored during index creation. The label-path specific components N_{p_l} needed to compute N_{s_k} are attached to the respective index nodes (P level). Similarly, n_{t_j,s_k} is split into PT components n_{t_j,p_l} stored in individual rows of the CA Table. Each element listed for a particular term and label path in the table is combined with its *freq* value for that term. As explained in Section 3, storing *TN* information on *PTN* level in this way does not cause any redundancy since no element appears twice for the same term. Finally, the *maxfreq* values for all elements are kept in a separate node table.

Query Evaluation. The relevance $\varrho((t_j, s_k), e)$ of an element e w.r.t. a single role s_k and a term t_j is calculated as follows. First two counters c^N and c^n are created for iterative computation of N_{s_k} and n_{t_j,s_k}, respectively. During path matching in the IR-CADG tree, s_k is mapped to the set $p_1, \ldots, p_r$ of matching label paths. For each pair (t_j, p_l), $1 \le l \le r$, the corresponding elements along with their *freq* values are fetched from the CA Table. These are exactly the elements matching s_k which also contain t_j. For each element e fetched for a pair (t_j, p_l), $c^N := c^N + N_{p_l}$ and $c^n := c^n + n_{t_j,p_l}$. Moreover, a counter c^f_e holds e's own *tf* value *freq*(t_j, e), which equals $f((t_j, s_k), e)$ if there are no descendants of e which also match (t_j, s_k). All ancestors of the fetched elements inherit their *freq* values in a bottom-up manner. To this end, a fetched element e increments the counters $c^f_{e'}$ of all its ancestors e' to be retrieved as part of the query result. When all label paths p_l for s_k have been processed this way, relevance values $\varrho((t_j, s_k), e)$ for all elements matching (t_j, s_k) and their ancestors are computed from the formula above using c^f_e, c^N and c^n as well as $maxfreq_e$, which is obtained from the node table. All other elements are ranked 0, as implied by the XPRES formula.

During the evaluation of a query q, each pair $(t_j, s_k) \in q$ is looked up separately in the IR-CADG to retrieve an ordered set of elements with relevance values. The overall relevance $\varrho(q, e)$ of element e w.r.t. q is calculated by adding up all relevance values $\varrho((t_j, s_k), e)$.

Benefits of using the IR-CADG. [15] describes a content index and a structure index for use with the XPRES model. The content index is an inverted node list holding the *ief* for a given term as well as *maxfreq* values of all elements containing the term. The *ief* values are computed at indexing time for all roles. Note that this is only feasible for a query language restricted to a predefined set of roles, since there exist infinitely many roles. By contrast, the IR-CADG maps roles to label paths in an efficient main-memory matching procedure, and then iteratively computes the *ief* while visiting matching label paths during retrieval. This way any query expression can be evaluated and ranked. The structure index employed in XPRES consists of parent/child links among elements, such that path matching operates on the document tree, with the drawback described in Section 2.2. Besides, label paths are not represented in the index. Instead each document node holds a bit vector indicating its roles. Hence for each query role, the bit vectors of all document nodes need to be checked to find matching elements, which entails numerous disk operations. The IR-CADG only visits document nodes which satisfy both the structural and the textual query conditions, reducing disk access to the miminum required for result construction.

4.2 *BUS*

Like XPRES, the *Bottom-Up Scheme (BUS)* [12] ranks document nodes based on *tf·idf* with structured *idf*. The main difference between both models lies in the definition of *tf*, which BUS restricts to direct term occurrences in a document node (as opposed to those in its descendant nodes, which also count in XPRES). Otherwise the models and hence the ways to integrate them with the IR-CADG are similar. Due to its native index structure, BUS also computes *ief* iteratively as described above for XPRES with the IR-CADG. BUS uses a path index similar to the DataGuide tree, but without the path table. Besides, an inverted node list maps terms to document nodes along with their term frequency and a link to the corresponding index node. This enables DataGuide-style main-memory path matching (albeit without the benefits of content awareness). Moreover, the content/structure join is simplified, because selecting those document nodes linked to the right index node is less expensive than intersecting them with a second node set. Still all document nodes with a given term are visited during the join, including those with mismatching label paths. [5] augments inverted lists in a way similar to BUS (but lacks support for on-line ranking), and adds pointers for chaining entries for the same index node. By contrast, the IR-CADG avoids any iteration over the disk-resident inverted file, retrieving document nodes based on their path and content directly, thanks to its materialized join. This reduces the number of disk operations and accumulators needed during retrieval. As another approach close to the DataGuide, *SemIndex* [16] can be supported by the IR-CADG in this way.

4.3 *XIRQL*

The *XML Information Retrieval Query Language (XIRQL)* [3] has a rich query model with a wide variety of operators for propagating and combining computed relevance values through the query. The probabilistic model is based on events of the form “term t occurs in index node i”, where index nodes are fixed portions of documents determining retrieval granularity. Beneath the level of index nodes, individual elements cannot be distinguished w.r.t. frequency distributions of terms. This technical construct permits to tune the model between the two extremes of high structural precision and heavy online query processing on the one hand and little structural precision and reduced processing workload on the other hand. The query language of XIRQL extends the XML query language XQL by constructs used to influence result ranking. Given a query, every document node can be assigned a relevance score. XIRQL relies on a *tf·idf*-based formula for computing term weights. Its signature $\sigma_{\text{XIRQL}} = \{TN, T, N\}$ contains T which is needed for *idf*, N for *maxfreq*, and TN for *tf*. Note that P and PT levels are not involved, since XIRQL uses a flat *idf* model.

XIRQL comes with a node list and an inverted node list as index structures, i.e. path matching is performed at document node level with the entailed performance penalty discussed in Section 2.2. The IR-CADG can be used to support query evaluation similarly as discussed for XPRES: TN information (*tf*) can be stored on PTN level in the CA Table, whereas N information (*maxfreq*) has to be stored in an extra node table. The *idf* is represented on PT level without redundancy, as explained in Section 3. Analogously, the IR-CADG may be adapted to the *XFIRM* model [9].

4.4 *XXL*

The *Flexible XML Search Language (XXL)* [13] comes with a query language that enriches SQL’s SELECT-FROM-WHERE clauses with patterns for describing textual containment and structure constraints. Given a query, *tf·idf*-based relevance values can be computed for every element. As in XIRQL, a flat *idf* is used, therefore the P and PT levels do not occur in XXL’s functional signature (see Table 1). As a special feature, XXL integrates ontology-based similarities into relevance values. Apart from a dedicated ontology index supporting the latter feature, XXL uses a node list and an inverted node list for representing N and T information, respectively. A label index (mapping element names to document nodes) prevents paths from being matched entirely on document node level. However, since the index only represents label path fragments of length 1, numerous joins of document node sets are needed during path matching. For storing ranking weights (*tf* and *idf*), XXL relies on IR-technologies of the underlying relational database backend. Since XXL has the same functional signature as XIRQL, the IR-CADG can be integrated into XXL in exactly the same way as elaborated for XIRQL.

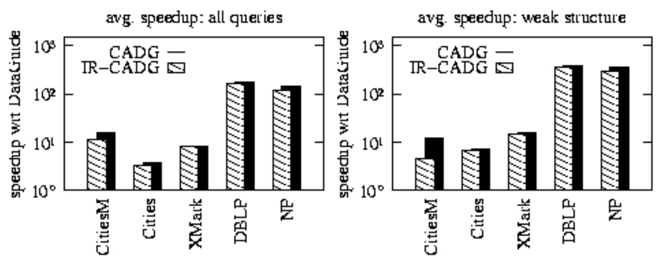

Figure 2: Response Time Speed-Up Statistics

5. EXPERIMENTAL EVALUATION

This section discussed the computational overhead caused by computing relevance scores during query evaluation. It also summarizes experiments [14] that showed the benefit of content awareness compared to the original DataGuide. IR-CADG, CADG and DataGuide were integrated into the XML retrieval system X^2 [6]. Extensive tests were performed on four different XML tree corpora (see Table 3).

name	XML size	nodes	keywords	label paths	depth
Cities	1.3 MB	16,000	19,000	253	7
XMark	30 MB	417,000	84,000	515	13
DBLP	188 MB	4,440,000	1,044,000	129	7
NP	510 MB	4,585,000	130,000	2,349	40

Table 3: Document Collections

All collections were tested with 100-600 automatically generated queries. We ensured that queries differing in characteristics such as search term frequency, number of `descendant` steps (as opposed to number of `child` steps), or existence of matches in the respective document collection are equally represented in the query sets. Additionally, the *Cities* collection was tested with a set of 90 hand-crafted queries (*CitiesM*), which were also equally distributed over the query classes. Manually and synthetically generated queries were paths containing up to 20 steps and $*$ node tests.

Figure 2 shows the performance results for two selected sets of query classes, as indicated by the plot titles. Weakly structured queries are those containing mainly `descendant` steps and $*$ node tests. The five bar charts on the left depict, on a logarithmic scale, the IR-CADG's (▨) and CADG's (■) *average speedup* over the DataGuide in the respective test suite. The speedup is defined as the ratio of the DataGuide's to the CADG's or IR-CADG's response time.

The diagrams show that the computational overhead incurred by ranking is very small and does not spoil the overall speedup of the CADG over the DataGuide, which is up to three orders of magnitude. We can also observe the capability of the CADG to deal with specific difficulties imposed by corpus structure and size on the one hand and query structure on the other hand: For the most challenging collections *DBLP* and *NP* the performance gain is by far the highest. Also, for the highly challenging weakly structured queries, which are common to realistic applications [14], the performance gain of the CADG is between two and five times higher than its average performance gain over all queries.

Of course, the performance gain of the CADG has to be paid by increased storage due mainly to the materialized join. Compared to the huge performance gain, the overhead is small: The space consumption for CADG and IR-CADG (35 MB for *NP*) lies for all corpora between two and three times of that used by the DataGuide (15 MB for *NP*).

6. FUTURE WORK

We plan to integrate the IR-CADG with further XML ranking models. Adaptation to models not based on term frequencies like [10] seems obvious. More challenging is the *s-term model* [11] with its generalized notion of *structured terms*, i.e. document and query subtrees. It enforces strict bottom-up evaluation, conflicting with the top-down CADG. Hence we are investigating the idea of an *inverted CADG* representing the document tree bottom-up. This leads to *content-centric* indexing as discussed in [14].

7. REFERENCES

[1] Y. Chen and K. Aberer. Combining Pat-Trees and Signature Files for Query Evaluation in Document DBs. In *Proc. 10th Int. Conf. on DB & Expert Systems Applic.*, 1999.

[2] B. Cooper, N. Sample, M. Franklin, G. Hjaltason, and M. Shadmon. A Fast Index for Semistructured Data. In *Proc. 27th Int. Conf. on Very Large DB*, 2001.

[3] N. Fuhr and K. Großjohann. XIRQL: A Query Language for IR in XML Documents. In *Research and Development in IR*, pages 172–180, 2001.

[4] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured DBs. In *Proc. 23rd Int. Conf. on Very Large DB*, 1997.

[5] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the Integration of Structure Indexes and Inverted Lists. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2004.

[6] H. Meuss, K. Schulz, and F. Bry. Visual Querying and Exploration of Large Answers in XML DBs with X^2. In *Proc. 19th Int. Conf. on Data Engineering*, 2003.

[7] T. Milo and D. Suciu. Index Structures for Path Expressions. In *Proc. 7th Int. Conf. on DB Theory*, 1999.

[8] G. Salton and M. J. McGill. *Introduction to Modern IR*. McGraw-Hill, 1983.

[9] K. Sauvagnat and M. Boughanem. XFIRM : A Flexible IR Model for Indexing and Searching XML Documents. Poster at *26th Europ. Conf. on IR*, 2004.

[10] T. Schlieder. Similarity Search in XML Data using Cost-Based Query Transformations. In *Proc. 4th Int. Workshop on the Web and Databases*, 2001.

[11] T. Schlieder and H. Meuss. Querying & Ranking XML Documents. *JASIS Special Topic XML/IR 53(6)*, 2002.

[12] D. Shin, H. Jang, and H. Jin. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. In *Proc. 3rd Int. Conf. on Dig. Lib.*, 1998.

[13] A. Theobald and G. Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *Proc. 8th Int. Conf. on Extending DB Technology*, pages 477–495, 2002.

[14] F. Weigel, H. Meuss, F. Bry, and K. U. Schulz. Content-Aware DataGuides: Interleaving IR and DB Indexing Techniques for Efficient Retrieval of Textual XML Data. In *Proc. 26th Europ. Conf. on IR*, 2004.

[15] J. E. Wolff, H. Flörke, and A. B. Cremers. Searching and Browsing Collections of Structural Information. In *Proc. IEEE Forum on Research and Technology Advances in Dig. Lib.*, pages 141–150, 2000.

[16] H. Zargayouna and S. Salotti. SemIndex: A Model of Semantic Indexing on XML Documents. Poster at *26th Europ. Conf. on IR*, 2004.