System for Querying Syntactically Annotated Corpora

Petr Pajas
Charles Univ. in Prague, MFF ÚFAL
Malostranské nám. 25
118 00 Prague 1 – Czech Rep.
pajas@ufal.mff.cuni.cz

Jan Štěpánek
Charles Univ. in Prague, MFF ÚFAL
Malostranské nám. 25
118 00 Prague 1 – Czech Rep.
stepanek@ufal.mff.cuni.cz

Abstract

This paper presents a system for querying treebanks. The system consists of a powerful query language with natural support for cross-layer queries, a client interface with a graphical query builder and visualizer of the results, a command-line client interface, and two substitutable query engines: a very efficient engine using a relational database (suitable for large static data), and a slower, but paralel-computing enabled, engine operating on treebank files (suitable for "live" data).

1 Introduction

Syntactically annotated treebanks are a great resource of linguistic information that is available hardly or not at all in flat text corpora. Retrieving this information requires specialized tools. Some of the best-known tools for querying treebanks include TigerSEARCH (Lezius, 2002), TGrep2 (Rohde, 2001), MonaSearch (Maryns and Kepser, 2009), and NetGraph (Mírovský, 2006). All these tools dispose of great power when querying a single annotation layer with nodes labeled by "flat" feature records.

However, most of the existing systems are little equipped for applications on structurally complex treebanks, involving for example multiple interconnected annotation layers, multi-lingual parallel annotations with node-to-node alignments, or annotations where nodes are labeled by attributes with complex values such as lists or nested attribute-value structures. The Prague Dependency Treebank 2.0 (Hajič and others, 2006), PDT 2.0 for short, is a good example of a treebank with multiple annotation layers and richly-structured attribute values. NetGraph was a tool traditionally used for querying over PDT, but still it does not directly support cross-layer queries, unless the layers are merged together at the cost of loosing some structural information.

The presented system attempts to combine and extend features of the existing query tools and resolve the limitations mentioned above. We are grateful to an anonymous referee for pointing us to ANNIS2 (Zeldes and others, 2009) – another system that targets annotation on multiple levels.

2 System Overview

Our system, named PML Tree Query (PML-TQ), consists of three main components (discussed further in the following sections):

- an expressive *query language* supporting cross-layer queries, arbitrary boolean combinations of statements, able to query complex data structures. It also includes a sublanguage for generating listings and non-trivial statistical reports, which goes far beyond statistical features of e.g. TigerSearch.
- client interfaces: a graphical user interface with a graphical query builder, a customizable visualization of the results and a command-line interface.
- two interchangeable engines that evaluate queries: a very efficient engine that requires the treebank to be converted into a relational database, and a somewhat slower engine which operates directly on treebank files and is useful especially for data in the process of annotation and experimental data.

The query language applies to a generic data model associated with an XML-based data format called Prague Markup Language or PML (Pajas and Štěpánek, 2006). Although PML was developed in connection with PDT 2.0, it was designed as a universally applicable data format based on abstract data types, completely independent of a

particular annotation schema. It can capture simple linear annotations as well as annotations with one or more richly structured interconnected annotation layers. A concrete PML-based format for a specific annotation is defined by describing the data layout and XML vocabulary in a special file called PML Schema and referring to this schema file from individual data files.

It is relatively easy to convert data from other formats to PML without loss of information. In fact, PML-TQ is implemented within the TrEd framework (Pajas and Štěpánek, 2008), which uses PML as its native data format and already offers all kinds of tools for work with treebanks in several formats using on-the-fly transformation to PML (for XML input via XSLT).

The whole framework is covered by an open-source license and runs on most current platforms. It is also language and script independent (operating internally with Unicode).

The graphical client for PML-TQ is an extension to the tree editor TrEd that already serves as the main annotation tool for treebank projects (including PDT 2.0) in various countries. The client and server communicate over the HTTP protocol, which makes it possible to easily use PML-TQ engine as a service for other applications.

3 Query Language

A PML-TQ query consists of a part that selects nodes in the treebank, and an optional part that generates a report from the selected occurrences.

The selective part of the query specifies conditions that a group of nodes must satisfy to match the query. The conditions can be formulated as arbitrary boolean combinations of subqueries and simple statements that can express all kinds of relations between nodes and/or attribute values. This part of the query can be visualized as a graph with vertices representing the matching nodes, connected by various types of edges. The edges (visualized by arrows of different colors and styles) represent various types of relations between the nodes. There are four kinds of these relations:

- topological relations (*child, descendant depth-first-precedes, order-precedes, same-tree-as, same-document-as*) and their reversed counterparts (*parent, ancestor, depth-first-follows, order-follows*)
- inter- or cross-layer ID-based references
- user-implemented relations, i.e. relations whose low-level implementation is provided by the user as an extension to PML-TQ[1] (for example, we define relations *eparent* and *echild* for PDT 2.0 to distinguish effective dependency from technical dependency).
- transitive closures of the preceding two types of relations (e.g. if `coref_text.rf` is a relation representing textual coreference, then `coref_text.rf{4,}` is a relation representing chains of textual coreference of length at least 4).

The query can be accompanied by an optional part consisting of a chain of output filters that can be used to extract data from the matching nodes, compute statistics, and/or format and post-process the results of a query.

Let us examine these features on an example of a query over PDT 2.0, which looks for Czech words that have a patient or effect argument in infinitive form:

```
t-node $t := [
  child t-node $s := [
    functor in { "PAT", "EFF" },
    a/lex.rf $a ] ];
a-node $a := [
  m/tag ~ '^Vf',
  0x child a-node [ afun = 'AuxV' ] ];

>> for $s.functor,$t.t_lemma
   give $1, $2, count()
   sort by $3 desc
```

The square brackets enclose conditions regarding one node, so `t-node $t := [...]` is read *'t-node `$t` with ...'*. Comma is synonymous with logical `and`. See Fig. 3 for the graphical representation of the query and one match.

This particular query selects occurrences of a group of three nodes, `$t`, `$s`, and `$a` with the following properties: `$t` and `$s` are both of type `t-node`, i.e. nodes from a tectogrammatical tree (the types are defined in the PML Schema for the PDT 2.0); `$s` is a child of `$t`; the `functor` attribute of `$s` has either the value `PAT` or `EFF`; the node `$s` points to a node of type `a-node`, named `$a`, via an ID-based reference `a/lex.rf` (this expression in fact retrieves value of an attribute `lex.rf` from an attribute-value structure stored in the attribute `a` of `$s`); `$a` has an attribute `m` carrying an attribute-value structure with the attribute

[1] In some future version, the users will also be able to define new relations as separate PML-TQ queries.

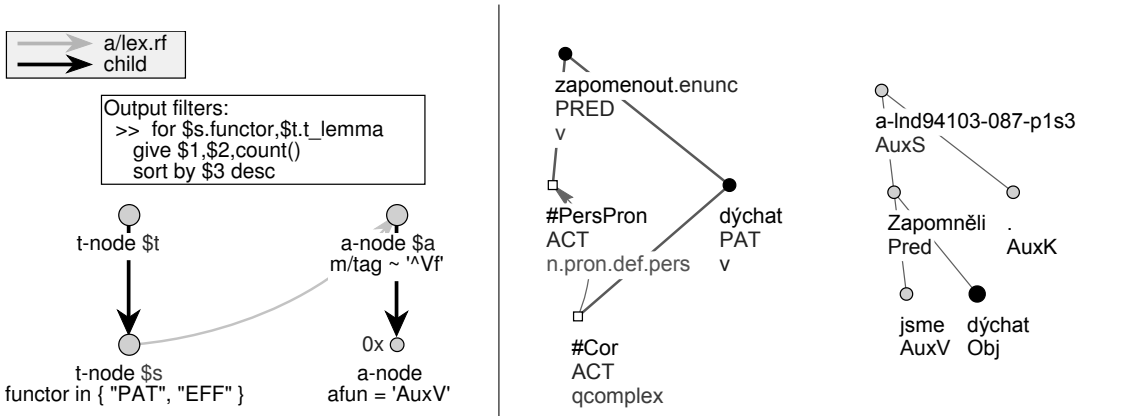

Figure 1: Graphical representation of a query (left) and a result spanning two annotation layers

`tag` matching regular expression `^Vf` (in PDT 2.0 tag set this indicates that `$a` is an infinitive); `$a` has no child node that is an auxiliary verb (`afun = 'AuxV'`). This last condition is expressed as a sub-query with zero occurrences (`0x`).

The selective part of the query is followed by one output filter (starting with >>). It returns three values for each match: the functor of `$s`, the tectogrammatical lemma of `$t`, and for each distinct pair of these two values the number of occurrences of this pair counted over the whole matching set. The output is ordered by the 3rd column in the descending order. It may look like this:

```
PAT     možnost          115
PAT     schopný          110
EFF     a                 85
PAT     #Comma            83
PAT     rozhodnout_se     75
```

In the PML data model, attributes (like `a` of `$t`, `m` of `$a` in our example) can carry complex values: attribute-value structures, lists, sequences of named elements, which in turn may contain other complex values. PML-TQ addresses values nested within complex data types by attribute paths whose notation is somewhat similar to XPath (e.g. `m/tag` or `a/[2]/aux.rf`). An attribute path evaluated on a given node may return more than one value. This happens for example when there is a list value on the attribute path: the expression `m/w/token='a'` where `m` is a list of attribute-value structures reads as *some one value returned by `m/w/token` equals 'a'*. By prefixing the path with a `*`, we may write *all values returned by `m/w/token` equal 'a'* as `*m/w/token='a'`.

We can also fix one value returned by an attribute path using the `member` keyword and query it the same way we query a node in the treebank:

```
t-node $n:= [
  member bridging [
    type = "CONTRAST",
    target.rf t-node [ functor="PAT" ]]]
```

where `bridging` is an attribute of `t-node` containing a list of labeled graph edges (attribute-value structures). We select one that has type `CONTRAST` and points to a node with functor PAT.

4 Query Editor and Client

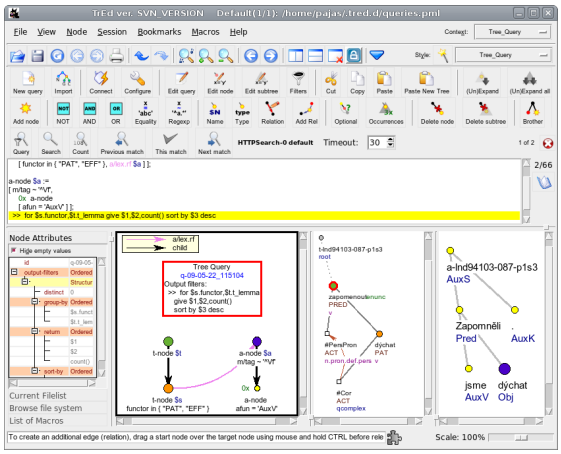

Figure 2: The PML-TQ graphical client in TrEd

The graphical user interface lets the user to build the query graphically or in the text form; in both cases it assists the user by offering available node-types, applicable relations, attribute paths, and values for enumerated data types. It communicates with the query engine and displays the results (matches, reports, number of occurrences).

Colors are used to indicate which node in the query graph corresponds to which node in the result. Matches from different annotation layers are displayed in parallel windows. For each result, the user can browse the complete document for context. Individual results can be saved in the PML format or printed to PostScript, PDF, or SVG. The user can also bookmark any tree from the result set, using the bookmarking features of TrEd. The queries are stored in a local file.[2]

5 Engines

For practical reasons, we have developed two engines that evaluate PML-TQ queries:

The first one is based on a translator of PML-TQ to SQL. It utilizes the power of modern relational databases[3] and provides excellent performance and scalability (answering typical queries over a 1-million-word treebank in a few seconds). To use this engine, the treebank must be, similarly to (Bird and others, 2006), converted into read-only database tables, which makes this engine more suitable for data that do not change too often (e.g. final versions of treebanks).

For querying over working data or data not likely to be queried repeatedly, we have developed an index-less query evaluator written in Perl, which performs searches over arbitrary data files sequentially. Although generally slower than the database implementation (partly due to the cost of parsing the input PML data format), its performance can be boosted up using a built-in support for parallel execution on a computer cluster.

Both engines are accessible through the identical client interface. Thus, users can run the same query over a treebank stored in a database as well as their local files of the same type.

When implementing the system, we periodically verify that both engines produce the same results on a large set of test queries. This testing proved invaluable not only for maintaining consistency, but also for discovering bugs in the two implementations and also for performance tuning.

6 Conclusion

We have presented a powerful open-source system for querying treebanks extending an established framework. The current version of the system is available at `http://ufal.mff.cuni.cz/~pajas/pmltq`.

[2]The possibility of storing the queries in a user account on the server is planned.

[3]The system supports Oracle Database (version 10g or newer, the free XE edition is sufficient) and PostgreSQL (version at least 8.4 is required for complete functionality).

Acknowledgments

This paper as well as the development of the system is supported by the grant Information Society of GA AV ČR under contract 1ET101120503 and by the grant GAUK No. 22908.

References

Steven Bird et al. 2006. Designing and evaluating an XPath dialect for linguistic queries. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 52. IEEE Computer Society.

Jan Hajič et al. 2006. The Prague Dependency Treebank 2.0. CD-ROM. Linguistic Data Consortium (CAT: LDC2006T01).

Wolfgang Lezius. 2002. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Ph.D. thesis, IMS, University of Stuttgart, December. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4.

Hendrik Maryns and Stephan Kepser. 2009. Monasearch – querying linguistic treebanks with monadic second-order logic. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories (TLT 2009)*.

Jiří Mírovský. 2006. Netgraph: A tool for searching in Prague Dependency Treebank 2.0. In *Proceedings of the 5th Workshop on Treebanks and Linguistic Theories (TLT 2006)*, pages 211–222.

Petr Pajas and Jan Štěpánek. 2008. Recent advances in a feature-rich framework for treebank annotation. In *The 22nd International Conference on Computational Linguistics - Proceedings of the Conference*, volume 2, pages 673–680. The Coling 2008 Organizing Committee.

Petr Pajas and Jan Štěpánek. 2006. XML-based representation of multi-layered annotation in the PDT 2.0. In *Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006)*, pages 40–47.

Douglas L.T. Rohde. 2001. TGrep2 the next-generation search engine for parse trees. `http://tedlab.mit.edu/~dr/Tgrep2/`.

Amir Zeldes et al. 2009. Information structure in african languages: Corpora and tools. In *Proceedings of the Workshop on Language Technologies for African Languages (AFLAT), 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL-09), Athens, Greece*, pages 17–24.