Supporting Temporal Queries on XML Keyword Search Engines

Edimar Manica[1,3], Carina F. Dorneles[2], Renata Galante[1]

[1] Universidade Federal do Rio Grande do Sul, Brazil
{edimar.manica, galante}@inf.ufrgs.br
[2] Universidade Federal de Santa Catarina, Brazil
dorneles@inf.ufsc.br
[3] Instituto Federal do Rio Grande do Sul - Campus Avançado Ibirubá, Brazil

Abstract. In this paper, we propose an approach for providing support to temporal queries on XML keyword search engines. Our proposal is based on identifying temporal constraints in a keyword query and intercepting the query processing, executed by a conventional XML search engine, in order to evaluate those constraints. Our approach allows users to find the temporal information that they are interested in without having to learn a complex query language or needing prior knowledge of the structure of the underlying data. In this paper, we present a web search log analysis, which is our main motivation for considering temporal queries in keyword search and describe the proposal that has been defined for allowing such a query. In order to demonstrate the effectiveness of our proposal, we have executed a set of experiments over three real XML datasets.

Categories and Subject Descriptors: H. Information Systems [**H.m. Miscellaneous**]: Databases

Keywords: Temporal keyword search, temporal XML search

1. INTRODUCTION

The use of temporal data for recording historical information is a common practice, and essential in many applications (for example, web server logs, financial data, online transaction logs, workflow process logs, georeferencing applications for the localization and evaluation of a culture, scientific data and so on). In this context, a common question that arises when the topic of temporal query languages is introduced is "why are they necessary"? First, SQL-92 includes date and time data types and many applications seem to have gotten along fine using SQL. After, extensions of SQL have been proposed to support temporal queries. Furthermore, in recent years, the use of temporal expressions has emerged in Web search queries once Web documents also have temporal information.

The ability to correctly access such a data in an environment like Web is still a challenge due to scalability and diversity of data representation. Consider, for example that a simple date can be stored as "`2009 - October 12th`", "`2009-10-12`", "`12/10/2009`". How do users write their queries in a search engine? Are they really interested in temporal search? An exploratory study [Nunes et al. 2008] has concluded that the temporal expressions are used in 1.5% of queries, and they are related to current and past events. Although temporal expressions appear in only a small fraction of all queries, the scale of the Web translates this percentage into a large number of users. We speculate that the reason that might explain this situation is that the conventional search engines do not have any specific treatment for the temporal information presented in the query or in the documents. Moreover, when they offer such a functionality, most users do not know how to access or to use it.

This work was partially supported by grants from project INCTWeb (573871/2008-6 CNPq) and by a CNPq scholarship to Edimar Manica.

A temporal constraint in a keyword search can be expressed in different ways that are not treated correctly by a conventional XML search engine. We have performed a web query analysis in order to identify the different meanings of the temporal expression posed in Web queries. We have used a dataset containing Brazilian web search queries from the extinct real search engine called TodoBR[1]. We found queries where the user restricted query using a point in time ("`... in 2008`", for example) or an interval ("`... between 2008 and 2010`", for example), using temporal operators (`after` and `before`, for example) and granularities (`year`, `month` and `day`, for example). Therefore, our main goal is to fill this gap by defining an approach for automatically identifying and perform a special treatment to the temporal constraints posed in keyword queries according to web query analysis performed. We have treated this challenge in-depth when compared with related works [Wang et al. 2010]. We have covered all the temporal relations defined by Allen [Allen 1983], also including a new temporal operator (`intersect`) that considers the intersection relation. The temporal operator `A intersect B` restricts that a temporal interval `A` shares, at least, 1 temporal instant with the temporal interval `B`. It is important because it was the second relation more frequent in query analysis. We believe that the treatment of temporal expression can be used successfully in public search engines to improve ranking or result clustering. Besides, the user does not need to know the details of query languages nor uses advanced interfaces.

Another front is concerned with giving temporal treatment for XML documents. Works in this field require that the XML documents follow a specific temporal model [Li et al. 2007; Rizzolo and Vaisman 2008]. However, most XML documents that contain temporal information do not follow a specific temporal model. In this case, a temporal data might be a point in the time, which could be represented by means of a simple element, such as `<date>15/09/2010</date>`, or a fragmented instant, represented by more than one element, such as `<date><day>15</day><month>09</month><year>2010</year> </date>`. Another possible representation in a XML document is to express time as a temporal period, like `<period><begin>10/09/2009</begin><end>10/12/2009</end></period>`.

Therefore, beyond identifying temporal information presented in the query, we identify the temporal information contained in the XML documents. Our proposal is based on identifying temporal constraints in a keyword query and intercepting the query processing, executed by a conventional XML search engine, in order to evaluate those constraints. Our approach allows users to find the temporal information that they are interested in without having to learn a complex query language or needing prior knowledge of the structure of the underlying data. This paper presents four main contributions to our community: (i) web search log analysis, which illustrate how the users pose their temporal queries over the web; (ii) a temporal query classification that defines the meaning of the different types of temporal expressions used in the queries, that is used in our proposal on supporting temporal queries; (iii) a proposal to support temporal queries on XML search engines through the interception of the query processing; and (iv) experimental evaluation, showing that our implementation is effective since it improves significantly the precision and does not put down the conventional XML search engine performance.

The rest of this paper is structured as follows. Section 2 goes over related work. Section 3 presents a web query analysis for verifying how temporal expressions are posed in keyword search in order to define what kind of temporal constraint must be treated. We also describe the most important concepts related to temporal keyword search. Based on log analysis, in Section 4 we first specify a classification to be used as guidance for our temporal XML keyword search approach. The experiments that demonstrate the effectiveness of our approach are described in Section 5. The main ideas of this paper, and future work, are summarized in Section 6.

[1]TodoBR is a trademark of Akwan Information Technologies, which was acquired by Google in July 2005.

2. RELATED WORK

Several works [Tyler and Teevan 2010; Mendoza and Baeza-Yates 2008; Graells and Baeza-Yates 2008; Nunes et al. 2008] have been addressed to analyze user query logs topic in order to understand on-line user behavior and how it improves Web IR. We believe that temporal expression can be used successfully in public search engines to improve ranking or result clustering. As an example, Google has released the Google News Timeline[2], which is a web application that organizes search results chronologically. Google Timelines allows users to view news and other data sources on a browsable, graphical timeline. Our contribution in this paper, about analyzing temporal information in Web search queries, is on evaluating how users built a query using temporal predicates in a search engine.

In the field of temporal query languages, [Rizzolo and Vaisman 2008; Gao and Snodgrass 2003] extend XML query languages, respectively XPath and XQuery, to add time support by extending the syntax and semantics of the query. Timely YAGO(T-YAGO) [Wang et al. 2010] is a prototype system that extracts temporal facts from Wikipedia, which are represented as semi-structured data, such as infoboxes, categories, and lists, and integrates them into a knowledge base. T-YAGO provides a time-aware query language (SPARQL-style language) to access the temporal data. Our work is focused on extracting temporal information from XML documents while T-YAGO extracts temporal information from HTML sources, and [Rizzolo and Vaisman 2008] and [Gao and Snodgrass 2003] require a specific temporal data model. Moreover, our work provides keyword search while T-YAGO, [Rizzolo and Vaisman 2008] and [Gao and Snodgrass 2003] allows queries into a structure query language. Our approach allows users to find the information that they are interested in without having to learn a complex query language or needing prior knowledge of the structure of the underlying data.

There are three main categories for determining returned nodes in XML keyword searches. First, returning the subtrees rooted at the lowest common ancestor (LCA) nodes of the matches of the keyword queries [Guo et al. 2003; Cohen et al. 2003; Xu and Papakonstantinou 2005]. Second [Bhalotia et al. 2002; Hristidis et al. 2006], returning the paths in the XML tree from each LCA node to its descendants that match with a keyword query. Last [Liu and Chen 2007], inferring the semantics of the search to identify returned nodes. These works have general solutions for keyword searching within semi-structured documents, but do not treat specifically the problem of temporal information. Although temporal expressions appear in only a small fraction of all queries, we speculate that the reason that might explain this situation is that the search engines do not have special treatment for the temporal information.

Our approach can be applied in any conventional XML search engine. To validate it, we use the XML search engine XSeek [Liu and Chen 2007], which allows users to execute keyword searches in XML documents, and identifies meaningful return nodes without identifying the user preferences. Return nodes refer to the node names that are the goal of user searches. XSeek analyzes both XML data structure and keyword match patterns. The XML data are separated in three types of information: (*i*) entities in the real world; (*ii*) attributes of entities; and (*iii*) connection nodes. Keywords are also categorized in two types: (*i*) search predicates; and (*ii*) return information that user is looking for. In this way, XSeek generates return nodes, which can be explicitly inferred from keywords, or dynamically constructed according to the entities in the data that are relevant to the search. In this paper, we propose an approach that allows temporal keyword searching. We adopt XSeek in order to process the non-temporal keywords of the query. It means that we extract and treat temporal keywords of temporal queries once that this feature is not supported by other XML keyword search engines, such as XSeek.

[2]http://newstimeline.googlelabs.com

3. USING TEMPORAL PREDICATES IN WEB SEARCH

In this section, we present a web query analysis whose objective is to verify how temporal expressions are posed in keyword search in order to define what kind of temporal constraint must be treated. Before we describe our analysis, we shortly present some concepts about temporal queries that are important to understand the analysis itself.

3.1 Temporal Queries

Informally speaking, a temporal query is composed by temporal expressions that act as a predicate to the query. This predicate represents a binary temporal condition between the temporal expression and the other keywords of the query. This temporal condition, or temporal relation [Allen 1983], can be expressed explicitly by a temporal operator or inferred by the query semantics and context.

The temporal expression represents an operand of this operator (a temporal operand) and the other keywords express the other operand (a non-temporal operand). For instance, in the query "president after 2000", `after` is a temporal operator, `president` is the non-temporal operand and `2000` is the temporal operand.

Allen [Allen 1983] defines 13 temporal relations: `after`, `before`, `contains`, `during`, `equal`, `finished by`, `finishes`, `meets`, `met by`, `overlapped by`, `overlaps`, `started by` and `starts`. From these disjoint basic relations, it is possible to define other relations. For instance, the temporal relation `intersect` combines the relations `equal`, `overlaps` and `overlapped by`. Figure 1 depicts these relations.

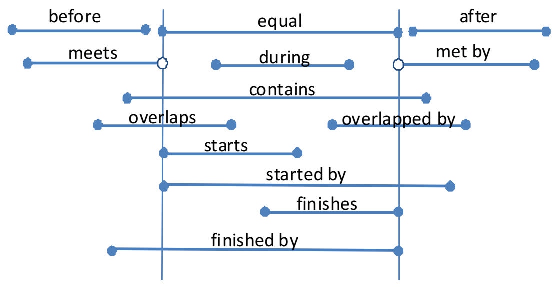


Fig. 1. Temporal Relations

3.2 Web query analysis

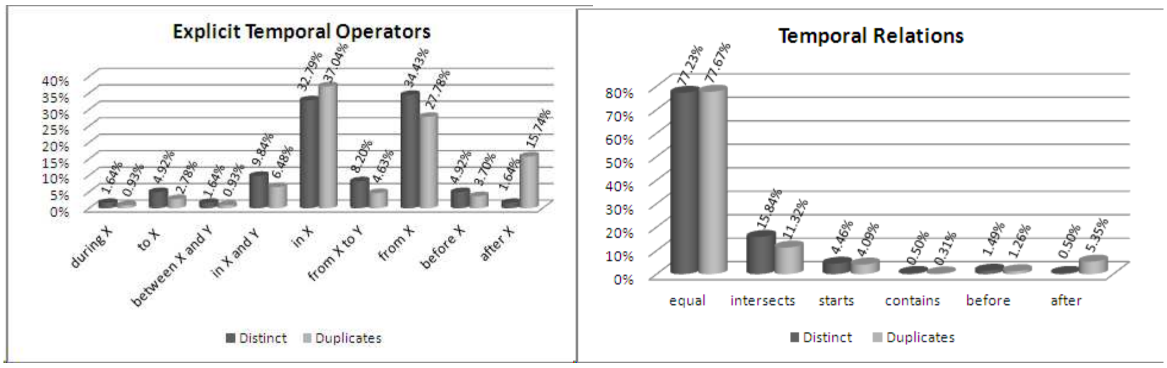
In this work, we have analyzed the temporal predicates used in queries with temporal expressions. We have used a dataset containing web search queries from the extinct real search engine called TodoBR. This dataset contains the queries performed on the Brazilian web in November 2005, and includes 65,737 duplicate queries and 33,154 distinct queries. In order to find the queries with temporal expressions, we have submitted each query to GUTime [GUTime 2009] and ANNIE [ANNIE 2009], which are tools for annotating temporal expressions. After, we analyzed manually each query that was annotated by at least one of the tools.

First of all, we distinguished queries with temporal expressions, called temporal queries, from queries without temporal expression (in both cases, duplicate and distinct queries). We also distinguished queries with temporal expression from queries with a software version (Windows 2000, for example), which we do not consider as temporal expression. As we can see in Table I, 318 (0.48%) of queries have temporal expressions and 223 (0.34%) have information about version of some specific software. Removing duplicate queries, 202 (0.61%) of the queries have temporal expressions and 96 (0.29%) have information about version. In [Nunes et al. 2008], the authors present a higher rates of queries with temporal expressions (1.5% in query with duplicates and between 1.6% and 1.9% in distinct queries), since they have analyzed the entire Web, while in our study we have just analyzed queries on Brazilian web. Moreover, they also have considered as temporal expressions dates and periods represented by names, such as Christmas, Easter, Paleolithic, while we do not consider these cases. Furthermore, they do not distinguish versions from temporal expressions, since the temporal expression identification is fully automatic. In this study we have distinguished version from temporal expressions, because the software version can not be exactly its release, for instance, "Windows 2000 Advanced Server - Limited Edition" was released in 2001.

	Distinct Queries	Duplicate Queries	Distinct Queries (%)	Duplicate Queries (%)
Total queries	33,154	65,737	-	-
Queries with temporal expressions	202	318	0.61	0.48
Queries with versions	96	223	0.29	0.34

Table I. Use of Temporal Expressions on Brazilian web searches.

Considering queries with temporal expressions (temporal queries), our next step was to verify the use of explicit temporal operators in order to specify temporal relations between the temporal expression and the others query keywords. We have checked, manually, all the 202 distinct, and 318 duplicates, temporal queries, where we have found 61 (30.20%) explicit temporal operators in distinct queries, and in 108 (33.96%) in duplicate queries. The temporal operators we found are presented in Figure 2(a). It is important to note that we cluster some operators: `in X` (em X) includes `no ano X` (in the year X), `nos anos X` (in the years X) and `na década X` (in the decade X); `from X to Y` (de X a Y) includes `de X a Y` (from X to Y), `de X e Y` (from X and Y), `de X Y` (from X Y), `no período de X a Y` (in the period from X to Y) and `dos anos X e Y` (from the years X and Y); and `from X` (de X) includes `de X` (from X), `do ano X` (from year X) and `da década X` (from decade X).



(a) Use of explicit Temporal Operators in Temporal Queries

(b) Temporal Relations inferred in the temporal queries

Fig. 2. Temporal Operators and Temporal Relations

Based on this analysis, we could observe that some temporal operators used in the queries represent the same temporal relation, for instance, `in X` and `from X` represent the temporal relation `intersect`. Moreover, queries with temporal expressions, but without explicit temporal operator also have a temporal relation between the temporal expression and the other query keywords that can be inferred by the semantics of the query. We classified manually each query with temporal expressions (including queries without explicit temporal operators) according to temporal relations defined by [Allen 1983] more the relation `intersect`. It is a subjective classification that considers the query semantic, the query context and the explicit temporal operators (when it was explicit). For instance, the query "`deputadas federais em 1988`" (federal deputies in 1988) contains the explicit temporal operator `em X`; as in Brazil the mandate of federal deputies is fours years, we infer that the temporal relation for this query is `intersect`, i.e., the user wants all federal deputies whose mandate was still valid in 1988. Another example, the query "`regime militar 64 84`" (military regime 64 84) does not contain explicit temporal operator, but we know that the military regime in Brazil occurs from 1964 to 1984, then we infer that the temporal relation is `equal`, i.e., the user wants web pages about the military regime, whose period was from 1964 to 1984. As we can see in Figure 2(b), six temporal relations were found: `equal`, `intersect`, `starts`, `contains`, `before` and `after`.

We also could check the frequency of the use of periods or instants in the temporal operand. As it can be observed in Table II, 184 (91.09%) temporal queries had an instant as temporal operand, while 18 (8.91%) temporal queries had a period. Considering duplicates queries, 297 (93.40%) queries had an instant as temporal operand, while 21 (6.60%) had a period. In these periods used as operand, it were found periods composed of two instants, an initial instant and a final instant. But, it were also found periods composed of the expression "`last x gran`", where `x` is a value and `gran` is a granularity (`last 5 years`, for example), representing a displacement from the current date.

	Distinct	Duplicates	Distinct (%)	Duplicates (%)
Instants	184.00	297.00	91.09	93.40
Periods	18.00	21.00	8.91	6.60

Table II. Proportion of instants and periods in the temporal queries.

As we can see in Figure 3, most of temporal queries (72.77% in the distinct queries and 65.41% in the duplicate queries) have granularity year, followed by decade (21.78% in the distinct queries and 28.62% in the duplicate queries). It is important to note that the granularity of the query can be different of granularity of the data. For instance, the query "`músicas nos anos 60`" (`songs in the years 60`) should return songs of 1960, 1961, until 1969.

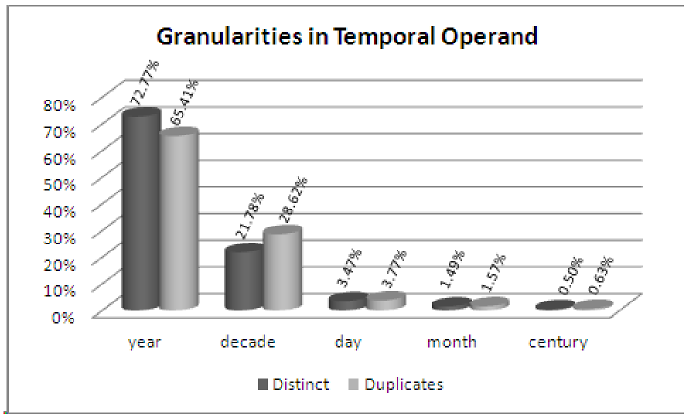


Fig. 3. Granularities in Temporal Operand

When analyzing the position of a temporal expression in the query (Table III), we discover that in the most queries (93.22% of distinct queries and 94.62% of the duplicate queries), temporal expressions are in the end of the query, i.e., after all the other keywords. Later, this feature is considered to identify what kind of temporal query has to be processed by query processor. It is important to note that there are 25 distinct queries (58 duplicate queries), in which all keywords are temporal expressions, and then these queries were not considered in this analysis of location.

	Distinct	Duplicates	Distinct (%)	Duplicates (%)
Beginning	2	2	1.13	0.77
Middle	10	12	5.65	4.62
End	165	246	93.22	94.62

Table III. Temporal expressions localization.

Figure 4 shows the number of non-temporal keywords in the temporal queries (removed the stop words). As we can see, most queries have between 0 and 3 non-temporal keywords.

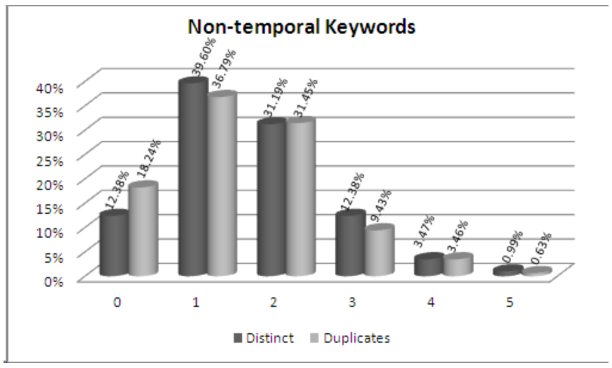


Fig. 4. Number of Non-temporal Keywords

4. SUPPORTING TEMPORAL SEARCH

In this section, we present our proposal for supporting temporal XML keyword search. First, we introduce a temporal predicate classification we have defined, based on the web query analysis, to be used as guidance for constructing our temporal search proposal itself. After, we briefly describe, the temporal indexing processing, which is based in some concepts introduced in the classification. Finally, we present the two phase interceptor (TPI), which has been built to handle temporal requirements in a query user.

4.1 Temporal predicate classification

From the results we have obtained in the web query analysis, we propose a predicate classification that is used to identify what kind of temporal query has to be processed by query processor over XML documents. Figure 5 presents an overview about our classification proposal.

A Temporal Predicate is a temporal restriction applied to some text node with temporal data, and is composed of: (i) a First Operand that represents a text, element or attribute node, and it is characterized by the non-temporal keywords of the query; (ii) a Temporal Operator that represents a temporal relation (as explained before) between first and second operands; and a (iii) Second Operand that express the time. A second operand can be a Simple Instant, a Simple Interval or a Composite Interval. A Simple Instant is characterized by using a point in time. For instance, `13/11/2010` and `2008`. A Simple Interval is a period in the time and is characterized by using two simple instants, representing the initial and the final instant of a period. For instance, `[15/03/2009, 20/05/2010]`.

A Composite Interval is a period characterized by a shift from the current date. It is composed of a Reserved Word, a Granularity and optionally a Value. A Reserved Word indicates the direction of the displacement in time, a Granularity defines the time unit used, and the `Value` specifies how many time units the displacement should be done. If the value is not given in the query, then 1 is considered as default value. We specified three reserved words to temporal query structure: *(i)* `Next`,

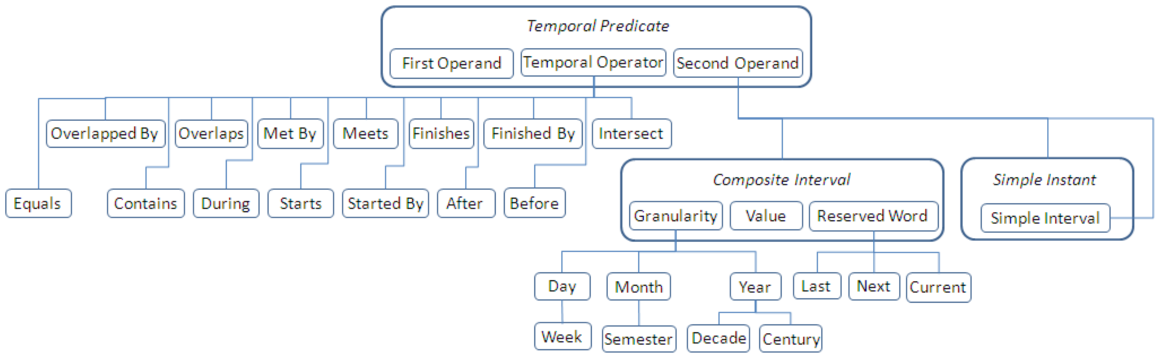


Fig. 5. Classification of temporal queries

which starts the displacement in the current date and advances *n* time units; *(ii)* `Last`, which ends the displacement in the current date and returns *n-1* time units; and *(iii)* `Current`, which covers the current period according to a given granularity. For example, the expression "*current month*" represents a period that starts in the first day of the current month and ends in the last day of the current month.

4.2 Processing Temporal Queries

In this section, we describe an overview of our proposal for querying temporal data on conventional XML keyword search engines.

4.2.1 *Indexing.* Generally speaking, the indexing processing stores the temporal values, which have been previously normalized to a common format. This common format has been defined for allowing a faster and consistent access to the documents during the execution of a temporal query. The temporal indexing process has five steps, as presented in Figure 6, which are described below.

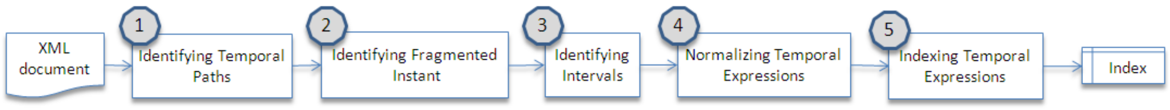

XML document
1
Identifying Temporal Paths
2
Identifying Fragmented Instant
3
Identifying Intervals
4
Normalizing Temporal Expressions
5
Indexing Temporal Expressions
Index

Fig. 6. Steps of the Indexing Process

Step 1: Identifying Temporal Paths. The main objective in this step is the disambiguation of data formats through the path behavior. It is done by clustering temporal expressions occurrences according to the paths that contains them. As examples of temporal paths, we present in Table IV some temporal paths extracted from XML documents. Each temporal path is associated with its temporal expressions, which are the node values, a type and a format. For instance, the last path contains two temporal expressions 03/05/2007 and 24/11/2007. Analyzing only the temporal expression 03/05/2007 we can not discover its format, but analyzing all temporal expressions of the path we can find out that the format for all temporal expressions of this path is DMY4, because there is no format ambiguity for the temporal expression 24/11/2007 belonging to this path.

Temporal Path	Temp. Expression	Type	Format
/clinic/vaccine/when/day	01, 15	DAY	D
/clinic/vaccine/when/month	01, 1	MONTH	M
/clinic/vaccine/when/year	1998, 2000	YEAR	Y4
/clinic/hospitalization/patient/departure	03/05/2007, 24/11/2007	DATE	DMY4

Table IV. Examples of Temporal Paths.

Step 2: Identifying Fragmented Instants. The second step of the indexing processing is to identify all fragmented instants. A fragmented instant is composed of temporal nodes that are children of the same parent node of different, but complementary, types. These temporal nodes must have values of the following domains: YEAR, MONTH and DAY (optional). In the following example, we can identify one fragmented instant:

```
<person>
   <born_year>1980</born_year>
   <born_month>05</born_month>
   <born_day>29</born_day>
</person>
```

Step 3: Identifying Temporal Intervals. The third step executed in the indexing processing is the identification of temporal intervals. A temporal interval is identified when a XML fragment has two temporal nodes (or fragmented instants) that are children of the same node where the behavior of these paths implies in the value of one of them being always less than the value of other. A temporal interval can have fragmented instants that are grandson of a same node and children of nodes with different tags. The following XML fragments present, respectively, examples about a simple temporal interval (a) and two fragmented temporal intervals (b) and (c):

(a)

```
<event>
   <begin>2010-10-05</begin>
   <end>2010-10-08</end>
</event>
```

(b)

```
<event>
   <begin>
    <year>2010</year>
    <month>10</month>
    <day>05</day>
   </begin>
   <end>
    <year>2010</year>
    <month>10</month>
    <day>08</day>
   </end>
</event>
```

(c)

```
<event>
  <begin_year>2010</begin_year>
  <begin_month>10</begin_month>
  <begin_day>05</begin_day>
  <end_year>2010</end_year>
  <end_month>10</end_month>
  <end_day>08</end_day>
</event>
```

Step 4: Normalizing Temporal Expressions. The fourth step executed in the indexing processing is simple and relates to the normalization of all temporal expressions identified in the XML documents. This procedure consists in to normalize the temporal expressions to a common format. As an example, if a temporal path has a type DATE and format XXX, it means that the format is undefined. In this case, each temporal expression in this path is normalized to six distinct values, in order to consider all possible formats: DMY2, MDY2, Y2MD, Y2DM, MY2D and DY2M.

Step 5: Indexing Temporal Expressions. Temporal intervals and temporal instants, which do not compose a temporal interval, are stored in the same index structure. They are indexed as a normalized temporal period, so a temporal instant is stored as a period that starts and finishes in the same instant. Each stored period points to a XML node. We adopt the data model from XSeek, using the concept of Dewey Label in order to represent XML nodes. When the indexed value is composed of values from more than one XML node (a fragmented instant, for example), it points to the XML node that represents the lowest common ancestor of these nodes.

We also create another index we call `ClosestTemporalNode`, which returns the `ids` of the temporal closest nodes of a given node, as well as the distance between them. We use the follow distance formula:

$$distance = (level_m - level_{lca}) + (level_{nt} - level_{lca})$$

where $level_m$ is the level of the given node, $level_{lca}$ is the level of the lowest common ancestor node between the given node and the candidate temporal node and $level_{nt}$ is the level of the candidate temporal node.

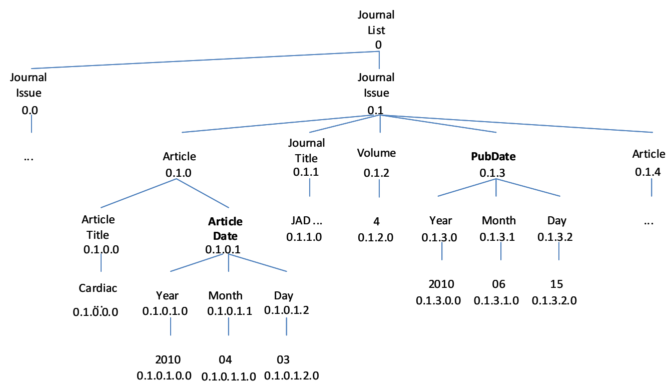


Fig. 7. XML Example

For instance, considering the example presented in Figure 7 that contains 2 temporal nodes (`ArticleDate` and `PubDate`). The distance between `ArticleTitle` and `ArticleDate` is 2 and the distance between `ArticleTitle` and `PubDate` is 3, then the closest node to `ArticleTitle` is `ArticleDate`. To avoid the need to compare each XML node with all temporal nodes, we just consider as candidates temporal nodes those nodes belonging to the same entity of the given node. We use the entity identification defined by XSeek, that infers that a node represents an entity if it corresponds to a *-node in the DTD. Though these inferences do not always hold, they provide good heuristics in the absence of the E-R model. When the schema information is not available, we infer the schema based on data summarization, similar as [Deutsch et al. 1999; Yu and Jagadish 2006]. According to XSeek heuristics, `JournalIssue` and `Article` are entities. Therefore, `ArticleTitle` belongs to the entity `Article`, once that it is its descendant and the unique temporal node belonging to this entity is `ArticleDate`, and then it is the temporal node related to `ArticleTitle`.

It is important to note that the predefined logics used, in the steps 2 and 3, to identify fragmented instants and temporal intervals assume a simplified vision of the XML data model. This is because our focus is data-oriented XML, which are structured and usually exported from a relation database, such as DBLP and Lattes.

4.2.2 *Two Phase Interceptor.* In this section, we briefly describe the Two Phase Interceptor (TPI) that has been defined to handle with the temporal expression of the queries. The TPI has two steps: one extracts the temporal predicate of the queries and the other treats it. Figure 8 illustrates both steps (numbers `(1)` and `(2)`) of the TPI architecture. Figure 8 also shows the indexing module that creates temporal (explained before) and conventional indexes.

When the user poses a temporal query into a conventional XML search engine, the TPI intercepts the query (first step in the figure - `(1)`) to extracts the temporal predicate (`Temporal Predicate Extraction` step in the figure), and the rest of the query (which represents all non-temporal keywords) is sent to the conventional XML search engine, which computes the matches for the rest of the query (`Keywords Matching`). Meanwhile, TPI identifies the temporal relation by analyzing the temporal predicate, which is performed in the `Temporal Predicate Mapping` step. From that, the temporal predicates are matched with the temporal values indexed to the temporal nodes closest to the query (`Temporal Predicate Matching` step). The processor finds in the index structures the temporal nodes closest to each match of each keyword. After, it chooses the nodes with shortest distance. For

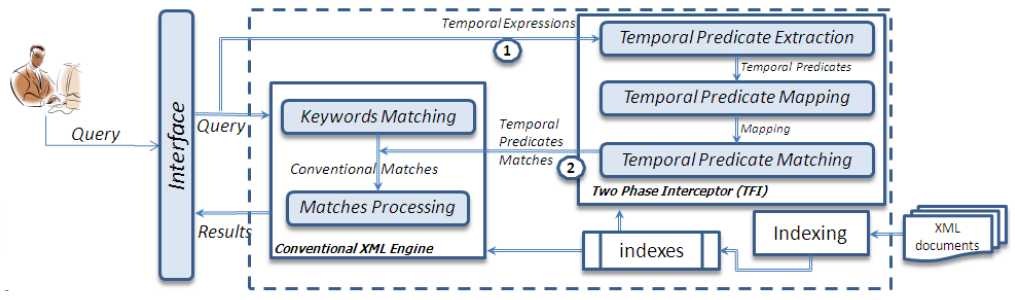


Fig. 8. Detailed Two Phase Interceptor (TPI) module

instance, considering the XML fragment presented in Figure 7, for the query "`ArticleTitle PubDate intersect 2005`", the temporal node closest to `ArticleTitle` is `ArticleDate` with distance 2 and the temporal node closest to `PubDate` is itself with distance 0, then the temporal node closest to the query is `PubDate`. For the query "`ArticleTitle JournalTitle intersect 2005`" the temporal node closest to `ArticleTitle` is `ArticleDate` with distance 2 and the temporal node closest to `JournalTitle` is `PubDate` with distance 2, so `ArticleDate` and `PubDate` are the temporal nodes closest to the query. After finding the temporal nodes closest to the query, the processor chooses just the nodes that satisfy the temporal predicate. The second interception (second step in the figure - `(2)`) happens when the matches of the temporal predicate are sent to the traditional search engine, which are appended to the non-temporal matches. Finally, the following steps are performed by conventional XML keyword search engine without any intervention.

5. EXPERIMENTS

In this section, we describe the experiments we have performed in order to evaluate the effectiveness of our approach. We have run the experiments using two implementation: (i) the XSeek as originally implemented, which we consider our baseline; and (ii) an extension of XSeek, in which we inserted the Two Phase Interceptor module (TPI). For clarity, in this section, the original implementation of XSeek is called baseline, and our proposal of extension is called TPI. We have compared our approach with a conventional XML search engine because we have not found an XML search engine with temporal treatment, and the approaches presented in the related work use structured languages, which always retrieval all and only desired results, because of the expressiveness of structured languages.

In order to analyze the results of experiments, we have evaluated them under the following aspects: (i) recall; (ii) precision; (iii) F1-measure; (iv) query processing; and (v) query scalability. Recall, precision and F1-measure are well known quality measure metrics in IR community [Baeza-Yates and Ribeiro-Neto 1999]. In the query processing aspect we evaluated the time the system has expended on executing the queries. Finally, the scalability is calculated in terms of the number of query keywords, and in terms of the interval size used as second operand.

Since TPI differs from the baseline on handling the temporal part of query, for both tests, time processing and scalability, we compare just the time to perform it. In this way, we compute in TPI the time for mapping the temporal predicate and the time for finding matches for it. In the baseline, we compute the time to find matches to the keywords that compose the temporal instant given, for instance, in query *"country name 10 01 1948"* we compute the time to find the matches to the keywords *"10"*, *"01"* and *"1948"*.

5.1 Setup

We have used Java 1.6 for our implementation and the experiments have been performed on a 2GHZ Intel Core 2 Duo running Ubuntu 9.10, with 3GB of main memory and 120GB of disk space (7200rmp).

Data Sets. We have tested three real XML data sets, Mondial, Pubmed and Lattes.

—Mondial[3] is a world geographic database integrated from CIA World Factbook, TERRA database, among other sources. For our experiments, we have used a 1.8MB document, 4 temporal paths, 0 fragmented instants, 0 temporal intervals, 3247 temporal nodes indexed, and 105178 XML nodes indexed.

—Pubmed[4] is a free digital archive of biomedical and life sciences journal literature. The data set we have used in our experiments is an XML data set obtained as a result from the query *"alzheimer's disease pathology prevention"* submitted to Pubmed. The experiments have been run over a 2.6MB document, 6 temporal paths, 2 fragmented instants, 0 temporal intervals, 2362 temporal nodes indexed, and 72590 XML nodes indexed.

—Lattes[5] contains all the curriculums of Brazilian researchers, which are stored in a centralized database that is hosted in a Federal Government funding agency. The experiments have been executed over a subset of this data set with 6.9MB, 73 temporal paths, 22 fragmented instants, 22 temporal intervals, 6996 temporal nodes indexed, and 214546 XML nodes indexed.

We have tested 32 queries for each data set, created by 4 users. For each query the user filled out the query goal in natural language, the keyword query for our proposal (TQuery) and the keyword query for a conventional XML search engine (CQuery). In the following examples, we can observe one query for each dataset, respectively, for Mondial, Pubmed and Lattes data sets:

```
Query Goal: Return the countries whose independence occurred between 2000 and 2002
TQuery: country name indep_date intersect [2000, 2002]
CQuery: country name indep_date 2000 2001 2002

Query Goal: Return the articles published in 2009
TQuery: articletitle pubdate equal 2009
CQuery: articletitle pubdate 2009

Query Goal: Return the research projects that finishes in 2000
TQuery: RESEARCH-PROJECT finishes 2000
CQuery: RESEARCH-PROJECT FINAL-YEAR 2000
```

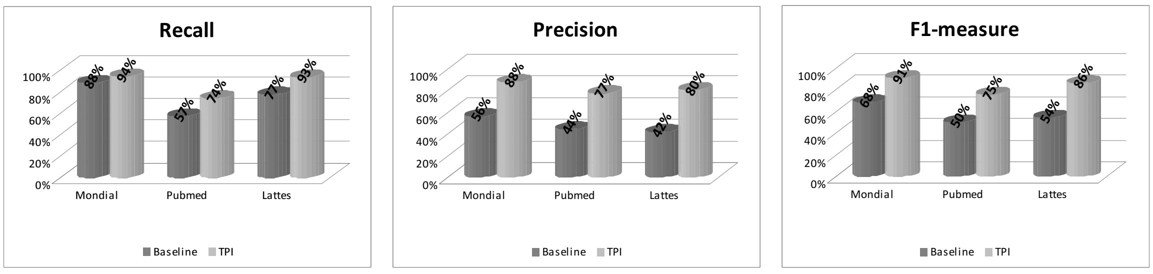
We submitted the TQuery for TPI, CQuery for baseline and used the query goal to manually create queries in the structured XML language XQuery in order to discover the desired data.

5.2 Search Quality

In order to measure the search quality, we have evaluate whether desired data, obtained through queries in XQuery manually created according to the query goals described by users in natural language, are present in the results. We have used precision, recall, and F1-measure metrics. Recall measures the percentage of the desired data that are output. Precision measures the percentage of the output data that are desired. F1-measure is the weighted harmonic mean of recall and precision, where recall and precision are evenly weighted. Figure 9 presents recall, precision and F1-measure averaged for all queries in each dataset.

TPI improves significantly (p-value of T-test < 0.05) the recall, the precision and the F1-measure for temporal queries in all data sets, except the recall of Mondial dataset (p-value of T-test $= 0.16$). That is because baseline treats the temporal expressions the same way that other keywords while TPI applies the temporal constraint just in the temporal nodes closest to the query, supports several temporal operators and granularities, uses the behavior of the temporal path in order to disambiguate the format and identifies fragmented instants and temporal intervals. In Mondial dataset there is no

[3]http://www.cs.washington.edu/research/xmldatasets/
[4]http://www.ncbi.nlm.nih.gov/sites/entrez
[5]http://lattes.cnpq.br/



(a) (b) (c)

Fig. 9. Recall, Precision and F1-measure averaged for all queries of each dataset

statistical significant difference between the recall of the TPI and the recall of the baseline because this dataset does not contain any fragmented instants or temporal intervals. Nevertheless, TPI improves significantly the precision and the F1-measure in this dataset because Mondial has several nodes with numeric value which are not temporal nodes, such as `inflation`, `total_area` and `population`. The baseline considers these nodes as matches for the temporal expressions of the query.

TPI has obtained the largest value in F1-measure in the Lattes dataset. It has happened because Lattes contains several temporal intervals. This is a feature that affects the recall of conventional XML search engines since they are not able to identify that 2007 is a valid value for a temporal node whose initial year is 2006 and final year is 2008. Moreover, they do not have knowledge that a temporal node with an open interval (in this case, the final year is open) means that the information is currently valid. This feature also affects the precision of conventional XML search engines. For instance, the query "`RESEARCH-PROJECT INITIAL-YEAR 2002`", whose goal is to return the research projects that started in 2002, returns both the research projects whose `INITIAL-YEAR` is equal to 2002, considering INITIAL-YEAR as a predicate and the research projects whose FINAL-YEAR is equal to 2002, considering `INITIAL-YEAR` as a desired return node. In TPI, the query would be "`RESEARCH-PROJECT starts 2002`", which apply the temporal constraint just on the INITIAL-YEAR of the research project, once TPI is able to identify and to treat temporal intervals.

In a preliminary experiment, we have tested the application of the temporal constraint in any temporal node, but this procedure dramatically decreases the precision. We agree that in some situations the more relevant nodes may be far from a given keyword. But, we have adopted the strategy of applying the temporal constraint in the temporal nodes closest to the query. However, we report to the user which temporal paths the temporal constraint was applied. If the user is not satisfied with the results, so it is possible to visualize all temporal paths ranked in proximity order with the query keywords. Together with each temporal path, a keyword is presented so that the user must use it in the query to ensure that the temporal constraint will be applied in that temporal path.

Baseline and TPI have had recall and precision averaged for all queries affected by some queries that do not return any significant value due to an erroneous inference of the return nodes. This behavior has happened mainly in Pubmed dataset. For example, there is an element `title` referring to the title of the journal and an element `articletitle` referring to the title of the article. In some queries, users have used the keyword `title` in order to return the title of the article, so it was inferred that the user wanted the title of the journal and no article title was returned. In the query "`country name government republic indep_date last 3 decades`", whose goal is to return the country names whose government is republic and its independence occurred in the last 3 decades, some countries whose government is not republic was returned when "`Republic`" is a part of their name, such as `Czech Republic`. These problems are inherent of the XML search engine used. It is important to note that our approach is independent of XML search engine used. We also emphasize that even in these cases, TPI always applies the temporal restriction on the correct node. A detected problem in TPI is the identification of return nodes, which is not the focus of this work.

5.3 Time processing

Figure 10 presents the time processing averaged for all queries in each dataset. Although TPI adds a new layer to the search engine querying processing, what normally increases the time processing, in Pubmed and Lattes datasets, the time processing cost was lower. This is due to the fact that in queries having keywords that represent periods as second operand, the query for the baseline needs to contain one temporal expression for each instant, then the baseline needs to traverse the index for each temporal expression. On the other hand, in queries with instant as second operand, where just one temporal expression is posed in the query, the time processing cost of TPI has been lower too. This is justified by the fact that TPI searches for matches of temporal predicate in an index that contains only values of temporal nodes, while the baseline looks for matches of keywords that compose the temporal instant in all XML nodes.

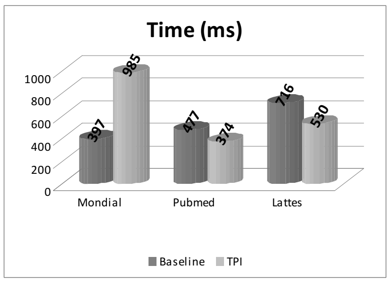


Fig. 10. Time to handle the temporal part of query

In Mondial data set, TPI has had a worst result on those queries that have a simple instant as second operand. In this case, some keywords have several matches (because of the dataset structure) and, as we can see in the next section, the processing time of TPI increases linearly when the number of matches for a query increases. For instance, `country` is a child element of `mondial` and an attribute of `city`, `border`, `province`, `members`, `located`, etc.

5.4 Scalability

For testing scalability, we have used only the Pubmed data set over two parameters: query size and interval size.

Number of Keywords: the experiments have been performed using those queries with an increasing number of non-temporal keywords, a constant temporal operator and a second operand. We have defined two different cases. In the first case (Figure 11(a)), all **keywords** have the same number of the matches (720), while in second case (Figure 11(b)), all **queries** have the same number of the matches (720), i.e., for a query having 1 non-temporal keyword, this keyword has 720 matches and a query with 8 non-temporal keywords, each keyword has 90 matches. Furthermore, for both cases, all keywords refer to the same temporal node and have the same distance from this node. We define eight queries to be tested, from "`ArticleTitle intersect 2010`" to "`ArticleTitle Language Affiliation Abstract Pagination Journal AuthorList PublicationTypeList intersect 2010`". Queries performed in the baseline do not have the explicit temporal operator `intersect`.

In the second case, the processing time, of both approaches, is constant when the number of the keywords increases, keeping constant the total number of matches. In the first case, the processing time of the baseline is constant while the processing time of TPI increases linear when the number of matches increases. This is because the baseline only finds matches to temporal expression on the XML document, while TPI analysis all other keywords matches to identify the temporal node closest to the non-temporal keywords.

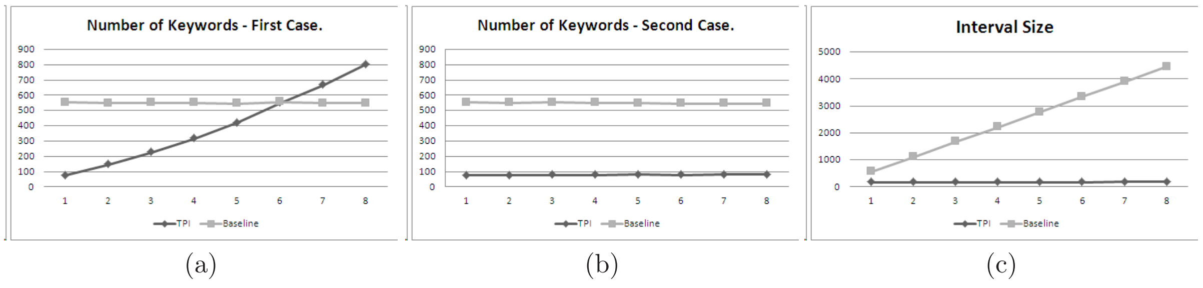


Fig. 11. Scalability on the PubMed

Interval Size: the experiments have been performed using those queries with an increasing interval size, a constant number of non-temporal keywords, and a constant number of matches for each non-temporal keyword. All keywords refer to the same temporal node and have the same distance from this node. We have defined eight queries to be tested, from "`ArticleTitle Language intersect [2000,2000]`" to "`ArticleTitle Language intersect [2000,2007]`" in TPI and from "`ArticleTitle Language 2000`" to "`ArticleTitle Language 2000 2001 ... 2006 2007`" in the baseline. As we can see (Figure 11(c)), the processing time of TPI is constant while the processing time of baseline is linear when the interval size increases. It happens because, for any interval size, TPI requires the same structure, while the baseline requires one temporal expression as keyword for each instant of the interval. For example, the query "`ArticleTitle Language intersect [2000,2002]`", executed over TPI, must be written with three temporal expressions as keywords in the baseline: "`ArticleTitle Language 2000 2001 2002`". Therewith, baseline needs to traverse the conventional index for each instant (2000, 2001 and 2002) while TPI traverses the temporal index only once. TPI and baseline traverse the conventional index for each non-temporal keyword.

In summary, TPI has an improved search quality compared with our baseline (XSeek) by intercepting the query and performing the special treatment to the temporal predicate. The processing time overhead, for this treatment, is reasonable. TPI scales well (constant) when the query size and the interval size increase and it scales reasonable (linear) when the number of query matches increases.

5.5 Discussion

We have empirically shown that TPI keeps the overall system performance and significantly improves the retrieval quality. Then, we believe that our method contributes to practical use. This is because, TPI performs a special treatment of temporal information present at the queries and the XML documents contents. Besides the advantage of allowing several temporal operators, some reserved words and different granularities, our proposal also differs from conventional XML keyword search engines once that takes into account the values pattern in the XML paths to decide if a node is temporal or not, to identify fragmented instants and temporal intervals. Other proposals (conventional XML keyword search engines) treat the temporal expressions the same way that other keywords, i.e., each text node that contains that keyword is considered a match, including dates, prices, codes, heights, or any other numeric value. Moreover, they just consider each isolated value for computing the matches for the temporal expressions without format disambiguation and there is no fragmented instant and temporal interval identification.

TPI does not recover always all and only desired results as extensions of temporal structured languages presented in related work, however these structured languages require that the user learn a complex query language and needing a large prior knowledge of the structure of the underlying data while keyword search is a user-friendly information discovery technique that has been extensively studied for text [Bhalotia et al. 2002; Hristidis et al. 2006; Liu and Chen 2007].

6. CONCLUSIONS

In this paper, we highlight the problem of supporting temporal queries on conventional XML keyword search engines describing an analysis on web searches logs and proposing an implementation that executes an interception in a XML search engine processor. We have analyzed how the temporal predicates are used in temporal queries in order to create a temporal predicate classification. In order to identify temporal information in the XML documents, we used the behavior of the paths and some heuristics to perform path disambiguation, fragmented instant and temporal interval identification. We have executed some experiments over three real XML data sets that show our proposal improves significantly the precision of the XSeek for temporal queries. In this paper, we only treat queries of temporal selection, where the user uses temporal expressions to restrict the query results. As future work we intend to support queries of temporal output, where the user is interested in discovering when a given event occurred. For instance, "When did Brazil win the World Cup?" or "for how long ...".

REFERENCES

Allen, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26 (11): 832–843, November, 1983.

ANNIE. Open source information extraction, 2009. <http://www.aktors.org/technologies/annie/>.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. Keyword searching and browsing in databases using banks. In *Proceedings of the International Conference on Data Engineering*. San Jose, USA, pp. 431–440, 2002.

Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y. Xsearch: A semantic search engine for xml. In *Proceedings of the International Conference on Very Large Data Bases*. Berlin, Germany, pp. 45–56, 2003.

Deutsch, A., Fernández, M. F., and Suciu, D. Storing semistructured data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Philadelphia, USA, pp. 431–442, 1999.

Gao, D. and Snodgrass, R. T. Temporal slicing in the evaluation of xml queries. In *Proceedings of the International Conference on Very Large Data Bases*. Berlin, Germany, pp. 632–643, 2003.

Graells, E. and Baeza-Yates, R. A. Evolution of the chilean web: A larger study. In *Proceedings of the Latin American Web Conference*. Vila Velha, Brazil, pp. 108–114, 2008.

Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. XRANK: ranked keyword search over xml documents. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. San Diego, USA, pp. 16–27, 2003.

GUTime. Adding TIMEX3 tags, 2009. <http://www.timeml.org/site/tarsqi/modules/gutime/index.html>.

Hristidis, V., Koudas, N., Papakonstantinou, Y., and Srivastava, D. Keyword proximity search in xml trees. *IEEE Transactions on Knowledge and Data Engineering* 18 (4): 525–539, 2006.

Li, Y., Yang, H., and Jagadish, H. V. NaLIX: A generic natural language search environment for xml data. *ACM Transactions on Database Systems* 32 (4): 30:1–30:44, 2007.

Liu, Z. and Chen, Y. Identifying meaningful return information for xml keyword search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Beijing, China, pp. 329–340, 2007.

Mendoza, M. and Baeza-Yates, R. A. A web search analysis considering the intention behind queries. In *Proceedings of the Latin American Web Conference*. Vila Velha, Brazil, pp. 66–74, 2008.

Nunes, S., Ribeiro, C., and David, G. Use of temporal expressions in web search. In *Proceedings of the European Conference on Information Retrieval*. Glasgow, UK, pp. 580–584, 2008.

Rizzolo, F. and Vaisman, A. A. Temporal xml: modeling, indexing, and query processing. *The VLDB Journal* 17 (5): 1179–1212, 2008.

Tyler, S. K. and Teevan, J. Large scale query log analysis of re-finding. In *Proceedings of the International Conference on Web Search and Web Data Mining*. New York, USA, pp. 191–200, 2010.

Wang, Y., Zhu, M., Qu, L., Spaniol, M., and Weikum, G. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from wikipedia. In *Proceedings of the International Conference on Extending Database Technology*. Lausanne, Switzerland, pp. 697–700, 2010.

Xu, Y. and Papakonstantinou, Y. Efficient keyword search for smallest lcas in xml databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Baltimore, USA, pp. 537–538, 2005.

Yu, C. and Jagadish, H. V. Schema summarization. In *Proceedings of the International Conference on Very Large Data Bases*. Seoul, Korea, pp. 319–330, 2006.