The DTAG treebank tool. Annotating and querying treebanks and parallel treebanks

Matthias Buch-Kromann
Copenhagen Business School
Copenhagen, Denmark
mbk.isv@cbs.dk

Abstract

DTAG is a versatile annotation tool that supports manual and semi-automatic annotation of a wide range of linguistic phenomena, including the annotation of syntax, discourse, coreference, morphology, and word alignments. It includes commands for editing general labeled graphs and graph alignments, comparing annotations, managing annotation tasks, and interfacing with a revision control system. Its visualization component can display graphs and alignments for entire texts in a compact format, with a highly flexible and configurable formatting scheme. It also provides a powerful search-replace mechanism with queries based on full first-order logic, which can be used to search for linguistic constructions and automatically apply graph transformations to collections of annotated graphs.

1 Introduction

Treebanks and other linguistically annotated corpora have emerged as important resources in language technology and linguistic research. Since their creation involves a large amount of human effort, it is important to have flexible and efficient annotation tools that provide optimal support for the specific annotation task. There are many excellent tools that support the creation and use of various kinds of linguistic annotations, for example the TrEd tool for annotating dependency tree structures (Pajas and Štěpánek, 2008), the MMAX2 tool targeted primarily towards coreference annotation (Müller and Strube, 2006), the NITE XML toolkit for text and video markup (Carletta et al., 2005), the WordFreak tool for annotating syntax trees, named entities, and discourse (Morton and LaCivita, 2003), the Annotate tool for annotating syntax trees (Plaehn and Brants, 2000), and the Yawat word alignment tool (Germann, 2008).

In this paper, we will present the most recent version of DTAG (Buch-Kromann, 2010),[1] an open-source Linux-based treebank annotation tool written in Perl. DTAG is used in the annotation of the Copenhagen Dependency Treebanks (Buch-Kromann et al., 2009), a parallel treebank based on a Danish 80,000 word corpus translated into English, German, Italian and Spanish. In these treebanks, a text is analyzed as a primary tree structure which encodes discourse structure, syntax and morphology, augmented with secondary edges; word alignments are used to link the source text to its translations. This kind of annotation is not well-supported by existing annotation tools. DTAG complements existing annotation tools by providing particular support for the annotation of large, complex graphs with alignments.

The paper is organized as follows. In section 2, we describe the data structures supported by DTAG and provide examples of the visualizations used in DTAG. In section 3, we describe some of the tasks for which DTAG can be used. In section 4, we describe the file formats used by DTAG. Section 5 presents our conclusions.

2 Annotation structures

We will now describe the two kinds of annotation supported by DTAG: graphs and alignments.

2.1 Graphs

DTAG graphs consist of a set of linearly ordered nodes with an arbitrary number of attribute-value pairs, linked by an arbitrary set of directed edges with atomic string labels.[2] As an illustration, Fig-

[1] (Kromann, 2003) presents an early version of DTAG.

[2] The graph may contain comment nodes whose main function is to retain information losslessly from richer XML-based annotation formats, such as CES/XCES (Ide et al., 2000). This is an issue we ignore in this paper.

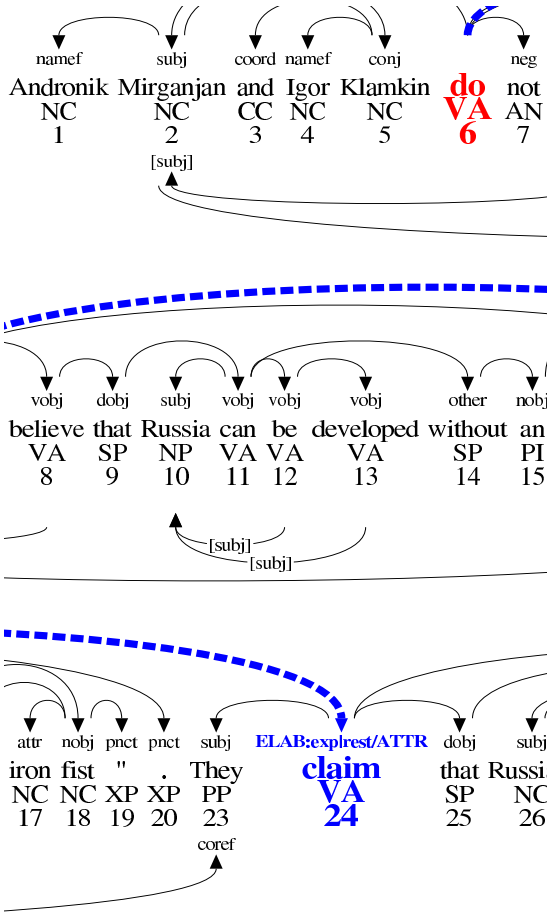

Figure 1: Excerpt from a larger dependency graph for a text annotated for part-of-speech, syntax, discourse structure, and coreference.

ure 1 shows a partial analysis of a text from the Copenhagen Dependency Treebanks. The analysis consists of a primary tree structure (shown above the words) which encodes the primary dependency relations between the words in the text, augmented by secondary relations (shown below the words) that encode coreference relations and secondary dependencies in filler-gap constructions. The arrows go from head to dependent, with the relation name shown at the arrow tip or the middle of the arrow. Phrase-structure analyses can be represented as DTAG graphs by inserting the phrasal nodes at an arbitrary point in the linear order (eg, right before any node within its subtree), as exemplified by Figure 2.

DTAG can display graphs in real time and export them to PostScript, PDF, and PNG format.[3]

[3]DTAG itself uses UTF-8 encoded text, but the visualization tools are based on PostScript which does not have built-in support for Unicode, so characters outside the ISO-8859-1 character set are not displayed properly.

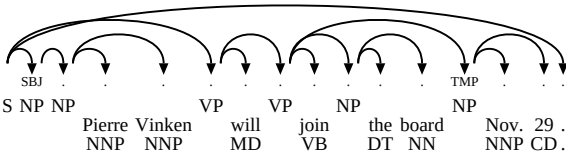

Figure 2: DTAG encoding of a phrase-structure tree from the Penn Treebank.

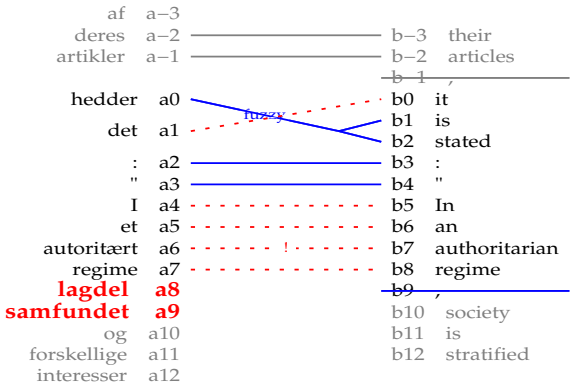

Figure 3: Excerpt from a larger word alignment (as displayed by DTAG's autoaligner).

The visualization is highly flexible, using customizable formatting rules that can be used for example to format the graph for discourse annotation by letting purely syntactic nodes appear in tiny font size, or to identify potential errors and visually highlight them to warn human annotators. In PDF and PostScript format, graphs can extend over several pages, allowing an entire text to be displayed as a single multi-page document.

2.2 Alignments

DTAG alignments link up a set of DTAG graphs by means of a set of labelled alignment edges, ie, labeled, directed hyperedges that connect sets of nodes rather than single nodes. DTAG alignments do not limit the number of graphs in the alignment, nor do they place any particular restrictions on the alignment edges themselves, except that all the out-nodes in an alignment edge must belong to the same graph, and similarly for all the in-nodes. An example of a DTAG alignment is shown in Figure 3. Like with graphs, the layout of DTAG alignments is highly flexible.

3 Supported annotation tasks

DTAG has a command-line based user interface in which all annotation commands are typed on the keyboard or read from script files. Sequences of commands can be defined as macros, and DTAG

COMMAND	EXPLANATION
load 0001-en.tag	*load graph from file*
viewer	*display graph in viewer*
1 namex 2	*add "namex" edge to 1 from 2*
del 1 namex 2	*delete it again*
1 namef 2	*add "namef" edge to 1 from 2*
2 subj 6	*add "subj" edge to 2 from 6*
2 [subj] 8	*add "[subj]" edge to 2 from 8*
<*return*>	*update viewer*
...	
oshow 13	*display from node 13, changing offset*
...	
save	*save graph*

Figure 4: Typical usage situation for the annotation of syntax, discourse, and coreference.

COMMAND	EXPLANATION
load 0001-da-en.tag	*load alignment from file*
viewer	*display alignment in viewer*
autoalign *.atag	*start autoaligner*
a0 b1	*align node a0 to b1 (no label)*
del a0	*delete all alignments at a0*
a0 fuz b1	*align node a0 to b1 with label "fuz"*
1+2 2..4+7	*align a1,a2 to b2,b3,b4,b7*
b0 b0	*align b0 to itself (null alignment)*
<*return*>	*update viewer*
ok	*accept current window and move on*
...	
save	*save alignment*

Figure 5: Typical usage situation for the annotation of alignments with the autoaligner.

provides a simple syntax for calling external programs via the UNIX shell. Any number of graphs and alignments can be loaded into memory and displayed on screen in real-time. We will now briefly describe some of the annotation tasks supported by DTAG.

3.1 Annotating graph structure: syntax, discourse, and coreference

DTAG provides commands for manually adding and deleting nodes and edges, and for semi-automatically revising edge labels for all edges that match a given set of relation names. The typical usage situation for the annotation of syntax, discourse, and coreference is exemplified by Figure 4: the annotator loads an automatically tagged or parsed text in TAG or CoNLL-X format, starts a PostScript viewer, and begins adding (and occasionally deleting) edges from the graph. The graph is updated whenever the annotator hits return on a blank line. When a chunk of text is completed, the annotator can ask DTAG to redisplay the graph from a given node, restarting the node numbering from that node if desired. When done, the corrected graph is saved to file again.

3.2 Annotating node attributes: part-of-speech and morphology

DTAG provides commands for manually editing node attributes, and for semi-automatically revising the attributes for all nodes that match a particular search query, reusing previous annotations as annotation suggestions. This is particularly useful for editing part-of-speech tags and morphological features, both when creating the annotations from scratch and when revising annotations produced by external automatic annotation tools.

3.3 Annotating graph alignments

Graph alignments can be edited manually, or semi-automatically using the built-in autoaligner. During the annotation, the autoaligner presents a window of 20 unaligned tokens to the annotator, as exemplified in Figure 3. The annotator's manual alignments are shown in solid blue, automatic suggestions in dotted red. Whenever the annotator adds or deletes an alignment, the autoaligner recomputes its alignment suggestions, always giving preference to human alignments. When finished, the annotator accepts the alignments in the current window, prompting the autoaligner to move to the next window of unaligned words (after being accepted, automatic alignments are shown in solid black). The autoaligner generates its alignment suggestions from a set of heuristic rules and a large alignment lexicon produced from a corpus of existing alignments; it can also take advantage of the automatic alignments produced by an external aligner. A typical usage situation for the annotation of alignments is given in Figure 5.

3.4 Comparing annotations

Comparing annotations is an important part of quality control in human annotation. DTAG provides commands for finding and visualizing the differences between two graphs or alignments with the same underlying text. An example is shown in Figure 6, where two dependency annotations of the same text have been compared with each other. The first graph is shown above the nodes and the second graph below, with differences highlighted in red. This feature can also be used to visualize the errors made by a dependency parser compared to a gold parse.

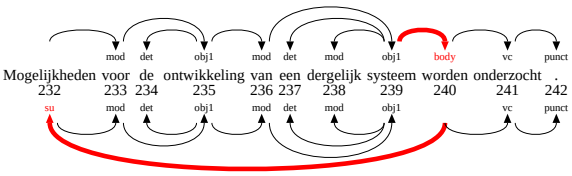

Figure 6: Excerpt from a visualization of the difference between two graphs.

3.5 Search and replace

DTAG provides a powerful query language that makes it possible to search for all nodes that match a qiven query in a given file or set of files. The query language is based on full first-order logic with negation and quantification.[4] Variables are recognized by being prefixed with "$" and always refer to nodes in the graph. DTAG defines a set of atomic predicates that can be used to place constraints on a node with respect to its attributes, its labelled edges, and its linear order in relation to other nodes.[5] For example, the command *find $X vobj $Y & ! exists($Z, $Z subj $X)* will find all tuples (X, Y) where X is a verbal object of Y, and X does not have any subject.

DTAG also has a powerful search-replace mechanism which allows the execution of a sequence of DTAG commands for each match. This search-replace mechanism can be used to perform automatic transformations of the annotation, such as renaming relation labels or attributes, editing attribute values, or modifying the graph structure. For example, in order to rename all *pred* edges to *preds*, we can use the command *find $X pred $Y -do(del $X pred $Y; $X preds $Y)*.

3.6 Tasks, version control, and annotation status

DTAG can interface with the Subversion version control system in order to save the entire revision history of the annotations and make sure that every annotator has the most recent revision of the annotation files. We have used the macro and scripting facilities in DTAG to implement a convenient interface to external scripts that control the revision control system, records the completion status of the different dimensions of the annotation, and allows the annotator to maintain a task list with the texts waiting to be annotated. These scripts also provide statistics about the completion status of the annotation project.

[4] Queries are treated as a constraint-satisfaction problem in which free variables must be resolved. The query algorithm transforms the query into disjunctive normal form, and then processes the atomic predicates in each of the conjunctions in a parallel most-restrictive-first manner.

[5] This makes DTAG queries as expressive as in the query system FSQ (Kepser, 2003), and more powerful in some respects than query systems such as Tgrep2 (Rohde, 2001), TIGERSearch (König et al., 2003), Netgraph (Mírovský, 2008), and MonaSearch (Maryns and Kepser, 2009), which either restrict themselves to tree structures, or do not provide full support for negation and quantification.

```
<DTAGalign>
<alignFile key="a" href="0001-da.tag"/>
<alignFile key="b" href="0001-en.tag"/>
...
<align out="a86" type="f" in="b49 b50"/>
<align out="a87" type="" in="b48"/>
<align out="a88" type="" in="b51"/>
...
</DTAGalign>
```

Figure 8: Excerpt from the ATAG file used to encode the alignment shown in Figure 3, which was generated using offsets a86 and b48.

4 File formats

DTAG has its own native file formats for storing graphs and alignments, the TAG graph format and the ATAG alignment format, respectively. The TAG graph format, exemplified by the excerpt in Figure 7, is a simple line-based format in UTF-8 encoding in which non-comment nodes are specified as <W> elements; all other lines are retained verbatim as comment nodes. The *in* and *out* attributes of <W> are used to encode the node's in- and out-edges, all other attributes represent node attributes. Edges are given as pairings of relation labels with relative line positions.

The ATAG alignment format is also an XML-like line-based format in UTF-8 encoding, as exemplified by the excerpt in Figure 8. The ATAG format contains a sequence of *alignFile* lines that identify the graphs that are being aligned and their associated identifier keys, followed by a sequence of *align* lines that encode the alignment edges.

DTAG additionally supports three other graph formats for import and export: TIGER-XML format (Mengel and Lezius, 2000), CoNLL format (Buchholz and Marsi, 2006), and Joakim Nivre's MALT-TAB format. Many other formats (including Penn Treebank format) can be imported into DTAG via TIGER-XML using the conversion filters provided with TIGERSearch (König et al., 2003). Other formats are supported upon request.

```
<s>
<W msd="NC" tag="NP--U==-" in="1:namef" out="">Andronik</W>
<W msd="NC" tag="NP--U==-" in="4:subj" out="-1:namef|3:conj|6:[subj]">Mirganjan</W>
<W msd="CC" tag="CC" in="2:coord" out="">and</W>
<W msd="NC" tag="NP--U==-" in="1:namef" out="">Igor</W>
<W msd="NC" tag="NP--U==-" in="-3:conj" out="-1:namef|-2:coord">Klamkin</W>
```

Figure 7: The first six lines of the TAG file for Figure 1. The first line is interpreted as a comment node.

5 Conclusion

We have described DTAG, a versatile annotation tool which can be used to produce human-edited annotations for a wide range of phenomena, including syntax, discourse, coreference, part-of-speech, morphology, and alignments. DTAG has a highly configurable visualization component for both graphs and alignments which provides compact visualizations of large texts. DTAG provides commands for comparing annotations visually, and for semi-automatic annotation of alignments and node attributes. It is equipped with a search-replace mechanism where queries are expressed as formulas within a powerful query language based on full first-order logic (with negation and quantification), and replacement actions are specified as arbitrary sequences of DTAG commands; the search-replace mechanism makes it easy to search for specific linguistic constructions, and to automatically rename labels and attributes or perform powerful graph operations in large annotated corpora.

References

Matthias Buch-Kromann, Iørn Korzen, and Henrik Høeg Müller. 2009. Uncovering the 'lost' structure of translations with parallel treebanks. In Fabio Alves, Susanne Göpferich, and Inger Mees, editors, *Methodology, Technology and Innovation in Translation Process Research*, volume 38 of *Special issue of Copenhagen Studies of Language*, pages 199–224.

Matthias Buch-Kromann. 2010. The DTAG treebank tool. http://code.google.com/p/copenhagen-dependency-treebank/wiki/DTAG.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on Multilingual Dependency Parsing. In *Proc. CoNLL-2006*.

J. Carletta, S. Evert, U. Heid, and J. Kilgour. 2005. The NITE XML toolkit: data model and query. *Language Resources and Evaluation Journal*, 39(4):313–334.

Ulrich Germann. 2008. Yawat: Yet Another Word Alignment Tool. In *Proc. ACL-08: HLT Demo Session*, pages 20–23.

N. Ide, P. Bonhomme, and L. Romary. 2000. XCES: An XML-based standard for linguistic corpora. In *Proc. LREC 2000*, pages 825–30.

Stephan Kepser. 2003. Finite Structure Query – a tool for querying syntactically annotated corpora. In *Proc. EACL 2003*, pages 179–186.

Esther König, Wolfgang Lezius, and Holger Voormann. 2003. TIGERSearch 2.1 user's manual. IMS, Univ. of Stuttgart.

Matthias T. Kromann. 2003. The Danish Dependency Treebank and the DTAG treebank tool. In *Proc. Treebanks and Linguistic Theories 2003*.

Hendrik Maryns and Stephan Kepser. 2009. MonaSearch – a tool for querying linguistic treebanks. In *Proc. Treebanks and Linguistic Theories 2009*, pages 29–40.

Andreas Mengel and Wolfgang Lezius. 2000. An XML-based encoding format for syntactically annotated corpora. In *Proc. LREC 2000*, pages 121–126.

Jiří Mírovský. 2008. Netgraph – making searching in treebanks easy. In *Proc. IJCNLP 2008*, pages 945–950.

Thomas Morton and Jeremy LaCivita. 2003. Wordfreak: an open tool for linguistic annotation. In *Proc. NAACL 2003*, pages 17–18.

Christoph Müller and Michael Strube. 2006. Multi-level annotation of linguistic data with MMAX2. In Sabine Braun, Kurt Kohn, and Joybrato Mukherjee, editors, *Corpus Technology and Language Pedagogy: New Resources, New Tools, New Methods*, pages 197–214. Peter Lang, Frankfurt a.M., Germany.

Petr Pajas and Jan Štěpánek. 2008. Recent advances in a feature-rich framework for treebank annotation. In *Proc. Coling 2008*.

Oliver Plaehn and Thorsten Brants. 2000. Annotate — an efficient interactive annotation tool. In *Proc. ANLP 2000*.

Douglas L. T. Rohde. 2001. Tgrep2 user manual.