Netgraph: a Tool for Searching in Prague Dependency Treebank 2.0

Jiří Mírovský

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
Malostranské nám. 25, 118 00 Prague, Czech Republic
e-mail: `mirovsky@ufal.mff.cuni.cz`

Abstract

We present Netgraph — a tool for searching in linguistically annotated treebanks. It works in the Internet environment, on multi–user and client–server basis. Netgraph has a graphically oriented hardware independent user–friendly interface. The query system is very intuitive, easy to learn, understand and use, nevertheless it can perform advanced searches. We show how Netgraph can be used for such complex annotation schemes as the Prague Dependency Treebank 2.0 with its non–1:1 relation among nodes on different layers. We also mention main properties of two other searching tools available for this treebank.

1 Introduction

Searching in a linguistically annotated treebank is a principal task that requires a sophisticated tool. Netgraph has been designed to perform the searching with maximum comfort and minimum requirements on its users. It is well adapted to features of Prague Dependency Treebank 2.0 and it solves the non–trivial relation among nodes on the analytical and tectogrammatical layers.

In section 2 we briefly describe the Prague Dependency Treebank 2.0 (PDT 2.0). We mention those properties of the treebank that must be taken into account while designing a searching tool and those that are required to understand the examples of queries that follow. In section 3 we briefly introduce the architecture of Netgraph. In section 4 we describe the query language of Netgraph along with the idea of meta–attributes and present several examples of queries. We also present hidden nodes in this section — the way of handling non–1:1 relation among annotation layers. In section 5 we write shortly about the requirements on treebanks set by Netgraph and about the reasonable size of treebanks

Netgraph is able to search quickly enough. In section 6 we briefly describe two other tools available for searching in PDT 2.0 - Tred and Oraculum. Finaly, in section 7 we present some concluding remarks.

2 The Prague Dependency Treebank 2.0

The Prague Dependency Treebank 2.0 ([7], [1]) is a manually annotated corpus of Czech. It is a sequel to the Prague Dependency Treebank 1.0 (PDT 1.0, [6], [4], [5]).

The texts in PDT 2.0 are annotated on three layers — the morphological layer, the analytical layer and the tectogrammatical layer.

The corpus size is almost 2 million tokens (115 thousand sentences) on the morhological layer, 1.5 million tokens (88 thousand sentences) on the analytical layer, and 0.8 million tokens (49 thousand sentences) on the tectogrammatical layer. By ’tokens’ we mean word forms, including numbers and punctuation marks. All the texts that are annotated on the tectogrammatical layer are also annotated on the two lower layers. In the following, we take into account only this part of PDT 2.0 that is annotated on all three layers.

On the morphological layer ([2]), each token of every sentence is annotated with one lemma (attribute *m/lemma*) and one tag (attribute *m/tag*). Sentence boundaries are annotated here, too.

The analytical layer roughly corresponds to the surface syntax of the sentence; the annotation is a single–rooted dependency tree with labeled nodes ([3], [4]). The nodes on the analytical layer (except for technical roots of the trees) also correspond 1:1 to the tokens of the sentences. The order of the nodes from left to right corresponds exactly to the surface order of tokens in the sentence. Non–projective constructions (that are quite frequent in Czech – see [9]) are allowed. Analytical functions are kept at nodes (attribute *a/afun*), but in fact they are names of the dependency relations between a dependant (child) node and its governor (parent) node.

The tectogrammatical layer captures the linguistic meaning of the sentence in its context. Again, the annotation is a dependency tree with labeled nodes. The correspondance of the nodes to the lower layers is more complex here. It is often not 1:1. In fact, it can be both 1:N and N:1. Every tool designed for searching in PDT 2.0 has to address this issue. Let us describe the tectogrammatical layer in a little more detail now. It will help us later understand some examples of queries in Netgraph.

2.1 The Tectogrammatical Layer

The tectogrammatical layer is the most elaborated and complex of the three layers. It is also the most theoretically based layer ([12], recently [13]). Many nodes found on the analytical layer disappear here (such as functional words, prepositions, subordinating conjunctions, etc.). The information carried by these nodes is stored in attributes of the remaining nodes and can be reconstructed. On the other hand, some nodes representing for example obligatory positions of verb frames, deleted on the surface, are regenerated on this layer.

The tectogrammatical layer annotation scheme is divided into several sublayers:

Dependencies, Functors, and Grammatemes:

The tectogrammatical layer goes beyond the surface structure of the sentence, replacing notions such as “Subject" and “Object" by notions like “Actor", “Patient", “Addressee" etc ([8]). Every sentence is represented as a dependency tree, the nodes of which correspond to the autosemantic words of the sentence, and the labeled edges name the dependencies between a dependent and its governor. Attribute *functor* that describes the dependency is again stored at the child–nodes. A tectogrammatical lemma (attribute *t_lemma*) is assigned to every node. Grammatemes are rendered as a set of 16 attributes grouped by the “prefix" *gram* (e.g. *gram/verbmod* for verbal modality). The total of 39 attributes are assigned to every non–root node of the tectogrammatical tree, although (based on the node type) only a certain subset of the attributes is necessarily filled in.

Topic, Focus and Deep Word Order:

Topic and focus ([10]) are marked (attribute *tfa*), together with so–called deep word order reflected by the order of nodes in the annotation (attribute *deepord*). It is in general different from the surface word order, and all the resulting trees are projective by the definition of deep word order.[1]

Coreference:

Coreference relations between nodes of certain category types are captured ([14]), distinguishing also the type of the relation (textual or grammatical). Each node has an identifier (attribute *id*) that is unique throughout the whole corpus. Attributes *coref_text.rf* and *coref_gram.rf* contain ids of coreferential nodes of the respective types.

3 Introduction to Netgraph

The development of Netgraph started in 1998 ([15]), and has been proceeding along with the ongoing annotations of the Prague Dependency Treebank 1.0 and later the Prague Dependency Treebank 2.0. Now it is a fully functional tool for complex searching in PDT 2.0.

Netgraph is a client–server application that allows multiple users to search a treebank online and simultaneously through the Internet. The server searches the treebank (the treebank and the server are located at the same computer or local network). The client serves as the front–end for users and may be located at any node in the Internet. It sends user queries to the server and receives results from it. Both the server and the client also can, of course, reside at the same computer. Authentication by the means of login names and passwords is provided. Users can have various access permissions.

[1]By deep word order we mean such ordering of nodes at the tectogrammatical layer that puts the “newest" information to the right, and the “oldest" information to the left, and all the rest inbetween, in the order corresponding to the notion of “communicative dynamism" ([11].

Netgraph server is written in C and C++ and works smoothly on Linux, other Unix–like systems, and Apple Mac OS. (An experimental version exists for MS Windows, too.) Netgraph client is written in Java and as such is platform–independent. A detailed desctiption of the inner architecture of Netgraph and of the communication between the server and the client has been given in [17].

4 Searching in Netgraph

After the user has connected to the server, there are three steps to follow: defining a subcorpus to be searched, setting a query, and seeing the result of the query.

The user can browse the directory structure of the corpus and define a set of individual files and/or directories for searching. The final selection is sent to the server. The server goes through all the selected files and retrieves information about available attributes from the file heads. Thus, the corpus does not have to be homogenous. Different files in the corpus can define different sets of attributes. One server can also handle several different corpora. A union of all available attributes from the file selection is created and sent back to the client.

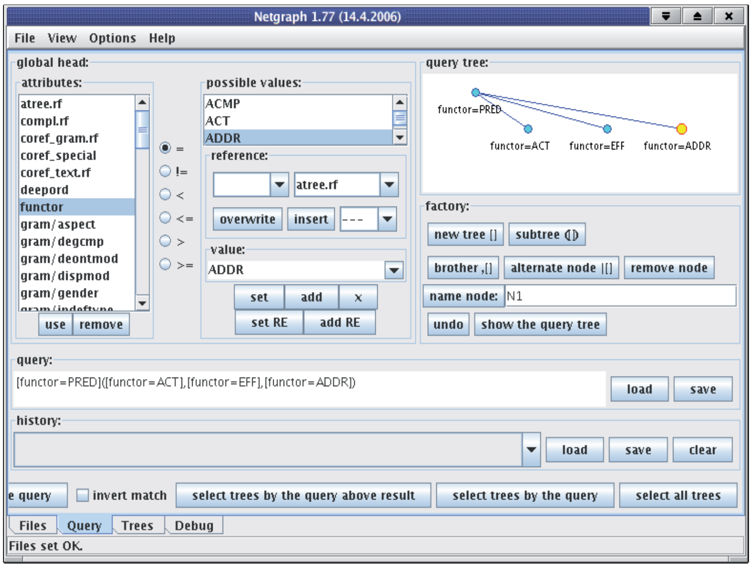
A query in Netgraph is a single node or a subtree with user–specified properties which he wishes to find in the corpus. Searching the corpus thus means searching for sentences (in the form of annotated trees) that contain the query as a subtree. The properties of the subtree which the user can specify range from the most simple ones (such as searching for all trees in the corpus that contain a given word) to very elaborate ones (such as searching for all sentences with any verb that is modified by an Addressee which is not in the dative case and by at least one directional adverbial, etc.). There may be alternative values of one attribute defined in the query or alternative evaluations of the whole node even with different sets of attributes. Wild cards are supported in the values of attributes, too.[2] A detailed desctiption of the basics of the Netgraph query language has been given in [16].

Since version 1.77 of Netgraph, Perl–like regular expressions can be used as values of attributes in queries. For example, *m/tag="[AN]...[ˆ4].*"* means a tag with *A* or *N* as its first character and anything but *4* as its fifth character, which means that the node (on the analytical layer) is a noun or adjective that is not in accusative.[3]

This simple query language is extended with so called meta–attributes in order to allow setting even more complex queries. The meta–attributes allow setting conditions on transitive edges, optional nodes, position of the query nodes in the trees, size of trees, order of nodes, relations between attributes at different nodes in the trees, negation, and many other such things. A detailed description of meta–attributes is given in section 4.1.

[2]'?' stands for any single character and '*' stands for any (possibly empty) sequence of characters.

[3]Regular expressions are enclosed in *'"'*. Just like in the normal form of a value (meaning not being a regular expression), it is necessary to use the escape character *'\'* before some special characters (those which constitute the *fs* structure of the query), namely *'['*, *']'*, *'('*, *')'*, *'='*, *','* and *'|'*. The above example in the text form of the query would be: *m/tag="\[AN\]...\[ˆ4\].*"*.



value: ADDR

name node: N1

query: [functor=PRED]([functor=ACT],[functor=EFF],[functor=ADDR])

Files set OK.

Figure 1: Creating a query in Netgraph. There are two tables listing available attributes and possible values of a selected attribute. The graphic version of the query also has its text form (in text field *query*). Both versions of the query can be edited simultaneously.

It is possible to chain the queries so that a query can search in the result of a previous query. Thus the user can refine the previous selection.

Queries in Netgraph are created in a user–friendly graphic environment. An example of a simple query is in Figure 1. In this query, we are interested in all trees containing a node labeled as Predicate and governing at least three nodes labeled as Actor, Effect, and Addressee. There is no condition on the order of the nodes.

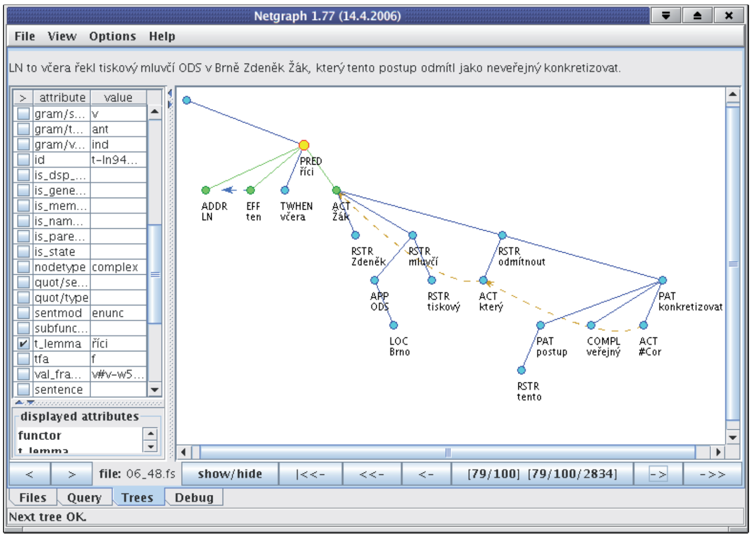
An example of a result (sent back by the server) is given in Figure 2.

The nodes matching the query are highlighted by yellow and green color. In this particular result, the predicate has got more children than we have specified in the query. This is in accordance with the definition of searching in Netgraph — it is sufficient that the query tree is only included in the result tree as a subtree. Also note that the order of the nodes in the result is different from their order in the query. Meta–attributes allow controlling both the real number of children and the order of nodes, if required.

4.1 Meta attributes

Meta attributes add a real power to the query language of Netgraph. They are called *meta* because they do not appear in the corpus but the user can use them the same way like the normal attributes. So far, nine meta attributes are available:

_transitive — by defining this as true, nodes between this node and its parent (in the query)



LN to včera řekl tiskový mluvčí ODS v Brně Zdeněk Žák, který tento postup odmítl jako neveřejný konkretizovat.

>	attribute	value
☐	gram/s...	v
☐	gram/t...	ant
☐	gram/v...	ind
☐	id	t-ln94...
☐	is_dsp_...	
☐	is_gene...	
☐	is_mem...	
☐	is_nam...	
☐	is_pare...	
☐	is_state	
☐	nodetype	complex
☐	quot/se...	
☐	quot/type	
☐	sentmod	enunc
☐	subfunc...	
✓	t_lemma	říci
☐	tfa	f
☐	val_fra...	v#v-w5...
☐	sentence	

file: 06_48.fs [79/100] [79/100/2834]

Next tree OK.

Figure 2: A result tree in Netgraph. The node matching the root of the query is selected and as such highlighted by yellow color. The other nodes matching the query are green. Attributes *functor* and *t_lemma* are displayed. Values of all attributes of the selected node are listed in the left table.

are allowed (in the result tree); in other words, the child node (in the query) is only required to be anywhere in the whole subtree (in the result tree) of its parent (in the query).

_optional — if set true, the node (lets call it B) may but need not be present in the result. Its parent (node A) and child (node C) in the query must be in the result tree, but node B itself is optional. It may appear between A and C so that A is the grandparent of C in the result, or it may be ommited in the result and C is a direct child of A. An example of searching with an optional node is given in Figure 3. In this query, we are interested in an Actor governed by a Predicate with an optional coordination inbetween.

_depth — this meta attribute defines the distance between a node and the root (in the result tree); it restricts the position of the query tree in the result tree.

#sons — this meta attribute defines the exact number of immediate children of a node (in the result tree); for example, *#sons=0* defines the node as a leaf.

_#descendants — this meta attribute defines the exact number of all descendants of a node (in the result tree) — all nodes in its subtree (excluding the node itself); this meta attribute restricts the size of the result tree.

_#lbrothers — this meta attribute defines the exact number of left brothers of the node in the result tree.

_#rbrothers — this meta attribute defines the exact number of right brothers of the node in the result tree.

_#occurrences — this meta attribute specifies the exact number of occurrences of a par-

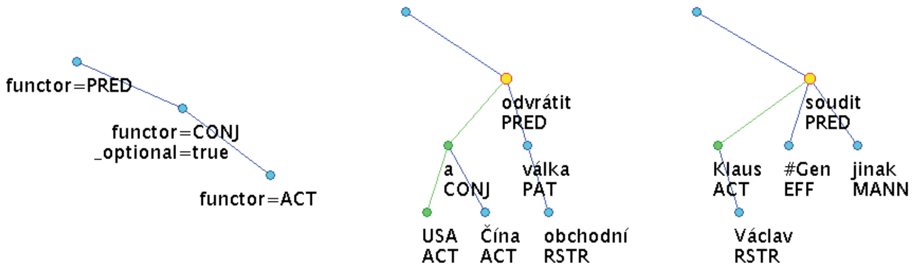


Figure 3: A query with an optional node (on the left) and two possible results. In the middle, there is a result with the optional node present. The sentence is *USA a Čína odvrátily obchodní válku.* (Literally: *The USA and China warded off a trade war.*) The tree on the right represents a result without the optional node. The sentence is *Václav Klaus soudí jinak.* (Literally: *Václav Klaus thinks otherwise.*) Attributes *t_lemma* and *functor* are displayed in the result trees.

ticular node at a particular place in the result tree. By defining it at a node in the query one can specify how many nodes of the kind may occur in the result tree as direct children of the node matching the parent of the node (including the node itself). If used together with meta attribute *_transitive*, it specifies how many nodes of the kind can occur in the whole subtree of the parent. Only non–negative values of this meta attribute are allowed; if set to zero, it specifies that such a node does not appear there at all. If we were looking, for example, for a Predicate governing an Actor and an Addressee, but not a Patient, we would create a query with a Predicate as a root, two children representing the Actor and Addresse, and a third child representing our request "not Patient". It would not be correct to define the third child as *functor!=Pat* because this node can match any node that is not Patient and does not prevent a Patient to occur in the result as its brother. It is necessary to use meta–attribute *_#occurrences* here and make sure that there are no Patients among the children of the Predicate at all. The third child in our query should be defined as *functor=Pat, _#occurrences=0*. In section 4.2 there is an example of such a usage of this meta–attribute.

_name — this meta attribute is used for naming a node in the query. Then, it is possible to make a reference to values of attributes of this node in the result tree when specifying a value of an attribute of another node. There are two kinds of references. The first one is a reference to the whole value of an attribute (the format is *{node_name.attr_name}*). The second one is a reference to one character of the value of an attribute (the format is *{node_name.attr_name.character_order}*). For example, if a node has been named *n1*, one can define attribute *m/tag* at a different node using a reference to the fourth character of attribute *m/tag* of node *n1* — *m/tag=???{n1.m/tag.4}**. References can be parts of value masks and can be used repeatedly when specifying one value of an attribute.

A simple use of references is setting the order of the query nodes in the result tree. It requires that there is a numeric attribute in the corpus that describes the order of the nodes. In PDT 2.0, there are two such attributes. Attribute *a/ord* controls the order of the nodes on the analytical layer, attribute *deepord* does the same thing on the tectogrammatical layer.

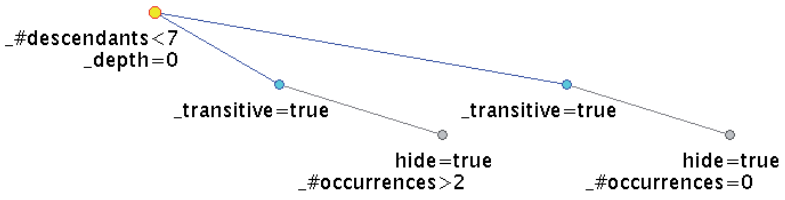


Figure 4: Searching for “a nice tree" with hidden nodes.

The example in section 4.3 sets the order of nodes using attribute *deepord*.

4.2 Hidden nodes

Already in PDT 1.0, hidden nodes served to connect the tectogrammatical layer with the analytical layer. The hidden nodes were a part of the annotation. They had slightly different meaning, though, from the one we present in this paper. Only non–autosemantic nodes (words) from the analytical layer were marked as hidden on the tectogrammatical layer, while the autosemantic nodes carried both the analytical and tectogrammatical information.

In PDT 2.0, there are no hidden nodes any more. Each layer is annotated in its own file, the files are interlinked in order to preserve the relation between the contents ([18]). Netgraph, on the other hand, presents all the available information in one tree.

We present hidden nodes with a new meaning as a solution to the non–1:1 relation among nodes on the analytical and tectogrammatical layers in PDT 2.0. On the tectogrammatical layer (as presented in Netgraph), each node can have additional hidden children that represent *all* nodes from the analytical layer that “belong" to it. There may be zero, one or several such nodes belonging to one tectogrammatical node. The hidden nodes are usually not displayed – they are “hidden".[4] They are not a part of the tectogrammatical layer, they only provide connection to the lower layers. All the nodes from the analytical layer (except for the technical root) become hidden nodes on the tectogrammatical layer in Netgraph, both autosemantic and non–autosemantic. The non–hidden nodes on the tectogrammatical layer do not carry any information from the lower layers. This information is accessible through the hidden nodes.

Also unlike in PDT 1.0, most meta–attributes do not take hidden nodes into account at all. For example, meta–attribute *_#descendants* only counts non–hidden nodes in a subtree. Meta–attribute *_#occurrences*, on the other hand, if used at a hidden node, treats hidden nodes as normal nodes (see the following example of a query). The searching algorithm itself ignores hidden nodes entirely, unless a node in the query is marked as hidden (*hide=true*).

Lets make searching for a nice tree with hidden nodes another example of making queries

[4]The hidden nodes are displayed in black and gray colors in order to be easily recognized from normal nodes. The user can choose whether the hidden nodes are displayed or not.

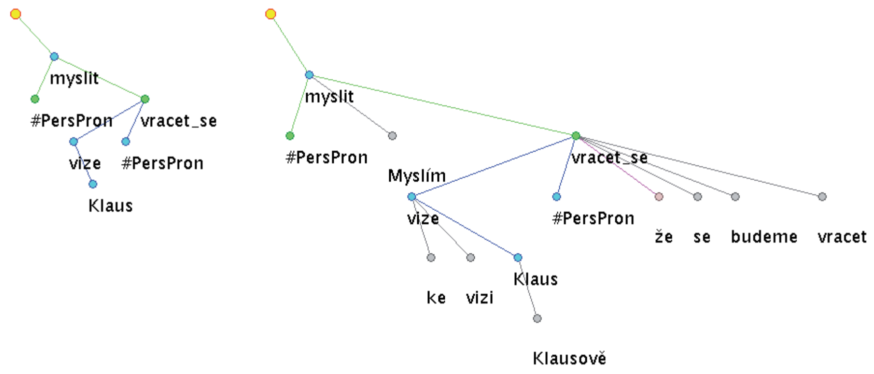


Figure 5: An example of a tree on the tectogrammatical layer with hidden nodes hidden (on the left) and displayed (on the right, hidden nodes are gray). The sentence is *Myslím, že ke Klausově vizi se budeme vracet.* (Literally: *I think that to Klaus's vision we will get back.*) Attributes *t_lemma* (tectogrammatical lemma, empty at hidden nodes) and *m/token* (the actual token from the sentence, empty at non–hidden nodes on the tectogrammatical layer) are displayed.

in Netgraph. We want the tree to be small and show both nodes with zero hidden children and nodes with more than two hidden children. Figure 4 shows such a query. First, we set the root of the query to be the root of the result tree, too, by defining *_depth=0* and restrict the number of non–hidden nodes in the rusult by *_#descendants<7*. Then we add two transitive children (*_transitive=true*) to the root (it means "there are two non–hidden nodes anywhere in the tree") and define that one of the children has at least three hidden child–nodes (*hide=true, _#occurrences>2*), while the other has non (*hide=true, _#occurrences=0*). Figure 5 shows one of the trees found by this query, both with hidden nodes hidden and displayed.

4.3 Coreferences

As described in section 2.1, PDT 2.0 captures two types of coreferences between nodes on the tectogrammatical layer. Netgraph provides a general system for displaying coreferences. It only requires an attribute serving as an identifier of the end–node of the coreference and another attribute (or attributes, in case of several types of coreference) serving as a pointer to the end–node from the start–node. The actual names of the attributes, as well as shapes and colors of arrows of the coreferences, possibly of several different types at the same time, are defined in a text file at the Netgraph server.

Figure 6 shows a query searching for a small tree (*_depth=0, _#descendants<10*) with a grammatical coreference from a node evaluated as Actor (*functor=ACT, coref_gram.rf={N1.ic*
to a node evaluated as Patient (*functor=PAT, _name=N1*). We also require that the Patient is on the right side in the tree from the Actor (*deepord<{N1.deepord}*). One of the trees

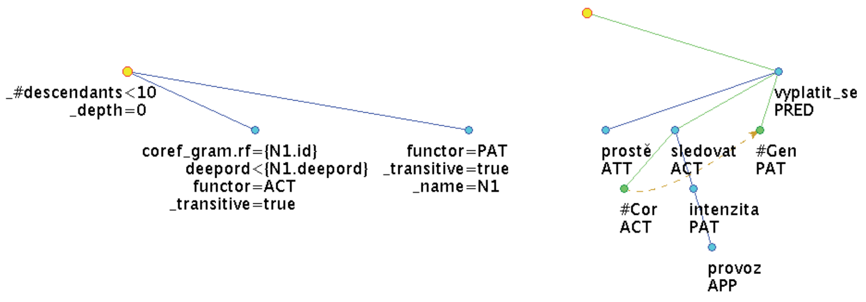


Figure 6: Searching for a coreference. The query is on the left, a result on the right. The sentence is *Sledovat intenzitu provozu se prostě vyplatí.* (Literally: *To watch the intensity of traffic is simply worth while.*) Attributes *t_lemma* and *functor* are displayed in the result.

matching this query[5] is in the picture, too.

5 Data Format

Netgraph server requires the treebank in FS format ([6]), encoded in UTF-8. FS is a simple text format designed during the early phase of annotation of PDT 1.0 ([19]). Later, it was replaced by CSTS format ([6]) and finally in PDT 2.0 by PML format([18]). For its simplicity and fast readibility, FS has remained the format of treebanks used by Netgraph. Data in any other format has to be transformed to FS before it can be searched by Netgraph.

Although there is no special support for it, Netgraph can handle both the dependency trees and the constituent–structure trees. FS does not recoginze two sorts of nodes (terminals and non–terminals) but they can be distinguished by values of attributes.

The speed of the search depends both on the size of the corpus and on the complexity of the query. Netgraph achieves a reasonably fast response to queries, as long as the corpus is not larger than several hundred thousand sentences (which is true for manually annotated treebanks). Such a corpus is usually searched in a few minutes (on a fast PC).[6]

6 Other Tools

Apart from Netgraph, there are two other tools available for searching in PDT 2.0 — Tred and Oraculum.

[5]Actualy, there are only three trees matching this query in PDT 2.0.

[6]The searching time through all tectogrammatical trees of PDT 2.0 for every example query from this paper was about 1 minute on a PC with Intel Xeon 2.80 GHz.

Tred ([7]) is primarily a tool for editing trees but has been widely used for searching, especially during post–annotation corrections. Its query power is unlimited in the sense of computers, users can write complex queries in Perl programming language and access tree structures in object–oriented way. The search can by parallelized. Other advantages of Tred are the possibility to process the data non–interactively using scripts and change the content of the data. A disadvantage is that Tred is a tool for programmers. The creation of a query requires at least a limited knowledge of Perl programming language. Another disadvantage is that Tred is not client–server based. All components including the treebank must reside at the same computer.

Oraculum ([21]) is a tool developed for searching in PDT, although it can be used for other treebanks, too. It is client–server based. There is no authentication implemented, though. Oraculum is able to combine several data sources in one query and to use the full power of logical programming in the queries. Its client part is web–browser based and as such it does not need any installation. Making queries is a combination of clicking on buttons and writing logical formulas. The interface suffers a bit from being purely web–browser based, though, and the work with it is not comfortable. Writing more complex queries requires a knowledge of logical programming. Oraculum is a product of a students project and it seems that its development has been discontinued.

7 Conclusion

Netgraph is a sofisticated user–friendly tool for searching in Prague Dependency Treebank 2.0. It is client–server based and provides access to PDT 2.0 to anyone interested simply and effectively through the Internet. Netgraph is well adapted to the features of PDT 2.0 but it can handle any other treebank as well, provided it exists in *FS* format. The query language is very easy to learn and use, nevertheless it is strong enough for creating complex queries, using the mechanism of meta–attributes. The system of hidden nodes is used for non–trivial connection between the tectogrammatical and analytical layers. An extensive manual covering all features is available.

Netgraph is stable and the development continues. Netgraph is free for academic purposes and its newest version is always available to download from the Netgraph home page ([20]).

Acknowledgments

This work has been supported by the Grant Agency of the Academy of Sciences of the Czech Republic, project IS-REST (No. 1ET101120413).

References

[1] Hajič J.: *Complex Corpus Annotation: The Prague Dependency Treebank*, Jazykovedný ústav Ľ. Štúra, SAV, Bratislava, Slovakia, 2004.

[2] Hana J., Zeman D., Hajič J., Hanová H., Hladká B., Jeřábek E.: *Manual for Morphological Annotation, Revision for PDT 2.0*, ÚFAL Technical Report TR-2005-27, Prague, 2005.

[3] Hajič J. et al.: *A Manual for Analytic Layer Tagging of the Prague Dependency Treebank*, ÚFAL Technical Report TR-1997-03, Charles University, Czech Republic, 1997.

[4] Hajič J.: *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*, In: Issues of Valency and Meaning, Karolinum, Praha 1998, pp. 106–132.

[5] Hajič J., Pajas P. and Vidová Hladká B.: *The Prague Dependency Treebank: Annotation Structure and Support*, In: IRCS Workshop on Linguistic databases, 2001, pp. 105–114.

[6] Hajič J., Vidová-Hladká B., Panevová J., Hajičová E., Sgall P., Pajas P.: *Prague Dependency Treebank 1.0 (Final Production Label)*, CDROM, LDC2001T10, LDC, Philadelphia, 2001.

[7] Hajič J. et al.: *Prague Dependency Treebank 2.0*, CDROM, LDC2006T01, LDC, Philadelphia, 2006.

[8] Hajičová E.: *Prague Dependency Treebank: From analytic to tectogrammatical annotations*, In: Proceedings of 2nd TST, Brno, Springer-Verlag Berlin Heidelberg New York, 1998, pp. 45–50.

[9] Hajičová E., Havelka J., Sgall P., Veselá K., Zeman D.: *Issues of Projectivity in the Prague Dependency Treebank*, MFF UK, Prague, 81, 2004.

[10] Hajičová E., Partee B. and Sgall P.: *Topic-Focus Articulation, Tripartite Structures and Semantic Content*, Dordrecht, Amsterdam, Netherlands: Kluwer Academic Publishers, 1998.

[11] Sgall P., Hajičová E., Buráňová E.: *Aktuální členění věty v češtině*, Academia, Praha, 1980.

[12] Sgall P., Hajičová E., Panevová J.: *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*, Academia, Prague, 1986.

[13] Hajičová E.: *Theoretical description of language as a basis of corpus annotation: The case of Prague Dependency Treebank*, In: Prague Linguistic Circle Papers, (4), Amsterdam/Philadelphia: John Benjamins, 2002, pp. 111–127.

[14] Kučová L., Kolářová-Řezníčková V., Žabokrtský Z., Pajas P., Čulo O.: *Anotování koreference v Pražském závislostním korpusu*, ÚFAL Technical Report, 19, MFF UK, Prague, 2003.

[15] Ondruška R.: *Tools for Searching in Syntactically Annotated Corpora*, Diploma Thesis, Charles University, Praha, 1998.

[16] Mírovský J., Ondruška R.: *NetGraph System: Searching through the Prague Dependency Treebank*, In Prague Bulletin of Mathematical Linguistics, Prague, 2002, pp. 101–104.

[17] Mírovský J., Ondruška R., Průša D.: *Searching through Prague Dependency Treebank-Conception and Architecture*, In: Proceedings of The First Workshop on Treebanks and Linguistic Theories, Sozopol, 2002, pp. 114–122.

[18] Pajas P., Štěpánek J.: *A Generic XML-Based Format for Structured Linguistic Annotation and Its Application to Prague Dependency Treebank 2.0*, In: ÚFAL Technical Report, 29, MFF UK, Prague, 2005.

[19] Křen M.: *Editor grafů*, Diploma Thesis, Charles University, Praha, 1996.

[20] Mírovský J.: *Netgraph home page*: http://quest.ms.mff.cuni.cz/netgraph/index.html.

[21] Ljubopytnov V., Němec P., Pilátová M., Reschke J., Stuchl J.: *Oraculum, a System for Complex Linguistic Queries* In: Proceedings of SOFSEM 2002, Student Research Forum, Milovy, 2002.