Supporting Annotation Layers for Natural Language Processing

Preslav Nakov, Ariel Schwartz, Brian Wolf
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
{nakov,sariel}@cs.berkeley.edu

Marti Hearst
SIMS
University of California, Berkeley
Berkeley, CA 94720
hearst@sims.berkeley.edu

Abstract

We demonstrate a system for flexible querying against text that has been annotated with the results of NLP processing. The system supports self-overlapping and parallel layers, integration of syntactic and ontological hierarchies, flexibility in the format of returned results, and tight integration with SQL. We present a query language and its use on examples taken from the NLP literature.

1 Introduction

Today most natural language processing (NLP) algorithms make use of the results of previous processing steps. For example, a word sense disambiguation algorithm may combine the output of a tokenizer, a part-of-speech tagger, a phrase boundary recognizer, and a module that classifies noun phrases into semantic categories. Currently there is no standard way to represent and store the results of such processing for efficient retrieval.

We propose a framework for annotating text with the results of NLP processing and then querying against those annotations in flexible ways. The framework includes a query language and an indexing architecture for efficient retrieval, built on top of a relational database management system (RDBMS). The model allows for both hierarchical and overlapping layers of annotation as well as for querying at multiple levels of description.

In the remainder of the paper we describe related work, illustrate the annotation model and the query language and describe the indexing architecture and the experimental results, thus showing the feasibility of the approach for a variety of NLP tasks.

2 Related Work

There are several specialized tools for indexing and querying treebanks. (See Bird et al. (2005) for an overview and critical comparisons.) *TGrep2*[1] is a a grep-like utility for the Penn Treebank corpus of parsed Wall Street Journal texts. It allows Boolean expressions over nodes and regular expressions inside nodes. Matching uses a binary index and is performed recursively starting at the top node in the query. *TIGERSearch*[2] is associated with the German syntactic corpus TIGER. The tool is more typed than TGrep2 and allows search over discontinuous constituents that are common in German. TIGERSearch stores the corpus in a Prolog-like logical form and searches using unification matching. *LPath* is an extension of XPath with three features: immediate precedence, subtree scoping and edge alignment. The queries are executed in an SQL database (Lai and Bird, 2004). Other tree query languages include *CorpusSearch*, *Gsearch*, *Linguist's Search Engine*, *Netgraph*, *TIQL*, *VIQTORYA* etc.

Some tools go beyond the tree model and allow multiple intersecting hierarchies. *Emu* (Cassidy and Harrington, 2001) supports sequential levels of annotations over speech datasets. Hierarchical relations may exist between tokens in different levels, but precedence is defined only between elements within the same level. The queries cannot

[1] http://tedlab.mit.edu/∼dr/Tgrep2/
[2] http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/

express immediate precedence and are executed using a linear search. *NiteQL* is the query language for the MATE annotation workbench (McKelvie et al., 2001). It is highly expressive and, similarly to TIGERSearch, allows quering of intersecting hierarchies. However, the system uses XML for storage and retrieval, with an in-memory representation, which may limit its scalability.

Bird and Liberman (2001) introduce an abstract general annotation approach, based on *annotation graphs*.[3] The model is best suited for speech data, where time constraints are limited within an interval, but it is unnecessarily complex for supporting annotations on written text.

3 The Layered Query Language

Our framework differs from others by simultaneously supporting several key features:

- Multiple overlapping layers (which cannot be expressed in a single XML file), including self-overlapping (e.g., a word shared by two phrases from the same layer), and parallel layers, as when multiple syntactic parses span the same text.
- Integration of multiple intersecting hierarchies (e.g., MeSH, UMLS, WordNet).
- Flexible results format.
- Tight integration with SQL, including application of SQL operators over the returned results.
- Scalability to large collections such as MEDLINE (containing millions of documents).[4]

While existing systems possess some of these features, none offers all of them.

We assume that the underlying text is fairly static. While we support addition, removal and editing of annotations via a Java API, we do not optimize for efficient editing, but instead focus on compact representation, easy query formulation, easy addition and removal of layers, and straightforward translation into SQL. Below we illustrate our *Layered Query Language (LQL)* using examples from bioscience NLP.[5]

[3]http://agtk.sourceforge.net/

[4]http://www.nlm.nih.gov/pubs/factsheets/medline.html

[5]See http://biotext.berkeley.edu/lql/ for a formal description of the language and additional examples.

Figure 1 illustrates the layered annotation of a sentence from biomedical text. Each annotation represents an interval spanning a sequence of characters, using absolute beginning and ending positions. Each layer corresponds to a conceptually different kind of annotation (e.g., word, gene/protein[6], shallow parse). Layers can be *sequential*, *overlapping* (e.g., two concepts sharing the same word), and *hierarchical* (either *spanning*, when the intervals are nested as in a parse tree, or *ontologically*, when the token itself is derived from a hierarchical ontology).

Word, POS and shallow parse layers are *sequential* (the latter can skip or span multiple words). The gene/protein layer assigns IDs from the LocusLink database of gene names.[7] For a given gene there are as many LocusLink IDs as the number of organisms it is found in (e.g., 4 in the case of the gene Bcl-2).

The MeSH layer contains entities from the hierarchical medical ontology MeSH (Medical Subject Headings).[8] The MeSH annotations on Figure 1 are overlapping (share the word *cell*) and hierarchical both ways: *spanning*, since *blood cell* (with MeSH id D001773) orthographically spans the word *cell* (id A11), and *ontologically*, since *blood cell* is a kind of *cell* and *cell death* (id D016923) is a kind of *Biological Phenomena*.

Given this annotation, we can extract potential protein-protein interactions from MEDLINE text. One simple approach is to follow (Blaschke et al., 1999), who developed a list of verbs (and their derived forms) and scanned for sentences containing the pattern PROTEIN ... INTERACTION-VERB ... PROTEIN. This can be expressed in LQL as follows:

```
FROM
[layer='sentence' { ALLOW GAPS }
  [layer='protein'] AS prot1
  [layer='pos' && tag_type="verb" &&
    content='activates']
  [layer='protein'] AS prot2
] SELECT prot1.content, prot2.content
```

This example extracts sentences containing a protein name in the gene/protein layer, followed by any sequence of words (because of ALLOW GAPS), followed by the interaction verb *activates*, followed by any sequence of words, and finally followed by an-

[6]Genes and their corresponding proteins often share the same name and the difference between them is often elided.

[7]http://www.ncbi.nlm.nih.gov/LocusLink

[8]http://www.nlm.nih.gov/mesh/meshhome.html

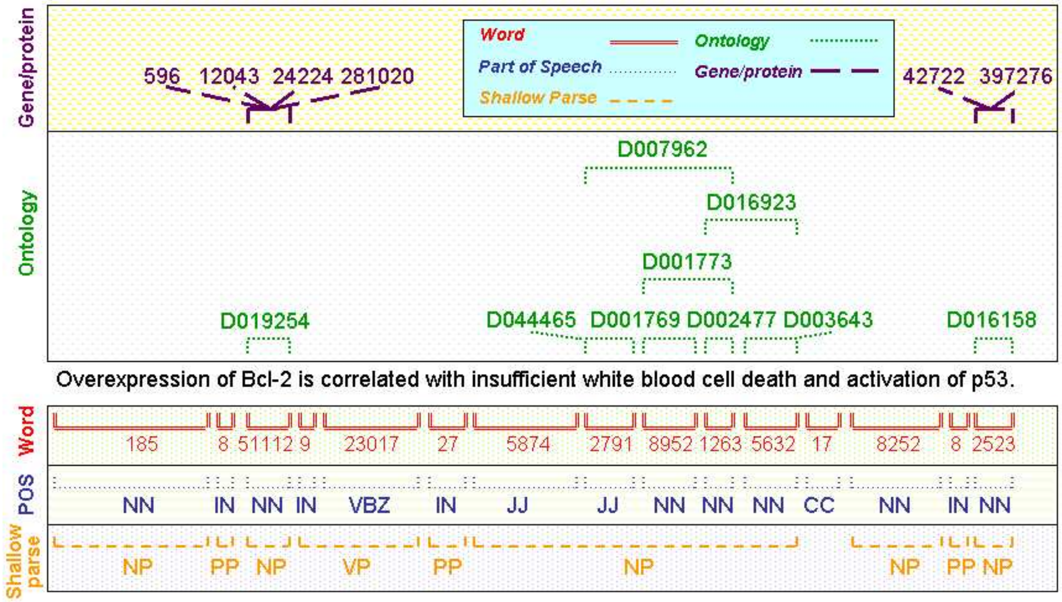

Figure 1: Illustration of the annotation layers. The *full parse*, *sentence* and *section* layers are not shown.

other protein name. All possible protein matches within the same sentence will be returned. The results are presented as pairs of protein names.

Each query level specifies a layer (e.g., sentence, part-of-speech, gene/protein) and optional restrictions on the attribute values. A binding statement is allowed after the layer's closing bracket. We can search for more than one verb simultaneously, e.g., by changing the POS layer of the query above to `[layer='pos' && (content='activates' || content='inhibit' || content='binds')]`. Further, a wildcard like `content ~ 'activate%'` can match the verb forms *activate*, *activates* and *activated*. We can also use double quotes " to make the comparison case insensitive. Finally, since LQL is automatically translated into SQL, SQL code can be written to surround the LQL query and to reference its results, thus allowing the use of SQL operators such as `GROUP BY`, `COUNT`, `DISTINCT`, `ORDER BY`, etc., as well as set operations like `UNION`.

Now consider the task of extracting interactions between chemicals and diseases. Given the sentence *"Adherence to statin prevents one coronary heart disease event for every 429 patients."*, we want to extract the relation that *statin* (potentially) prevents *coronary heart disease*. The latter is in the MeSH hierarchy (id D003327) with tree codes C14.280.647.250 and C14.907.553.470.250, while the former is listed in the MeSH supplementary concepts (ID C047068). In fact, the whole C subtree in MeSH contains diseases and all supplementary MeSH concepts represent chemicals. So we can find potentially useful sentences (to be further processed by another algorithm) using the following query:

```
FROM
[layer='sentence' {NO ORDER, ALLOW GAPS}
 [layer='shallow_parse' && tag_type='NP'
  [layer='chemicals'] AS chem $
 ]
 [layer='shallow_parse' && tag_type='NP'
  [layer='MeSH' && label BELOW "C"] AS dis $
 ]
] AS sent
SELECT chem.content,dis.content,sent.content
```

This looks for sentences containing two NPs in any order without overlaps (`NO ORDER`) and separated by any number of intervening elements. We further require one of the NPs to *end* (ensured by the `$` symbol) with a chemical, and the other (the disease) to end with a MeSH term from the C subtree.

4 System Architecture

Our basic model is similar to that of TIPSTER (Grishman, 1996): each annotation is stored as a record, which specifies the character-level beginning and ending positions, the layer and the type. The basic table[9] contains the following columns: **(1) annotation_id**; **(2) doc_id**; **(3) section**: *title*, *abstract* or *body*; **(4) layer_id**: layer identifier (*word*, *POS*, *shallow parse*, *sentence*, etc.); **(5) start_char_pos**: beginning character position, relative to *section* and *doc_id*; **(6) end_char_pos**: ending character position; **(7) tag_type**: a layer-specific token identifier.

After evaluating various different extensions of the structure above, we have arrived at one with some additional columns, which improves cross-layer query performance: **(8) sentence_id**; **(9) word_id**; **(10) first_word_pos**; and **(11) last_word_pos**. Columns (9)-(11) treat the *word* layer as atomic and require all annotations to coincide with word boundaries.

Finally, we use two types of *composite* indexes: *forward*, which looks for positions in a given document, and *inverted*, which supports searching based on annotation *values*.[10] An index lookup can be performed on any column combination that corresponds to an index *prefix*. An RDBMS' query optimizer estimates the optimal access paths (index and table scans), and join orders based on statistics collected over the stored records. In complex queries a combination of *forward* (F) and *inverted* (I) indexes is typically used. The particular ones we used are:[11]

(F) +doc_id+section+layer_id+sentence
+first_word_pos+last_word_pos+tag_type

(I) +layer_id+tag_type+doc_id+section+sentence
+first_word_pos+last_word_pos

(I) +word_id+layer_id+tag_type+doc_id+section
+sentence+first_word_pos

We have experimented with the system on a collection of 1.4 million MEDLINE abstracts, which include 10 million sentences annotated with 320 million multi-layered annotations. The current database size is around 70 GB. Annotations are indexed as they are inserted into the database.

[9]There are some additional tables mapping token IDs to entities (the string in case of a word, the MeSH label(s) in case of a MeSH term etc.)

[10]These *inverted* indexes can be seen as a direct extension of the widely used *inverted file* indexes in traditional IR systems.

[11]There is also an index on *annotation_id*, which allows for annotating relations between annotations.

Our initial evaluation shows variation in the execution time, depending on the kind and complexity of the query. Response time for simple queries is usually less than a minute, while for more complex ones it can be much longer. We are in the process of further investigating and tuning the system.

5 Conclusions and Future Work

We have provided a mechanism to effectively store and query layers of textual annotations, focusing on compact representation, easy query formulation, easy addition and removal of layers, and straightforward translation into SQL. Using a collection of 1.4 MEDLINE abstracts, we have evaluated various structures for data storage and have arrived at a promising one.

We have also designed a concise language (LQL) to express queries that span multiple levels of annotation structure, allowing users to express queries in a syntax that closely resembles the underlying annotation structure. We plan to release the software to the research community for use in their own annotation and querying needs.

Acknowledgements This research was supported by NSF DBI-0317510 and a gift from Genentech.

References

Steven Bird and Mark Liberman. 2001. A formal framework for linguistic annotation. *Speech Communication*, 33(1-2):23–60.

Steven Bird, Yi Chen, Susan Davidson, Haejoong Lee, and Yifeng Zheng. 2005. Extending XPath to support linguistic queries. In *Proceedings of PLANX*, pages 35–46.

Christian Blaschke, Miguel Andrade, Christos Ouzounis, and Alfonso Valencia. 1999. Automatic extraction of biological information from scientific text: Protein-protein interactions. In *Proceedings of ISMB*, pages 60–67.

Steve Cassidy and Jonathan Harrington. 2001. Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1-2):61–77.

Ralph Grishman. 1996. Building an architecture: a CAWG saga. *Advances in Text Processing: Tipster Program Ph. II.*

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

David McKelvie, Amy Isard, Andreas Mengel, Morten Moeller, Michael Grosse, and Marion Klein. 2001. The MATE workbench - an annotation tool for XML coded speech corpora. *Speech Communication*, 33(1-2):97–112.