AlpinoGraph: A Graph-based Search Engine for Flexible and Efficient Treebank Search

Peter Kleiweg
University of Groningen
p.c.j.kleiweg@rug.nl

Gertjan van Noord
University of Groningen
g.j.m.van.noord@rug.nl

Abstract

AlpinoGraph is a graph-based search engine which provides treebank search using SQL database technology coupled with the Cypher query language for graphs. In the paper, we show that AlpinoGraph is a very powerful and very flexible approach towards treebank search. At the same time, AlpinoGraph is efficient. Currently, AlpinoGraph is applicable for all standard Dutch treebanks. We compare the Cypher queries in AlpinoGraph with the XPath queries used in earlier treebank search applications for the same treebanks. We also present a pre-processing technique which speeds up query processing dramatically in some cases, and is applicable beyond AlpinoGraph.

1 Introduction

Traditionally, treebanks are, of course, collections of trees. Search engines for treebanks therefore often exploit this tree-like nature. Early treebank search tools such as `tgrep`, `tgrep2`, `lpath` (Rohde, 2001; Bird and Lai, 2010) provide a specialized query language over trees. For Dutch, similarly, current tools (van Noord, 2009; Augustinus et al., 2012; van Noord et al., 2013; Odijk et al., 2017; Augustinus et al., 2017; van Noord et al., 2020) are built with the XPath query language which is a standard query language for XML documents. XML documents are, in essence, trees too.

Obviously, not all linguistic annotations fit the concept of trees, and in most treebanks there are ways to encode, for instance, discontinuous constituents, secondary edges, enhanced dependencies etc. Also, feature structures such as those that arise in constraint-based grammatical frameworks (LFG, HPSG, ...) are directed graphs, not trees. It can be argued, therefore, that graphs are a better representation for linguistic annotation. And indeed, several treebank search systems have been based on graphs (Mírovský, 2008; Proisl and Uhrig, 2012; Bonfante et al., 2018).

In this paper, we argue in addition that a graph-like representation is useful because it allows for a straight-forward combination of different types of annotation and annotation layers. In the AlpinoGraph application, four different annotation layers are combined (automatically), including two layers for Universal Dependencies (standard and enhanced) (Nivre et al., 2018), (Bouma and van Noord, 2017), the original Lassy annotation layer (van Eynde, 2005; van Noord et al., 2019), and a simple layer of word pairs inherited from PaQu (Odijk et al., 2017).

The representations used in AlpinoGraph are automatically derived from existing treebanks in the Lassy XML format, a hybrid dependency format with some categorical information as well, originally based on and developed as an alternative of the format used in the Tiger treebank (Brants et al., 2004) and the Dutch Spoken Corpus (CGN)(Schuurman et al., 2003). In addition, information is derived from the CoNLL-U format for Universal Dependencies. In fact, the UD treebanks for Dutch are automatically derived from the treebanks in the Lassy XML format. It should be straightforward to map treebanks in CoNLL-U format (including all UD treebanks) into AlpinoGraph.

In this paper, we do not consider the potential linguistic advantages of graph-based representations, since the linguistic annotations are derived from existing resources, and no further manual annotation efforts have been invested for AlpinoGraph.

AlpinoGraph is built on AgensGraph. AgensGraph provides database technology (PostgreSQL) with the standard search language for graphs, Cypher. This combination provides, on the one hand, a very powerful query language which allows to express very complex linguistic patterns. On the other hand, the database tools ensure a very flexible tool which not only is capable of identifying relevant sentences, but also provides a wealth of functionality for aggregating the information of relevant sentences, structures or words.

In the next sections, we describe how treebanks are represented as graphs, and how we can formulate simple queries over such treebanks. In the fourth section, we compare AlpinoGraph on the full set of more than a hundred queries that are available in the SPOD extension of PaQu (van Noord et al., 2020). This comparison illustrates not only that the tool provides the required expressive power, but also shows that the tool is much faster for our purposes. In section 5, we present a search optimization technique which improves speed for some queries enormously. The technique appears to be applicable for most other treebank search systems.

2 Treebanks as Graphs

Graphs consist of vertices and edges. In AlpinoGraph, vertices can be words as well as constituents. A vertex is written as `()`. A vertex of type word is written as `(:word)`, and a vertex of a higher level constituent, a "node", is written as `(:node)`. We can use the notation `(:nw)` as an alias for a vertex that could be either a word or a node.

If we want to provide further information of a vertex, we use attribute and values within curly brackets. For instance, `(:node{cat:'np'})`, denotes a noun phrase. If we need to refer to a particular vertex, we can place a variable directly after the opening bracket: `(v:node{cat:'np'})`. Here, `v` functions as a variable that we can refer to later.

Edges are represented in much the same way, except that square brackets are used. We use edges to represent universal dependencies. Such dependencies are of type `ud`. For example, the direct object universal dependency is written as `[:ud{rel: 'obj'}]`.

We can use path expressions to combine vertices and edges. Such expressions look like:

```
() -[]-> ()
() <-[]- ()
```

Between brackets, we can specify further requirements. For instance, the following expression describes the direct object relation between a verb and some word `n`:

```
(:word{upos:'verb'}) -[:ud{rel:'obj'}]-> (n:word)
```

If this expression is used in a search, the variable `n` would be instantiated to (heads of) direct objects of verbs.

Each sentence in AlpinoGraph is represented by a graph where the vertices are words and nodes, and a single vertex of type `:sentence`. The attributes of the words are all the attributes available in the standard Lassy annotation guidelines (van Eynde, 2005; van Noord et al., 2019), as well as all the attributes in the UD representation. The attributes of nodes include the attribute `cat` and a few others that we can ignore for now. Multi-word units also have attributes for `word` and `lemma`.

The edges come in four different types for the representation of dependencies: standard universal dependencies `:ud`, enhanced universal dependencies `:eud`, Lassy dependencies `:rel` and simplified Lassy dependencies `:pair` (inherited from the word-pair part of the PaQu search tool). A fifth type of edge is `:next` which links each word to the next word in the sentence.

The Lassy-type dependencies look as follows:

```
(:sentence) -[:rel{rel: 'top'}]-> (:node{cat: 'top'})
(:node) -[:rel]-> (:node)
(:node) -[:rel]-> (:word)
(:node) -[:rel]-> (:nw)
```

The standard UD-type dependencies look as follows:

```
(:sentence) -[:ud{rel: 'root'}]-> (:word)
(:word) -[:ud]-> (:word)
```

And words are connected by means of the `:next` edges:

```
(:word) -[:next]-> (:word)
```

Paths can be longer than an edge connecting two vertices. This path identifies the root of the sentence that has a subject:

```
(:sentence) -[:ud]-> () -[:ud{rel:'nsubj'}]-> ()
```

3 AlpinoGraph by Example

Simple AlpinoGraph queries can be built using the path expressions of the previous section. For example:

```
match (:word{lemma:'drinken'})-[:ud{rel:'obj'}]->(o:word{upos:'NOUN'})
return o
```

For a given corpus, this query will return all direct object nouns of the verb with lemma *to drink*. It is straightforward to combine edges of different types in a query. For instance, suppose you are interested to find (heads of) direct objects which are double-quoted. This can be accomplished by identifying direct objects (in the first clause), and then requiring that both the words to the left and the right are double-quotes:

```
match ()-[:ud{rel:'obj'}]->(o:word),
      (:word{lemma:'"'}) -[:next]-> (o)-[:next]->(:word{lemma:'"'})
return o
```

Queries can also return multiple values. And the values need not be vertices, but could also be edges. Using the ’.’-operator you can also return the attributes of vertices or edges. The following example finds nodes with a verb as the head and an indirect object. The result is a table of pairs consisting of the lemma of the verb and the category of the indirect object.

```
match (v:word{pt:'ww'})<-[:rel{rel:'hd'}]-(:node)-[:rel{rel:'obj2'}]->(w:node)
return v.lemma, w.cat
```

It is straightforward to add further conditions on a pattern. The following example provides an illustration, where we want to collect direct objects of the verb "to eat", but ignoring the cases where the direct object is a pronoun:

```
match (:word{lemma:'eten'})-[:ud{main:'obj'}]->(w2:word)
where w2.upos != 'PRON'
return w2
```

In addition to simply returning the matches, we can perform a variety of aggregations on those.

```
match (:word{lemma:'eten'})-[:ud{main:'obj'}]->(w2)
where w2.upos != 'PRON'
return w2.lemma, count(w2.lemma) as frequency
order by frequency desc
```

This results in a table of lemmas with their respective frequencies in decreasing order.

4 Representing secondary edges

In the Lassy treebank, secondary edges are represented using an index attribute associated to nodes of the tree to indicate reentrancies in the graph. In AlpinoGraph, such secondary edges are represented in much the same way as primary edges (although an attribute is added to ensure that the difference can be recovered in the relevant cases). An example will illustrate this.

In the annotation of passives, the subject of the passive auxiliary is also annotated to be the object of the embedded verb. An as example, sentence 1 gets analysed as in the left part of figure 1. In contrast, such "secondary edges" are represented in AlpinoGraph as first class citizens. Since AlpinoGraph is graph-based, there is no problem by having two edges connecting to "het brood". In AlpinoGraph, the resulting graph is displayed on the right of figure 1 (including the UD representation layer for further illustration).

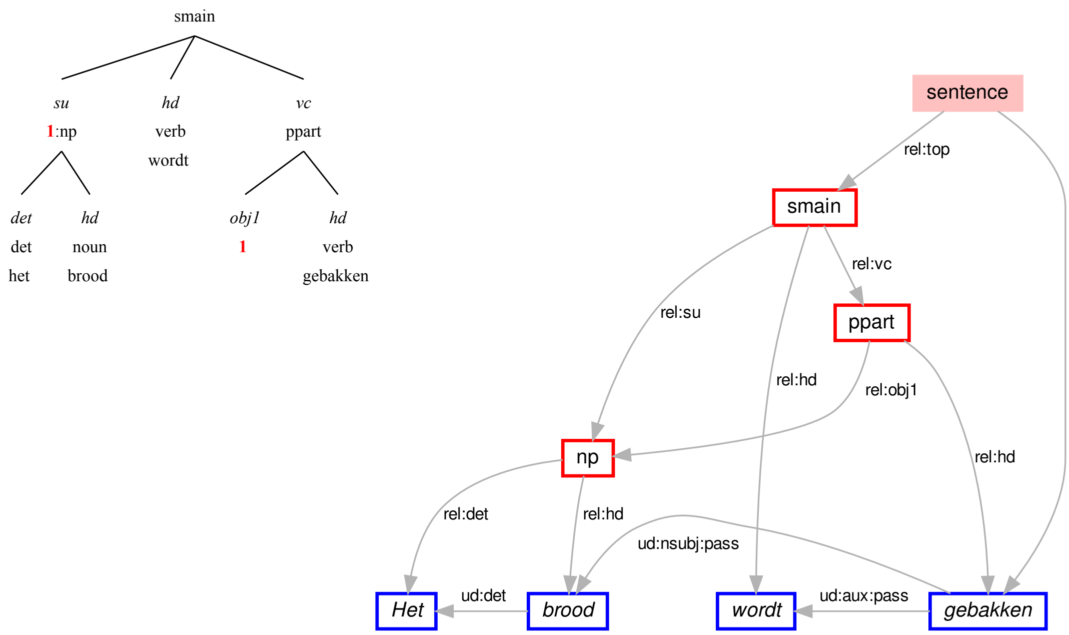

Figure 1: Original Lassy annotation of *Het brood wordt gebakken* (left) and the representation in AlpinoGraph (right), displaying two (:rel and :ud) of the available representation layers.

(1) Het brood wordt gebakken
The bread is baked

5 Comparison with XPath

5.1 Treebanks and queries

In this section, we compare the Cypher queries of AlpinoGraph with equivalent XPath queries used in the earlier treebank search systems DACT(van Noord et al., 2013), GrETEL(Augustinus et al., 2012), PaQu(Odijk et al., 2017). The comparison between XPath and Cypher is based on the same treebanks, for a large number of queries. We thus need a large number of relevant linguistic queries. This representative set of linguistic queries is taken from the SPOD extension of PaQu. SPOD (Syntactic Profiler of Dutch) (van Noord et al., 2020) provides an interface to a set of over a hundred linguistic queries which can be used to compare texts and corpora. These queries are supposed to be generally useful to obtain a good characterization of the syntactic properties of a text. SPOD has been used to study, for instance, the writing development of Dutch school children. The list of queries has been established in close connection with linguists.

The queries are applied for four different treebanks, described here as follows.

Alpino Treebank. The Alpino Treebank (van der Beek et al., 2002) contains over 7 thousand manually annotated sentences which constitute the newspaper ("cdbl") part of the Eindhoven corpus (uit den Boogaart, 1975). This treebank is one of the UD treebanks. It is available both in CoNLL-U and Lassy XML format.

CGN. The CGN treebank contains the manually syntactically annotated part of CGN ("Corpus of Spoken Dutch") (Schuurman et al., 2003). The treebank consists of 1 million words. The CGN annotation format has been automatically converted to the Lassy XML format.

Eindhoven. The Eindhoven treebank contains over 40 thousand *automatically* annotated sentences. The annotations are provided by the Alpino parser (van Noord, 2006), in the Lassy XML format.

Lassy Small. The Lassy Small treebank (van Noord et al., 2013) is the de facto standard treebank of written Dutch. The size of the manually annotated corpus is 1 million words, and the corpus consists of a variety of text types. Part of this treebank is available as one of the UD treebanks (the limitation is due to copyright reasons).

The list of linguistic queries from SPOD contains 102 items (we ignore the queries about parser performance since most of our treebanks are manually developed). Of those, 18 queries are not available for the timing experiment because the Cypher queries exploit an efficiency improvement which we will only discuss in section 5. Since that improvement is somewhat independent of the actual query engine, including those queries here would be unfair. A further complication is that the automatically annotated treebanks include some information on separable verb prefixes that is not available in the manually annotated treebanks. SPOD includes 6 queries which focus on that information, so naturally those 6 queries are only applied for the Eindhoven treebank. Finally, the CGN treebank pre-dates the other treebanks and does not include certain types of secondary edges which have been added systematically to later treebanks. For that reason, three queries are not applicable to the CGN treebank. Table 1 summarizes the number of queries used per treebank. We list both the number of queries used to compare the results (left) and the number of queries used in the timing experiment (right).

	results	timing
queries in SPOD	102	84
Eindhoven	102	84
Lassy Small	96	78
Alpino Treebank	96	78
CGN	93	75

Table 1: Number of queries used in the two experiments per treebank. On the left, the number of queries used to compare the number of results. On the right, the number of queries used for the timing experiment.

5.2 Differences in query results

The queries available in SPOD have all been re-implemented in AlpinoGraph. As a consequence, we can compare the results of running the original XPath queries on the one hand, and running the newly implemented Cypher queries in AlpinoGraph on the other hand. During the development of the Cypher queries, we carefully compared if the Cypher queries returned the same hits as the corresponding XPath query. In a limited number of cases, it turned out quite hard to obtain precisely the same set of hits. There are two classes of cases where the number of hits differs for some of the queries. Firstly, while we were re-implementing the queries in AlpinoGraph we found a number of subtle problems with the original XPath queries. A few cases are reported below. Secondly, a further important difference is the representation of "secondary edges".

5.2.1 Query improvements

During the process of re-implementing the SPOD queries in AlpinoGraph, we encountered a small number of subtle problems with the original XPath queries.

A simple example concerns the identification of noun phrases. Word groups are labeled by a category attribute, so any node with category "np" is a noun phrase. However, category features are used only for word groups and not for single words. Therefore, single-word noun phrases such as pronouns do not have a category attribute. If noun phrases have to be identified in XPath queries, a disjunction is used to include both word groups with the relevant category attribute as well as single words with appropriate part-of-speech attributes. A further complication arises for coordination. A coordination of two noun phrases is assigned "conj" as category attribute, not "np". In PaQu, a macro is defined to specify what it means to be a noun phrase. That macro essentially states that you are a noun phrase if you are a basic noun phrase, or if you are a coordination of basic noun phrases. And a basic noun phrase is a word group with category "np", or a word with the appropriate part of speech tag (noun, pronoun, proper name). This definition missed the cases where a conjunction was built up of two NP conjunctions, as in:

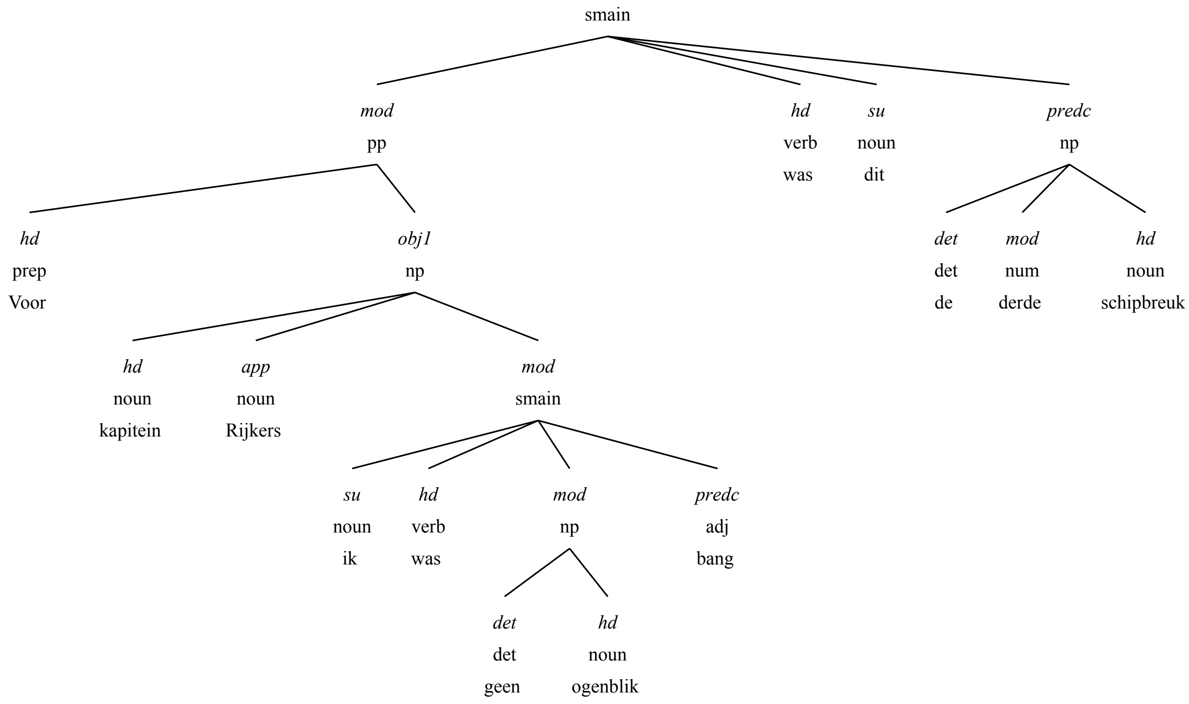

Figure 2: Annotation of *Voor kapitein Rijkers "(ik was geen ogenblik bang)" was dit de derde schipbreuk*

(2) de problemen van misdaad en straf , schuld en vergeving
the problems of crime and punishment , guilt and forgiveness

A more intricate example concerns the definition of the topic position in main sentences, in Germanic studies often called the "vorfeld": the word group that precedes the finite verb in V2 main sentences. In the hybrid dependency annotation format, it is somewhat complicated to define this word group. The word group should be a dependent (either direct or indirectly) of the finite verb, and it should (directly) precede that finite verb.

(3) het huis op de heuvel wordt verkocht
the house on the hill is sold

In this example, "het huis op de heuvel" satisfies those conditions and is a potential vorfeld. In order to rule out dependents of the actual vorfeld constituent, in the example "op de heuvel", the XPath query furthermore required that the vorfeld candidate should not be part of a constituent which is itself a vorfeld candidate. However, after comparing the results of the XPath query and the Cypher variant, it became apparent that this added condition was a bit too strict. That condition also rules out vorfelds of embedded main sentences. An example is listed where, with the analysis illustrated in figure 2. The original XPath query thus missed the fact that "ik" here also should be considered a vorfeld constituent.

(4) Voor kapitein Rijkers " (ik was geen ogenblik bang) " was dit de derde schipbreuk .
for captain Rijkers " (I was no moment afraid) " was this the third shipwreck .

5.3 Differences for secondary edges

Complements of fixed verbal expressions are labeled using the relation "svp". In a few cases, such a fixed part of a fixed verbal expression also functions as the subject in a passive-like construction, as in example 5, analysed as in the left part of figure 3.

(5) Wel werd meteen groot alarm geslagen
Indeed was immediately major alarm raised
"however, a major alarm was immediately raised"

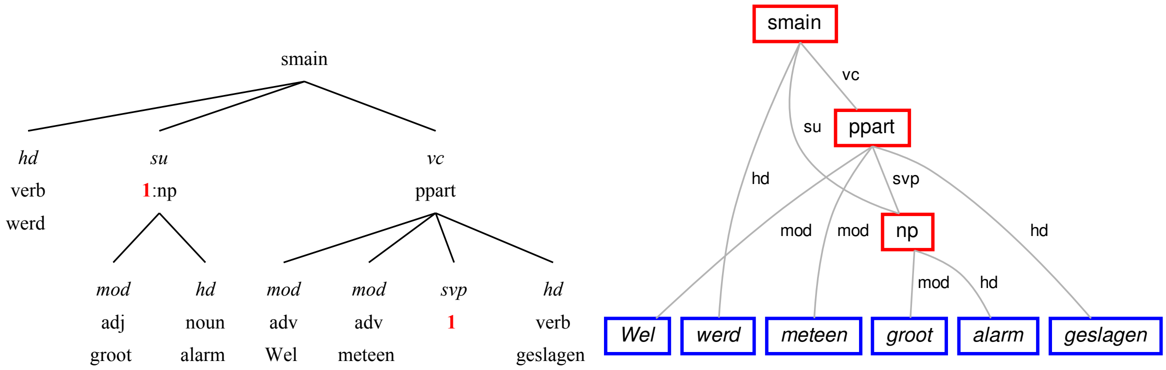

Figure 3: Representation of *Wel werd meteen groot alarm geslagen* in Lassy XML (left) and the :rel representation layer in AlpinoGraph (right)

One of the SPOD queries identifies complements of fixed verbal expressions. It does so using a simple query which identifies all nodes that have a "svp" dependency with a verbal head. In that query, however, no special consideration was made for cases where that node only contains an index. Therefore, the word group "groot alarm" is not found by the XPath query. The right part of figure 3 illustrates the representation used in AlpinoGraph. As a consequence, the AlpinoGraph variant of the query to identify complements of fixed verbal expressions will identify the "groot alarm" word group as a hit.

Almost all differences between XPath and Cypher are caused by this difference in representation of secondary edges, and in most of these cases, the Cypher version of the query is in fact closer to the linguistic intention of the query - as in our running example. As a side note, going over the differences revealed quite a few manual annotation mistakes too.

5.4 Timing experiment

In addition to a comparison of the results of the various queries, it is also interesting to consider the speed of the various queries for both XPath and Cypher.

As explained in the first paragraph of this section, we compare the cputime requirements for about 80 queries applied to four different treebanks. The results are presented in figure 4. Both axes of the graph are in logarithmic scale. Each dot in the graph represents the cputime it took to finish a particular query for a particular treebank. The Y-axis represents the cputime taken by the XPath queries, whereas the X-axis represents the time taken by the Cypher queries.

As can be observed in the graph, in most cases, but not all, the evaluation of the Cypher queries by AlpinoGraph is much faster than the evaluation of the XPath queries. For the few cases for which the Cypher query is slower, the difference is relatively small.

6 Search optimization

Both Cypher and XPath are expressive enough to define complex syntactic patterns. Some of these patterns occur quite frequently. For example, in the Lassy dependency structures, the topological fields known from Germanic syntax, such as *vorfeld*, *mittelfeld* and *nachfeld* are not explicitly encoded. Yet, it is possible to define Cypher expressions and XPath expressions which recover this information. However, such complicated patterns are relatively hard to compute.

The properties of nodes that we regularly want to refer to can be pre-computed. For instance, a special attribute `_vorfeld` has been added in the representation of treebanks in AlpinoGraph. This attribute is assigned the value "True" for the relevant nodes at the time when the corpus is loaded into AlpinoGraph.

Without such an attribute, it would be possible to identify vorfeld constituents using a Cypher query, but that query is quite complicated, since it must recover the surface syntax of the sentence on the basis of a dependency graph. The actual query identifies potential vorfelds which are (potentially indirect) dependents of the finite verb which precede that finite verb. From those potential vorfelds, the query

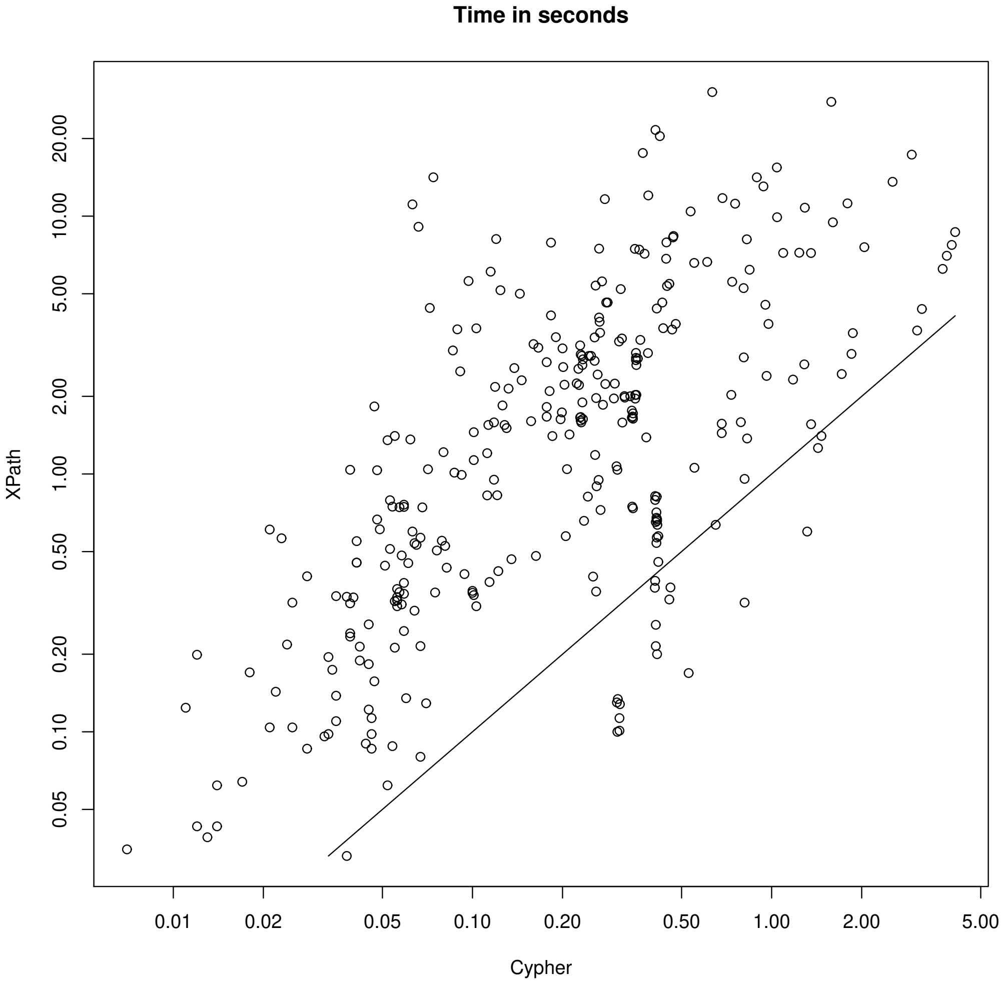

Figure 4: Timing all queries. Note the logarithmic scale. All results above the straight line are cases where the Cypher queries are faster. The few results below the line indicate queries for which the Cypher queries were slower.

then further extracts the maximal one. A further complication is that parts of the vorfeld constituent may actually be extraposed. The full query is given in the appendix.

Running the complicated query over the Lassy Small corpus to identify vorfelds in the corpus takes almost four minutes. After coding the property as an attribute of the relevant nodes, the following, trivial, query finishes within 100 msec:

```
match (n:nw{_vorfeld: true})
return n
```

Properties of nodes that are often used in treebank queries can be encoded by simple attributes. We developed a tool which takes a treebank, a query, an attribute and a value. Each node in the treebank that satisfies the query is augmented with the given attribute and value. This way, treebanks can be enriched with, essentially, redundant information. The benefit will be that queries which rely on that information can be expressed much simpler and will be evaluated much faster.

7 Concluding remarks

In this paper, we introduced AlpinoGraph, a novel graph-based treebank search engine, based on the Cypher query language for graphs. We argued that graphs are an appropriate representation for linguistic annotation, in particular if several annotation layers are combined. We have compared the Cyper queries of AlpinoGraph with the XPath queries that can be used in PaQu, a popular existing treebank search tool for Dutch treebanks. This comparison is based on a large set of relevant syntactic queries, taken from SPOD. Both in XPath and Cypher, it is possible to recover fairly subtle and complicated syntactic patterns. And typically, the Cypher queries are evaluated much faster.

We also described a simple search optimization technique by adding special attributes to nodes which represent properties which are often referred to in queries, but slow to be evaluated on-line. This preprocessing technique is applicable to other treebank search engines too.

AlpinoGraph is open-source and can be used on-line, free of charge. The system is available via `https://urd2.let.rug.nl/kleiweg/alpinograph/`, and the sources are available via `https://github.com/rug-compling/alpinograph`.

Appendix: Query for vorfeld

In order to identify the vorfeld, the following query first identifies the head of main sentences (the finite verb) and then selects embedded dependents for which it is the case that their head precede this finite verb. These potential vorfeld constituents include the actual vorfeld, but also most of the dependents of the vorfeld. Therefore, the query is complicated by removing from the set of potential vorfelds all those nodes that are dominated by a potential vorfeld.

Further complications arise because of the possibility of multi-word-units, and because of the fact that not only real heads (with relation "hd") are treated as heads here, but also dependents of type "crd" and "cmp".

```
select sentid, id
from (
    match (n:node{cat:'smain'}) -[:rel{rel:'hd'}]-> (fin:word)
    match (n) -[:rel*{primary:true}]-> (topic:nw) -[rel:rel*0..1]-> (htopic:nw)
    where (  ( not htopic.lemma is null )
             and htopic.begin < fin.begin
             and ( length(rel) = 0 or rel[0].rel in ['hd','cmp','crd'] )
          ) or
          ( topic.begin < fin.begin and topic.end <= fin.begin )
    return topic.sentid as sentid, topic.id as id, n.id as nid
    except
    match (n:node{cat:'smain'}) -[:rel{rel:'hd'}]-> (fin:word)
    match (n) -[:rel*{primary:true}]-> (topic:nw) -[rel:rel*0..1]-> (htopic:nw)
    where (  ( not htopic.lemma is null )
             and htopic.begin < fin.begin
             and ( length(rel) = 0 or rel[0].rel in ['hd','cmp','crd'] )
          ) or
          ( topic.begin < fin.begin and topic.end <= fin.begin )
    match (topic) <-[:rel*1..]- (nt:node)  <-[:rel*]- (n)
    match (nt) -[relt:rel*0..1]-> (hnt:nw)
    where (  ( not hnt.lemma is null )
             and hnt.begin < fin.begin
             and ( length(relt) = 0 or relt[0].rel in ['hd','cmp','crd'] )
         ) or
         ( nt.begin < fin.begin and nt.end <= fin.begin )
    return topic.sentid as sentid, topic.id as id, n.id as nid
) as foo
```

References

Liesbeth Augustinus, Vincent Vandeghinste, and Frank Van Eynde. 2012. Example-based treebank querying. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC-2012)*, pages 3161–3167, Istanbul, Turkey.

Liesbeth Augustinus, Vincent Vandeghinste, Ineke Schuurman, and Frank Van Eynde. 2017. GrETEL. a tool for example-based treebank mining. In Jan Odijk and Arjan van Hessen, editors, *Clarin in the low countries*. Ubiquity Press, London.

Steven Bird and Catherine Lai. 2010. Querying linguistic trees. *Journal of Logic, Language and Information*, 19:53–73, 06.

Guillaume Bonfante, Bruno Guillaume, and Guy Perrier. 2018. *Application of Graph Rewriting to Natural Language Processing*. Wiley.

Gosse Bouma and Gertjan van Noord. 2017. Increasing return on annotation investment: The automatic construction of a universal dependency treebank for Dutch. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 19–26, Gothenburg, Sweden, May. Association for Computational Linguistics.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. Tiger: Linguistic interpretation of a german corpus. *Journal of Language and Computation*, 2:597–620, 12.

Jiří Mírovský. 2008. Netgraph - making searching in treebanks easy. In *IJCNLP 2008 Proceedings of the Third International Joint Conference on Natural Language Processing*, pages 945–950, Hyderabad, India. International Institute of Information Technology.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adedayo Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi

Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Tak-sum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Jan Odijk, Gertjan van Noord, Peter Kleiweg, and Erik Tjong Kim Sang. 2017. The parse and query (PaQu) application. In Jan Odijk and Arjan van Hessen, editors, *Clarin in the low countries*. Ubiquity Press, London.

Thomas Proisl and Peter Uhrig. 2012. Efficient Dependency Graph Matching with the IMS Open Corpus Workbench. In *Proceedings of LREC*, page 2750–2756, Istanbul. ELRA.

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

I. Schuurman, M. Schouppe, T. Van der Wouden, and H. Hoekstra. 2003. Cgn, an annotated corpus of Spoken Dutch. In A. Abbeilé, S. Hansen-Schirra, and H. Uszkoreit, editors, *Proceedings of 4th International Workshop on Language Resources and Evaluation*, pages 340–347, Budapest.

P. C. uit den Boogaart. 1975. *Woordfrequenties in geschreven en gesproken Nederlands*. Oosthoek, Scheltema & Holkema, Utrecht. Werkgroep Frequentie-onderzoek van het Nederlands.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino dependency treebank. In Hendri Hondorp Mariët Theune, Anton Nijholt, editor, *Computational Linguistics in the Netherlands 2001*. Rodopi.

Frank van Eynde. 2005. Part of speech tagging en lemmatizering van het D-COI corpus.

Gertjan van Noord, Gosse Bouma, Frank Van Eynde, Daniël de Kok, Jelmer van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste. 2013. Large scale syntactic annotation of written Dutch: Lassy. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch: Results by the STEVIN programme*, pages 147–164. Springer Berlin Heidelberg, Berlin, Heidelberg.

Gertjan van Noord, Ineke Schuurman, and Gosse Bouma. 2019. Lassy syntactische annotatie.

Gertjan van Noord, Jack Hoeksema, Peter Kleiweg, and Gosse Bouma. 2020. SPOD: Syntactic profiler of Dutch. *Computational Linguistics in the Netherlands Journal*, 10. Accepted.

Gertjan van Noord. 2006. **A**t **L**ast **P**arsing **I**s **N**ow **O**perational. In *TALN 2006 Verbum Ex Machina, Actes De La 13e Conference sur Le Traitement Automatique des Langues naturelles*, pages 20–42, Leuven.

Gertjan van Noord. 2009. Huge parsed corpora in Lassy. In Frank van Eynde, Anette Frank, Koenraad De Smedt, and Gertjan van Noord, editors, *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories (TLT 7)*, number 12 in LOT Occasional Series, pages 115–126, Utrecht, The Netherlands. Netherlands Graduate School of Linguistics.