VIQTORYA – A Visual Query Tool for Syntactically Annotated Corpora

Ilona Steiner*, **Laura Kallmeyer**†

*Seminar für Sprachwissenschaft, Universität Tübingen
Wilhelmstr. 113, D–72074 Tübingen, Germany
steiner@sfs.uni-tuebingen.de

†TALaNa-Lattice, Université Paris 7
2, place Jussieu, F–75251 Paris cedex 05, France
laura.kallmeyer@linguist.jussieu.fr

Abstract

This paper presents a query tool for syntactically annotated corpora. The query tool is developed to search the Tübingen Treebanks annotated at the University of Tübingen. However, in principle it also can be adapted to other corpora. The tool uses a query language that allows to search for tokens, syntactic categories, grammatical functions and binary relations of (immediate) dominance and linear precedence between nodes. The overall idea is to extract in an initializing phase the relevant information from the corpus and store it in a compact way in a relational database. An incoming query is then translated into a corresponding SQL query that is evaluated on the database. A graphical user interface allows to specify queries in a user-friendly way.

1. Introduction

With the increasing availability of large amounts of electronic texts, linguists have access to more and more material for empirically based linguistic research. Currently many corpora are tagged with morphosyntactic categories (part-of-speech) and there are already several syntactically annotated corpora. Examples are the Penn Treebank (Marcus et al., 1994; Bies et al., 1995) annotated at the University of Pennsylvania, the Negra corpus (Brants et al., 1999) developed in Saarbrücken, the French treebank annotated in Paris (Abeillé and Clément, 1999) and the Tübingen Treebanks (Hinrichs et al., 2000) annotated in the project *Verbmobil* at the University of Tübingen. A sample entry taken from the Tübingen German Treebank is shown in Fig. 1. However, in order to have access to these rich linguistic annotations, adequate query tools are needed.

In this paper we present the prototype version of VIQTORYA (A **Vi**sual **Q**uery **T**ool f**or** S**y**ntactically **A**nnotated Corpora) with a query language that allows to search for complex syntactic structures in corpora. In contrast to an earlier version of the tool presented in Kallmeyer (2000b), the query component now can process all elements defined by the query language (disjunction in a restricted version) and the architecture is extended by a graphical user interface. The tool is developed for the Tübingen Treebanks, but it can be adapted to other corpora as well.

1.1. The Tübingen Treebanks

The Tübingen Treebanks, annotated at the University of Tübingen, comprise a German, an English and a Japanese treebank consisting of spoken texts restricted to the domain of arrangement of business appointments. In this paper we focus on the German Treebank (TüBa-D) (Stegmann et al., 2000; Hinrichs et al., 2000) that contains approx. 38.000 trees (or rather tree-like annotation structures since the structures are not always trees).

The corpus is part-of-speech tagged using the Stuttgart Tübingen tagset (STTS) described in Schiller et al. (1995). One of the design decisions was that for the purpose of reusability of the treebank, the annotation scheme should not reflect a commitment to a particular syntactic theory. Therefore a surface-oriented annotation scheme was adopted to structure German sentences that is inspired by the notion of topological fields in the sense of Höhle (1985) (see Fig. 1, 2): The verbal elements have the categories `LK` (*linke Klammer*) and `VC` (*verbal complex*), and roughly everything preceding the `LK` forms the "vorfeld" `VF`, everything between `LK` and `VC` forms the "mittelfeld" `MF` and the "nachfeld" `NF` follows the verbal complex.

The corpus is annotated with syntactic categories as node labels, with grammatical functions as edge labels and with dependency relations. The syntactic categories are based on traditional phrase structure and on the theory of topological fields.

The data structures used for the Tübingen Treebanks are not always trees in order to cope with the characteristics of spontaneous speech. So, for example, an entry of the treebank can consist of several tree structures (see Fig. 1). Furthermore, an element of the corpus might contain completely disconnected nodes (see, e.g., the repetition of the finite verb *macht* in Fig. 1 which is not bound to the rest of the sentence). In contrast to Negra or Penn Treebank, there are neither crossing branches nor empty categories.

1.2. An Example

In the following, an example of a linguistically relevant construction is considered that illustrates how useful access to structural information in a corpus might be.

(1) der Kaiser hat dem Fürst den Maler empfohlen
the emperor has to the prince the painter recommended
"the emperor recommended the painter to the prince"

Linguistic research is often concerned with word order preferences in sentences and the factors that influence word order. Studies have shown, for example, that dative noun phrases (NPs) tend to precede accusative noun phrases in

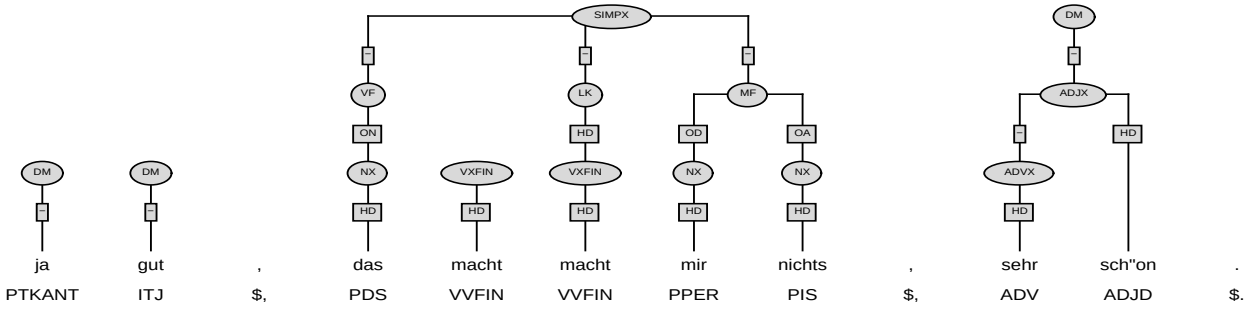

Figure 1: Annotated sentence "yes okay, it doesn't matter to me, very good." (taken from the Tübingen German Treebank)

the German 'mittelfeld'. An example is (1) where we have a dative NP (*dem Fürst*) preceding the accusative NP (*den Maler*). Both NPs are positioned between the two verbal elements (i.e. in the 'mittelfeld'). It seems, however, that the preferred word order is reversed if the NPs are pronouns (see, e.g., Featherston (2002)). The annotation of (1) according to the annotation scheme of the Tübingen German Treebank (TüBa-D) is shown in Fig. 2.

In these cases it is useful to search an adequate corpus for more natural data showing the same construction (see also Meurers (1999), for examples of the use of corpora for linguistic research). In order to find structures as in Fig. 2 in the German treebank one needs to search for a dative NP preceding an accusative NP in the 'mittelfeld', i.e., search for trees containing a node n_1 with label `NX` and grammatical function `OD` (dative object) preceding a node n_2 with label `NX` and grammatical function `OA` (accusative object) and a node n_3 with label `MF` that dominates n_1 and n_2.

Evaluating this query on TüBa-D gives results such as (2) showing a preference of dative NPs preceding accusative NPs even for two pronouns, at least if the pronoun in the accusative case is the demonstrative *das*.

(2) na, dann notiere ich mir das so .
well, then write down I myself that like this .
"well, then I'll write that down like this ."

This example illustrates the usefulness of syntactic annotations for linguistic research and the need of query languages and query tools that allow access to these annotations.

The query tool we propose in this paper is implemented at present for the Tübingen German Treebank and allows to search for underspecified tree fragments including parent,

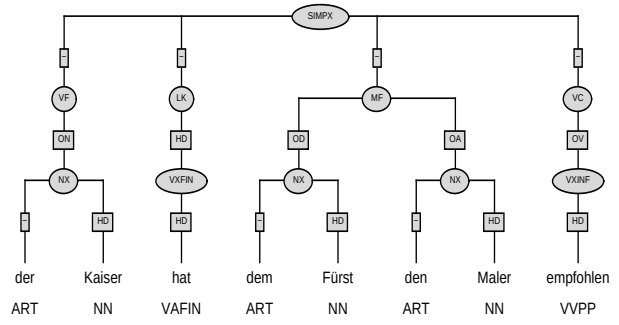

Figure 2: Annotation of (1) according to TüBa-D

dominance and linear precedence relations even in corpora annotated with structures different from trees.

2. The Query Language of VIQTORYA

2.1. Syntax

As query language, a first order logic without quantification is chosen. Variables are interpreted as existentially quantified. Negation is only allowed for atomic formula. Even this very simple logic already gives a high degree of expressive power with respect to the queries linguists are interested in (see for example Kallmeyer (2000a) for theoretical investigations of query languages).

Let C (the node labels, i.e., syntactic categories and part-of-speech categories), E (the edge labels, i.e., grammatical functions) and T (the terminals, i.e., tokens) be pairwise disjoint finite sets. `>`, `>>`, `..` are constants for the binary relations immediate dominance (parent relation), dominance (reflexive transitive closure of immediate dominance) and linear precedence. The set $\mathbb{N}$ of natural numbers is used as variables. Further, `&`, `|`, `!` are logical connectives (conjunction, disjunction, and negation).

Definition 1 ((C, E, T)-queries)
(C, E, T)-queries are inductively defined:

(a) for all `i`$\in \mathbb{N}$, `t`$\in T$*:*
`token(i)=t` *and* `token(i)!=t` *are queries,*

(b) for all `i`$\in \mathbb{N}$, `c`$\in C$*:*
`cat(i)=c` *and* `cat(i)!=c` *are queries,*

(c) for all `i`$\in \mathbb{N}$, `e`$\in E$*:*
`fct(i)=e` *and* `fct(i)!=e` *are queries,*

(d) for all `i`, `j`$\in \mathbb{N}$*:*
`i>j` *and* `i!>j` *are queries,*
`i>>j` *and* `i!>>j` *are queries,*
`i..j` *and* `i!..j` *are queries,*

(e) for all queries q_1, q_2:
q_1 `&` q_2 *and* `(`q_1 `|` q_2`)` *are queries.*

Note that different variables are considered to refer to different nodes. The intended models for this logic are defined as more general structures than finite trees, since, as already mentioned, in the case of the Tübingen German Treebank the data structures are not always trees. The structure in Fig. 1, for example, does not have a unique root, i.e., a node that dominates all other nodes, and two nodes do not necessarily have a dominance or linear precedence relation (e.g., the two nodes with labels `VXFIN` and `SIMPX` in

Fig. 1). The intended query models and the model-theoretic semantics are defined in Kallmeyer (2000b).

Of course, when adapting this language to another corpus, depending on the specific annotation scheme, other unary or binary predicates might be added to the query language. This does not change the complexity of the query language in general. However, it is also possible that at a later point negation needs to be allowed in a general way or that quantification needs to be added to the query language for linguistic reasons. Such modifications would affect the complexity of the language and the performance of the tool. Therefore the decision was taken to keep the language as simple as possible in the beginning.

2.2. Underspecification in VIQTORYA

An important aspect of the query language defined above is the possibility of underspecification. It is possible to specify abstract tree fragments and, furthermore, to leave underspecified particular aspects of these partial structures as, e.g., the relation of dominance and the linear precedence of nodes.

In the following, another example is considered that illustrates the usefulness of underspecification. (3) shows a complex prepositional phrase (PP) consisting of two parts: a first PP beginning with the preposition *von* ("from") and a second PP beginning with the preposition *bis* ("to").

(3) von der Stadt in der ich lebe bis nach Hannover
"from the town I live in to Hannover"

In order to find complex PPs as described above in the corpus, one needs to search for a preposition *von* linearly preceding a preposition *bis* and, moreover, a prepositional phrase (with syntactic category `PX`) that dominates both prepositions. Everything else remains underspecified. For this query the relations of general dominance and general linear precedence are needed since it is unknown how complex both PPs are and how much material is positioned between the two prepositions. This leads to the query in (4).

(4)
```
token(1)=von & token(2)=bis & 1..2
& cat(3)=PX & 3>>1 & 3>>2
```

3. The Architecture of VIQTORYA

The general idea of the tool is to store the information one wants to search for in a relational database and then to translate an expression in the query language presented above into an SQL expression that is evaluated on the database. The tool consists of the following components:

- an *initializing component* that extracts the relevant information (e.g., nodes/node pairs) from the trees in the corpus and stores them in the database (see Sec. 3.2.). This needs to be done only once for each corpus, usually by the corpus administrator.
- a *graphical user interface* for specifying in a user-friendly way the queries that are then passed on to the query component (see Sec. 3.4.),
- a *query component* that translates an incoming query (expressed in the query language described above) into an SQL expression and evaluates it on the database (see Sec. 3.3.),
- *query results* (including their annotation) can be viewed in the Annotate tool (Brants and Skut, 1998).

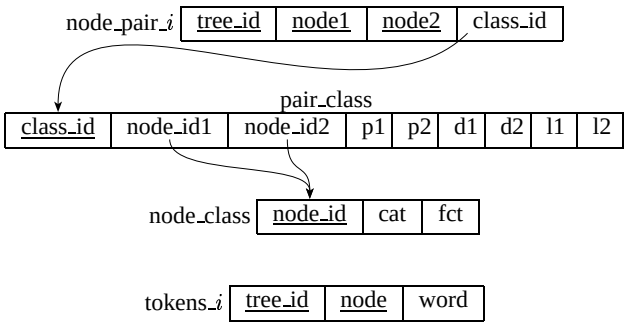

Figure 3: The relational database schema of VIQTORYA

VIQTORYA is implemented in Java and uses Java Database Connectivity (JDBC) as interface and MySQL as database management system.

3.1. The Relational Database Schema

The Tübingen German Treebank consists of several subcorpora. In the relational database there are two global tables, node_class and pair_class. Besides these, for each of the subcorpora identified by i there are tables tokens_i and node_pair_i. The database schema is shown in Fig. 3. The arrows represent foreign keys. The column class_id in the table node_pair_i, for example, is a foreign key referring to the column class_id in the table pair_class. This means that each entry for class_id in node_pair_i uniquely refers to one entry for class_id in pair_class.

The content of the tables is as follows:

- **node_class** contains node classes characterized by category (node label) and grammatical function (edge label between the node and its mother). Each node class has a unique identifier, namely the column node_id.
- **pair_class** contains classes of node pairs characterized by the two node classes and the parent, dominance and linear precedence relation between the two node classes. The columns p1, p2, d1, d2, l1 and l2 stand for binary relations and have values 1 or 0 depending on whether the relation holds or not. p1 signifies immediate dominance of the first node over the second, p2 immediate dominance of the second over the first, d1 dominance of the first over the second, etc. Each node pair class has a unique identifier, namely its class_id.
- **tokens**_i contains all leaves from subcorpus i with their tokens (words).
- **node_pair**_i contains all node pairs from subcorpus i with their pair class. Of course, only pairs of nodes belonging to one single annotation structure are stored.

The global tables node_class and pair_class represent abstract classes of nodes and of pairs of nodes. This abstraction allows a compact representation of the corpus: instead of storing concrete pairs of nodes with all their properties (binary relations, categories, grammatical functions etc.), one indicates just the class of pairs a concrete pair be-

```
#BOS 24 25 898511955 1
scheinbar   ADV      --   HD   500
nicht       PTKNEG   --   HD   501
beides      PIS      --   HD   502
zusammen    ADV      --   HD   503
.           $.       --   --   0
#500        ADVX     --   -    505
#501        ADVX     --   -    505
#502        NX       --   HD   504
#503        ADVX     --   -    504
#504        NX       --   HD   505
#505        NX       --   --   0
#EOS 24
```

Corresponding structure:

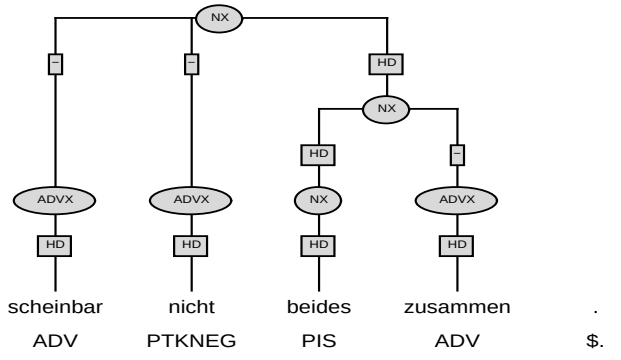

Figure 4: Export format of sentence 24 in cd20 and corresponding structure

longs to. Each class of pairs is an equivalence class on the sets of node pairs. In order to obtain a compact characterization of these classes, additional node classes are introduced. This double abstraction allows an efficient storage of large amounts of data and facilitates the search in the corpus. The only feature that is not part of the characteristics of node classes is the token of a node. The tokens of the corpus are stored in an extra table.

3.2. Initializing the Database

The storage of the corpus in the database is done by an initializing component. This component extracts information from the structures in export format (the format used for the Tübingen German Treebank) and stores them in the database. Details about the initializing process can be found in Kallmeyer (2000b).

As an example, consider the storage of sentence 24, subcorpus cd20 (identifier 20) in the database. This sentence was chosen for the simple reason that it is not too long but contains enough nodes to provide a useful example. Besides this, its construction and its tokens are not of any interest here.

Fig. 4 shows the sentence in its export format, i.e., the way it originally occurs in the corpus, together with a picture of the corresponding structure. Parts of the tables in the database concerning sentence 24 are shown in Fig. 5. Each line in the export format corresponds to one node. The nodes are assigned numbers 0, 1, ... The nodes with tokens (i.e., the leaves) are inserted into the table tokens_20.

tokens_20

tree_id	node	word
24	0	scheinbar
24	1	nicht
24	2	beides
24	3	zusammen
24	4	.

node_pair_20

tree_id	node1	node2	class_id
24	0	1	1459
24	0	2	2608
24	0	3	120
	...		
24	9	10	1327
	...		

pair_class

class_id	node_id1	node_id2	p1	p2	d1	d2	l1	l2
		...						
120	13	13	0	0	0	0	1	0
		...						
1327	24	25	0	1	0	1	0	0
		...						

node_class

node_id	cat	fct
	...	
13	ADV	HD
	...	
24	NX	HD
25	NX	–
	...	

Figure 5: Sentence 24 in the database

Each pair of nodes is inserted into the table node_pair_20 together with its pair class. Both orders of a pair are stored.[1] The pair classes and node classes belonging to a pair can be found in the global tables. Consider for example the nodes 9 and 10 in sentence 24 (the node labelled NX that dominates *beides zusammen* and the topmost node with label NX). The class_id of this pair is 1327. The corresponding entry in pair_class tells us that the second node is the mother of the first, that the second dominates the first, and that there is no linear precedence relation between the two nodes. Furthermore, the node classes identified by node_id1 and node_id2 are such that the first node has label NX and grammatical function HD whereas the second has label NX and no grammatical function.

After having stored the data in the data base, several indexes are created using the MySQL indexing functionalities, e.g., indexes are put on the column class_id in table node_pair_*i* and on node_id1 and node_id2 in pair_class. This accelerates the search of the corpus considerably.

In order to test the tool approximately one quarter of the Tübingen German Treebank is stored in the database,

[1] In a previous version just one order was stored but it turned out that for some queries this causes an exponential time complexity depending on the number of variables occurring in the query. This problem is avoided by storing both orders of a node pair.

i.e., 5 subcorpora containing all together 10112 trees and 98253 tokens. The table pair_class has 46048 entries and node_class has 213 entries. The current size of all data files is 58 MB and the size of the index files 453 MB. We expect the size of the global tables pair_class and node_class not to increase substantially when storing additional subcorpora.

3.3. The Query Component

In order to search the corpus, one needs of course to know the specific properties of the annotation scheme. These are described in the STTS guidelines (Schiller et al., 1995) and in the stylebook for the German Treebank (Stegmann et al., 2000), that must be both available to any user of the query tool.

The input of the query component is an expression in the query language. This is translated into an SQL expression, which is then passed to the database. As an example, consider again the construction in (1) which leads the query in (5) (searching a dative NP preceding an accusative NP and a node `MF` dominating the first two nodes).

(5)
```
cat(1)=NX & fct(1)=OD
  & cat(2)=NX & fct(2)=OA & 1..2
  & cat(3)=MF & 3>>1 & 3>>2
```

For query (5) as input performed on subcorpus cd24, the query component produces the following SQL query:

```
SELECT DISTINCT np1.tree_id
FROM node_class AS nc1,
node_class AS nc2, node_class AS nc3,
node_pair_24 AS np1, pair_class AS pc1,
node_pair_24 AS np2, pair_class AS pc2,
node_pair_24 AS np3, pair_class AS pc3
WHERE nc1.cat='NX' AND nc1.fct='OD'
AND nc2.cat='NX' AND nc2.fct='OA'
AND nc3.cat='MF'
AND pc1.node_id1=nc1.node_id
AND pc1.node_id2=nc2.node_id AND pc1.l1=1
AND pc2.node_id1=nc3.node_id
AND pc2.node_id2=nc1.node_id AND pc2.d1=1
AND pc3.node_id1=nc3.node_id
AND pc3.node_id2=nc2.node_id AND pc3.d1=1
AND np1.class_id=pc1.class_id
AND np1.node1=np2.node2
AND np1.node2=np3.node2
AND np1.tree_id=np2.tree_id
AND np2.class_id=pc2.class_id
AND np2.node1=np3.node1
AND np2.tree_id=np3.tree_id
AND np3.class_id=pc3.class_id;
```

As a second example, query (4) for complex prepositional phrases, involving three pairs of nodes but just one node class and two tokens, is repeated here as (6):

(6)
```
token(1)=von & token(2)=bis & 1..2
  & cat(3)=PX & 3>>1 & 3>>2
```

Performed on cd20, (6) as input leads to the following SQL query:

```
SELECT DISTINCT np1.tree_id
FROM node_class AS nc1,
node_pair_20 AS np1, pair_class AS pc1,
node_pair_20 AS np2, pair_class AS pc2,
node_pair_20 AS np3, pair_class AS pc3,
tokens_20 AS t1, tokens_20 AS t2
WHERE nc1.cat='PX' AND pc1.l1=1
AND pc2.node_id1=nc1.node_id AND pc2.d1=1
AND pc3.node_id1=nc1.node_id AND pc3.d1=1
AND np1.class_id=pc1.class_id
AND np1.node1=np2.node2
AND np1.node2=np3.node2
AND np1.tree_id=np2.tree_id
AND np2.class_id=pc2.class_id
AND np2.node1=np3.node1
AND np2.tree_id=np3.tree_id
AND np3.class_id=pc3.class_id
AND t1.word='von'
AND np1.tree_id=t1.tree_id
AND np1.node1=t1.node
AND t1.tree_id=t2.tree_id
AND t2.word='bis'
AND np1.node2=t2.node;
```

As can be easily seen, there is no direct correspondence between a conjunct in the query language of VIQTORYA and a conjunct in the WHERE-clause of the corresponding SQL statement. Instead, the translation process of a query depends on the use of the different node variables involved. Therefore, the first step of the computation is to determine for each node variable in the query in which predicates or relations the node variable appears and to collect the variable pairs used in binary relations. In query (6), for example, the variable pairs are $< 1, 2 >$, $< 3, 1 >$ and $< 3, 2 >$.

For each variable pair (e.g., $< 1, 2 >$), an SQL variable for the tables node_pair_i (`np1`) and pair_class (`pc1`) is needed with the same class_id (e.g., `np1.class_id=pc1.class_id`) and the appropriate relation is chosen (`pc1.l1=1`). If a node variable is involved in several binary relations (e.g., var. 1) the individual nodes in table node_pair, of course, must match (e.g., `np1.node1=np2.node2`, since variable 1 is first element of the first variable pair $< 1, 2 >$ and second element of the second pair $< 3, 1 >$). For each node variable involved in the predicates `cat` or `fct` (var. 3), an SQL variable for the table node_class (`nc1`) with the appropriate value is created (`nc1.cat='PX'`) that matches one of the two node_ids of the corresponding pair_class variable (e.g., `pc2.node_id1=nc1.node_id`), depending if the node variable is first or second element of the relation. For each node variable involved in the token predicate (e.g., var. 1), an SQL variable for the table tokens_i (e.g., `t1`) with the appropriate value is created (e.g., `t1.word='von'`) that matches the tree_id (e.g., `np1.tree_id=t1.tree_id`) and one of the two nodes for the corresponding node_pair_i variable (e.g., `np1.node1=t1.node`), depending if the node variable is first or second element of the relation. Finally, several variables for tables node_pair_i resp. tokens_i must match with regard to the tree_id (e.g., `t1.tree_id=t2.tree_id`). Note that the search for the category or function of a node that is not involved in a binary relation cannot be evaluated without the tables pair_class and node_pair_i, whereas this does not hold for the search for tokens.

We implemented the composition of an SQL expression in a modular way. Each conjunct in the WHERE-clause is assigned a particular rank and an additional subrank for each rank. The conjuncts are ordered according to this ranking. For example, the same rank is assigned for all conjuncts that are concerned with the same table (e.g., table `node_class`) and the subrank determines the ordering of the different variables for this table (e.g., `nc1`, `nc2`, etc.). If it turns out at a later point that a different ordering of the SQL conjuncts is favoured, this can be easily modified with the present implementation.

The performance depends crucially on the size of intermediate results. In cases where more than one node pair is searched for (as in the examples above) the order of the pairs is important since the result set of the first pair restricts the second pair etc. In (5) for example, two node pairs that involve the search for a node `NX` with function `OD` are searched for first. Afterwards, the search for node `NX` with function `OA` dominated by `MF` is restricted to those trees that were found when searching for the first two pairs. Obviously, each of the first two pairs is much more restrictive than the third. If the order is reversed and the third node pair is searched for first, the query takes much more time to process. Currently the ordering of the pairs needs to be done by the user, i.e., depends on the incoming query. However, we plan to implement at least partly an ordering of the binary conjuncts in the query depending on the frequency of the syntactic categories and grammatical functions involved in the pairs.

In contrast to the examples illustrated above, queries containing disjunctions are not translated directly into an SQL expression. Instead they are transformed first into disjunctive normal form. The different disjuncts are then translated separately into the corresponding SQL statements and evaluated iteratively. Following, the union of the search results is built. In most cases this procedure leads to a much better performance than the evaluation of a large and complex SQL statement, especially in those cases where the disjunction contains binary relations. But we also found that even disjunctions of category or function labels can best be treated this way. An additional advantage of transforming queries first into disjunctive normal form is that the query language defined in Sec. 2. and the query component can be extended easily in order to allow negation in a general way with just one more transformation step prior to the proposed procedure, namely transforming negation in general into atomic negation.

Currently, the query component can process almost all possible expressions in the query language. However, queries involving disjunctions are so far only allowed in disjunctive normal form. The query component will be completed very soon to process the full range of disjunction defined in Sec. 2.

The database and the tool are running at present on a Pentium II PC 400MHz 256MB under Linux. On this machine, example (5) takes 1.26 sec to be answered by MySQL, and example (6) takes 0.28 sec to be answered. This shows that although the queries are quite complex and involve many intermediate results, the performance of the system is quite efficient.

The obvious advantage of using a relational database to store the corpus is that some parts of the work are taken over by the database management system such as the search of the corpus. Furthermore, and this is crucial, the indexing functionalities of the database management system can be used to increase the performance of the tool.

3.4. The Graphical User Interface of VIQTORYA

Since VIQTORYA will be used by linguists a user-friendly handling of the tool is important. Therefore we developed a graphical user interface for the specification of the queries that are then passed on to the query component. In a comfortable way the user can choose a corpus, can specify nodes with their properties (including, e.g., the topological field a node is situated in) and relations between nodes. The user does not need to know the specific syntax of the query language since the queries can be generated automatically on the basis of the chosen specifications, at least for standard queries without complex disjunction. Disjunction in general will be supported by the user interface very soon. Furthermore, specified queries can be stored for later use and refinement.

Figure 6 shows the main window of the user interface containing the specification of query (6). Specified nodes together with their properties (PoS-tag or category, function and, in the case of terminals, token) are shown in the left window. For example, in Fig. 6 the first node ("Node 1") is specified only for token (*von*) and the third node ("Node 3") is specified only for category (`PX`). The right window displays the specified binary relations. On the basis of these specifications the query can be generated automatically and will show up in the query window on top. Then the query together with the chosen corpus (here subcorpus cd20) can be submitted to the query component.

Terminal nodes, nonterminal nodes and binary relations are specified with the bottom part of the main window. In Figure 7 the specification of "Node 3" (in Fig. 6) is shown, i.e., the specification of its syntactic category `PX`. The different properties of a node can be chosen with the register on top of the node window, i.e. category, function, topological field in case of nonterminal nodes. In case of terminal nodes there is an additional option for "token" and "category" is replaced by "PoS-tag". For each property the negation of the corresponding predicate can be chosen (see in Fig. 7 "Category of Node is not...") which leads to a modified surface of the node window in order to have the possibility to choose more than one value for a predicate.

For the specification of binary relations all nodes that are already specified are displayed in the relation window. The nodes involved in the binary relation and the relation itself can be chosen. Fig. 8 shows the specification of the first relation in Fig. 6 ("Node 1 linearly precedes Node 2").

For German corpora that are annotated with topological fields (such as the Tübingen German Treebank) there is an option for terminal and nonterminal nodes to determine the topological field the node is situated in. As an example consider again the construction in Fig. 2 with the corresponding query in (5). E.g., for the node (node variable 1) with syntactic category `NX` and grammatical function `OD`, the topological field 'mittelfeld' (`MF`) can be specified as additional

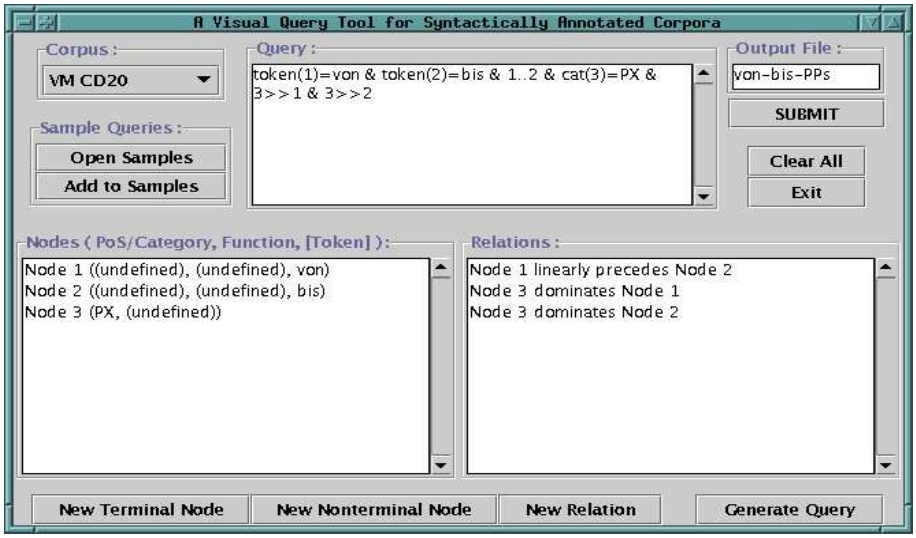

Figure 6: Specification of the query in (6)

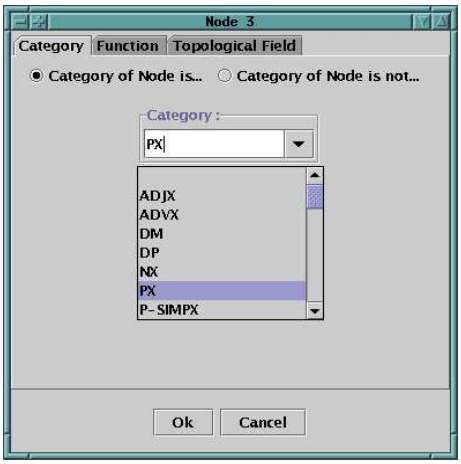

Figure 7: Specification of "Node 1" in Fig. 6

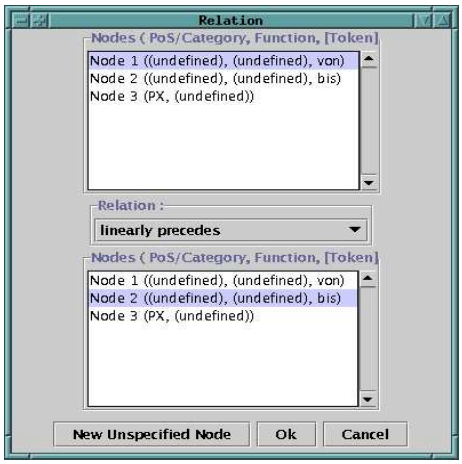

Figure 8: Specification of the first binary relation in Fig. 6

property. Since the theory of topological fields is specific for German and, furthermore, not every German corpus is annotated with topological fields, we implemented this option in a modular way, i.e., for corpora that are not annotated with topological fields it can be faded out easily.

Having specified and submitted a query, the corresponding search results can be viewed in the Annotate tool.

4. Related Work

Recently, some other query tools for syntactically annotated corpora have been proposed, e.g., TIGERSearch (Lezius and König, 2000), ICECUP III (Wallis and Nelson, 2000), CorpusSearch (Randall, 2000) and tgrep2 (Rohde, 2001). We will briefly compare VIQTORYA to them.

The most interesting tool with respect to a comparison to VIQTORYA is TIGERSearch since it was primarily developed for the data structures used in the Negra corpus (allowing crossing branches) and therefore has to deal with the specific problems of annotations that are not trees.

The query language of TIGERSearch allows the same logical connections as VIQTORYA (conjunction, disjunction and atomic negation). TIGERSearch comprises some binary relations that are not present in VIQTORYA but that either can be defined using the existing relations or can be easily integrated extending the pair_class table in a corresponding way.

There is a crucial difference between TIGERSearch and VIQTORYA concerning the way linear precedence between internal nodes is defined: in TIGERSearch a node n_1 linearly precedes a node n_2 if the leftmost leaf dominated by n_1 linearly precedes the leftmost leaf dominated by n_2. One problem of this definition is that it does not exclude a dominance relation between n_1 and n_2: in Fig. 2, according to this definition, the `SIMPX` node linearly precedes the `MF` node which seems counter-intuitive. Furthermore, for corpora without crossing branches but with subtrees 'wrapped' around other subtrees such as the Tübingen Treebanks this definition is not adequate. It means that n_1 linearly precedes n_2 even if the subtree rooted by n_1 is 'wrapped' around the subtree with root n_2 (e.g., in Fig. 1 the `SIMPX` node would linearly precede the `VXFIN` that dominates the

first *macht*). Therefore, in VIQTORYA we chose the following definition: n_1 linearly precedes n_2 if all nodes dominated by n_1 linearly precede all nodes dominated by n_2.

ICECUP, developed for the ICE-GB corpus, uses a query language with predicates that are slightly different from VIQTORYA. ICECUP, however, does not allow negation for binary predicates and, moreover, is restricted to trees and cannot handle the data structures used in the Tübingen Treebanks.

Neither tgrep, developed for the Penn Treebank, nor CorpusSearch, developed for the Penn Helsinki Corpus of Middle English, provide a graphical user interface. Both allow sets of unary and binary predicates slightly different from VIQTORYA. CorpusSearch, however, does not allow negation of these predicates and general disjunction.

Concerning the general architecture of the tools, a key feature of VIQTORYA is that, in contrast to the tools mentioned above, it uses a relational database schema that allows an efficient search in large amounts of data.

5. Conclusion and Future Work

In this paper, we have presented VIQTORYA, a visual query tool for syntactically annotated corpora, that is implemented for the Tübingen German Treebank annotated at the University of Tübingen. The key idea is to extract in an initializing phase the information one wants to search for from the corpus and to store it in a compact way in a relational database. The search itself is done by translating an input query, that is an expression in a simple quantifier free first order logic, into an SQL query that is then passed to the database system. A graphical user interface allows to specify queries in a user-friendly way.

An obvious advantage of this architecture is that a considerable amount of work is taken over by the database management system and therefore needs not to be implemented. Furthermore, the MySQL indexing functionalities can be used to directly affect the performance of the search.

The query tool is work in progress, and we briefly want to point out some of the things that still need to be done. First, the set of queries the tool can process needs to be extended to the full range of disjunction allowed in the query language. This will be done very soon. Another task for the near future is, as mentioned above, to add an ordering mechanism on binary conjuncts in order to ensure that the more restrictive node pairs are searched for first.

Besides these tasks, a more general issue to persue in the future is to adapt the tool to other corpora. In some cases, this implies a modification of the way binary relations are precompiled in the initialization process, and in some other cases this would even lead to a modification of the query language and the database schema, namely in those cases where other binary relations are needed, e.g., the coindexation relation in the case of the Penn Treebank.

6. Acknowledgments

The work presented here was done as part of the project A1 in SFB 441 “Linguistic Data Structures” at the University of Tübingen. We would like to thank our research assistant Thomas Schurtz for the implementation of the graphical user interface. For valuable discussions of the work presented here we would like to thank Detmar Meurers, Oliver Plaehn and, in particular, Stephan Kepser. Furthermore, we are grateful to the project A2 for support in the final phase of this work.

7. References

A. Abeillé and L. Clément. 1999. A tagged reference Corpus for French. In *Proceedings of EACL-LINC*, Bergen.

A. Bies, M. Ferguson, K. Katz, and R. MacIntyre. 1995. Bracketing Guidelines for Treebank II Style Penn Treebank Project. University of Pennsylvania.

T. Brants and W. Skut. 1998. Automation of treebank annotation. In *Proceedings of NeMLaP-3/CoNLL98, Sydney, Australia*, pages 49 – 57.

T. Brants, W. Skut, and H. Uszkoreit. 1999. Syntactic Annotation of a German Newspaper Corpus. In *Journées ATALA, 18–19 juin 1999, Corpus annotés pour la syntaxe*, pages 69–76, Paris.

S. Featherston. 2002. Coreferential objects in german: Experimental evidence on reflexivity. In *Linguistische Berichte*. Buske Verlag. In press.

E. W. Hinrichs, J. Bartels, Y. Kawata, V. Kordoni, and H. Telljohann. 2000. The VERBMOBIL Treebanks. In *Proceedings of KONVENS 2000*, October.

T. Höhle. 1985. Der Begriff ‘Mittelfeld’. Anmerkungen über die Theorie der topologischen Felder. In A. Schöne, editor, *Kontroversen alte und neue. Akten des 7. Int. Germanistenkongresses Göttingen*, pages 329–340.

L. Kallmeyer. 2000a. On the Complexity of Queries for Structurally Annotated Linguistic Data. In *Proceedings of ACIDCA’2000*, pages 105–110, March.

L. Kallmeyer. 2000b. A query tool for syntactically annotated corpora. In *Proceedings of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong, October.

W. Lezius and E. König. 2000. Towards a search engine for syntactically annotated corpora. In *Proceedings of the KONVENS Conference*.

M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. 1994. The Penn Treebank: Annotating Predicate Argument Structure. In *ARPA ’94*.

D. Meurers. 1999. Von partiellen Konstituenten, erstaunlichen Passiven und verwirrten Franken. Zur Verwendung von Korpora für die theoretische Linguistik. Handout at the DGfS Jahrestagung, February.

B. Randall, 2000. *CORPUSSEARCH USER’S MANUAL*. University of Pennsylvania.

D. L. T. Rohde, 2001. *Tgrep2 User Manual - version 1.2*. Carnegie Mellon University, Pittsburgh.

A. Schiller, S. Teufel, and C. Thielen. 1995. Guidelines für das Tagging deutscher Textcorpora mit STTS. Manuscript Universitäten Stuttgart und Tübingen.

R. Stegmann, H. Telljohann, and E. W. Hinrichs. 2000. Stylebook for the German Treebank in VERBMOBIL. Technical Report 239, Verbmobil.

S. A. Wallis and G. Nelson. 2000. Exploiting fuzzy tree fragments in the investigation of parsed corpora. *Literary and Linguistic Computing*, 15(3):339–361.