Searching through Prague Dependency Treebank

Conception and Architecture

Jiří Mírovský[1], Roman Ondruška[1], and Daniel Průša[2]

[1] Center for Computational Linguistics
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
{mirovsky,ondruska}@ckl.mff.cuni.cz

[2] Department of Theoretical Informatics
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
prusa@barbora.mff.cuni.cz

Abstract

In this paper we present our conception of searching in syntactically annotated corpora. In the first part we briefly introduce the Prague Dependency Treebank—one of the key projects at our center. Its goal is to build a large corpus of Czech with a rich annotation scheme. After that we describe architecture and usage of the software system we have originally developed for searching through the treebank. This tool works in the Internet environment and has a graphically oriented hardware independent user interface. As a theoretical background we also present a proof that the subtree counting problem is $\#P$-complete.

1 The Prague Dependency Treebank

The Prague Dependency Treebank (PDT) ([1]), ([2]) and ([3]) is a manually annotated corpus of Czech. The corpus size is approx. 1.5 million words. The texts are annotated in three layers:

- the morphological layer, where lemmas and tags are being annotated based on their context
- the analytical layer, which roughly corresponds to the surface syntax of the sentence
- the tectogrammatical layer, or linguistic meaning of the sentence in its context

Unique annotation for every token in every sentence is used on all three layers. Most of them are annotated manually, using the necessary human judgment.

1.1 The Morphological Layer

The annotation at the morphological layer is an unstructured classification of the individual tokens (words and punctuation) of the utterance into morphological classes (morphological tags) and lemmas. The tagset size used is 4257 with about 1100 different tags actually appearing in the PDT.

There are 13 categories used for morphological annotation of Czech: Part of speech, Detailed part of speech, Gender, Number, Case, Possessor's Gender and Number, Person, Tense, Voice, Degree of Comparison, Negation and Variant. So-called positional tag system is used, where each position in the actual tag representation corresponds to one category.

1.2 The Analytical Layer

At the analytical layer ([4]), two additional attributes are being annotated:

- (surface) sentence structure
- analytical function

A single-rooted dependency tree is being built for every sentence as a result of the annotation. Every item (token) from the morphological layer becomes (exactly) one node in the tree and no nodes (except for the single technical root of the tree) are added. The order of nodes in the original sentence is being preserved in an additional attribute, but non-projective constructions are allowed. Analytical functions are kept at nodes, but in fact they are names of the dependency relations between a dependant (child) node and its governor (parent) node.

There are 24 analytical functions used, such as Sb (Subject), Obj (Object), Adv (Adverbial), Pred, Pnom (Predicate / Nominal part of a predicate for the (verbal) root of a sentence), Atr (Attribute in noun phrases), Atv, AtvV (Verbal attribute / Complement), AuxV (auxiliary verb), Coord, Apos (coordination/apposition head), etc.

1.3 The Tectogrammatical Layer

The tectogrammatical layer is the most elaborated, complicated, but also the most theoretically based layer of syntactico-semantic representation ([5]). The tectogrammatical layer annotation scheme is divided into four sublayers:

- dependencies and functional annotation
- the topic/focus annotation including reordering according to the deep word order
- coreference
- the fully specified tectogrammatical annotation (including the necessary grammatical information)

Let us shortly describe at least the first two of the sublayers.

1.3.1 Dependencies and Functors

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. The nodes in the tectogrammatical tree are autosemantic words only. Dependencies between nodes represent the relations between the words in a sentence. The dependencies are labeled by functors, which describe the dependency relations. Every sentence is thus represented as a dependency tree, the nodes of which are autosemantic words, and the labeled edges name the dependencies between a dependent and its governor.

Many nodes found at the morphological and analytical layers disappear (such as function words, prepositions, subordinating conjunctions, etc.). The information carried by the deleted nodes is not lost and can be reconstructed.

1.3.2 Topic, Focus and Deep Word Order

Topic and focus ([6]) are marked, together with so-called deep word order reflected by the order of nodes in the annotation, is in general different from the surface word order, and all the resulting trees are projective by the definition of deep word order.

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".

2 Introduction to Netgraph

The current version of the Prague Dependency Treebank is PDT 1.0 ([7]) and contains approx. 100,000 sentences on the analytical level. A non-automatic searching for concrete dependencies through so large set is impossible. For this purpose we have developed a special software called Netgraph.

Netgraph is a multiuser system with a net architecture ([8]). This means that more than one user can access it at the same time and its components may be located in different nodes of the Internet (see Fig. 1). Netgraph generally consists of a server part, which mainly realizes the corpus searching itself, and a client part, which provides a user interface to the system.

3 A view into the inner parts of the Netgraph system

The communication between a client and a server is always started and ended by the client.

The client part, to be flexible, is available in two forms – as a Java2 application and as a Java2 applet. The client connects to the server using a specified port, a login name and a password[1], obtains some initial information and then communicates with the server by sending and recieving several types of messages. The client can ask server for the directory structure of the treebank, select subcorpora, send queries to the server above the subcorpora and obtain results of the queries—trees from the treebank.

The treebank itself in its natural format is located on the server node. The server part of Netgraph consists of two main programs, written in C++ in order to be as fast as possible:

- *netser* - it waits for incoming connections from clients and provides the network communication
- *dotser* - it handles all incoming messages and is responsible for query resolving. It is the central part of the system

[1]An anonymous login is possible too, but anonymous users are restricted in some ways - they cannot save result trees to local disks and the amount of trees in the result of one query is limited. Netgraph client as an applet allows only the anonymous login.

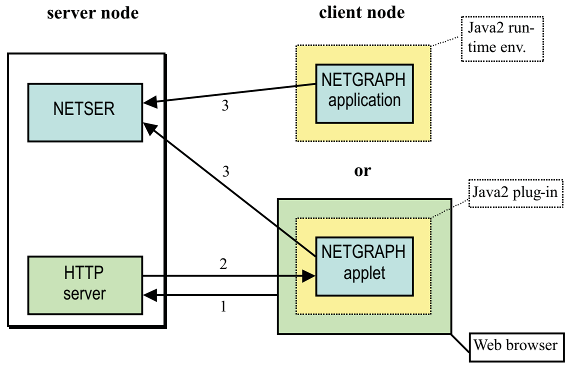

Figure 1: Start of Netgraph client. The arrows #1 and #2 represent the first two steps of the work with the Netgraph client as an applet. First, the user clicks in a web browser on a link to Netgraph client applet (the arrow #1). Then the applet is loaded into the web browser (the arrow #2) and connects to the program netser waiting for connections on the same server the applet has been loaded from (the lower arrow #3). Using the Netgraph client as an application, the user runs the client, then selects a server (the name and the port) and the client connects to the server (the upper arrow #3). In both of the cases, the Netgraph client needs Java2 as its environment - either Java2 plug-in (the Netgraph applet) or Java2 run-time environment (the Netgraph application).

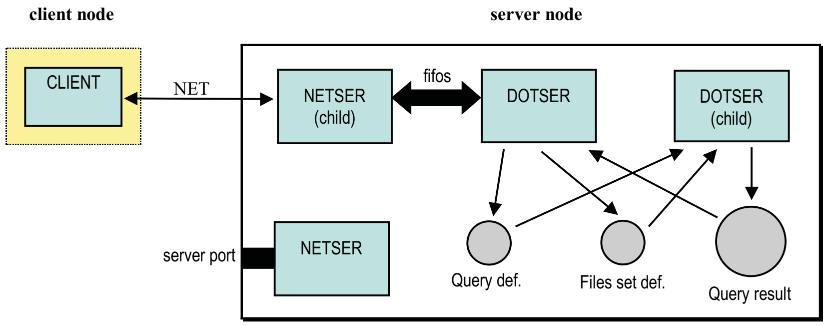

Figure 2: Typical situation during the work of a user - *dotser* communicates with the client through *netser* (child) while *netser* (parent) waits for another connection. *Dotser* (child) performs a particular query and communicates with *dotser* (parent) using several temporal files.

Netser waits on the server node for connections from clients. After a connection from a client is requested, *netser* (parent) immediately creates a new process *netser* (child), which will manage the communication with the client (see Fig. 2). The *netser* (child) creates a new process *dotser*, which will perform the responses to the client requests. The *netser* (parent) is ready to accept another connection from another client.

When an incoming message is identified as a query, *dotser* creates a new process - *dotser* (child). This child processes the query and answers immediately to the client that the query has been accepted. This mechanism ensures that the system is not blocked in the case of a query which requires larger amount of time to be processed. The user can send another message or enter a new query during the processing of a query (the processing of the previous query is then canceled).

4 Searching in Netgraph

After the user has connected to the server, s/he has to go through three steps: define a subcorpus to be queried, define an object of a query, and fetch and display the result of the query. A query has the form of a labeled tree structure, which is derived from the tree structure that represents sentences in the corpus files. The result of the query consists of trees from the subcorpus that contain the tree from the query as a subtree. The matching of the nodes evaluation is checked as well.

All the steps can be done using an easy-to-use graphical interface.

4.1 Subcorpus definition

The user can browse the directory structure of the corpus provided by the server and select files s/he wants to use for searching; this set can be saved to disk (and loaded back). The subcorpus may be also defined as the result of the previous query.

4.2 Query definition

Queries determine which trees will be included in the result of searching. The user defines a tree (see Fig. 3) s/he wants to be included as a subtree in each tree of the result. For defining such a tree, including the labels of its nodes, a graphical interface can be used. The graphically created tree is simultaneously displayed in a linear text form; the text form can also be edited directly.

If an unlabeled tree is used for a query then the searching process only considers the tree structure itself, the node matching is not checked in this case. However, the user usually demands some restrictions on some of the node attributes. In Netgraph one may enter them for every attribute by defining so called masks. The masks are expressions written into particular nodes of a query in brackets, separated by commas. Two special characters can be used in the mask definitions: the asterisk character `'*'` represents a sequence of characters and the dot `'.'` represents a single character.

Example:
```
[origf=.resident*]
```

This mask indicates that the attribute *origf* can be of the value *'President'*, *'presidents'* and so on.

Another useful operation is the alternation—the logical conjunction, represented by the separator `'|'`.

Example:
```
[lemma=president,afun=Sb]|
[lemma=president,afun=Obj,tag=N...4*|N...6*]
```

This requires the lemma *president* as the *(subject or (object in (accusative or locative)))*. The analyti-

```
[lemma=chodit]
        [origf=na]
              [tag=N...4*]
```

Figure 3: Example of a simple query with attributes: *lemma*—an identifier of the underlying lexical unit, *origf*—an original word form as found in text, *tag*—a morphological category.

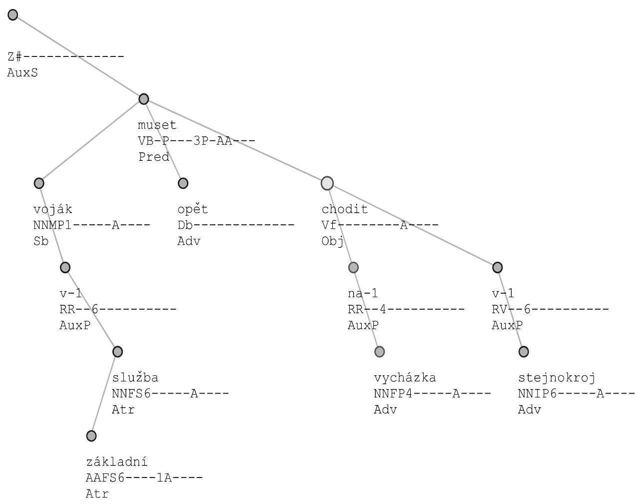

Figure 4: Example of tree depiction in Netgraph. The nodes in the tree represent words and their linguistic attributes and the edges represent analytical dependencies. Although the depicted tree is projective, this treatment also supports non-projective dependencies (present in Czech surface shapes of sentences and thus on the analytical level of PDT). The user can select attributes to be displayed at the nodes of the tree.

cal function is represented by the attribute *afun*.[2]

4.2.1 Meta attributes

To define the query in more detail, a system of meta attributes—attributes not really present in the corpus—can be used. There are five meta attributes at this moment:

- *_transitive* – by defining this as true, nodes between this node and its parent (in the query) are allowed (in a result tree)
- *_optional* – if true, then the node (lets call it A) may but need not be present in the result. In the result tree, there must be a node matching the node A or its son on the place of the node A
- *_depth* – this meta attribute defines the distance between a node and the root (in a result tree)
- *_#sons* – this meta attribute defines the exact number of immediate sons of a node (in a 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 a result tree) - all nodes in its subtree (excluding the node itself)

The last three meta attributes allow to restrict the position of a query tree in a result tree and also to restrict the size of a result tree.

4.3 Viewing the result

After the first tree matching the query is found, it is immediately displayed; the subtree matching the query is highlighted. The order of words is viewed from left to right. According to that, the tree in

[2] The exact description of the meaning of all attributes can be found in [7].

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \qquad\qquad \pi(1) = 2,\ \pi(2) = 3,\ \pi(3) = 1$$

Figure 5: Matrix A and its coverage

Fig. 4, obtained as the result of the query from Fig. 3, represents the sentence: *"Vojáci v základní službě musejí opět chodit na vycházky ve stejnokrojích." (Soldiers in compulsory service must again go on walks in uniforms.)*

The order of nodes on the analytical level may be different from the order on the tectogrammatical level. Also, some nodes from the analytical level (esp. those representing function words and punctuation marks) should be hidden on the tectogrammatical level. Netgraph offers two modes of tree displaying according to these two levels.

5 $\#P$-completeness of the Subtree Counting Problem

In this section we study complexity of the subtree counting problem. We assume the reader is familiar with the basic notions of the theory of graphs and the theory of complexity.

We consider the subtree counting problem to be defined as follows:

Definition (the subtree counting problem): *For two trees T_Q (a query) and T compute the number of all different subtrees of T that match the query.*

We do not require vertices of T_Q and T to have some labels. Our goal will be to prove that computing the number of all different subtrees of T matching T_Q is $\#P$-complete (which can be interpreted that it is highly improbable that the number of subtrees returned by a query can be computed effectively in polynomial time). This result has impact on searching all subtrees. Note that it can be decided in polynomial time if at least one searched subtree exists in T (see [9]).

We start by the description of a known $\#P$-complete problem that we want to transform to the subtree counting. Let us consider a square matrix A of the size n, where each element is 0 or 1 (so called 0-1 matrix). We denote elements of A by $A(i, j)$, for $i, j = 1, \ldots, n$. A *coverage* of A is every permutation π of the set $\{1, \ldots, n\}$ such that $A(i, \pi(i)) = 1$ for all $i = 1, \ldots, n$. A coverage can be interpreted as a placement of n rooks on the chessboard of the size n using fields corresponding to value 1 so that no two rooks threat each other. Fig. 5 shows a matrix and its coverage. To compute the number of all different coverages of A is a well known $\#P$-complete problem (the number of coverages is called *permanent* of A).

As a preparation to the proof we continue by defining some auxiliary trees. Let n, i be positive integers, $i \le n$. $S_{n,i}$ denotes the tree consisting of a chain of $n + 1$ vertices and one extra leaf appended to the i-th vertex of the chain (we consider the vertices to be numbered in the ascendant order $1, \ldots, n + 1$ starting by the root, see Fig. 6 for examples of $S_{n,i}$ trees). Note that the defined trees have one important property. If n is fixed there are no two different trees $S_{n,i}$, $S_{n,j}$ such that one of them is a

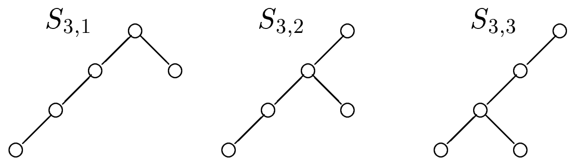

Figure 6: Trees $S_{n,i}$

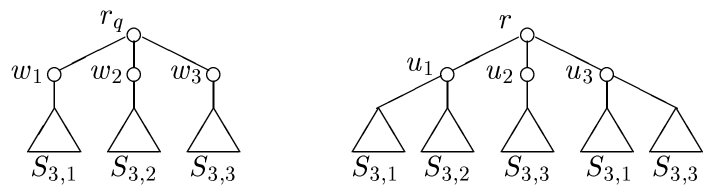

Figure 7: T_Q and T

subtree of the other one. Moreover, the depth of such trees is always $n + 1$.

We are ready to show now, how the transformation looks like. Let A be a 0-1 matrix of the size n. We construct an instance of the subtree counting problem, i.e. trees T_Q, T, such that the number of all different subtrees of T matching T_Q is the same as the number of all different coverages of A. The construction is done as follows. The root of T_Q (denoted r_Q) has n direct descendants $w_1, \dots, w_n$, for each of them, $S_{n,i}$ is appended to w_i. The root of T (denoted r) has n direct descendants again. Let they be denoted $u_1, \dots, u_n$. For each u_i, $S_{n,j}$ is appended to u_i if and only if $A(i, j) = 1$. See Fig. 7 for the example of T_Q and T that correspond to A in Fig. 5.

We have to prove there is a one to one correspondence between solutions of both problems. Let us suppose π is a coverage of A. T_Q can be mapped to a subtree of T as follows: r_Q is mapped to r, each w_i is mapped to $u_{\pi(i)}$. Since $S_{n,i}$ is appended to $u_{\pi(i)}$, the whole subtree having w_i as the root can be mapped to vertices under $u_{\pi(i)}$.

On the other hand, let us suppose T_Q is mapped to some subtree of T. Since the depths of both trees are the same, r_Q has to be mapped to r. It implies the vertices $w_1, \dots, w_n$ are mapped to the vertices $u_1, \dots, u_n$, let say each w_i to $u_{\pi'(i)}$, where π' is the corresponding permutation. $S_{n,i}$ appended to w_i has to be mapped to $S_{n,i}$ appended to $u_{\pi'(i)}$ (since $S_{n,i}$ cannot be a subtree of another $S_{n,j}$, $j \neq i$). So, it is evident that there is $S_{n,i}$ appended to $u_{\pi'(i)}$, which further implies that $A(\pi'(i), i) = 1$, hence π'^{-1} is a coverage of A.

To complete the proof it is sufficient to note that the transformation is polynomial, since a pair T_Q, T can be easily computed in time polynomial in n.

We have proved the subtree counting problem is $\#P$-complete. In theory, it indicates we can expect difficulties during evaluation of queries. However, the number of vertices of examined trees and the number of subtrees matching a query is usually relatively small, thus it is possible to achieve an acceptable response on queries that can be, in addition, improved by preprocessing information stored in trees (to be able to eliminate immediately some queries that cannot match) and by choosing suitable searching strategies (that prioritize the order of comparisons of some labels).

6 Conclusion

Netgraph provides access to PDT to anyone interested simply and effectively. PDT itself is an amorphous set of trees without any structure. With Netgraph a user can add structure into PDT and obtain any linguistic information that PDT contains. The searching problem is generally $\#P$-complete, but in our special case it is possible to achieve an acceptable response to queries. Netgraph also works corpus-language independently, so it is not restricted only to Czech corpuses. It only demands a corpus in fs ([7]) format. Netgraph is stable, but is still being developed and other features are planned to be added in the future—for example relations among numeric values of attributes cannot be defined in queries yet. Some technical enhancements like XML support are planned too. The newest version of Netgraph is available for downloading on the Netgraph home page ([10]).

Acknowledgments

This work has been supported by the Ministry of Education—project *Center for Computational Linguistics* (No. LN00A063).

The studies of complexity have been supported by Grant Agency of the Czech Republic, *Grant-No. GA CR 201/02/1456*.

References

[1] Böhmová A., Hajič J., Hajičová E., Hladká B.: The Prague Dependency Treebank: Three-Level Annotation Scenario, In: *Treebanks: Building and Using Syntactically Annotated Corpora*, ed. by Anne Abeille. Kluwer Academic Publishers, in press.[3]

[2] Hajič J.: Building a Syntactically Annotated Corpus: The Prague Dependency Treebank, In: *Issues of Valency and Meaning*, ed. by E. Hajičová, pp.106-132, Karolinum, Praha 1998.

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

[4] 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.

[5] Hajičová E., Panevová J., Sgall P.: *A Manual for Tectogrammatical Tagging of the Prague Dependency Treebank. ÚFAL/CKL Technical Report* TR-2000-09, 2000.

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

[7] CD-ROM PDT 1.0, available at http://shadow.ms.mff.cuni.cz/pdt.

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

[9] Matula D. W.: An algorithm for subtree identification. SIAM Rev., 10:273-274, 1968.

[10] Mírovský J.: *Netgraph home page* – http://shadow.ms.mff.cuni.cz/~mirovsky/netgraph/index.html.

[3] PDT documentation can be found at http://shadow.ms.mff.cuni.cz/pdt/Corpora/PDT_1.0/References/index.html