A Search Tool for Parallel Treebanks

Martin Volk, Joakim Lundborg and Maël Mettler
Stockholm University
Department of Linguistics
106 91 Stockholm, Sweden
volk@ling.su.se

Abstract

This paper describes a tool for aligning and searching parallel treebanks. Such treebanks are a new type of parallel corpora that come with syntactic annotation on both languages plus sub-sentential alignment. Our tool allows the visualization of tree pairs and the comfortable annotation of word and phrase alignments. It also allows monolingual and bilingual searches including the specification of alignment constraints. We show that the TIGER-Search query language can easily be combined with such alignment constraints to obtain a powerful cross-lingual query language.

1 Introduction

Recent years have seen a number of initiatives in building parallel treebanks. Our group has participated in these efforts by building a tri-lingual parallel treebank called SMULTRON (Stockholm MULtilingal TReebank).[1] Our parallel treebank consists of syntactically annotated sentences in three languages, taken from translated (i.e. parallel) documents. In addition, the syntax trees of corresponding sentence pairs are aligned on a sub-sentential level. This means they are aligned on word level or phrase level (some phrases can be as large as clauses). Parallel treebanks can be used as training or evaluation corpora for word and phrase alignment, as input for example-based machine translation (EBMT), as training corpora for transfer rules, or for translation studies.

Similar projects include Croco (Hansen-Schirra et al., 2006) which is aimed at building a German-English parallel treebank for translation studies, LinES an English-Swedish parallel treebank (Ahrenberg, 2007), and the Czech-English parallel dependency treebank built in Prague (Cmejrek et al., 2005).

SMULTRON is an English-German-Swedish parallel treebank (Samuelsson and Volk, 2006; Samuelsson and Volk, 2007). It contains the first two chapters of Jostein Gaarder's novel "Sofie's World" with about 500 sentences. In addition it contains 500 sentences from economy texts (a quarterly report by a multinational company as well as part of a bank's annual report). We have (semi-automatically) annotated the German sentences with Part-of-Speech tags and phrase structure trees (incl. edges labeled with functional information) following the NEGRA/TIGER guidelines for German treebanking.

For English we have used the Penn Treebank guidelines which are similar in that they also prescribe phrase structure trees (with PoS tags, but only partially annotated with functional labels). However they differ from the German guidelines in many details. For example the German trees use crossing edges for discontinuous units while the English trees introduce symbols for empty tokens plus secondary edges for the representation of such phenomena.

For Swedish there were no appropriate guidelines available. Therefore we have adapted the German

[1]We gratefully acknowledge financial support for the SMULTRON project by Granholms stiftelse and Rausings stiftelse.

Proceedings of the Linguistic Annotation Workshop, pages 85–92,
Prague, June 2007.

guidelines to Swedish. The general annotation strategy for Swedish was the same as for German: PoS tags, phrase structure trees (incl. functional edge labels) and crossing branches for discontinuous units. But, of course, there are linguistic differences between German and Swedish that required special attention (e.g. Swedish prepositions that introduce sentences).

The treebanks for all three languages were annotated separately with the help of the treebank editor ANNOTATE. After finishing the monolingual treebanks, the trees were exported from the accompanying SQL database and converted into an XML format as input to our alignment tool, the Stockholm TreeAligner.

In this paper we will first describe this alignment tool and then focus on its new search facility. To our knowledge this is the first dedicated tool that combines visualization, alignment and searching of parallel treebanks (although there are others who have experimented with parallel corpus searches (Nygaard and Johannesen, 2004; Petersen, 2006)).

2 The Stockholm TreeAligner

When our monolingual treebanks were finished, the trees were exported from the editor system and converted into TIGER-XML. TIGER-XML is a line-based (i.e. not nested and thus database-friendly) representation for graph structures which supports crossing edges and secondary edges.[2] TIGER-XML has been defined as input format for TIGER-Search, a query tool for monolingual treebanks (see section 3). We use this format also as input format for our alignment tool, the Stockholm TreeAligner (Volk et al., 2006).

The TreeAligner program is a graphical user interface to specify (or correct) word and phrase alignments between pairs of syntax trees.[3] The TreeAligner is roughly similar to alignment tools such as I*Link (Ahrenberg et al., 2002) or Cairo (Smith and Jahr, 2000) but it is especially tailored to visualize and align full syntax trees (including trees with crossing edges).

[2]For information about TIGER-XML see www.ims.uni-stuttgart.de/projekte/TIGER

[3]The TreeAligner is freely available at www.ling.su.se/DaLi/downloads/treealigner/index.htm

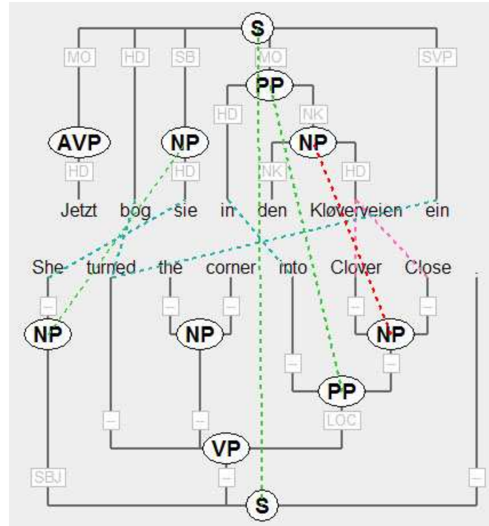

Figure 1: Tree pair German-English in the TreeAligner.

The TreeAligner operates on an alignment file in an XML format developed by us. This file describes the alignments between two TIGER-XML treebanks (specified in the alignment file) holding the trees from language one and language two respectively. For example the alignment between two nodes is represented as:

```
<align type="exact">
  <node id="s13_505" tb_id="DE"/>
  <node id="s14_506" tb_id="EN"/>
</align>
```

This says that node 505 in sentence 13 of the German treebank is aligned with node 506 in sentence 14 of the English treebank. The node identifiers refer to the ids in the TIGER-XML treebanks. The alignment is given the label “exact” if the corresponding token sequences are equivalent in meaning.

The alignment file might initially be empty when we want to start manual alignment from scratch, or it might contain automatically computed alignments for correction. The TreeAligner displays tree pairs with the trees in mirror orientation (one top-up and one top-down). See figure 1 for an example. The trees are displayed with node labels and edge labels. The PoS labels are omitted in the display since they are not relevant for the alignment task.[4]

[4]During the development of our treebanks we discovered

Each alignment is displayed as a dotted line between two nodes (or words) across two trees. Clicking on a node (or a word) in one tree and dragging the mouse pointer to a node (or a word) in the other tree inserts an alignment line. Currently the TreeAligner supports two types of alignment lines (displayed in different colors) which are used to indicate exact translation correspondence vs. approximate translation correspondence. However, our experiments indicate that eventually more alignment types will be needed to precisely represent different translation differences. The alignment type attribute can be used to describe many different levels or types of alignment. These distinctions could prove useful when exploiting the aligned treebanks for Machine Translation and other applications.

Often one tree needs to be aligned to two (or more) trees in the other language. The TreeAligner therefore provides the option to browse the trees independently. For instance, if we have aligned only a part of a tree T_i from language one to tree T_k of language two, we may scroll to tree T_{k+1} of language two in order to align the remaining parts of T_i. Special [Forward] and [Back] buttons are provided to browse through the multiple-aligned trees systematically.

The TreeAligner is designed as a stand-alone tool (i.e. it is not prepared for collaborative annotation). It stores every alignment in an XML file (in the format described above) as soon as the user moves to a new tree pair.

The TreeAligner was implemented in Python by Joakim Lundborg. Python has become popular in Language Technology in recent years. It is a high level programming language that allows different programming styles including a good support for object-oriented programming. It is an interpreted language that uses a dynamic type system. It is therefore mostly compared to its siblings Perl, Tcl and Ruby, even though the influence of other languages like Smalltalk and Haskell are probably stronger on a conceptual level.

One of Python's strengths is the ease with which a programmer can manipulate primitive data types like strings or numbers. Python's string objects are an excellent match to the needs of linguistic processing. In addition to the primitive data types, Python also features higher level data types: lists, tuples and dictionaries. The combination of these built-in data types, the vast standard library and the simple and straightforward syntax make Python the perfect tool for a wide range of scientific programming.

The TreeAligner served us well for creating the alignments, but it soon became evident that we needed suitable tools to explore and exploit the aligned data. The most apparent need was a search module for aligned trees. We decided to design our search module after TIGER-Search.

3 TIGER-Search

TIGER-Search is a powerful treebank query tool developed at the University of Stuttgart by Wolfgang Lezius (cf. (König and Lezius, 2002; Lezius, 2002). Its query language allows for feature-value descriptions of syntax graphs. It is similar in expressiveness to tgrep (Rohde, 2005) but it comes with graphical output and highlighting of the syntax trees plus nice frequency tables for objects identified in the query. TIGER-Search has been implemented in Java and is freely available for research purposes. Because of its clearly defined input format (TIGER-XML) and its powerful query language, it has become the corpus query system of choice for many linguists.

The TIGER-Search query language is based on feature-value descriptions of all linguistic objects (tokens and constituents), dominance, precedence and sibling relations in the tree, graph predicates (e.g. with respect to token arity and continuity), variables for referencing objects, regular expressions over values for varying the query precision, and queries over secondary edges (which constitute a secondary graph level).

A complex query might look like in the following example (with > denoting direct dominance, >* denoting general dominance, the dot denoting immediate precedence, and the # symbol introducing variables). This query is meant to find instances of two ambiguously located PPs that are both attached to the first noun (as illustrated by the example tree in figure 2).

that the TreeAligner is also useful for displaying different versions of the same treebank (e.g. before and after corrections, or manually vs. automatically parsed). Therefore we plan to add a tree-diff module which will highlight the differences between a pair of trees over the same token sequence.

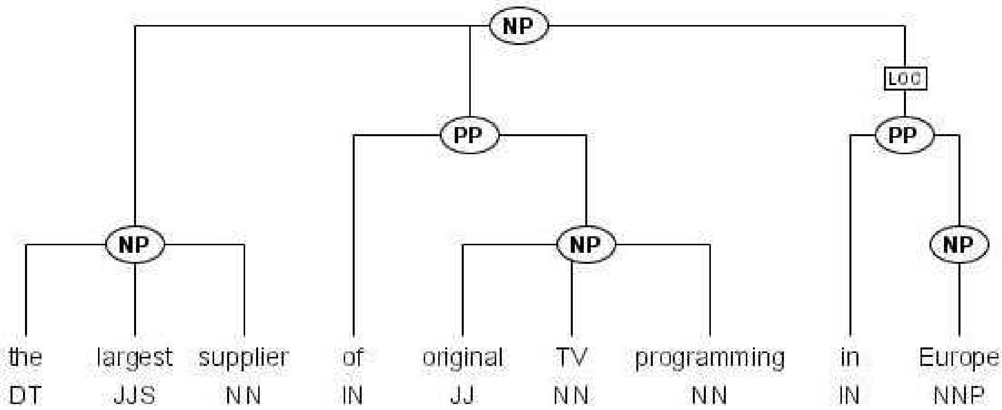

Figure 2: Noun phrase tree from the Penn Treebank

```
#np:[cat="NP"] >* #n1:[pos="NN"]&
#np  >  #pp1:[cat="PP"] &
#n1  .  #pp1 &
#pp1 >* #n2:[pos="NN"] &
#np  >  #pp2:[cat="PP"] &
#n2  .  #pp2
```

This query says: Search for an NP (call it #np) that dominates a noun #n1 (line 1) and two PPs (lines 2 and 5). #pp1 must follow immediately after the noun #n1 (line 3), and #pp2 must follow immediately after the noun within the #pp1 (lines 4 and 6).

TIGER-Search handles such queries efficiently based on a intricate indexing scheme. It finds all matching instances in a given treebank and allows to browse (and to export) the resulting trees. The matching objects in the resulting trees are highlighted.

TIGER-Search is limited in that it only allows manually entered queries (rather than processing a batch of queries from a file). Furthermore it is limited with regard to negation. The TIGER-Search query language includes a negation operator but this is of limited usefulness. The reason is that “For the sake of computational simplicity and tractability, the universal quantifier is (currently) not part of the TIGER language” (quoted from the TIGER-Search online help manual). This means that typical negated queries such as “Find all VPs which do **not** contain any NP” are not possible.

And clearly TIGER-Search is a tool for querying monolingual treebanks and thus needed to be extended for our purposes, i.e. querying parallel treebanks.

4 The TreeAligner Search Module

(Merz and Volk, 2005) had listed the requirements for a parallel treebank search tool. Based on these we have now re-implemented TIGER-Search for parallel treebanks and integrated it into the TreeAligner.

The idea is to allow the power of TIGER-Search queries on both treebanks plus additional alignment constraints. For example, a typical query could ask for a verb phrase VP dominating a prepositional phrase PP in treebank one. This query can be combined with the constraint that the VP in treebank one is aligned to a sentence S in treebank two which also dominates a PP. Such a query would be expressed in 3 lines as:

```
#t1:[cat="VP"] > [cat="PP"]
#t2:[cat="S"] > [cat="PP"]
#t1 * #t2
```

These three lines are entered into three separate input fields in the user interface (cf. the three input fields in the bottom left in figure 3). Lines 1 and 2 contain the queries over the monolingual treebanks 1 and 2. And line 3 contains the alignment constraint. Note that the treebank queries 1 and 2 closely follow the TIGER-Search syntax. In particular they allow the binding of variables (marked with #) to specific linguistic objects in the query. And these variables are used in the alignment constraint in line 3. The reuse of the variables is the cru-

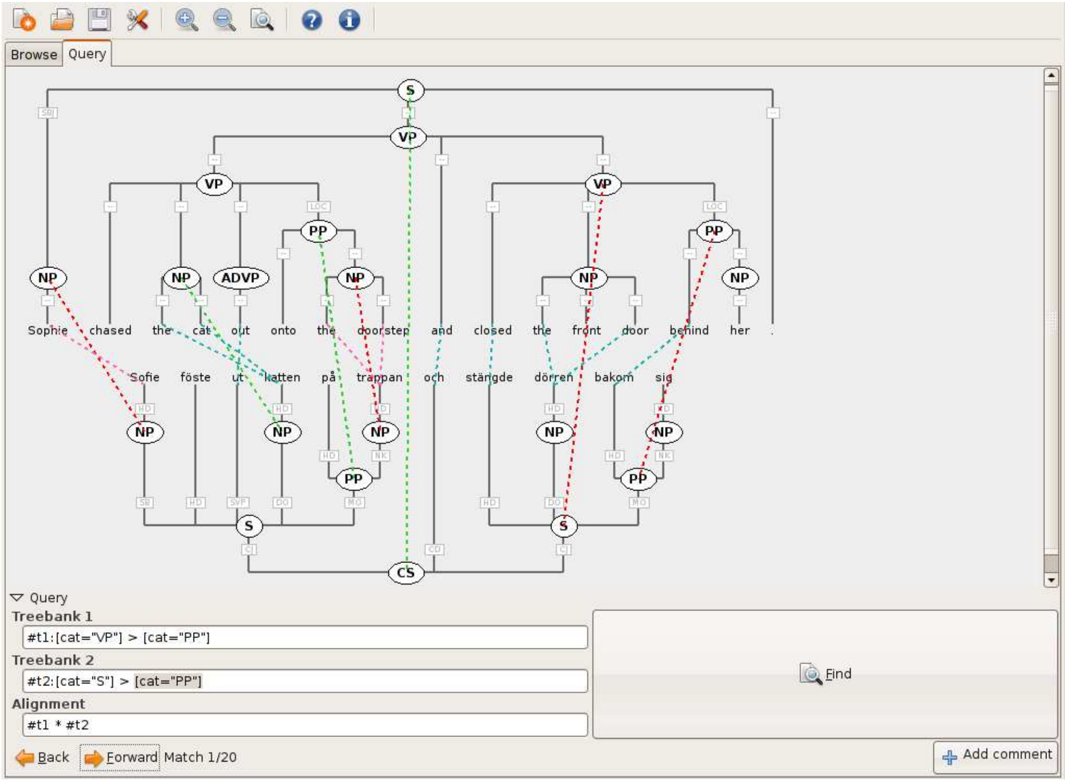

Figure 3: Screenshot of the TreeAligner with the Search Module.

cial idea which enabled a clear design of the Search Module by keeping the alignment constraints separate from the queries over the two treebanks.

So the above query will find the tree pair in figure 3 because it matches the alignment between the English VP *closed the front door behind her* and the elliptical Swedish sentence *stängde dörren bakom sig* (which lacks the subject, but is still annotated as S).

The Search Module has recently been added to the TreeAligner. It is intended to be used with any parallel treebank where the monolingual treebanks can be converted into TIGER-XML and where the alignment information can be converted to the SMULTRON alignment format. The separation of these parts makes it possible to query each treebank separately as well. The system is divided into a monolingual query facility and an alignment query facility that makes use of the former to perform its job. This design choice made it necessary to (re)implement the following in Python:

1. TIGER-Search
2. The alignment query facility
3. The integration into the TreeAligner

The choice of reimplementing TIGER-Search in Python influenced the feature set. Even though the implementation of TIGER-Search is well documented (in (Lezius, 2002) among others) and the source codes are available under an Open Source license, this is still a non-trivial task. In order to narrow down the amount of work in a first phase, it was decided to restrict the implementation to a subset of the TIGER-Search query language. The implementation of negation within the queries was therefore postponed (with the exception of negations used in regular expressions within a feature definition). As discussed in section 3, negations are limited even in TIGER-Search, and we plan to implement a comprehensive support for negation at a later stage. The code already has hooks for this extension.

The language for the alignment constraints is kept simple as well. The user can specify that two linguistic objects must be aligned (with exact alignment or approximate alignment). And such constraints can be combined with *AND* statements into more complex constraints. Currently, we cannot foresee exactly how a parallel treebank will be queried. We have therefore focused on a clear design of the Search Module rather than overloading it with features. This will facilitate the integration of more features as they are requested by users.

4.1 Implementation Details

The implementation of the Search Module started as a close re-implementation of the TIGER-Search system described in (Lezius, 2002). During the development it became apparent that some of Lezius' design choices did not translate well into Python. Moreover, the advancements concerning speed and memory in computer hardware in recent years have made it possible for us to deviate from the original design towards a more Python-oriented and simpler code with less considerations for resource limitations (see (Mettler, 2007)).

This code base can be divided into four types of functionality classes: helper, index, parser and processor. The **helper** classes are the smallest pieces of code and perform trivial tasks like sorting or set operations and are called from the other classes. The query system as such consists of the index, the parsers and processors. The **parsers** are used to transform a string such as the TIGER-XML files or the queries into objects. These parse objects are then used to create the index or are passed to a processor object to get the results of a query.

The **index** consists of four classes. The Corpus class governs the three others which are used to store the data for the graphs and the attribute value register that is defined in the TIGER-XML head. Each graph is contained within its own object. The attribute value register consists of one object that governs a range of attribute value lookup tables. There are three parser classes and one parser method. Each of these parser classes handles a different input. The first parses TIGER-XML, the second parses the node definitions within a TIGER-Search query (contained within the square brackets), and the third parser class uses them to parse complete TIGER-Search queries. As the syntax for the alignment constraints is simple, this was done within a method of the parallel query processor class. This is likely to change with the increasing feature set for parallel queries.

The last part of the system consists of two **processor** classes. The first is the class used for monolingual queries. On instantiation the class takes an index object and a query parser object as arguments. When the object's query method is called with a query string, the object lets the query parser produce a parse object from the string. The parse object is then processed to produce an object that contains the matching graph parts using the index. The processor for parallel queries works similarly. On instantiation a monolingual processor for each language is passed as arguments to the object. When the query method is called, the parallel processor objects gets the results from the monolingual processors first and then parses and processes the parallel query using the results from the monolingual processing step. The result of a query is a list with the two aligned sentence IDs.

4.2 Evaluation of the Search Module

The TreeAligner Search module was first tested by running a set of representative queries over a part of our English-German parallel treebank (500 tree pairs). This test set included:

- dominance relations (direct dominance, general dominance, labeled dominance, right and left corner dominance)
- precedence relations (immediate precedence, general precedence, sibling precedence, precedence distance)
- queries over secondary edges
- graph predicates (root, arity, tokenarity)

For the monolingual queries we checked whether the number of hits in our TreeAligner Search corresponded to the number of hits in TIGER-Search. This worked nicely. For bilingual queries we manually checked the correctness of the results.

We also tested the system for robustness and scalability. Since we currently do not have a large parallel treebank, we took the German NEGRA treebank with 10,000 trees and used it for both language one and language two in our TreeAligner. This means we used each tree aligned to a copy of itself as the basic data. This treebank contains around 81,000 nodes. We automatically generated an alignment file that contains each node aligned to its copy in the corresponding tree. This means we were using an alignment file with 81,000 alignments.

Unfortunately the time for loading this data set into the TreeAligner was prohibitively long (while loading a monolingual treebank with 10,000 trees into TIGER-Search takes less than a minute for indexing it once, plus few seconds for loading the index before starting the searches). Obviously, we need to improve the scalability of the TreeAligner.

When we redid the experiment with 1000 trees from the NEGRA treebank (with 35,756 alignments), it worked fine. Loading takes about one minute, and queries like the one given in the example above are processed in less than one minute. The system is currently not optimized for speed. It is a proof-of-concept system to demonstrate that the (monolingual) TIGER-Search query language can be elegantly extended with alignment constraints for parallel treebank searches.

Lately we have tested the use of serialized indexes. We have observed that they are much faster, but that the speed-up factor decreases with increasing file size. It seems that eventually we will have to switch to a custom binary format as was done in TIGER-Search, if we want to provide a smooth work experience with parallel treebanks of 10,000 and more trees.

5 Conclusions

We have built a TreeAligner for displaying and searching parallel aligned trees. The tool is written in Python and freely available. In particular it allows to align nodes and words across languages by drawing lines. We distinguish between exact and approximate alignment types. The search module which was recently added supports queries over both treebanks in combination with alignment constraints. The query language follows TIGER-Search (though negation is not included yet). The alignment constraints use the variables bound to linguistic objects in the monolingual queries.

In the future we will improve the TreeAligner in three directions: features, usability and evaluation. The feature part consists of providing full support for TIGER-Search queries (in particular the implementation of negation) and improving the parallel query facilities (with a variety of alignment constraints).

Moreover we are in the process of extending the TreeAligner to handling dependency trees. The TreeAligner currently imports only treebanks in TIGER-XML. This format is well suited for representing phrase structure trees but less for dependency trees. We will therefore extend the support to appropriate XML import formats.

Usability is the broadest group and aims at improvements like creating an installation routine for all operating systems, improving speed and making sure that UTF8 support works properly.

Finally, more systematic evaluations are needed. We plan to enlarge our standard set of queries to cover all possible combinations. This query set could then be used to test the speed and performance of our system (and for the comparison with other systems). We hope that the TreeAligner will gain a broad user community which will help to drive improvements in alignment and querying.

References

Lars Ahrenberg, Magnus Merkel, and Mikael Andersson. 2002. A system for incremental and interactive word linking. In *Proc. of LREC-2002*, pages 485–490, Las Palmas.

Lars Ahrenberg. 2007. LinES: An English-Swedish parallel treebank. In *Proc. of Nodalida*, Tartu.

Martin Cmejrek, Jan Curín, and Jirí Havelka. 2005. Prague Czech-English dependency treebank. Resource for structure-based MT. In *Proceedings of EAMT 10th Annual Conference*, Budapest.

Silvia Hansen-Schirra, Stella Neumann, and Mihaela Vela. 2006. Multi-dimensional annotation and alignment in an English-German translation corpus. In *Proceedings of the EACL Workshop on Multidimensional Markup in Natural Language Processing (NLPXML-2006)*, pages 35– 42, Trento.

Esther König and Wolfgang Lezius. 2002. The TIGER language - a description language for syntax graphs. Part 1: User's guidelines. Technical report.

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.

Charlotte Merz and Martin Volk. 2005. Requirements for a parallel treebank search tool. In *Proceedings of GLDV-Conference*, Sprache, Sprechen und Computer / Computer Studies in Language and Speech, Bonn, March. Peter Lang Verlag.

Maël Mettler. 2007. Parallel treebank search - the implementation of the Stockholm TreeAligner search. C-uppsats, Stockholm University, March.

Lars Nygaard and Janne Bondi Johannesen. 2004. SearchTree - a user-friendly treebank search interface. In *Proc. of 3rd Workshop on Treebanks and Linguistic Theories*, pages 183–189, Tübingen, December.

Ulrik Petersen. 2006. Querying both parallel and treebank corpora: Evaluation of a corpus query system. In *Proc. of LREC*, Genua.

Douglas L. T. Rohde, 2005. *TGrep2 User Manual*. MIT. Available from http://tedlab.mit.edu/ ∼dr/Tgrep2/.

Yvonne Samuelsson and Martin Volk. 2006. Phrase alignment in parallel treebanks. In Jan Hajic and Joakim Nivre, editors, *Proc. of the Fifth Workshop on Treebanks and Linguistic Theories*, pages 91–102, Prague, December.

Yvonne Samuelsson and Martin Volk. 2007. Alignment tools for parallel treebanks. In *Proceedings of GLDV Frühjahrstagung 2007*.

Noah A. Smith and Michael E. Jahr. 2000. Cairo: An alignment visualization tool. In *Proc. of LREC-2000*, Athens.

Martin Volk, Sofia Gustafson-Capková, Joakim Lundborg, Torsten Marek, Yvonne Samuelsson, and Frida Tidström. 2006. XML-based phrase alignment in parallel treebanks. In *Proc. of EACL Workshop on Multi-dimensional Markup in Natural Language Processing*, Trento, April.