Fangorn: A system for querying very large treebanks

Sumukh GHODKE[1] *Steven BIRD*[1,2]
(1) Department of Computing and Information Systems, University of Melbourne
(2) Linguistic Data Consortium, University of Pennsylvania
sumukh.ghodke@gmail.com, sbird@unimelb.edu.au

ABSTRACT
The efficiency and robustness of statistical parsers has made it possible to create very large treebanks. These serve as the starting point for further work including enrichment, extraction, and curation: semantic annotations are added, syntactic features are mined, erroneous analyses are corrected. In many such cases manual processing is required, and this must operate efficiently on the largest scale. We report on an efficient web-based system for querying very large treebanks called Fangorn. It implements an XPath-like query language which is extended with a linguistic operator to capture proximity in the terminal sequence. Query results are displayed using scalable vector graphics and decorated with the original query, making it easy for queries to be modified and resubmitted. Fangorn is built on the Apache Lucene text search engine and is available under the Apache License.

KEYWORDS: language resources, database query, annotation, syntax, parsing.

Proceedings of COLING 2012: Demonstration Papers, pages 175–182,
COLING 2012, Mumbai, December 2012.

1 Introduction

Treebanks play a central role in the analysis of language structures in a diverse range of areas including language modelling, machine translation, information extraction, and syntactic description. Manual annotation requires painstaking work by specialists, and this is too expensive to do on a large scale. Instead, it is standard to use a state-of-the-art parser on massive quantities of text and then manually post-edit the output. At the point of discovering and correcting a parse error, it is desirable to quickly locate other instances of the same error, regardless of where they appear in the corpus. Syntactic research depends on manual exploration of conditioning factors that allow us to identify constructions of interest, and these constructions will usually be rare. Such activities require efficient query over very large treebanks.

The last decade has seen the development of several corpus query tools, including TGrep2, TIGERSearch, Emu, Nite NXT Search, Netgraph, fsq, and Emdros (Rohde, 2001; Brants et al., 2002; Cassidy and Harrington, 2001; Heid et al., 2004; Mírovský, 2006; Kepser, 2003; Petersen, 2004). These tools are effective when the corpus fits in main memory, or when work can be done in batch mode (requiring a linear pass through the entire corpus on disk, typically taking tens of seconds). When the corpus is large, and when fast query processing is required, an entirely different approach is needed. *Fangorn* is designed to fill this gap. It is the outcome of an interdisciplinary research project combining the scaling properties of general purpose semi-structured databases and information retrieval engines with the features commonly found in corpus query tools (Ghodke and Bird, 2008, 2010).

In this paper we present the features of *Fangorn*, including its query language, its user interface, and its architecture. Finally, we describe areas for further research.

2 Background

Treebank query tools are often designed for specific annotation structures. Some tools are designed for phrase structure trees (e.g. TGrep2, TIGERSearch), while others are designed for dependency trees (e.g. Netgraph). Some are intended for corpora with multiple annotation types (e.g. Nite NXT Search, Emu). Some permit phrase structure trees with crossing branches (e.g. TIGERSearch) while others require strict trees (e.g. TGrep2). Some support query of extra properties on tree nodes or edges. Despite this diversity, there are still some abstract requirements that are common across all corpus query tools.

All tools support expressive query languages, although the great variety of syntax obscures the expressive similarity of the languages. Lai and Bird (2004) compare several corpus query languages and present a few generic requirements for treebank query languages. They state that the query language should include more than just simple navigational operators and include features such as: subtree matching, non-tree navigation (e.g. the "immediate following" operator explained later), secondary edges, and closure operators. They should also handle Boolean operators: conjunction, disjunction, and negation. The language should be able to specify the granularity of results. While query language operators depend largely on the structure of the data, some features such as finding secondary edges in a tree are specific to annotation style. Query languages usually support regular expressions over node labels, allowing search for label prefixes and suffixes (e.g. `N.*` for any noun-like syntactic category). Support for popular file formats is another requirement. Most corpus query tools accept one or more of the following annotation formats as inputs: Penn Treebank (Marcus et al., 1993), NEGRA (Skut et al., 1997), TIGER corpus (Brants et al., 2002), or custom XML formats.

Most tree query tools are not designed for very large data since they perform a linear pass over the entire collection, e.g. TGrep. More advanced tools store and index tree data using relational databases. However, relational databases are not efficient for storing and accessing trees (Zhang et al., 2001). On the other hand, semistructured databases have indexes that are specialised for efficient operations on large tree collections. Unfortunately, their query languages lack the required expressiveness. An ideal system for large scale treebank data should inherit the query language and annotation awareness from corpus query tools while borrowing the scaling features from semi-structured databases.

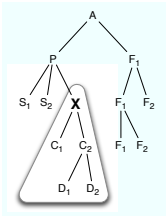
3 Query Language

Fangorn queries involve "path expressions", a series of navigations around the tree, e.g. from a node to its sibling, descendent, or ancestor. At each step, the type of navigation is specified by an operator and the type of node or terminal is specified with a string. For instance, `/NP` navigates to an NP child, `=>PP` navigates to a PP following sibling, and `/NP=>PP` combines both of these steps. Paths always begin at the root, and so a path expression that starts with `/S` can only navigate to the S node at the root of the tree. The descendent operator allows a path to skip to any descendent of the current node, so a path expression beginning with `//NP` navigates from the root to any NP node in the whole tree, in a single step. Paths can be arbitrarily long, e.g. `//NP=>S//VP->NN/table`; each time we add a step, we further restrict the number of possible matching trees. Paths can branch, and we use "filter expressions" (sub-paths contained in square brackets) to specify restrictions on the branches. Furthermore, these filter expressions can be combined using Boolean operators. The language used by *Fangorn* is defined in BNF as follows, where the axis operators and logical operators are given in Tables 1 and 2.

```
<expr> ::= <term> [<term>]*
<term> ::= <axis-operator><node-label> [<filter-expr>]
<filter-expr> ::= "[" <filter-element> [(AND|OR) <filter-element>]* "]"
<filter-element> ::= [NOT] <expr>
<node-label> ::= annotation_label | word | punctuation
```

Operator	**Symbol**	Y
Descendant	$X//Y$	C_*, D_*
Child	X/Y	C_*
Ancestor	$X\backslash\backslash Y$	P, A
Parent	$X\backslash Y$	P
Following sibling	$X==>Y$	
Immediately following sibling	$X=>Y$	
Preceding sibling	$X<==Y$	S_*
Immediately preceding sibling	$X<=Y$	S_2
Following	$X-->Y$	F_*
Immediately following	$X->Y$	F_1
Preceding	$X<--Y$	S_*
Immediately preceding	$X<-Y$	S_2

Table 1: Navigation operators



Operator	**Symbols**
Conjunction	AND, and, &
Disjunction	OR, or, |
Negation	NOT, not, !

Table 2: Logical operators

Filter expressions are responsible for the great expressiveness of path queries. Consider the query: `//VBG[<-is AND =>S AND -->PRP]`. Here, we are searching for any gerund VBG such

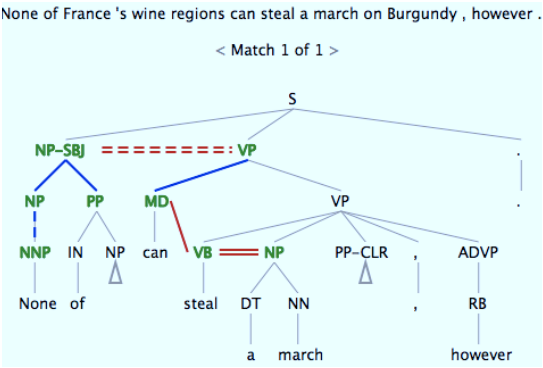


Figure 1: A result sentence from a page listing all results for a path query

that three conditions hold: it is immediately preceded by the word 'is', it has an immediately following sibling S, and it is followed somewhere later in the tree by a personal pronoun PRP.

The square brackets in the above query delimit the filter expression. Note that the contents of the filter expression can be arbitrarily complex. The above query could be extended by adding another step to one of the conjuncts as follows: `//VBG[<-is AND =>S AND -->PRP->TO]`.

The NOT operator may appear before any path expression inside a filter expression. For example, we can negate the earlier requirement concerning S as follows: `//VBG[<-is AND NOT =>S AND -->PRP]`. Similarly, the query to search for all occurrences of gerunds not preceded by the word 'is' is written as: `//VBG[NOT <-is]`.

Finally, filter expressions can be nested. For example, we can limit the S, requiring that it does not contain a PP, as follows: `//VBG[<-is AND =>S[NOT //PP] AND -->PRP]`.

4 User interface

Fangorn runs in a web browser. The entry point contains a simple search box where the query can be entered, along with a drop-down menu listing available corpora.[1] The page also displays the total number of sentences in the corpus that match the query and the time for executing the search.

Each matched sentence is rendered as an interactive scalable vector graphics (SVG) image. Figure 1 shows one annotated result sentence for the query `//NP-SBJ[/NP//NNP AND /PP]==>VP/MD->VB=>NP`. The NP-SBJ node has two child nodes NP and PP, whose edges are annotated using solid blue lines and a following sibling VP edge annotated using a double dashed red line. Blue lines are used for vertical operators (ancestor, descendant, parent, or child operators), while red lines are used for horizontal navigations. Double lines signify a sibling relationship, solid lines an immediate relationship, and dashed lines are used for closure operators. The trees are rendered in a minimally expanded form that ensures that the annotated result is always visible and cannot be accidentally collapsed.

[1] A demonstration system permits searches over Penn Treebank and the English Wikipedia (see `http://nltk.ldc.upenn.edu:9090`).

The other nodes in the sentence may be expanded or collapsed by clicking on the nodes or the triangle below collapsed nodes. Buttons are provided to expand/collapse all nodes in a tree, and to export each sentence as an SVG image or as text in Penn Treebank format. Displayed above each result tree is the corresponding sentence without annotations. When the mouse is positioned over a tree node, the corresponding span of words in the sentence is highlighted. When a sentence matches a query more than once, only the first match is displayed on screen while other matches can be viewed by clicking on the arrows on either side of the result.

Result trees are annotated with the path expression that was used to find them, so that users can immediately see why the tree was matched. This annotation can itself be edited, in order to modify and resubmit the query. The edit mode can be activated by clicking the "Build query from tree" button at the top right of each matched result in the result display screen. This window does not allow nodes in the tree to be collapsed on a mouse click, however, collapsed nodes can be expanded by clicking on the triangle below such nodes. The following operations can be performed in the edit query screen: (1) extend a query starting at a node, (2) edit label or delete query terms, and (3) change or negate operators joining two query terms.

5 Architecture

Fangorn uses a web client-server architecture. An embedded Jetty web server hosts the search engine software. The search engine is built using Apache Lucene, a popular and versatile open source text search engine toolkit. A corpus reader loads text corpora and converts the annotated sentences into a format easily digestible by the search engine. This step is called the analysis step and is explained later in this section. An embedded database is used to store corpora metadata, but this database is not used to answer tree queries. Searches can be performed from a browser once the web server is started.

The client browser receives results as Javascript objects from the server and renders them as SVG images. The SVG format was chosen because it is a compact vector format, and because event handlers can be easily attached to SVG elements to make them interactive. Not all browsers have adequate support for dynamically embedded SVG images, and so we designed the user interface specifically for the Mozilla Firefox browser.

The analysis step converts a treebank tree into tokens for indexing. Each node is assigned a 4-tuple of position numbers (left, right, depth, parent) that uniquely identify its location in the tree following the scheme used by LPath (Bird et al., 2006). This position information is enough to test whether two nodes satisfy an operator condition, avoiding the need to store the trees explicitly. The output of the analysis step is a sequence of tokens – essentially node labels in the treebank together with their position numbers – that are then indexed in the text search engine.

Lucene is configured to use two sets of inverted lists for the index. The first is the frequency index, which is a list of frequency postings lists. Each token's postings list maintains a sequence of document ids that contain the token together with the term-frequency in each document. (Note that each sentence is treated as its own document.) The second is the position index. This index typically stores an integer that identifies the sequential location of a token, and is used in phrase and proximity queries. However, in *Fangorn* the tree analysis step transforms a tree structure into a depth-first ordered sequence of tokens, and the integer position of a tree token is its depth-first position. The position index provides extra storage at each position (byte-array payloads), and we use these to store the tree position information.

Tree queries are processed using the frequency and position indexes. First, the frequency index identifies the sentences that contain the required terms in the query. Then, using the position index, the position lists of the query terms are joined pairwise, based on the structural relation between the two terms in a pair. This method of performing pairwise joins is called a path join. *Fangorn* implements a join algorithm similar to a path join, called the staircase join algorithm (Grust et al., 2003). It uses both the depth-first position and the tree position information to reduce the number of comparisons while executing the join. This join algorithm is faster in part because it does not find all matches in every sentence. Instead, it checks if a sentence has any match at all. When a match is spotted, the sentence id is registered and the search continues with a new sentence. Later, all matches within each sentence are found, but only for the limited number of sentences that are displayed in one page of results.

Fangorn can be installed on commodity hardware with at least about 2 GB of RAM. The server software is written in Java and can be installed on POSIX systems with little or no modifications. Microsoft Windows users may require tools that provide a POSIX-like environment to run maintenance scripts. A Java runtime version 5 or higher is required to be installed on the machine. Corpora can be added or removed from the search engine by running shell scripts distributed with the software. Currently, the Penn Treebank format is the only input format supported by the system.

6 Conclusion

Treebanks play a central role in the modelling of natural language grammar. The cyclic process of curating treebanks and retraining parsers on the improved annotations is set to continue for some time, and will continue to generate ever better treebanks. The bottleneck in this process is manual curation, and a major part of curation involves discovery of complex tree patterns that need to be edited in some way.

We have implemented an efficient system for querying very large treebanks, and it is available for download under the terms of the Apache License 2.0 from `http://code.google.com/p/fangorn`. A demonstration version is available at `http://nltk.ldc.upenn.edu:9090`. The system is built on top of Lucene, and its scaling performance can be expected to mimic that of text retrieval engines in general. The system has already been used for curating an ERG-style treebank (Bender et al., 2012).

The system inherits some limitations of the underlying query language concerning matching of node labels. For example `NP`, `NP-SBJ`, and `NP-SBJ-1` are all distinct labels. Yet we would expect a query containing `NP` to match any of these. Possible solutions are to extend the language to support regular expressions over node labels, or to encode grammatical relations (such as SBJ) as node attributes and refer to them using filter expressions. The system is also limited in the sense that it only works with phrase structure trees. It cannot be applied to dependency trees in its current form, since the descendent and ancestor relations cannot be checked using a single span-inclusion test. In spite of these expressive limitations, *Fangorn* sets a new standard for efficiency and ease of use for tree query systems.

References

Bender, E. M., Ghodke, S., Baldwin, T., and Dridan, R. (2012). From database to treebank: On enhancing hypertext grammars with grammar engineering and treebank search. *Electronic Grammaticography*. To appear.

Bird, S., Chen, Y., Davidson, S. B., Lee, H., and Zheng, Y. (2006). Designing and evaluating an XPath dialect for linguistic queries. In *Proceedings of the 22nd International Conference on Data Engineering*, pages 52–61. IEEE Computer Society.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. In *Proceedings of the workshop on Treebanks and Linguistic Theories*, pages 24–41.

Cassidy, S. and Harrington, J. (2001). Multi-level annotation of speech: an overview of the Emu Speech Database Management System. *Speech Communication*, 33:61–77.

Ghodke, S. and Bird, S. (2008). Querying linguistic annotations. In McArthur, R., Thomas, P., Turpin, A., and Wu, M., editors, *Proceedings of the Thirteenth Australasian Document Computing Symposium*, pages 69–72, Hobart, Tasmania.

Ghodke, S. and Bird, S. (2010). Fast query for large treebanks. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 267–275, Los Angeles, California. Association for Computational Linguistics.

Grust, T., Keulen, M., and Teubner, J. (2003). Staircase join: Teach a relational dbms to watch its (axis) steps. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB)*, pages 524–535.

Heid, U., Voormann, H., Milde, J.-T., Gut, U., Erk, K., and Pado, S. (2004). Querying both time-aligned and hierarchical corpora with NXT search. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*.

Kepser, S. (2003). 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.

Lai, C. and Bird, S. (2004). Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of Australasian Language Technology Workshop*, pages 139–146.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–30.

Mírovský, J. (2006). Netgraph: a tool for searching in Prague Dependency Treebank 2.0. In *Proceedings of 5th International Conference on Treebanks and Linguistic Theories*, pages 211–222.

Petersen, U. (2004). Emdros: a text database engine for analyzed or annotated text. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1190–1193.

Rohde, D. (2001). Tgrep2 user manual. http://citeseer.ist.psu.edu/569487.html.

Skut, W., Krenn, B., Brants, T., and Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*, pages 88–95, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., and Lohman, G. (2001). On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 425–436, New York. ACM.