Querying Linguistic Annotations

Sumukh Ghodke and Steven Bird

Department of Computer Science and Software Engineering
University of Melbourne, Victoria 3010, Australia

{sghodke,sb}@csse.unimelb.edu.au

Abstract *Over the past decade, a variety of expressive linguistic query languages have been developed. The most scalable of these have been implemented on top of an existing database engine. However, with the arrival of efficient, wide-coverage parsers, it is feasible to parse text on a scale that is several orders of magnitude larger. We show that the existing database approach will not scale up, and speculate on a new approach that leverages proximity search in the context of an IR engine. We also propose a simple syntax for querying linguistic annotations, avoiding the usability problems with existing tree query languages.*

Keywords Information Retrieval, Natural Language Techniques and Documents, XML Document Standards

1 Introduction

High quality part-of-speech taggers and syntactic parsers are able to annotate large quantities of English text [5]. With suitable indexing methods, it should be possible for users to express queries that are sensitive to this additional information, and support more focussed search.

In some cases, the part-of-speech of a word may disambiguate the primary senses of the word, e.g. *wind*, *park*. A user could easily select the intended POS-tag using a query format like: `wind/N` or `park/V`. In other cases we want to do a proximity search but need to constrain the syntax of the intervening material, e.g. `give NP up` will find instances of "give up" wrapped around a noun phrase (NP), and won't include results which have other intervening material.

Expecting users to annotate their queries adds a significant burden, without guaranteeing that the result will be sufficiently improved to justify the effort. However, we hypothesise that a specialised query engine can cluster results from a conventional ad-hoc query using extra information present in the linguistic annotations. Exemplars from each cluster could be presented to the user, together with the annotations that characterise each cluster. In this way, users are educated about the relevant linguistic properties and can start to annotate their own queries.

Proceedings of the 13th Australasian Document Computing Symposium, Hobart, Australia, 8 December 2008.

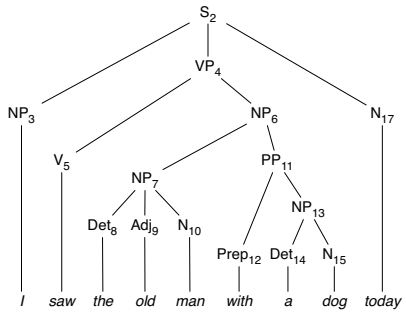

Figure 1: Syntax Tree

For example, an ad hoc query for documents concerning acquisitions of QANTAS mostly returns documents concerning acquisitions *by* QANTAS. The linguistic difference between these two cases is the grammatical role of QANTAS relative to the main verb (subject vs object). If this difference could be discovered, a user would be able to look through results of the refined query: `"acquire QANTAS/NP-OBJ"`. Similar analysis could detect differences in tense and cluster the results as pertaining to the past or the future.

This paper explores the suitability of existing work on linguistic tree query as the basis for such a query engine. The paper is organised as follows. First we give an overview of existing work on linguistic tree query (§2) and XML indexing (§3). Our scaling experiments are presented in §4. Our negative conclusions about scaling lead to a more speculative discussion (§5) on the prospects for using an IR engine for performing the desired query tasks.

2 Linguistic Tree Query

The problem of representing and querying linguistic annotations has been an active area of research for several years [3, 7]. It has grown out of work on curating large databases of annotated text such as *treebanks* [10] for use in developing and testing language technologies. Figure 1 gives an example of a parsed sentence; it represents the constituent structure of a sentence, and involves non-terminal nodes for noun phrases, verb phrases, prepositional phrases, and so on. Trees also encode relationships between these constituents, and permit us, amongst other things, to discover the subject and object noun phrases corresponding to a particular verb.

At least a dozen linguistic tree query languages have been developed for interrogating treebanks (see [8] for a survey). One of these languages, called LPath, extends XPath [4] with extra navigational operators tailored to the needs of linguistic tree query [2].

The syntax of such languages is arcane, and we would need to provide a more accessible, high-level syntax for use by a non-specialised audience. For instance, a high-level query for the word *wind* used as a noun, `wind/N`, could be automatically translated to the LPath expression `//N[@lex="wind"]` (and then compiled into SQL for execution). The high-level query `"acquire QANTAS/NP-OBJ"` could be translated into the LPath expression `//_[@lex="acquire"] -> _[@lex="QANTAS"]\\NP-OBJ`.

A more serious issue is scalability. Treebanks typically contain millions of words; the Penn Treebank, for instance, has 4.5 million words [10]. The scale of data on the Web is several orders of magnitude larger again. We will explore the scalability of this approach to querying linguistic annotations using the Penn Treebank, using multiple copies when necessary in order to simulate larger data sizes.

3 Indexing Hierarchical Data in Databases

Many approaches to storing hierarchical data have been carefully analysed in recent years. In the past, relational databases were preferred to specialised semi-structured database approaches, since the relational formalisms were more mature and offered superior performance [14].

However, more recently, several features found in relational databases have been incorporated into native XML databases, and indexes have been designed specifically for XML data. Commercial relational databases such as Oracle and DB2 now support native XML storage and retrieval, albeit with widely varying indexing and query evaluation techniques. Oracle uses a hybrid approach to store XML within relational tables [9], while DB2 allows XML data to be stored natively and builds value and full text indexes over them [12]. Both offer varying levels of XQuery support, and a primitive XML datatype called XMLType which can be used across all queries.

Yet another native XML database is the eXist open source database. It builds three primary indexes on XML data: the structure, range and full-text indexes [11]. A structure index is similar to an inverted index of all nodes and attributes of XML documents along with their document and node ids. The node ids help in identifying hierarchical and sibling relationships without tree traversal. Range indexes permit comparisons based on typed values while full-text indexes support queries over sequences of words or tokens.

Hierarchical data can be decomposed into sets of relations and stored in a relational database. In our experiments, we store all the elements and attributes in a single node relation, as it is best suited for a diverse structure such as that present in annotated linguistic data. A common feature linking such a relational representation and the eXist database representation is the evaluation of path expressions by decomposing them into smaller components.

Each path expression is treated as sequence of elements interleaved with operators such as *parent*, *child*, *ancestor*, or other navigational constructs. These expressions are evaluated by converting the path sequence into one or more binary expressions involving a single operator. Indexes are used to search for matching elements on either side of the binary expression and the resulting sets are reduced using a join based on the operator. This join operation is termed the structural join and several optimisations have been proposed to improve the efficiency of such joins [1].

4 Experiments

In this section we describe the experimental setup used to evaluate performance of databases for linguistic queries. We ran the experiments on an Intel core 2 Duo 2.4 GHz processor with 2 GB of RAM, running openSUSE Linux 11.0. The task of choosing the right database system and optimising parameter settings for each of the experiments introduce multiple variables within the experiment. As our study attempts to highlight the scalability of particular systems rather than compare relative performance, we feel that fine tuned optimisations will not drastically change our observations on scalability. Instead we focus on scalability by varying the size of the datasets.

Annotated texts from the Wall Street Journal section of the Penn Treebank corpus were used in our experiments. This corpus contains around 50,000 sentences annotated with POS and syntactic tags. In order to study the variation of performance with the size of the datasets we either selected a subset of the corpus, or used multiple copies to simulate larger datasets.

Tests using the relational approach use the Oracle 11g Standard Edition, while the eXist XML database ver-1.2.2 is used in the native XML approach. The relational database schema contains a node relation storing all elements and attributes of the treebank. This schema is similar to the one used by LPath to query treebanks and more details can be found in Bird et al.'s work on LPath [2]. For the XML database approach we store each sentence as a separate XML document.

The LPath queries used during evaluation are listed in Table 3. These queries were converted to SQL for the relational database and into XQuery for the XML database. Some of these queries include highly selective nodes, while others search for commonly occurring terms. Queries 3–6 evaluate the effects of a simple join on nodes with varying selectivity. Queries 7 and 8 find the occurrences of words within the treebank irrespective of their POS tag. Other queries include features like scoping, edge alignment, and negation; all

Table 1: Query execution time in Native XML DB

	Dataset sizes					
	500	5k	∼25k	∼50k	∼100k	∼200k
1	.06	.51	**.99**	**1.99**	**4.59**	**16.29**
2	.01	.12	.60	.73	1.23	3.13
3	.08	.24	.95	1.90	5.80	20.58
4	.08	.11	.86	2.32	7.98	29.81
5	.01	.01	.50	1.01	1.99	2.63
6	.03	.05	.40	.87	7.20	23.62
7	.38	.83	**3.98**	**9.38**	**33.26**	**156.60**
8	.10	.63	**4.17**	**9.31**	**29.93**	**116.36**
9	.19	.82	2.16	3.73	31.24	97.72
10	.82	2.36	9.24	17.43	43.28	86.90
11	.17	.54	2.06	4.23	17.48	76.28
12	.64	3.20	**14.85**	**27.99**	**58.19**	**160.28**
13	.03	.17	1.19	2.43	10.45	37.95

Table 2: Query execution time in Relational DB

	Dataset sizes					
	500	5k	∼25k	∼50k	∼100k	∼200k
1	.10	.32	1.29	2.48	4.80	9.46
2	.07	.07	.08	.07	.07	.11
3	.08	.18	.63	1.20	2.35	4.94
4	.10	.35	1.47	**2.83**	**5.54**	**12.15**
5	.07	.08	.07	.07	.08	.13
6	.07	.09	.16	.24	.40	.13
7	.18	1.07	5.04	9.94	19.93	6.79
8	.07	.07	.08	.07	.07	.12
9	.09	.16	.49	**.90**	**1.65**	**12.33**
10	.08	.14	.41	.76	1.30	2.75
11	.10	.29	1.17	**2.15**	**4.12**	**12.11**
12	.08	.12	.30	.55	.90	2.09
13	.08	.08	.09	.10	.10	.18

features commonly found in linguistic queries. Adjacency is another common linguistic query operator and is represented by queries 12 and 13.

In the Oracle setup, the buffer cache and shared pool of the database were cleared after every query, but the first run always took the greatest time to execute. Subsequent queries had stable and repeatable query times. Table 2 lists the minimum time (in seconds) taken by each query over a sample of 3 runs. The eXist queries seemed to perform more uniformly between runs, but random slowdowns were observed in some cases. Each value in Table 1 corresponds to the minimum execution time (in seconds) of 3 consecutive runs of each query, on the eXist database. The minimum execution time was chosen instead of an average to avoid including random slowdown times in the measurement.

One of the main observations in the eXist experiment was that the rate of increase in execution time increases with dataset size; especially for larger datasets. For some queries, when we double the size of the dataset from 100k to 200k sentences, the time taken by eXist increases drastically; see the results for queries 1, 7, 8 and 12 in Table 1, where values of interest appear in boldface. However, these queries do not display such a trend in the Oracle setup. On

Table 3: Test Queries

	LPath query
1	`//NP`
2	`//PP_LOC_MNR`
3	`//NP/NP`
4	`//NP//NP`
5	`//RRC/PP_TMP`
6	`//VP/PP_TMP`
7	`//_[@lex=saw]`
8	`//_[@lex=rapprochement]`
9	`//VP{//VB-->NN}`
10	`//VP//NP$`
11	`//NP[not(//JJ)]`
12	`//NP=>NP`
13	`//ADVP=>ADJP`

further inspection, we can see that queries 7 and 8 probably involve simple index lookup. It is unclear why these queries exhibit such an increase in spite of using the full-text index in eXist. The poor scaling behaviour of Query 1 and 12 could be attributed to the low selectivity of NP.

Queries 6 and 9 are exceptions to the pattern mentioned above; their execution times grows by a factor of 8–10 between 50k and 100k sentence datasets, but drops to a factor of 3 for larger datasets. A plausible explanation for this behaviour could be the thrashing of memory, caused by intermediate object creations. Query 5 is very similar to query 6 in its construction but does not exhibit such a phenomenon as it is composed of high selectivity elements. Overall, the query times in eXist almost always seem to increase by a factor of 3–5 from 100k to 200k sentences.

A linear increase in time was observed in fewer Oracle tests when compared to eXist. High selectivity queries in Oracle displayed almost constant execution time, indicating optimal use of indexes. However, query 4, 9 and 11 (in boldface), show an increase in execution time with the size of the dataset.

5 Searching Linguistic Annotations Using an IR Engine

From our experiments we observe that the database approach using structural joins does not scale for queries containing low selectivity elements. To address this shortcoming we intend to evaluate systems where paths are indexed in their entirety and not as a combination of element pairs. One such approach has been proposed by Cooper et. al., where paths are inserted into an indexing data structure called Index Fabric [6]. Paths from the root to each of the leaf nodes of every tree are treated as string sequences and are inserted into the data structure.

Patricia tries form a core component of Index Fabric. They are unbalanced structures and not very efficient for main memory operations. Hence, in Index Fabric, access to different fragments of a trie is broken down into multiple layers. Pointers are created at each node to navigate to deeper tree fragments within lower

layers. A multi-layered approach balances the overall data-structure and results in a constant number of I/O look-ups for all searches. The Patricia trie is also known to use an aggressive key compression algorithm, making it a scalable architecture for large number of keys and for paths of varied lengths. The only drawback of this approach is that it suits queries where the paths are defined from the root to the leaves or for selected predefined paths, and not for arbitrary partial expressions.

An alternative approach – using a combination of IR and database systems – has been developed by Park et al. and is known as XIR. Here, paths are treated as sentences, and individual elements in a path form the words within the sentence [13]. They identify paths as representations of the document schema and hence are indexed independently from the data, which is considered to occupy only the leaf nodes. Each unique path is stored in a path table, while individual elements comprising the paths are indexed in an inverted index. Every unique element appears in the inverted index with a postings list containing information regarding the element's occurrence in different paths, an offset indicating its relative position within the path and the total length of the path. The data and element information is stored in a separate table with document and node identifiers.

The two key contributions of the XIR system include the concept of using inverted indexes to search path expressions and the conversion of path queries into equivalent IR style proximity searches. For instance, a query such as `VP/NP/NNP`, which searches for a singular proper noun phrase (`NNP`) in the specified hierarchical relationship with a noun phrase (`NP`) and verb phrase (`VP`), could be converted into an IR query where the *near* operator specifies the proximity relation: `VP near(1) NP near(1) NNP`. Similarly, a descendent query `VP//NP`, could be rewritten as `VP` near(∞) `NP`, indicating that the second element can appear anywhere after the first element in the path string. Once the set of paths is known from the inverted index, a select query on the data and element information table retrieves the final results.

For linguistic queries, sequential navigation is as significant as hierarchical navigation. We expect that by extending the XIR approach, we could create independent indexes for hierarchical and sequential relationships in a document. Queries containing hierarchical and sequential expressions would be converted into appropriate proximity queries and the resulting nodes could be reduced by a join operation. This method would essentially reduce the number and cardinality of joins, as they would occur only when the expression changes from path to sequence or vice-versa and not at every element in the expression.

As a part of ongoing work in this area, we would also like to compare the performance of such a system with a pure database implementation containing an indexing algorithm similar to Index Fabric.

Acknowledgements

We gratefully acknowledge the support of Dr A. Kumaran and Microsoft Research India.

References

[1] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava and Yuqing Wu. Structural joins: a primitive for efficient XML query pattern matching. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 141, Washington, DC, USA, 2002. IEEE Computer Society.

[2] Steven Bird, Yi Chen, Susan B. Davidson, Haejoong Lee and Yifeng Zheng. Designing and evaluating an XPath dialect for linguistic queries. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 52, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Steven Bird and Jonathan Harrington (editors). *Speech Communication: Special Issue on Speech Annotation and Corpus Tools*, Volume 33 (1–2). Elsevier, 2001.

[4] James Clark and Steve DeRose. *XML Path language (XPath)*. W3C, 1999. `http://www.w3.org/TR/xpath`.

[5] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, Volume 33, Number 4, pages 493–552, 2007.

[6] Brian Cooper, Neal Sample, Michael J. Franklin, Gisli R. Hjaltason and Moshe Shadmon. A fast index for semistructured data. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 341–350, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[7] Stephan Kepser. Finite structure query: a tool for querying syntactically annotated corpora. In *Proceedings of the Tenth Conference of the European Chapter of the Association for Computational Linguistics*, pages 179–186, 2003.

[8] Catherine Lai and Steven Bird. Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop*, pages 139–146, 2004.

[9] Zhen Hua Liu, Muralidhar Krishnaprasad and Vikas Arora. Native XQuery processing in oracle XMLDB. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 828–833, New York, NY, USA, 2005. ACM.

[10] Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, Volume 19, Number 2, pages 313–30, 1993.

[11] Wolfgang Meier. *Web, Web-Services, and Database Systems*, Volume Volume 2593/2008, Chapter eXist: an open source native XML database, pages 169–183. Springer Berlin / Heidelberg, 2008.

[12] Matthias Nicola and Bert van der Linden. Native XML support in DB2 universal database. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1164–1174. VLDB Endowment, 2005.

[13] Young-Ho Park, Kyu-Young Whang, Byung Suk Lee and Wook-Shin Han. Efficient evaluation of partial match queries for XML documents using information retrieval techniques. *Database Systems for Advanced Applications*, pages 95–112, 2005.

[14] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt and Jeffrey F. Naughton. Relational databases for querying XML documents: limitations and opportunities. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.