The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents

Jens Graupmann Ralf Schenkel Gerhard Weikum

Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany
{graupman,schenkel,weikum}@mpi-sb.mpg.de

Abstract

This paper presents the novel SphereSearch Engine that provides unified ranked retrieval on heterogeneous XML and Web data. Its search capabilities include vague structure conditions, text content conditions, and relevance ranking based on IR statistics and statistically quantified ontological relationships. Web pages in HTML or PDF are automatically converted into XML format, with the option of generating semantic tags by means of linguistic annotation tools. For Web data the XML-oriented query engine is leveraged to provide very rich search options that cannot be expressed in traditional Web search engines: concept-aware and link-aware querying that takes into account the implicit structure and context of Web pages. The benefits of the SphereSearch engine are demonstrated by experiments with a large and richly tagged but non-schematic open encyclopedia extended with external documents.

1 Introduction

1.1 Problem

In recent years information retrieval on XML data, XML-IR for short, has received great attention [2, 6, 11, 15, 28]. The goal is to support structured queries on semistructured data without a global schema, a situation arising in large intranets, federations of loosely coupled databases such as digital libraries or scientific data repositories, and the Web (if it already had large amounts of XML data). XML search engines have to cope with the diversity in the structures and annotations (i.e., tag names) of the data, and should thus employ the ranked retrieval paradigm for producing relevance-ordered result lists rather than merely using SQL or XQuery for Boolean retrieval. To this end, prior and ongoing research projects such as XIRQL [22] or XXL [42] have combined XPath-style pattern matching with relevance scoring based on similarity functions for text contents, hierarchical and link structure, and tag names and their ontological relationships.

Web search engines, on the other hand, are much less expressive in their querying capabilities, with keyword queries being the dominant search paradigm. The following queries demonstrate the shortcomings of current Web search engines and, at the same time, show the potential benefit of applying XML-IR to Web data:

- Searching for facts about the great physics researcher Max Planck. Simply typing the keywords `researcher Max Planck` yields many results about researchers who work at institutes of the Max Planck Society (Germany's leading scientific organization). What is missing is the capability for expressing that Max Planck should appear in a particular role in a Web page, namely, as the researcher himself. So a better but fictitious *concept-aware* query formulation would be `researcher person="Max Planck"`. This would benefit from richer tagging of the data, which in turn could be provided by state-of-the-art techniques for named entity recognition (e.g., persons, locations, companies and organizations) in natural-language text data.
- Searching for professors from Germany who teach database courses and do research on XML. This query cannot be answered by Web search engines because no single Web page may be a match. Rather a typical answer would be a closely connected set of pages with, for example, a researcher's homepage containing his address and pointing to a page (or pages) with her courses and to a page with her research projects. On the Web, successfully processing such a query thus requires *link-awareness* or, more generally, *context-awareness*. Note that, unlike the usual navigation axes in XML, context should go beyond trees and consider the graph structure that

Proceedings of the 31th VLDB Conference, Trondheim, Norway, 2005

is spanned by XLink/XPointer references and, especially, href hyperlinks.

- Searching for dramas where a woman makes a prophecy to a nobleman that he would become king. This query cannot be answered because a good match does not necessarily contain the keywords `woman`, `prophecy`, `nobleman`, etc. but may rather say something like "Third witch: All hail, Macbeth, thou shalt be king hereafter!" and the same document may contain the text "All hail, Macbeth! hail to thee, thane of Glamis!". This query requires some form of ontology-enabled or *abstraction-aware* processing to recognize that a witch is a woman, "shalt be" refers to a prophecy, and thane is a title for a Scottish nobleman.

From the above examples we can derive the following desiderata for a next-generation Web search engine:

- It should support querying XML and current Web data in HTML in a unified manner, with relevance ranking of results
- It should support concept-aware, context-aware, and abstraction-aware search. For XML data these are natural features, but for Web data these would be a big step forward beyond the state of the art.

1.2 Contribution

This paper addresses the above desiderata. Its key contribution lies in showing how to apply XML-IR techniques, in particular, a more expressive query language, to Web data. We present a query language that allows us to search within heterogeneous XML and Web as well as combinations in a unified manner. The language is implemented in the SphereSearch Engine. Its design has been influenced by our earlier work on the XXL language for XML IR [42], on one hand, and the desire to handle also current Web data in HTML, on the other hand. But SphereSearch also deviates from and significantly extends prior work by interpreting all data as a graph structure rather than trees. The salient features of SphereSearch are:

- It is simpler than existing query languages for XML like XPath or XQuery, but it provides ranked retrieval with support for concept-, and abstraction-aware search.
- It is much more powerful than state-of-the-art Web search engines as it supports concept-aware, context-aware, and abstraction-aware search.
- It handles XML and Web data uniformly by automatically converting HTML data into XML, with heuristics and the use of linguistic tools for named entitity recognition to generate semantically meaningful XML tags.
- It extends XML-IR techniques to arbitrary graphs, with XPath-style search conditions across document/page boundaries and a scoring/ranking model that reflects the compactness of a matching subgraph.

The SphereSearch Engine is fully implemented in Java using Oracle10g as an underlying data manager. We present experimental results on large-scale datasets using four different setups:

1. the INEX benchmark [28] for XML-IR,
2. the open Internet encyclopedia Wikipedia consisting of more than 400,000, highly cross-linked lexicon entries,
3. Wikipedia in combination with structured data from IMDB, and
4. the DBLP data[31] converted to XML in combination with href links to researcher homepages and further Web pages about projects, courses, etc.

Our experiments demonstrate both the system's efficiency and its expressiveness and search result quality.

The paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces important concepts of SphereSearch, Section 4 describes data transformation and annotation. The query language of SphereSearch is introduced in Section 5, and Section 6 presents its formal query semantics. Section 7 discusses the architecture and implementation details. Section 8 gives experimental results.

2 Related Work

Ranked retrieval on (heterogeneous) XML data has recently been a very active research area. This includes approaches based on extending keyword-based search to XML [15, 26], combining text search with XPath-style conditions [14, 47], structural similarities [6, 39], ontology-enhanced content similarities [37, 42], and applying probabilistic IR and statistical language models to XML [2, 22]. None of this prior work has considered applying these recent concepts for XML search to Web data.

Applying structured search to Web data is not a novel idea. There is a considerable body of research on Web query languages that combine text matching with SQL-style conditions including joins and other complex predicates [8, 9, 1, 30]. The problem with these approaches is that they are based on Boolean retrieval. We strongly believe that ranked retrieval is crucial for dealing with large amounts of highly heterogeneous data even if the data contains structured fragments. So unless on-the-fly schema and data integration over many autonomous data sources (e.g., hundreds of bioinformatics databases or hundreds of sports portals) is solved, SQL- or XQuery-style Boolean retrieval is of limited value.

From a Web search viewpoint, several papers have addressed enhancing keyword-based search by combining it with ontologies [29, 32, 34].

Information extraction from text and HTML data is an area with intensive work. The approaches mostly follow a rule-based paradigm [7, 17, 23, 36], or employ learning techniques and/or linguistic methods [3, 16, 18, 19]. But actually using automatically converted and "semantically" enhanced Web data in a search engine has not been pursued in the literature on information extraction.

Ranked retrieval over graph-structured data has been pursued in [10, 27]. The graphs studied there are derived from foreign-key relationships in relational databases and are quite different from the settings of the current paper. Queries in this prior work were limited to keyword search on attributes scattered across different tables. [13] lists vague search over graph structures as one of the major open challenges in areas where DB and IR technologies meet.

3 SphereSearch Concepts

SphereSearch uses the following important concepts that makes it more powerful than simple keyword search engines:

- SphereSearch *transforms* all documents into XML, using a set of heuristic rules to add semantically meaningful tags instead of the pure layout tags available in standard HTML.
- Using information extraction tools, predefined classes of information like locations, persons, and dates are *annotated* with special tags.
- A simple yet expressive query language combines *concept-aware keyword-based search* with *abstraction-aware similarity search* and *context-aware ranking.* The language allows *grouping* of query conditions that refer to the same entity.
- The relevance of an element for a group of query conditions is not only determined by its own content, but also by the content of other neighboring elements in an environment (a so-called "*sphere*") of the element, including elements in other, linked documents.
- Query groups are evaluated independently; a result for the whole query is a set of results of each group that form a *compact subgraph*, with elements/nodes close to each other.
- Optional *join conditions* allow expressing that results of different query groups should have common properties.

We elaborate on these concepts in the following sections.

4 Data Transformation and Annotation

4.1 Transformation to XML

SphereSearch converts HTML, PDF, or plain text documents into XML. The conversion is done by type-specific transformation components that try to identify structure within textual content, using heuristic rules to generate meaningful tags instead of the generic layout tags used in HTML.

```
<H1>Experiments</H1>   <Experiments>
      ...Text1...        ...Text1...
<H2>Settings</H2>   =>   <Settings>
        ...Text2..           ...Text2..
<H1>...                  </Settings>
                       </Experiments>
```

Figure 1: Example of HTML2XML Conversion

As an example, consider headlines in HTML documents that are represented (using tags like <h1>) as sibling nodes of the paragraphs following the headline; see Figure 1 for an example. Thus, the paragraphs are not "naturally" connected to the headline, in addition to the lack of semantics in generic tags like <h1>. Our heuristic rules "promote" the text within the opening and closing headline tags into a "semantic" XML tag, and construct a properly nested structure.

Another heuristic rule concerns the transformation of HTML structures like `<b>Title:</b>War and Peace<br>`. Based on the heuristic rule that a bold term followed by a colon (in the example `Title:`) explains the following term (`War and Peace`), our HTML2XML converter creates the XML fragment `<Title>War and Peace</Title>`, transforming the sibling text node *War and Peace* into a child node of the *Title* node.

Other rules exist that convert HTML tables into XML fragments, using table headers as tag names whenever possible. Formatting tags (like <br>) that remain after the transformation process are removed from the documents.

4.2 Data Annotation

To automatically recognize and annotate named entities in the content of elements (thereby faciliating their use in concept-value conditions), SphereSearch applies the information extraction component ANNIE of GATE (*G*eneral *A*rchitecture for *T*ext *E*ngineering) [18]. GATE offers various modules for analyzing, extracting, and annotating text; its capabilities range from part-of-speech tagging (e.g., for noun phrases, temporal adverbial phrases, etc.) and lexicon lookups (e.g., for geographic names) to finite state transducers for annotations based on regular expressions (e.g., for dates or currency amounts). GATE provides a set of Java libraries to facilitate its integration into existing software. Currently, SphereSearch applies ANNIE's Gazetteer Module for named entity recognition based on part-of-speech tagging and a large dictionary containing names of cities, countries, person names (e.g., common first names), etc. Named entities that are found by the Gazetteer are annotated with a type-specific tag in the XML document; we use <location> for locations, <person> for persons, <date> for dates, and <money> for amounts of money The latter two also involve regular expression matching based on ANNIE'S JAPE[1] Transducer module, that provides finite state transduction over annotations based on regular expressions. New types of entities (like conferences, sport events, etc.) can be easily added by providing additional dictionaries.

5 Query Language

The query language of SphereSearch combines concept-aware keyword-based search with specific additions for abstraction-aware similarity search and context-aware ranking. Note that even though the query language is quite simple, it may still be too powerful for a typical end user, so SphereSearch provides a simple graphical interface to pose queries (see Sect. 7.5).

[1]Java Annotation Patten Engine

As a simple example for a SphereSearch query, the search for German professors who teach database courses and have projects on XML can be phrased as follows:

```
P(professor, location=∼Germany)
C(course, ∼databases)
R(∼project, ∼XML)
```

In this query, P, C, and R are *query groups* each of which refers to one class of entities; the example query searches for a professor, a course, and a project. (If the user cannot easily identify entities for her query, she can use SphereSearch like an ordinary keyword search engine by including all basic conditions in a single query group.) For each such group, a disjunction of basic conditions further characterize the entity; these are one or more keywords or, if the user has some idea of the underlying schema or latent structure, *concept-value conditions* that state that the user expects certain values, i.e., terms in tags with certain names. Concept-value conditions are typically used to take advantage of annotations, like in the example where we exploit the fact that locations are annotated. As the annotation is done by powerful tools that provide correct annotations with high probability, using this information helps in disambiguating terms (like denoting that "Max Planck" means the person, not the institute). If the user partially knows the schema of documents (like for documents that are originally XML), she can also use known tag names in concept-value conditions. For predefined, special-purpose tags like `<date>`, concept-value conditions may also use comparisons with '<' and '>', like `date>1970`, and range conditions like `1970<date<1980`. As this is very domain specific, SphereSearch currently supports this type of conditions only for automatically annotated dates and money amounts, but can be easily extended for other data types.

In addition, SphereSearch supports the similarity operator $\sim$ that was introduced in the XXL Search Engine in basic conditions. This operator expands a keyword, a concept or a value with similar terms supplied by a quantified ontology. For example, as the example query includes the similarity operator with `databases`, SphereSearch would also return matches with `information systems` and other highly similar terms. Likewise, the search condition `∼1970` is satisfied (with different scores) by years around 1970.

Additionally, query groups can be connected by joins. As an example, consider the following query that asks for gothic and romanic churches at the same location:

```
A(gothic, church)
B(romanic, church)
A.location=B.location
```

Like content-value conditions, joins usually exploit the additional markup introduced by the annotation process.

6 Query Semantics and Scoring

6.1 Data Model

As all documents are transformed to XML, we can consider a collection $\mathcal{X} = (\mathcal{D}, \mathcal{L})$ of XML documents $\mathcal{D}$ together with a set $\mathcal{L}$ of (href, Xpointer, or XLink) links between their elements. In our model, attributes are considered as if they were elements, and the documents are already annotated. We then maintain the element-level graph $G_E(\mathcal{X}) = (V_E(\mathcal{X}), E_E(\mathcal{X}))$ of the collection that has the union of the elements of all documents as nodes and undirected edges that correspond to parent-child edges and links. In this approach we could retain the orientation of links as directed edges, but we chose an undirected graph model for it is easier to phrase queries if the user does not have to think about the direction of links. It would be fairly straightforward to support a directed graph model in SphereSearch.

We maintain two labelings on this graph: For each element $x \in V_E(\mathcal{X})$, name(x) denotes the node's tag name, and content(x) its content. Each edge is assigned a nonnegative weight which is 1 for parent-child edges and λ for links. The distance function $\delta_{\mathcal{X}}(x, y)$ takes two elements as input and computes the weight of a shortest path (i.e., a path from x to y where the sum of edge weights is minimal) in $G_E(\mathcal{X})$ between them.

6.2 Formal Query Language

A SphereSearch query $S = (Q, J)$ consists of a set $Q = \{G_1, \ldots, G_g\}$ of one or more nonempty *query groups* and a (possibly empty) set $J = \{J_1, \ldots J_m\}$ of join conditions. Each query group Q_i consists of a (possibly empty) set of keyword conditions $t^i_1 \ldots t^i_{k_i}$ and a (possibly empty) set of concept-value conditions $c^i_1 = v^i_1 \ldots c^i_{l_i} = v^i_{l_i}$. Here, keywords, concepts and values are either exact-match conditions or include the similarity operator $\sim$.

Additionally, exact-match or similarity joins can be specified between query groups. A join has the form $Q_i.v = Q_j.w$ for exact-match joins and $Q_i.v \sim Q_j.w$ for similarity joins, where Q_i,Q_j are query groups and v,w are terms (typically tag names, e.g. those introduced by the annotation process).

6.3 Query Semantics

The result of a SphereSearch query $S = (Q, J)$ with g query groups is a list of g-tuples $(e_1, \ldots, e_g)$ of elements in $V_E(\mathcal{X})$ where each e_i is a result for query group G_i, sorted by a score that measures the expected quality of results. In this section we give a bottom-up definition of the scoring function, starting with scores for keyword and concept-value conditions.

Note that our score functions include a number of parameters that have to be carefully chosen. Even though a parameter-less scoring function would be preferrable, we are not aware of other IR systems that can do without such a suite of parameters.

6.3.1 Spheres and Query Groups

Existing retrieval systems consider only the content of an element (or document) itself to assess its relevance for a query, often using a scoring model that give high scores to elements where the keywords in the query appear frequently. In SphereSearch, this type of score is provided by the *node score* of a node. For a exact-match keyword condition t, the node score $ns(n,t)$ of a node n is computed using the well-known Okapi BM25 scoring model [35], adapted for XML (see [5, 45]). For a similarity keyword condition t that has the form ~`K`, we first compute the set $\exp(K)$ of all terms that are similar to K using the ontology. The node score $ns(n,t)$ of a node with respect to this condition is then defined as

$$\max_{x \in \exp(K)} \text{sim}(K,x) * ns(n,x)$$

where $\text{sim}(K,x)$ is the ontology-based similarity of K and another term x.

We think that this kind of "local" scoring is not sufficient (1) in the presence of linked documents and (2) when content is spread over several elements (like in an article with sections and paragraphs). Instead, we want to promote the scores of elements where the requested keyword appears frequently in the context of the element, i.e., in the content of other elements in its neighborhood. To reflect this notion of context-aware scoring, SphereSearch uses the concept of *spheres*, nodes at a fixed distance of a center node. The sphere score of a node is then aggregated from nodes in spheres around the node, with less weight to nodes that are in spheres at larger distances.

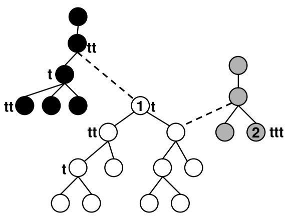


Figure 2: Example for linked documents

Formally, the *sphere* $S_d(n)$ of node n at distance d is the set of all nodes whose distance to n is d. The *sphere score* $s_d(n,t)$ at distance d of a node n with respect to an (exact-match or similarity) keyword condition t is then defined as

$$s_d(n,t) = \sum_{v \in S_d(n)} ns(v,t)$$

and the sphere score of n with respect to t is defined as

$$s(n,t) = \sum_{i=1}^{D} s_i(n,t) * \alpha^i$$

with a configurable nonnegative sphere size limit D and a configurable damping coefficient α between 0 and 1.

As an example, consider Figure 2 that shows the elements of three linked documents (denoted by different colors) and their elements, with 't's attached that correspond to the number of occurrences of the term 't' in their content. For a keyword query that asks for "t", node 2 would get the highest score in a local scoring model; in SphereSearch, it has the highest node score.

However, node 1 may be a better result for this query as the term "t" occurs much more frequently in 1's neighborhood than in 2's, hence its sphere score is higher than 2's. Figure 3 shows spheres of distances 1, 2, and 3 around node 1. Assuming $\alpha = 0.5$ and $D = 3$, we get $s(1,t) = 1 + 4 \cdot 0.5 + 2 \cdot 0.25 + 5 \cdot 0.125 = 4.175$ and $s(2,t) = 3 + 0 \cdot 0.5 + 0 \cdot 0.25 + 1 \cdot 0.125 = 3.125$, so 1 is a better result for t than 2 in our model. Note that the sphere scores for both nodes are formed from the scores of nodes in different documents, not a single document alone.

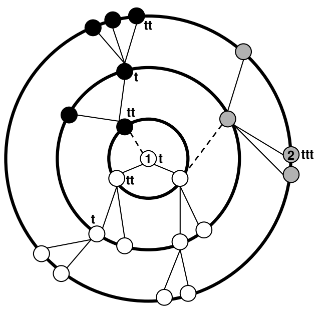


Figure 3: Spheres at distances 1, 2, and 3 around node 1

We apply a similar scoring model for concept-value conditions. Here, the node score of a node n with respect to a concept-value condition of the form `c=v` where `v` can optionally include a similarity operator is defined as

$$ns(n, \text{c=v}) = \begin{cases} 0 & \text{if name}(n) \neq c \\ ns(n,v) & \text{otherwise} \end{cases}$$

For a concept-value condition of the form ~`c=v` (again with an optional similarity operator for `v`), the score of a node n is defined as

$$\text{sim}(\text{name}(n), c) \cdot ns(n,v)$$

For domain specific comparisons with '<' and '>' and range conditions that are currently limited to automatically tagged dates and money amounts, the score of an element is computed by a domain specific similarity function.

The sphere score of a node n with respect to a concept-value condition is defined analogously to the sphere score for a keyword condition as the weighted sum of node scores in spheres around n up to distance D.

The sphere score of a node n with respect to a query group G is then the sum of the node's sphere scores for each condition of the query group:

$$s(n,G) = \sum_{j=1}^{k} s(n,t_j) + \sum_{j=1}^{l} s(n, c_j = v_j)$$

6.3.2 Queries Without Joins

Results for a query with g query groups, but without joins consist of a set of g nodes, one node for each query group of the query. Formally, we say that a *potential answer* to a query $S = (Q, \emptyset)$ without joins is a g-tuple $N = (n_1, \ldots, n_g)$ of nodes where n_i is a result for query group G_i, i.e., $s(n, G_i) > 0$.

As queries usually ask for related entities, the accumulated sphere scores of a potential answer alone is not enough to assess its relevance as the nodes in the answer may reside in completely different parts of the element-level graph, hence may be completely unrelated. Intuitively, we should prefer potential answers with nodes that are at short distances to each other in the graph, meaning that they are either in the same document or in a document that can be reached through only a few link traversals – and therefore often related as links typically connect related documents.

In SphereSearch, the score of a potential answer therefore is a combination of the sphere scores of the nodes in the answer and the compactness of the potential answer. To assess the compactness of a potential answer N, we create the connection graph $G(N) = (V(N), E(N))$ that has the elements from N as nodes and an undirected edge $\{x, y\}$ iff the distance $\delta_{\mathcal{X}}(x, y)$ of x and y in the element-level graph $G_E(\mathcal{X})$ is finite (i.e., x and y are connected); we assign this edge the weight $\frac{1}{\delta_{\mathcal{X}}(x,y)+1}$, yielding the best score for distance 0. The *compactness* $C(N)$ *of* N is then the sum of the total edge weights of a maximal spanning tree for $G(N)$; we set $C(N) = -\infty$ if the maximal spanning tree is not connected (i.e., if it is a forest, not a single tree). The score of a potential answer N to a query S is then defined as the weighted sum of the aggregated sphere scores and the compactness of the node set, i.e.,

$$s(N, S) = \beta C(N) + (1 - \beta) \sum_{i=1}^{g} s(n_i, G_i)$$

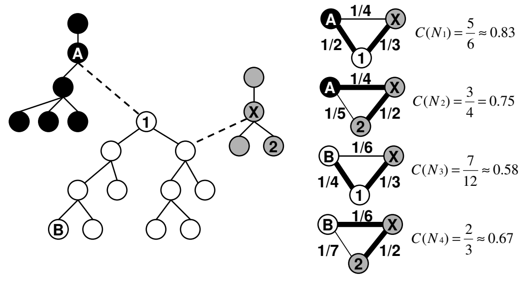

A
X
1
2
B
1/4
1/2
1/3
$C(N_1) = \frac{5}{6} \approx 0.83$
1/4
1/5
1/2
$C(N_2) = \frac{3}{4} = 0.75$
1/6
1/4
1/3
$C(N_3) = \frac{7}{12} \approx 0.58$
1/6
1/7
1/2
$C(N_4) = \frac{2}{3} \approx 0.67$

Figure 4: Compactness using maximal spanning trees

As an example, consider Figure 4 that shows (on the left) the same set of three documents that were shown in Figure 2. Assume a query with three query groups where the results for group G_1 are nodes A and B, for group G_2 node X, and for group G_3 nodes 1 and 2. From these sets of results we can generate four potential answers $N_1, \ldots, N_4$ whose corresponding connection graphs $G(N_i)$ are depicted on the right of Figure 4. Edges that belong to a maximal spanning tree are printed bold; it is evident that the potential answer N_1 consisting of A,X, and 1 is the most compact one. Note that the nodes are from two different documents, so this answer would not have been found by an engine without context awareness. Also note that the fact that documents can have arbitrary links between them make the identification of spanning trees in a rich graph structure a nontrivial task. We will discuss later how to efficiently compute them.

The ranked result of a query S is then (a prefix of) the list $N_1, \ldots$ of potential answers to S ordered by $s(N_i, S)$.

6.3.3 Queries Including Joins

An exact-match join condition `A.v=B.w` in a query requires elements with tag names `v` and `w` to have the same content and that these elements should be close neighbors of results of the query groups A and B. This again follows SphereSearch's basic design principle of context awareness, as it is unlikely that results will themselves have the requested tag names. As an example, consider a query that asks for two movies on specific themes with the same director. Here, the results for each group are probably elements describing the movies, and the director information may be in a subelement of each of them. Similarity join conditions extend this notion to contents that are similar (using an application-specific similarity measure for predefined tags or text-based similarity). While we expect that join conditions are most useful with tags like `location` introduced by the annotation process or already present in the (XML) data, SphereSearch also allows a less strict match where both the concept and the value merely occur in the content of the same element; in this case, the entire content is the basis for computing the similarity. This can be useful when information is only available as full text like 'The director of this movie is...'.

To fit this approach with our scoring model for queries without joins, we introduce new edges in our element-level graph that connect elements that match the join (i.e., have the correct tag names and the same content). The rationale for this is that results for query groups that are close neighbors to join results are moved closer together, hence the compactness of a result that includes them increases and, this way, its final score raises. As an example, consider again Figure 4 and assume the query contained an additional exact-match join between G_1 and G_2. If the contents of B's parent and X perfectly match, we introduce a new virtual link between B's parent and X; we assume that this is the only perfect match in the graph to keep the example simple. As the new distance of B and X is now reduced to 2, the new compactness of the potential answer $\{B, X, 2\}$ is increased to $\frac{5}{6}$, moving it higher in the list of results.

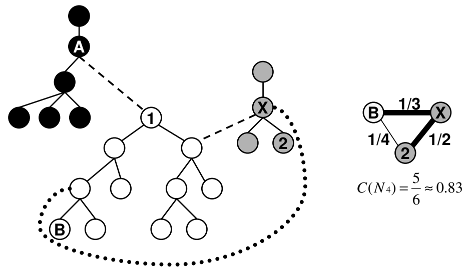


Figure 5: Effect of virtual links

Formally, we extend the collection $\mathcal{X}$ with new *virtual links*, forming an extended collection $\mathcal{X}' = (\mathcal{D}, \mathcal{L}')$. For a similarity join condition $Q_i.v \sim Q_j.w$, we consider the set $N(v)$ (resp. $N(w)$) that consists of all elements with name v (w) or contain v (w) in their content. For each pair $x \in N(v), y \in N(w)$ we add a link {x,y} with weight $1/\text{csim}(x, y)$, with csim measuring the similarity of the contents of x and y as a number between 0 and 1 where 1 represents equality. For exact-match joins, we add only links whose weight is 1 (i.e., where the corresponding nodes' contents match). csim computes either an application specific similarity measure for predefined comparison attributes (like locations or dates) or standard bag-of-words similarity of the contents of the elements (e.g., the cosine similarity based on tf*idf-style weights). The result of a query with join conditions is then the result of the query without the join conditions on the extended collection $\mathcal{X}$.

Note that this score model intentionally does not penalize potential answers that are compact, but that are not connected through shortest paths across virtual links. We believe that the elements of these answers are already closely related and therefore are a good answer to the query, too.

7 The SphereSearch Engine

7.1 System Architecture

Figure 6 illustrates the main components of the SphereSearch prototype system and their interactions. The system consists of

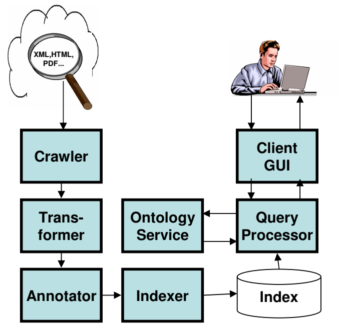


Figure 6: Architecture of the SphereSearch Engine

- the *Crawler* that collects the data, either from the Web or from a file system,
- the *Transformer* that converts non-XML formats into XML, using heuristic rules to generate meaningful tag names,
- the *Annotator* that adds annotations for named entities like locations and persons,
- the *Indexer* that inserts documents into a relational database, with specifically designed tables and index structures,
- the browser-based *Graphical User Interface* that help in graphically constructing queries,
- the *Ontology Service* that maintains an extensible ontology and provides similar terms, and
- the *Query Processor* that evaluates queries and returns a ranked list of results.

The system is implemented in pure Java, using the Tomcat application server for the graphical user interface and Oracle 10g as underlying relational database system.

7.2 Crawling and Indexing

The crawler can gather Web pages as well as local files and directories. It has as suite of control parameters for crawling strategies. The crawler supports thematically focused crawling based on an SVM classifier (which is beyond the scope of this paper), for which SphereSearch applies the BINGO! focused crawler [41].

After transforming and annotating documents (see Section 4), the indexer extracts elements, attributes and their contents from documents and stores them in an Oracle database together with information about edges and links. Content is stored in inverted lists together with corresponding tf*idf-style term statistics. The indexer also maintains position information for terms in the contents, to support phrase matching and reconstruction of whole documents. For efficient navigation within the same document, the indexer stores with each element the corresponding *Dewey encoding* of its position within the document.

In addition to edges, connections up to a configurable length in the element-level graph are precomputed and stored in the database to accelerate compactness computations. We plan to incorporate the HOPI index [38] that efficiently compresses such information in the future. If connections with longer distances are needed, they can be computed incrementally from the precomputed connections by (one or more) self joins.

Appropriate B^+ index structures are used to support an efficient evaluation of queries.

7.3 Ontology Service

The ontology service provides quantified ontological information to the system. It imports concepts and relationships between concepts from thesauri like WordNet [21] and other sources, e.g., geographic gazetteers [4], and constructs a graph of semantic relationships between concepts. In contrast to most ongoing efforts for Semantic-Web-style ontologies, our ontology service quantifies the strengths of semantic relationships based on corpus statistics [37]. To this end we have performed focused Web crawls and use their results to estimate statistical correlations between the characteristic words of related concepts. In the current version we use the Dice coefficient

$$Dice(c1, c2) = \frac{2|\{docs\ with\ c1\} \cap \{docs\ with\ c2\}|}{|\{docs\ with\ c1\}| + |\{docs\ with\ c2\}|}$$

The relationships that we quantify this way include hypernyms and hyponyms (i.e., generalizations and specializations) and holonyms and meronyms (i.e., part-of relationships). Synonyms are captured, too; their similarities are 1 by definition, regardless of correlations. We are currently extending this component to incorporate more is-instance-of knowledge (e.g., IBM is a computer manufacturer, IBM Thinkpad is a notebook) by crawling HTML tables and forms on the Web [24].

To evaluate the similarity operator $\sim$, the query evaluator extracts terms from the ontology that are semantically related to the given term in the query and whose similarity is greater than a predefined threshold.

7.4 Query Processor

The query processor first computes a result list for each query group, then adds virtual links for join conditions, and finally computes the compactness of a subset of all potential answers of the query in order to return the top-k results.

For each query group, the query processor computes a list of results (i.e., nodes in $V_E(\mathcal{X})$ together with their node scores) for each of its keyword and concept-value conditions. Candidate nodes for which the sphere score is computed are then nodes that are at distance at most D from any node that occurs in at least one of the lists, as exactly these nodes may have a nonzero sphere score. For all candidate nodes, the sphere score for the query group is then calculated by computing the spheres around the node up to distance D, looking up the node scores of nodes on the spheres in the result lists, and adding their node scores with the appropriate damping factor to the node's sphere score. The final list R_i for a query group G_i is then the list of nodes with nonzero sphere score for G_i.

When adding virtual links, the query processor considers only a limited set of possible end points of virtual links to facilitate an efficient computation, namely the nodes in the spheres up to distance D around nodes with nonzero sphere score for any query group. The rationale for this is that any other node has distance at least $D+1$ to any result node, so even if a virtual link is made through this node, it can contribute at most $\frac{1}{2(D+1)+1}$ to the compactness, which is negligible for typical values of D. Additionally, this set of candidate nodes can be computed on the fly when computing sphere scores. The set is further reduced by testing for the join attributes, yielding two sets of potential link end points. As an example, for the join condition `A.x=B.y`, one set consists of all candidate nodes whose tag name is `x` or that contain `x` in their content, and another set is computed with `y`. For all pairs of elements from these two sets, the contents are tested for equality (for exact-match joins) or similarity (for similarity joins).

To compute the final ranked list of answers, a naive solution would be to generate all possible potential answers from the answers to the query groups, compute their connection graphs with their compactness and finally their score, and sort this list in descending score order. However, as a user is typically interested only in the top-k answers (with k being in the order of 10), this would entail a lot of useless work, especially as most connection graphs will not have a connected maximal spanning tree anyway.

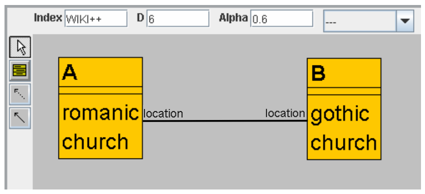
To avoid this potential performance problem, SphereSearch applies a top-k algorithm along the lines of Fagin's Threshold Algorithm [20, 25, 33] with sorted accesses only. The input to this algorithm are two kinds of sorted lists: (1) for each query group G_i in the query the list of results for each query group, sorted by their sphere score in descending order, and (2) for each two-element set $\{G_i, G_j\}$ of different query groups with their result sets R_i and R_j, a list with all two-element sets $\{r_i, r_j | r_i \in R_i, r_j \in R_j\}$ with score $\delta_{\mathcal{X}'}(r_i, r_j)$. The second kind of lists contain all possible edges of connection graphs of any potential answer.

The algorithm incrementally builds candidates for connected maximal spanning trees by reading the entry from any of the lists that currently has the highest score and combining it with already existing candidates. As an example, if an entry $\{r_i, r_j\}$ is read and there is a candidate with the same r_i, but without an edge to a result from R_j, this candidate is extended. The algorithm maintains the set of the k connected maximal spanning trees with the currently best scores; when a candidate that is extended forms a connected maximal spanning tree, it is added to this set if its score is better than the worst score of any other tree in this set, replacing this graph. The algorithm stops when no candidate that will be completed in the future can make it into the top-k, using Fagin's threshold condition. It can be shown that this algorithm computes exactly the k potential answers with the best scores; we skip the (straightforward) proof for space reasons.

While this basic algorithm is already quite efficient, we plan to extend it in the future by (1) incrementally creating the lists (especially those with the edges) as they are read, (2) additionally allowing random accesses to speed up evaluation, and (3) using an algorithm with probabilistic thresholds [43].

7.5 Client GUI

We provide a graphical user interface that allows users to construct search requests in an intuitive manner without any knowledge of the query language itself. Query groups are represented by boxes, and each box can hold a set of keyword or concept-value conditions. Joins can be expressed by drawing lines between the boxes that are annotated with the join condition. Additionally, parameters of the SphereSearch engine can be configured from the interface (like the size D of spheres, the damping factor α, etc.). Figure 7 shows the visual construction of the query from Section 5 that asks for gothic and romanic churches at the same location.



Index WIKI++ D 6 Alpha 0.6 ---

Figure 7: Sample Query in GUI

8 Experiments

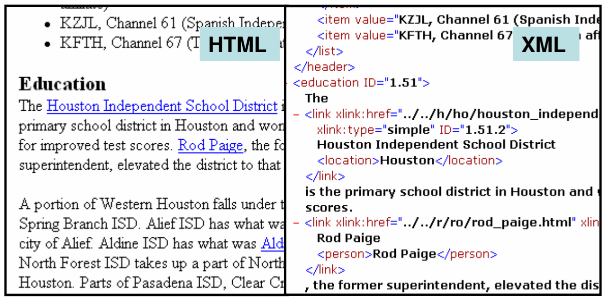
8.1 Setup

For all experiments in this section, the SphereSearch Engine was run on a dedicated machine (Sun V40z, 16

GB RAM) running Windows 2003 Server, using the Tomcat 4.0.6 environment and an Oracle 10g database running on the same machine.

Let us first comment on the difficulties of defining a meaningful benchmark for this kind of novel system. We considered the existing XML benchmarks like XMach [12], XMark [40], or INEX [28], as well as classical information retrieval benchmarks like TREC [44]. However, the first two XML benchmarks are designed only for XQuery-style exact-match, non-ranked queries on schematic XML data and thus are inappropriate for our purpose, whereas the current TREC benchmarks do not consider XML at all. The INEX benchmark for XML information retrieval would have come closest to our needs. But INEX is currently based solely on a collection of papers from IEEE CS journals and conference proceedings, full texts with XML tags, but with very simple, semantically poor tags like `<section>`, `<subsection>`, `<caption>` only and without any links. We nevertheless conducted experiments with the INEX Benchmark to demonstrate that our system performs well also on such poorly tagged XML trees. What we would need for an insightful stress test of our system is a large and structurally heterogeneous collection with semantically rich but diverse tags. None of the above choices seemed attractive from this viewpoint, so we defined three new collections together with sets of queries, in addition to the established, but not really appropriate INEX collection:

- The *Wikipedia Collection* consists of data from the Wikipedia project [46], a free Web encyclopedia that is collaboratively created by the Internet community. The English part of Wikipedia currently consists of more than 400,000 articles and is steadily growing. The articles do not follow specific schema- or content-related guidelines, nor any hierarchical structure. They are, however, highly interconnected using intra-Wikipedia links. The Wikipedia HTML collection can be created from data available from the Wikipedia website. We imported these files using the SphereSearch Crawler, transformed them to XML (which includes transforming HTML links to XLinks), and added annotations. Figure 8 shows a fragment of a transformed and automatically annotated page.



- KZJL, Channel 61 (Spanish Indepe
- KFTH, Channel 67 (T

Education

The Houston Independent School District i
primary school district in Houston and wor
for improved test scores. Rod Paige, the fo
superintendent, elevated the district to that

A portion of Western Houston falls under t
Spring Branch ISD. Alief ISD has what wa
city of Alief. Aldine ISD has what was Ald
North Forest ISD takes up a part of North
Houston. Parts of Pasadena ISD, Clear Cr

```
    <item value="KZJL, Channel 61 (Spanish Inde
    <item value="KFTH, Channel 67                 af
  </list>
</header>
<education ID="1.51">
  The
- <link xlink:href="../../h/ho/houston_independ
    xlink:type="simple" ID="1.51.2">
    Houston Independent School District
    <location>Houston</location>
  </link>
  is the primary school district in Houston and
  scores.
- <link xlink:href="../../r/ro/rod_paige.html" xlin
    Rod Paige
    <person>Rod Paige</person>
  </link>
  , the former superintendent, elevated the dis
```

Figure 8: Example HTML-to-XML on Wikipedia

- The *Wikipedia++ Collection* is an extension of the Wikipedia collection with information on movies derived from the Internet Movie Database IMDB. We downloaded the IMDB data, generated XML files for each movie and actor (containing information like plots and production locations for movies, birth places and vitae for actors, and a high number of links between documents representing castings etc.), and added links to Wikipedia pages on movies and actors.
- The *DBLP++ Collection* is based on the DBLP project [31] that provides bibliographic information on major computer science journals and proceedings. It currently indexes more than 480,000 publications and contains several thousand links to home pages of computer scientists. The DBLP database is available as a single, huge XML file; we created separate XML files for each author and publication and properly connected these files using XLinks. In addition to these XML files, we crawled the homepages of about 30,000 researchers listed in DBLP yielding a collection of XML and HTML data.
- *INEX* [28] provides a set of 12,107 XML documents (scientific articles from IEEE CS), a set of queries with and without structural constraints together with a manually assessed set of results for each query, and an evaluation environment to assess the effectiveness of XML search engines. We used the set of 47 queries from the 2004 evaluation round that were automatically converted to SphereSearch queries.

	Wikipedia++	DBLP++	INEX
documents	494,730	970,537	12,107
links	12,190,224	3,139,383	0
links per doc.	24.6	3.2	0
XML element nodes	28,697,928	9,319,820	8,283,874
elem. nodes per doc.	58	9.6	684
XML text nodes	27,196,960	7,849,264	8,267155
XML attr. nodes	29,185,399	8,174,065	3,858,653
source data size	3.1 GB	1.2 GB	504 MB

Table 1: Dataset Statistics

Table 1 illustrates the size and complexity of our data collections. We made preliminary experiments to find out reasonable values for the parameters of our engine. We chose the weight of links as $\lambda = 1$, the maximal sphere size as $D = 6$, the damping factor as $\alpha = 0.5$, and the weight for compactness in total score as $\beta = 0.5$ which gave good results for the vast majority of queries. However, a deeper analysis of the effects of parameter choices is beyond the scope of this paper.

8.2 Results for Wikipedia, Wikipedia++, and DBLP++

We asked colleagues for queries on the different collections (Wikipedia, Wikipdia++, DBLP++) that were not too easy (so that a simple keyword search could immediately find the answer) and not too difficult (so that even a very sophisticated XML query could not find the answer). We selected 50 of the submitted queries to obtain a query collection comprising queries taking advantage of the different query language features. To ensure that the queries followed some systematics, test users were told to view the data as if there were a "latent relation" with a virtual schema `Info (Person, Date/Time, Location, Event, Keywords, Description)` where each attribute could be a concept-value pair or simply text content. So usually each query group in a user query would consist of one more of these attributes (with one or more concept-value and/or keyword conditions, possibly in combination with ontology-based similarity), but the test users were also free to go beyond this view of the data.

We categorized the queries into different levels of increasing complexity, based on language features used:

- *SSE-basic*: the basic version limited to keyword conditions using sphere-based scoring, however it lacks concept-awareness and does not use multiple query groups (i.e. all conditions are included in one single group).
- *SSE-CV*: the basic version plus support for concept-value conditions.
- *SSE-QG*: the CV version plus support for query groups (i.e., full context awareness).
- *SSE-Join*: the full version with all features, including joins.

All of the above SphereSearch levels were abstraction aware by utilizing the statistically quantified ontology for query expansion. We compared the SphereSearch levels to SphereSearch operating in simple mode on document level and to Google keyword search in different modes:

- *SSE-KW*: a very restricted version of SphereSearch with simple keyword search without the notion of spheres so that score mass, using BM25 scores, is gathered only on the document level.
- *GoogleWiki*: Google search restricted to the wikipedia.org domain. Initial experiments have verified that virtually all Wikipedia data has been indexed by Google.
- *Google~Wiki*: Google on wikipedia.org with Google's ~ operator for query expansion.
- *GoogleWeb*: Google search on the entire Web (incl. Wikipedia).
- *Google~Web*: Google search on the entire Web with query expansion.

Our measures of interest include the macro-averaged precision@10 (i.e., the number of relevant results in the top-10 of a query, averaged over all queries, where relevance was manually assessed by the test users who posed the queries) and the elapsed run-time of the queries. Figures 9 and 10 show the aggregated results for the Wikipedia, Wikipedia++, and DBLP++ collections.

To apply the same collection of queries for all complexity levels, we emulated queries with higher-level features at lower levels by converting expressive conditions into keyword conditions. Queries from lower levels were added to higher levels without modification. Thus, the gain in average precicion from one to a higher level is only based on queries that were originally posed and are usually more difficult.

Figure 9: Aggregated results for Wikipedia

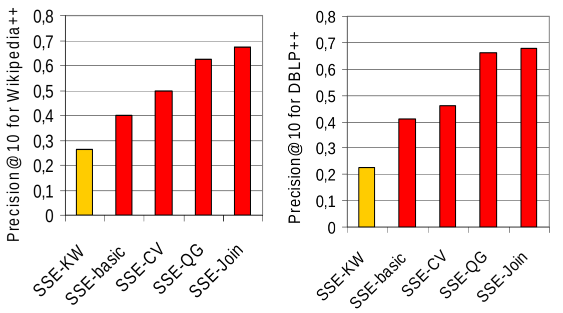


Figure 10: Aggregated results for Wikipedia ++ and DBLP++

The figures show that SphereSearch has a higher precision@10 than Google (with and without its similarity operator) on Wikipedia, and that the restricted version SSE-KW of SphereSearch is comparable to Google on this corpus. Adding more language features increases precision up to nearly twice the precision of Google, which shows that SphereSearch's structure-oriented language is helpful even on unstructured Web data. For the more structured data sets Wikipedia++ and DBLP++, the gain introduced by the new language features is even more evident.

Figure 11 shows the average runtime for different query types. With increasing complexity, runtime increases, too, but absolute runtimes are still quite efficient for a prototype implementation, given the huge size of the underlying corpus.

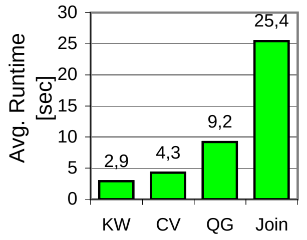


Figure 11: Avg. runtimes

In the following sections we discuss the results for the various language levels in more detail, including some anecdotic evidence for the behavior of SphereSearch.

8.2.1 Keyword Queries

Even for keyword queries SphereSearch performs better than Google and SSE-KW. One reason for this is that Google's PageRank computation gives high scores to hub-pages (like lists of events, pages for years, etc.) that contain very little specific information.

An example is the query `(Inventor,World Wide`

`Web)` to find out who invented the World Wide Web. Both Google and SSE-KW return almost the same results and the same ranking, but only the most prominent person that is considered as the inventor of the web, Tim Berners-Lee, can be found in the Top-10. Using SphereSearch, this query also returns Robert Cailliau, the co-inventor and former colleague of Berners-Lee. This is in particular based on the high scores contributed to this page by pages with outgoing and ingoing links to this page, generating a high sphere score.

8.2.2 Queries with Concept-Value Conditions

Using concept-value conditions especially increases the result quality when parts of the query are ambigous. When a user tries to find the first name of the American politician (Condoleezza) Rice with the keyword query `(American, politician, Rice)`, there is not a single correct result among the top-10. Stating the query as `(person=rice,politician)` increases precision@10 from 0 to 0.6 (in 4 of the top-10 pages the person called "Rice" is not a politician). Another query type that significantly gains from the usage of concept-value conditions are numerical annotations. The query for actors born between 1970 and 1980 `(1970<date<1980,actor)` can hardly be expressed with simple keywords.

8.2.3 Query Groups

As query groups can be used to group keywords belonging to the same entity, this additional structural element leads to higher result precision. When the results can only be found on connected pages, recall is increased as well. A typical query whose result quality can be significantly increased by using query groups is the keyword query `(California, governor, movie)` (find movies with an actor who is the governor of California) that results in a precision@10 of 0. Using groups, this query can be expressed as `G(California, governor) M(movie)`, yielding an increased precision@10 of 0.4.

8.2.4 Queries with Joins

A typical query that can exploit the power of joins is searching for movies directed by the husband of Madonna. Without knowing the name of Madonna's husband it is almost impossible to pose a keyword query returning correct results. Using the SphereSearch query `A(Madonna,husband) B(director) A.person=B.director` returns 4 relevant results among the Top-10. These results consist of pairs of elements where one element contains the information that Guy Ritchie is the husband of Madonna (this is automatically annotated as `<person>`) and another element representing a movie directed by Guy Ritchie.

8.3 Results for INEX

For the 2004 set of 47 queries with assessments, SphereSearch gained a mean average precision of 0.060 for keyword-only (CO) queries and 0.055 for queries with structural constraints (CAS), which is slightly better than our 'old' search engine XXL and would rank SphereSearch among the top-30 of all 70 systems that participated in INEX 2004. Note that SphereSearch has not been explicitly optimized for INEX (like some of the top-ranked systems) and the structural constraints of most queries cannot be easily mapped to SphereSearch's query language. On average, an INEX query took about 1-2 seconds to evaluate, which is much faster than the best published response times of INEX participants which were 13 seconds per query on similar hardware.

9 Conclusion

This paper has presented the SphereSearch Engine as a powerful search engine that provides unified ranked retrieval on heterogeneous XML and Web data. Its query language, while being much simpler than full-fledged XML query languages, includes enhanced search features for concept-awareness, context-awareness, and abstraction-awareness, yielding a much higher expressiveness than usual keyword-based text retrieval languages provide. These may not be fundamentally new for a rich XML query language, but they are innovative for a Web search engine and extremely beneficial on highly heterogeneous XML data where XPath or XQuery cannot be applied. In contrast to the mainstream of XML querying, SphereSearch is not limited to DOM trees but can handle arbitrary graph structures that emanate from XLink/XPointer references or Web hyperlinks.

All queries that we can express on XML produced ranked results and can be executed in exactly the same way on Web data that is automatically converted and heuristically enhanced with semantic tags and annotated with natural language processing tools. Most importantly, the ability to search mixed collections of XML and Web data opens up new search functionalities. Even though our experiments are preliminary, they clearly demonstrate the viable efficiency of SphereSearch and its expressiveness and good search result quality. Future work will include further experimentation, additional techniques for efficiency, and the integration of facilities for Deep-Web search.

References

[1] S. Abiteboul et al. The Lorel query language for semistructured data. *Int. Journal on Digital Libraries*, 1(1):68–88, May 1997.

[2] M. Abolhassani and N. Fuhr. Applying the divergence from randomness approach for content-only search in XML documents. In *ECIR 2004*, pages 409–419, 2004.

[3] M. Abolhassani, N. Fuhr, and N. Gövert. Information extraction and automatic markup for XML documents. In Blanken et al. [11], pages 159–174.

[4] Gazetteer Development at the Alexandria Digital Library Project. `http://www.alexandria.ucsb.edu/gazetteer/`.

[5] G. Amati, C. Carpineto, and G. Romano. Merging XML indices. In *INEX Workshop 2004*, pages 77–81, 2004. available from `http://inex.is.informatik.uni-duisburg.de:2004/`.

[6] S. Amer-Yahia, L. V. Lakshmannan, and S. Pandit. FleXPath: Flexible structure and full-text querying for XML. In *SIGMOD 2004*, pages 83–94, 2004.

[7] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *SIGMOD 2003*, pages 337–348, 2003.

[8] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases and webs. In *ICDE 1998*, pages 24–33, 1998.

[9] P. Atzeni, G. Mecca, and P. Merialdo. To weave the Web. In *VLDB 1997*, pages 206–215, 1997.

[10] G. Bhalotia et al. Keyword searching and browsing in databases using BANKS. In *ICDE 2002*, pages 431–440, 2002.

[11] H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. *Intelligent Search on XML Data*, volume 2818 of *LNCS*. Springer, Sept. 2003.

[12] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *German Database Conference (BTW) 2001*, pages 264–273, 2001.

[13] S. Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *ECML 2004*, pages 9–16, 2004.

[14] T. T. Chinenyanga and N. Kushmerick. An expressive and efficient language for XML information retrieval. *J. Am. Soc. Inf. Sci. Technol.*, 53(6):438–453, 2002.

[15] S. Cohen et al. XSEarch: A semantic search engine for XML. In *VLDB 2003*, pages 45–56, 2003.

[16] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *KDD 2004*, pages 89–98, 2004.

[17] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Automatic data extraction from data-intensive Web sites. In *SIGMOD 2002*, page 624, 2002.

[18] H. Cunningham. GATE, a general architecture for text engineering. *Comput. Humanit.*, 36:223–254, 2002.

[19] O. Etzioni et al. Web-scale information extraction in KnowItAll (preliminary results). In *WWW 2004*, pages 100–110, 2004.

[20] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[21] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[22] N. Fuhr and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *SIGIR 2001*, pages 172–180, 2001.

[23] G. Gottlob et al. The Lixto data extraction project – back and forth between theory and practice. In *PODS 2004*, pages 1–12, 2004.

[24] J. Graupmann, J. Cai, and R. Schenkel. Automatic query refinement using mined semantic relations. In *ICDE Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, 2005.

[25] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB 2000*, pages 419–428, 2000.

[26] L. Guo et al. XRANK: ranked keyword search over XML documents. In *SIGMOD 2003*, pages 16–27, 2003.

[27] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB 2003*, pages 850–861, 2003.

[28] G. Kazai et al. The INEX evaluation initiative. In Blanken et al. [11], pages 279–293.

[29] L. Kerschberg, W. Kim, and A. Scime. A semantic taxonomy-based personalizable meta-search agent. In *WISE 2001*, pages 41–52, 2001.

[30] D. Konopnicki and O. Shmueli. W3QS: A query system for the World-Wide Web. In *VLDB 1995*, pages 54–65, 1995.

[31] M. Ley. Computer science bibliography. `http://www.informatik.uni-trier.de/~ley/db/`.

[32] S. Liu et al. An effective approach to document retrieval via utilizing WordNet and recognizing phrases. In *SIGIR 2004*, pages 266–272, 2004.

[33] S. Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *ICDE 1999*, pages 22–29, 1999.

[34] Y. Qiu and H.-P. Frei. Concept-based query expansion. In *SIGIR 1993*, pages 160–169, 1993.

[35] S. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR 1994*, pages 232–241, 1994.

[36] A. Sahuguet and F. Azavant. Building light-weight wrappers for legacy Web data-sources using W4F. In *VLDB 1999*, pages 738–741, 1999.

[37] R. Schenkel, A. Theobald, and G. Weikum. Ontology-enabled XML search. In Blanken et al. [11], pages 119–131.

[38] R. Schenkel, A. Theobald, and G. Weikum. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections. In *ICDE 2005*, 2005.

[39] T. Schlieder and H. Meuss. Querying and Ranking XML Documents. *J. Am. Soc. Inf. Sci. Technol.*, 53(6):489–503, 2002.

[40] A. Schmidt et al. XMark: A Benchmark for XML Data Management. In *VLDB 2002*, pages 974–985, 2002.

[41] S. Sizov, M. Theobald, S. Siersdorfer, and G. Weikum. BINGO!: Bookmark-induced gathering of information. In *WISE 2002*, pages 323–332, 2002.

[42] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT 2002*, pages 477–495, 2002.

[43] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB 2004*, pages 648–659, 2004.

[44] Text REtrieval Conference (TREC), National Institute of Standards and Technology (NIST). `http://trec.nist.gov/`.

[45] J.-N. Vittaut, B. Piwowarski, and P. Gallinari. An algebra for structured queries in bayesian networks. In *INEX Workshop 2004*, pages 58–64, 2004. available from `http://inex.is.informatik.uni-duisburg.de:2004/`.

[46] Wikipedia, the free encyclopedia. `http://en.wikipedia.org/`.

[47] C. Yu, H. V. Jagadish, and D. Radev. Querying XML using structures and keywords in Timber. In *SIGIR 2003*, page 463, 2003.