Querying Diverse Treebanks in a Uniform Way

Jan Štěpánek, Petr Pajas

Charles University in Prague
MFF ÚFAL
{stepanek,pajas}@ufal.mff.cuni.cz

Abstract

This paper presents a system for querying treebanks in a uniform way. The system is able to work with both dependency and constituency based treebanks in any language. We demonstrate its abilities on 11 different treebanks. The query language used by the system provides many features not available in other existing systems while still keeping the performance efficient. The paper also describes the conversion of ten treebanks into a common XML-based format used by the system, touching the question of standards and formats. The paper then shows several examples of linguistically interesting questions that the system is able to answer, for example browsing verbal clauses without subjects or extraposed relative clauses, generating the underlying grammar in a constituency treebank, searching for non-projective edges in a dependency treebank, or word-order typology of a language based on the treebank. The performance of several implementations of the system is also discussed by measuring the time requirements of some of the queries.

1. Introduction

A treebank is a collection of texts annotated with syntactic structures. Usually, more information than just pure structure is involved, for example POS tags and values of morphological categories or lemmatization are often included; some treebanks include further annotations (coreference, predicate-argument annotation, named entities, etc.). Linguists, who need to work with different treebanks, typically face various obstacles resulting from the differences between the treebanks; for example:

a) Each treebank can have a different view on what "syntax" is. In the first place, we can classify treebanks as dependency or phrase structure based.

b) Values and names of the same categories are different.

c) Treebanks use various data formats, some of them based on XML, others just single-purpose ad-hoc formats.

d) Each format can encode a tree differently (structure may be represented by nesting or by reference and there is also more than one way to indicate the surface word order).

In this paper, we describe our experience with using a set of diverse treebanks for evaluation of a generic annotation format and a system for treebank querying.

Our initial setup consisted of 11 different treebanks using 5 different data formats plus some variants (see Section 3.). We have converted all of them into a generic pivot format called PML (Section 3.2.) in such a way, that all information is preserved and unified names are used for features that had no name in the particular source format (Section 4.). Then, we loaded the converted treebanks into a query system called PML Tree Query (Section 2.). In this setup we can effectively query, visualize, and explore all of the treebanks in a uniform way which eliminates at least the differences of types c, d, and partially b.

A given query may still vary across the treebanks, due to the differences of the types a and b: the same phenomenon can be annotated in different ways in the treebanks (e.g. coordination), lexical values depend on the language, some kinds of information might be missing in some of the treebanks (e.g. no lemmatization in Penn Treebank), etc. We demonstrate this on several examples of linguistically interesting queries (Section 5.).

Finally we present some results of performance comparison of the query system for portions of the Prague Dependency Treebank of increasing sizes.

The treebanks used in our setup were the following: the Prague Dependency Treebank 2.0 (Hajič et al., 2006) (training data set), Penn Treebank 3 (Marcus et al., 1993) (all four parts, i.e. Air Traffic Information System, Brown Corpus, Switchboard Corpus, and Wall Street Journal), Tiger Treebank 1.0 (Brants et al., 2002), Penn CU Chinese Treebank 6.0 (Xue et al., 2005), Penn Arabic Treebank 2 (Maamouri et al., 2004) and several treebanks from the CoNLL-2009 Shared Task (Hajič et al., 2009).

2. PML Tree Query

Our setup consisted of evaluation of a complex framework for treebank(-based) development, which gradually evolved from the tree editor TrEd (Pajas and Štěpánek, 2008). The native data format of the framework is the Prague Markup Language format (PML), described in Section 3.2. Our particular interest was the evaluation of a new subsystem of our framework, called PML Tree Query (PML-TQ).

PML-TQ is a system for searching and exploring treebanks (Pajas and Štěpánek, 2009). It offers a powerful query and report language for the generic PML data model. The system is driven by a modern SQL database engine or alternatively by an iterator-based search engine implemented in Perl. The SQL-based implementation is very efficient but requires the data to be first loaded into a database, which makes it best suited for large, stable datasets, such as released treebanks. The other implementation operates directly on PML files and is therefore targeted for querying local or work-in-progress data. PML-TQ uses the TrEd toolkit for treebank visualization and can run from the GUI

interface of the tree editor TrEd or as a web application with SVG-rendered trees.

2.1. Query Language

The PML-TQ language offers the following distinctive features:

- selecting all occurrences of one or more nodes from the treebanks with given properties and in given relations w.r.t. the tree topology, cross-referencing, surface ordering, etc.
- support for bounded or unbounded iteration (i.e. transitive closure) of relations[1]
- support for multi-layered or aligned treebanks with structured attribute values
- quantified or negated subqueries (as in "find all clauses with exactly three objects but no subject")
- referencing among nodes (find parent and child that have the same case and gender but different number)
- natural textual and graphical representation of the query (the structure of the query usually corresponds to the structure of the matched subtree)
- sublanguage for postprocessing and generating reports (extracting values from the matched nodes and applying one or more layers of filtering, grouping, aggregating, and sorting)
- support for regular expressions, basic arithmetic and string operations in the query and postprocessing

2.2. Other Approaches to Treebank Querying

The PML-TQ has already been compared to some other tree-searching approaches in an earlier paper (Pajas and Štěpánek, 2009). One of its main advantages is its universality, demonstrated in this paper: most of the existing projects can handle exclusively phrase structure trees, but not dependency trees (e.g. TGrep2 (Rohde, 2001), LPath+ (Lai and Bird, 2010), TigerSEARCH (König and Lezius, 2001)). Most of the features listed in section 2.1. are not common for other projects (especially support for multi-layered treebanks, postprocessing sublanguage and support for regular expressions and basic arithmetic and string operations). No system has ever been tested on so many diverse treebanks.

3. Treebank Formats

3.1. Standards

We do not use the term "standard" for any of the formats, not even for PML to which all the treebanks were converted. Merriam-Webster Online defines standard as

- something established by authority, custom, or general consent as a model or example
- something set up and established by authority as a rule for the measure of quantity, weight, extent, value, or quality

No authority has established PML as an model or example of a treebank format. There is only a small number of projects outside our institute that use the format natively, so we cannot say it has been established as such by custom or general consent either. We would be pleased if someone used our format and it helped them, because that is why we devised it. We hope that the rich framework that we have developed for it, including tools for annotation, validation, querying, might be attractive.

There are on-going efforts for establishing standards for linguistic annotations (such as ISO/TC 37/SC4), but we fear that authoritative declaration of a standard in still such an evolving field might rather constitute an obstacle of the progress and innovation. Also, there is no use for a standard unless a variety of suitable tools implementing the standard is available.

More practically, interoperability can be achieved by following good practices and sticking to fundamental standards such as XML (wherever it is possible and makes sense). Then standard technologies, such as XSLT, can be used to import and export data.

3.2. Prague Markup Language

The Prague Markup Language (PML) is a meta-format based on XML, upon which the native formats of the Prague Dependency Treebank 2.0 (PDT) are built (Pajas and Štěpánek, 2006).

Specific PML data formats, such as those used in PDT, are defined using so called PML Schemas and can be formed of the following abstract data types:

Atomic: a string whose value can further be restricted to a specific format (e.g. any, integer, date, etc.)

Enumerated: atomic type with a given set of possible values.

Structure: set of attribute-value pairs.

List: ordered or unordered list of constructs of one type.

Alternative: similar to unordered list, but with different semantics.

Sequence: similar to ordered list, but allowing members with diverse types and supporting mixed content.

PML Schemas can further declare some structures in the data as nodes and specify how trees are formed from these nodes by nesting. Nodes can carry attributes whose values can be of any of the above abstract data types; edge labels are typically stored within the child node. Additionally, PML supports ID-based inter- and cross-document referencing (used in PDT e.g. to represent coreference) and multi-layer annotations with individual layers stored in different files (PDT defines four inter-connected annotation layers).

[1] For example, `descendant{1,3}` (iterating parent relation) or `coref_gram.rf{1,}` (iterating coreference pointer in PDT), `sibling{-1,1}` (immediately preceding or following sibling), `order-precedes{-1,1}` (immediately preceding or following node in the ordering of the sentence)

```
<LM id="a-147-p1s1">
 <afun>AuxS</afun>
 <ord>0</ord>
 <children>
  <LM  id="a-147-p1s1w2">
   <form>nevylučuje</form>
   <lemma>vylučovat_:T</lemma>
   <tag>VB-S---3P-NA---</tag>
   <afun>Pred</afun>
   <ord>2</ord>
   <children>
    <LM id="a-147-p1s1w1">
     <form>Privatizace</form>
     <lemma>privatizace</lemma>
     <tag>NNFS1-----A----</tag>
     <afun>Sb</afun>
     <ord>1</ord>
    </LM>
    <LM id="a-147-p1s1w3">
     <form>bankroty</form>
     <lemma>bankrot</lemma>
     <tag>NNIP4-----A----</tag>
     <afun>Obj</afun>
     <ord>3</ord>
    </LM>
   </children>
  </LM>
 </children>
</LM>
```

Figure 1: The sentence "Privatizace nevylučuje bankroty" in the PML format for the PDT a-layer, merged (knitted) and simplified for brevity.

3.3. Penn Format

```
( (SBARQ
    (WHNP-1 (WP Who) )
    (SQ
      (NP-SBJ (-NONE- *T*-1) )
      (VP (VBZ cares) ))
    (. ?) ))
```

Figure 2: The sentence "Who cares?" in Penn Format.

The Penn Format is a plain text format introduced in Penn Treebank and adopted by some other projects for capturing phrase structure trees by means of bracketing. Nonterminal nodes can carry a category label and optionally a set of function labels while terminals are formed by the surface word forms optionally augmented by POS tags. Both types of nodes can also include numerical coindexes for cross-referencing. There are several dialects and extensions to this format (in fact, each part of the Penn Treebank 3 uses a slightly different variant and the Penn CU Chinese Treebank 6.0 uses this format enclosed in an SGML-like envelope). Penn Format is very concise and comprehensible, but possibly due to the lack of standard tools for its validation, we have discovered a few errors in bracketing and/or coindexing in all of the examined treebanks that were using it.

3.4. Tiger XML Format

```
<s id="s36784">
<graph root="s36784_501">
<terminals>
 <t id="s36784_1" word="Es" pos="PPER" />
 <t id="s36784_2" word="ist" pos="VAFIN" />
 <t id="s36784_3" word="eine" pos="ART" />
 <t id="s36784_4" word="Zwickmühle"
     pos="NN" />
 <t id="s36784_5" word="." pos="$." />
</terminals>
<nonterminals>
 <nt id="s36784_500" cat="NP">
  <edge label="NK" idref="s36784_3" />
  <edge label="NK" idref="s36784_4" />
 </nt>
 <nt id="s36784_501" cat="S">
  <edge label="SB" idref="s36784_1" />
  <edge label="HD" idref="s36784_2" />
  <edge label="PD" idref="s36784_500" />
 </nt>
</nonterminals>
</graph>
</s>
```

Figure 3: The sentence "Es ist eine Zwickmühle." in Tiger XML Format (Tiger Treebank 1.0).

The Tiger Treebank introduced an XML format for phrase structure treebanks; unlike in Penn Treebank, terminals, nonterminals, and even edges can carry arbitrary number of string-valued attributes, whose domains can be enumerated. Trees are formed as graphs defined as an ordered list of terminal nodes and a set of nonterminal nodes, where each nonterminal enumerates the edges which connect it to its children.

3.5. CoNLL-2009 ST Format

```
1    What WP   3    OBJ  _    _      R-A1
2    to   TO   0    ROOT _    _      _
3    do   VB   2    IM   Y    do.02  _
4    ?    .    2    P    _    _      _
```

Figure 4: The sentence "What to do?" in CoNLL-2009 ST format (some columns removed to fit the page).

CoNLL Shared Tasks traditionally use simple tabular plain-text formats, where each row represents a node, trees are represented by references to parents based on row numbers; further columns represent attribute values, which can be either strings or lists of strings using certain characters as delimiters. The format is easy to process, but cannot be reused, most notably because it lacks any header describing the meaning of the individual columns.

3.6. Penn Arabic Treebank Format

The Penn Arabic Treebank 2 uses a rather obscure mix of SGML (for the original text), Annotation Graph (AG) format (Bird and Liberman, 1999) (as an overly complex way of encoding lemmatization, morphological analysis,

and English glosses for the stream of input tokens), into which a plain-text bracketing similar to the Penn Format is embedded (with phrase trees whose terminals are numbers referring to XML elements in the AG annotation).

4. Conversion to PML

Compared e.g. to the concise plain-text formats, PML may seem quite heavy-weighted; on the other hand it is certainly the most versatile format among the formats considered.

In order to evaluate our PML-based framework and the query system, we first needed to convert the treebanks into the PML format. However, it should be noted that it was not our goal nor did we attempt to unify the actual labeling and tagging schemes of these treebanks. We simply wanted to create a common formal representation of the treebanks, using PML as a pivot format.

Therefore, for each of the treebanks, we have created a specialized PML schema which captures all information available in the original data. Nevertheless, the PML Schemas for phrase-based (and likewise for dependency) trees are all very similar: they all define nodes of three types (technical root, nonterminal and terminal), each of which carries some attributes; the main differences were only in the attributes or the enumerations of their possible values. During the conversion to PML, edge labels from Tiger XML Format were transformed to attributes of the child nodes and coindexes (occurring in most of the treebanks) were realized as ID-based references (every node in the PML format was assigned a unique ID if no original IDs were available). All of the conversion scripts were written in Perl, except for the conversion from Tiger XML, where we conveniently used XSLT.

5. Query Examples

The simplest way to demonstrate the query power of PML-TQ and the differences between treebanks is to provide example queries and their results.

5.1. Verb Clause without a Subject

All verb clauses without a subject can be easily identified in most treebanks with a query similar to Fig. 5. To understand the query language, one should note that nonterminal is just a node type identifier with no a priori semantics (but referring to type declarations in PML schema); * is a wild-card matching any node type, 0x is a quantifier (zero times), and cat and label are node attributes; square brackets contain comma separated constraints on a given node, so the string [...] usually reads "node which has ..." or "with ..."; the default relation between nodes in the query is "child". Thus, the query, translates from the PML-TQ syntax into English as follows: "Find all nonterminal nodes which has category 'S', zero child nodes with label 'Sb' and a nonterminal child node with category 'VP' ".

In TIGER, the query has 1155 matches, which can be easily counted by appending a postprocessing instruction ">> `count()`" to the end of the query.

In Penn CU Treebank, the head category is IP and subject function is SBJ, it can only be assigned to a nonterminal, see Fig. 6. Note that the attribute functions contains a list,

```
nonterminal [
 cat = 'S',
 0x * [
  label = 'SB'
 ],
 nonterminal [
  cat = 'VP'
 ]
]
```

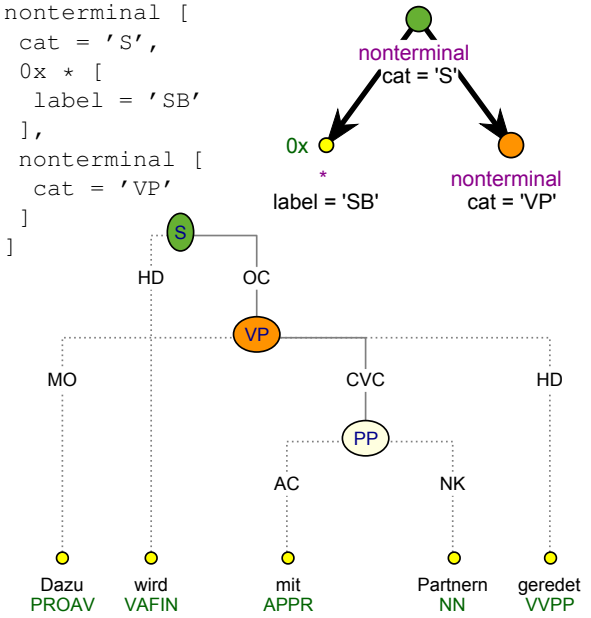


Figure 5: Query 5.1., its visualization, and a result tree from Tiger Treebank 1.0.

```
nonterminal [
  cat='IP',
  0x nonterminal [
    functions='SBJ' ],
  nonterminal [
    cat='VP'
] ]
```

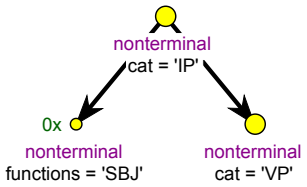


Figure 6: Query 5.1. for Penn CU Treebank.

and the constraint functions='SBJ' is true if any of the values in the list equals SBJ. Apart from these details, the query is the same as the one for Tiger Treebank 1.0.

```
nonterminal
[ cat = 'S',
  0x  nonterminal [
    functions = 'SBJ' ],
  nonterminal [
    cat = 'VP',
    coindex.rf nonterminal [
      cat = 'VP',
      sibling nonterminal [
        functions = 'SBJ'
] ] ] ]
```

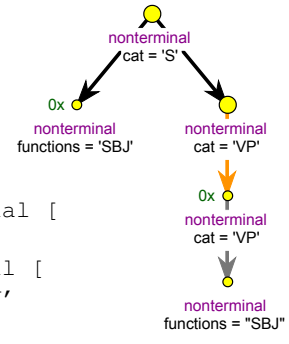


Figure 7: Query 5.1. for Penn Treebank 3 (WSJ part).

For Penn Treebank 3 (WSJ part), in addition, we have to exclude common subjects for parallel structures (e.g. "he joined CBS Sports (...) and, five years later, became its president", see Fig. 7.)[2] In the original format, the common

[2]This query still returns one match, a parallel structure with missing auxiliary verb "if schedules were reduced and the games

subject is captured by a co-index, which we translated into PML as a relation named coindex.rf; we can thus directly connect the nodes in the query by this relation.

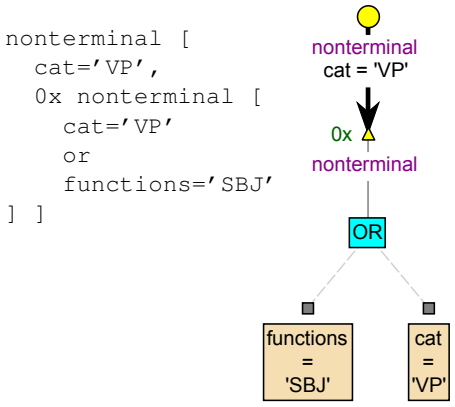


Figure 8: Query 5.1. for Penn Arabic Treebank.

In Penn Arabic treebank, except for 8 cases, a subject is dominated not by an S but by a VP that has no VP child; it therefore suffices to test those (see Fig. 8).

```
a-node [
 $$ = $aux and m/tag ~ '^V[^fs]'
    or $$ != $aux and m/tag ~ '^V[sf]',
  0x echild a-node [ afun = 'Sb' ],
     ? echild a-node $aux := [
       afun = 'AuxV', m/tag ~ '^V[^f]' ] ]
```

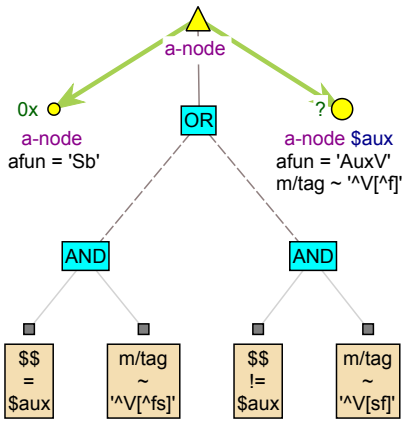


Figure 9: Query 5.1. for Prague Dependency Treebank 2.0 ($$ means *this node*).

In dependency treebanks, the query is quite different. For example, in Prague Dependency Treebank 2.0, it looks like Fig. 9. We are interested in finite verbs with no subject, but in Czech, even an infinitive or passive verb can function as finite, if it is accompanied by an auxiliary finite verb, matched in the query by so called optional node. An optional node (marked in the query by ?) is either disregarded (in which case the associated name, $aux, is identified with the parent node in the query), or matched normally according to its constraints. A custom relation echild, counting for coordinations and appositions, is used instead of the default child relation.

returned to the students".

5.2. Grammar Extraction

```
nonterminal $p := [ * $ch := [  ] ]
>> give $p, $p.cat,
        first_defined($ch.cat,$ch.pos),
        lbrothers($ch)
>> give $2 & " -> "
        & concat($3," " over $1 sort by $4)
>> for $1 give count(),$1 sort by $1 desc
```

Figure 10: Grammar extracting query for Penn Treebank 3 (WSJ part).

From phrase structure treebanks, their underlying grammar can be easily extracted. For example, in Penn Treebank 3 (WSJ part), we just look for a nonterminal parent and its child; we output the internal unique identifier and the category of the nonterminal, a label of the child (category if it is a nonterminal, or part of speech if it is a terminal) and its order among siblings (number of brothers on the left); we group this list by the parent, so all children of one parents get to one group; we then concatenate the children's labels in each group in the appropriate order to produce a grammar rule for each group, and finally, we simply compute the frequency of each rule in the list and sort by it (see Fig. 10). Table 1 lists the top 10 lines of the generated grammar (nonterminals set in bold face).

189856	**PP** → IN **NP**
128140	**S** → **NP VP**
87402	**NP** → **NP PP**
72106	**NP** → DT NN
65508	**S** → **NP VP** .
45995	**NP** → -NONE-
36078	**NP** → DT JJ NN
31916	**VP** → TO **VP**
28796	**NP** → NNP NNP
23272	**SBAR** → IN **S**
...	

Table 1: The grammar generated from Penn Treebank 3 (WSJ part). Each rule is preceded by number of applications.

5.3. Non-projective Edges

Classical phrase structure trees are projective by definition (traces are used to deal with crossing brackets). Tiger Treebank uses a hybrid approach with nonterminal constituent nodes, but with crossing branches and no traces. In dependency treebanks, non-projectivity is being studied as an important phenomenon (cf. (Havelka, 2007)).

For all the dependency treebanks a query similar to Fig. 11 can be used to find all the non-projective edges (just the names of the node types can be different). Table 2 shows how frequent non-projectivity is in several dependency treebanks. It displays the percentage of non-projective

```
node $upper := [
  same-tree-as $between,
  node $lower := [ ] ];
node $between := [ ! ancestor $upper,
  ( (order-precedes $upper
     and order-follows $lower)
    or (order-follows $upper
        and order-precedes $lower) ) ]
```

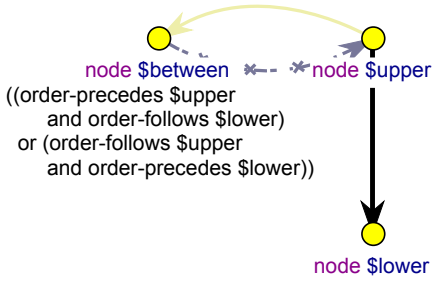


Figure 11: Query 5.3. for CoNLL 2009 dependency treebanks.

edges among all the edges, an average number of non-projective edges in a tree, and a real percentage of non-projective trees among all the trees. In Spanish CoNLL training data, all the edges are projective, which is probably not a feature of the language, but rather a feature of the formalism used.

Treebank	NPE/edge	NPE/tree	NPT
German CoNLL	2.33%	41.99%	28.10%
PDT 2.0	1.88%	32.14%	22.98%
English CoNLL	0.39%	9.48%	7.63%
Spanish CoNLL	0.00%	0.00%	0.00%

Table 2: Number of non-projective edges (NPE) and non-projective trees (NPT) in dependency treebanks.

5.4. Word Order Typology

The word order typology classifies languages according to the basic word order in transitive sentences. Query 12 extracts the facts from the German CoNLL-2009 Shared Task data. Depending on the exact definition, one might want to remove sentences with more than one object etc., but the query still can be used as a base. We just remove all the lines that do not contain both O and S. The results are shown in Table 3, the first two columns (capital "V") represent main clauses, the second two subordinate (small "v"). The difference between the two types of clauses is clearly visible. For other dependency treebanks, the query would be almost the same.

5.5. Extraposed Relative Clauses

Using the dependency approach, we can say that the relative clause normally follows its governing noun from which it can be separated only by other dependents of the noun. In constituent approach jargon, this can be rephrased as "The relative clause is normally a constituent of the noun phrase containing the head noun it modifies". Mirroring this difference between the two approaches, the queries searching

```
node $p := [ substr(pos,0,1) = 'V',
  ? node $ch := [
    deprel in {'SB','OA','OC','OA2','OP'}
] ];
>> give $p.xml:id,
     if($p=$ch,
       if($p.deprel = 'ROOT','V','v'),
       substr($ch.deprel,0,1)),
     $ch.order
>> give distinct $1,
     concat($2,'' over $1 sort by $3)
>> give
     substitute($2,'([OS])\\1+','\\1','g')
>> filter ($1 ~ 'O' and $1 ~ 'S')
>> for $1 give $1,count() sort by $2 desc
```

Figure 12: Word order typology (query 5.4.) for German CoNLL data.

Main clause	Num. of occurrences	Dependent clause	Num. of occurrences
SVO	11267	SOv	7556
VSO	7111	SvO	2273
OVS	2209	vSO	1113
VOS	625	OSv	606
SOV	110	OvS	109
OVSO	91	vOS	64
VOSO	64	SOvO	37
OVOS	31	OSOv	34
		...	

Table 3: Word order typology for German calculated by query 5.4. from German CoNLL data. Lines not containing both subject and object removed.

for extraposed relative clauses (the "non normal" case) are quite different for dependency and constituency treebanks. Two examples are given in Fig. 13.

In Penn Treebank, a special terminal ICH (Interpret constituent here) is used in the place where the relative clause would have been placed but for the extraposition. In Prague Dependency Treebank, we are looking for an attribute (`Atr`) expressed by a verb that is separated from its parent by a word that does not belong to the parent's subtree.

6. Comparison and Performance

Table 4 shows various quantitative differences between the treebanks, easily computed using PML-TQ queries. The meaning of particular columns can be found in the following list, the queries are shown in Fig. 14.

a. **Total number of nodes:** All types of nodes were counted (roots, terminals, nonterminals). In case of Prague Dependency Treebank, a-layer nodes were used.

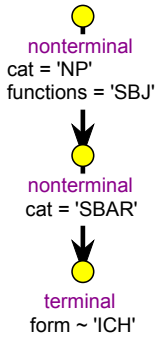
b. **Maximal rank:** By the rank we mean the number of sons.

c. **Median rank:** The rank of the middle tree from trees sorted by rank.

Treebank	total # nodes	max rank	median rank	max tree	median tree	max depth	max breadth	nodes / term.
PDT	1.59M	85	3	195	12	24	85	–
Tiger	0.95M	17	2	237	7	23	53	1.55
WSJ	2.28M	51	3	441	10	37	159	1.82
Atis	0.01M	8	2	81	10	16	17	2.13
Brown	0.92M	24	2	347	14	36	53	1.89
SWBD	2.73M	26	6	272	63	37	54	1.91
Chinese (Penn)	1.86M	64	2	558	4	30	169	2.18
Arabic	0.36M	25	2	602	173	52	73	2.16
Catalan	0.4M	37	9	215	23	24	56	–
Chinese (CoNLL)	0.63M	35	3	243	30	20	114	–
Spanish	0.44M	62	2	150	17	28	64	–

Table 4: Basic characteristics of the treebanks.

```
nonterminal [
  cat='NP',
  functions='SBJ',
  nonterminal [
    cat='SBAR',
    terminal [
      form ~ 'ICH'
] ] ]
```




```
a-node $n2 := [
  ! afun in {'Coord','Apos'},
  order-precedes a-node [
    ! ancestor $n2,
    order-precedes $n3
  ],
  a-node $n3 := [
    afun ~ 'Atr',
    m/tag ~ '^V' ] ]
```

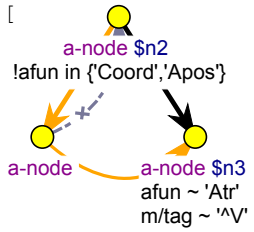


Figure 13: Query 5.5. for Penn Treebank 3 and PDT 2.0.

d. **Maximal tree:** The maximal number of nodes in a tree.

e. **Median tree:** Similar to median rank (c).

f. **Maximal depth:** The longest path from a root to a leaf node.

g. **Maximal breadth:** Maximal level size.

h. **Nodes to terminals ratio:** The query makes sense only in constituent treebanks. It reports both the nodes-to-terminals ratio and the nodes-to-nonterminals ratio.

We have collected various measurements of the system performance. For example, we ran the queries a–h over all the examined treebanks indicates high correlation between the treebank size and the running time of the query. For most treebanks, the running time for one query averaged over this

a.
```
* [ ] >> count()
```
b.
```
* $n := [ sons()>1 ]
>> sons($n) >> max()
```
c.
```
* $n := [ sons()>1 ]
>> sons($n)
>> $1, row_number(sort by $1),
     count(over all)
>> filter $2=($3 div 2) >> $1
```
d.
```
* $r := [ 0x parent * [ ] ]
>> descendants($r)+1 >> max()
```
e.
```
* $r := [ 0x parent * [ ] ]
>> descendants($r)+1
>> $1, row_number(sort by $1),
     count(over all)
>> filter $2=($3 div 2) >> $1
```
f.
```
* $n := [ sons()=0 ]
>> depth($n)+1 >> max()
```
g.
```
* $root := [ depth()=0,
  ? descendant * $n := [ ] ]
>> for $root,depth($n) give count()
>> max()
```
h.
```
* $n := [  ];
  >> give if(sons($n)>0,'n','t')
  >> for $1 give $1,count()
  >> for $1,$2
     give $1,sum($2 over all) div $2
```

Figure 14: Basic quantitative queries.

set of queries was 2.3 ± 0.2 seconds per 1 million of nodes; the exception was treebanks with less than half a million nodes, where the average was 1.1 ± 0.2 seconds (probably due to better use of caches and memory buffers for disk reads).

Another measurement shown in Fig. 15 illustrates the averaged performance of the system over a set of 50 linguistically meaningful queries (not listed in this paper) of varying complexity spanning over all four annotation layers of PDT 2.0. We run the queries over portions of the treebank of increasing sizes. The Oracle 10g XE engine appears to be the fastest by average, but we have noted several queries where it was outperformed by the PostgreSQL engine.

The Perl-driven engine is running in time linear to the size of the corpus. The times presented in Fig. 15 for the Perl-driven engine do not include loading the treebank data into memory, which adds additional linear overhead. The per-

formance of this engine can be highly accelerated by parallelization of the searching phase (independent documents can be processed in parallel). We have not yet developed an algorithm to parallelize the filter phase.

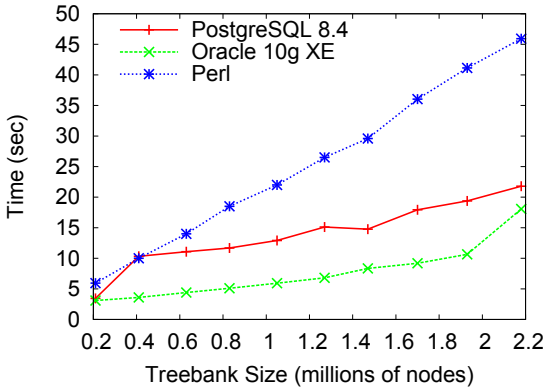


Figure 15: Comparison of average performance of two SQL-driven and the Perl-driven PML-TQ engines over 50 queries of varying complexity for all four annotation layers of PDT.

7. Conclusion

We have presented a system for querying treebanks in a uniform way. The abilities of the system are demonstrated on 11 different treebanks converted to its XML-based format without losing any information and loaded into the system. The query language used by the system provides many features not available in any other existing system.

We showed several examples of linguistically interesting queries and discussed briefly the performance of the system.

Acknowledgment

This paper was written with the support of post-doc grant 406/10/P193 of Grant Agency of Science of the Czech Republic and grant FP7-ICT-2007-3-231720 (EuroMatrix Plus, 7E09003 in the Czech Republic) of the Seventh Framework Programme of the European Union.

8. References

Steven Bird and Mark Liberman. 1999. A formal framework for linguistic annotation. *Speech Communication*, 33:23–60.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, Sozopol.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009), June 4–5*, Boulder, Colorado, USA.

Jan Hajič, Marie Mikulová, Alla Bémová, Eva Hajičová, Jiří Havelka, Veronika Kolářová-Řezníčková, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Razímová, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, and Zdeněk Žabokrtský. 2006. The Prague Dependency Treebank 2.0. CD-ROM. LDC.

Jiří Havelka. 2007. Relationship between non-projective edges, their level types, and well-nestedness. In *NAACL HLT 2007 Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 61–64, Rochester, NY, USA. Association for Computational Linguistics.

Esther König and Wolfgang Lezius. 2001. The TIGER language—a description language for syntax graphs. Part 1: Users' guidelines. Technical report, IMS, University of Stuttgart.

Catherine Lai and Steven Bird. 2010. Querying linguistic trees. *Journal of Logic, Language and Information*, 19(1):53–73.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. 2004. The Penn Arabic Treebank: Building a large-scale annotated Arabic corpus. In *NEMLAR Conference on Arabic Language Resources and Tools*, pages 102–109.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

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, Genova, Italy.

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. 2009. System for querying syntactically annotated corpora. In *Proceedings of the ACL-IJCNLP 2009 Software Demonstrations*, pages 33–36, Suntec, Singapore, August. Association for Computational Linguistics.

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

Nianwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.