Evaluating corpus query systems on functionality and speed: TIGERSearch and Emdros

Ulrik Petersen
Department of Communication, University of Aalborg
Kroghstræde 3
9220 Aalborg East, Denmark
ulrikp@hum.aau.dk
http://emdros.org/

Abstract

In this paper, we evaluate two corpus query systems with respect to search functionality and query speed. One corpus query system is TIGERSearch from IMS Stuttgart and the other is our own Emdros corpus query system. First, we show how the database model underlying TIGERSearch can be mapped into the database model of Emdros. Second, the comparison is made based on a set of standard linguistic queries culled from the literature. We show that by mapping a TIGERSearch corpus into the Emdros database model, new query possibilities arise.

1 Introduction

The last decade has seen a growth in the number of available corpus query systems. Some query systems which have seen their debut since the mid-1990ies include MATE Q4M (Mengel 99), the Emu query language (Cassidy & Bird 00), the Annotation Graph query language (Bird *et al.* 00), TGrep2 (Rohde 04), TIGERSearch (Lezius 02b), NXT Search (Heid *et al.* 04), Emdros (Petersen 04), and LPath (Bird *et al.* 05). In this paper, we have chosen to evaluate and compare two of these, namely TIGERSearch and Emdros.

TIGERSearch is a corpus query system made at the Institut für Maschinelle Sprachverarbeitung at the University of Stuttgart (Lezius 02a; Lezius 02b). It is a general corpus query system over so-called *syntax graphs* (König & Lezius 03), utilizing the TIGER-XML format for import (Mengel & Lezius 00). Converters have been implemented for the Penn Treebank, NeGRA, Susanne, and Christine formats, among others. It is available free of charge for research purposes.[1]

Emdros is also a general corpus query system, developed at the University of Aalborg, Denmark. It is applicable to a wide variety of linguistic corpora supporting a wide variety of linguistic theories, and is not limited to treebanks. It implements the EMdF model and the MQL query language described in (Petersen 04). Importers for the TIGER-XML and other corpus formats have been implemented, and more are under development. It is available free of charge as Open Source software from the address specified at the beginning of the paper.

The layout of the rest of the paper is as follows. First, we briefly introduce the EMdF database model underlying Emdros. Second, we introduce the database model underlying TIGERSearch. Next, we show how to map the TIGERSearch database model into the EMdF model. The next section explores how the TIGERCorpus (Brants & Hansen 02), now in Emdros format, can be queried with – in some instances – greater functionality and speed by Emdros than by TIGERSearch. Finally, we conclude the paper.

[1] See http://www.tigersearch.de/

2 The EMdF model of Emdros

The EMdF text database model underlying Emdros is a descendant of the MdF model described in (Doedens 94). At the backbone of an EMdF database is a string of *monads*. A monad is simply an integer. The sequence of the integers dictates the logical reading sequence of the text. An *object* is an arbitrary (possibly discontiguous) set of monads which belongs to exactly one *object type*. An object type (e.g., Word, Phrase, Clause, Sentence, Paragraph, Article, Line, etc.) determines what *features* an object has. That is, a set of attribute-value pairs are associated with each object, and the attributes are determined by the object type of the object. All attributes are strongly typed. Every object has a database-widely unique ID called its *id_d*, and the feature *self* of an object denotes its id_d. The notation $O.f$ is used to denote the value of feature f on an object O. Thus, for example, $O_1.self$ denotes the id_d of object O_1. An id_d feature can have the value NIL, meaning it points to no object. No object can have NIL as its id_d.

The sample tree in Figure 1 shows a discontiguous element, and is adapted from (McCawley 82, p. 95). The tree can be visualized as an EMdF database as in Figure 2. This figure exemplifies a useful technique used for representing tree-structures in Emdros: Since, in a tree, a child node always has at most one parent, we can represent the tree by means of id_d features pointing upwards from the child to its parent. If a node has no parent (i.e., is a root node), we can represent this with the value NIL. This technique will be used later when describing the mapping from TIGERSearch to EMdF.

3 The TIGERSearch database model

The database model underlying TIGERSearch has been formally described in (Lezius 02a) and (König & Lezius 03). The following description has been adapted from the former, and is a slight reformalization of the database model with respect to edge-labels.

Definition 1 A *feature record* F is a relation over $FN \times C$ where FN is a set of feature-names and C is a set of

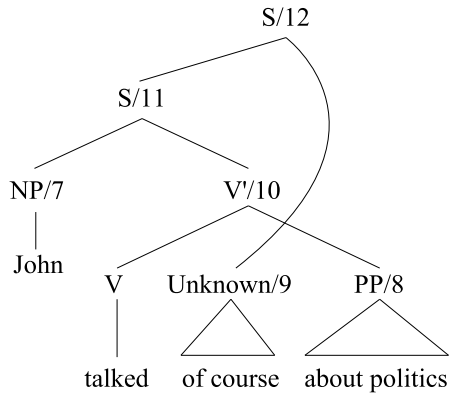

Figure 1: A tree with a discontiguous clause, adapted from (McCawley 82, p. 95).

	1	2	3	4	5	6
Word	id_d: 1 surf.: John pos: NProp parent: 7	id_d: 2 surf.: talked pos: V parent: 10	id_d: 3 surf.: of pos: P parent: 9	id_d: 4 surf.: course pos: N parent: 9	id_d: 5 surf.: about pos: P parent: 8	id_d: 6 surf.: politics pos: N parent: 8
Phrase	id_d: 7 type: NP parent: 11		id_d: 9 type: Unknown parent: 12		id_d: 8 type: PP parent: 10	
Phrase		id_d: 10 type: V' parent: 11			id_d: 10 type: V' parent: 11	
Clause	id_d: 11 type=S parent: 12				id_d: 11 type=S parent: 12	
Clause	id_d: 12 type=S					

Figure 2: An EMdF representation of the tree in Figure 1.

constants. The relation is defined such that for any $l_i = \langle f_i, c_i \rangle$ and any $l_j = \langle f_j, c_j \rangle$, $l_i \neq l_j \Rightarrow f_i \neq f_j$. That is, all f_i within a feature-record are distinct. The set of all feature-records over FN and C is denoted $\mathcal{F}$.

Definition 2 The set of all node ids is called ID and the relation $ID \subset C$ holds.

Definition 3 A *node* is a two-tuple $v \in ID \times \mathcal{F}$. That is, a node consists of a node id ν and a feature-record F.

Definition 4 A *syntax graph* G in the universe of graphs $\mathcal{G}$ is a six-tuple $G = (V_{NT}, V_T, L_G, E_G, O_G, R_G)$ with the following properties:

1. V_{NT} is the (possibly empty) set of non-terminals.
2. V_T is the non-empty set of terminals.
3. L_G is a set of edge labels where $L_G \subset C$.[2]
4. E_G is the set of labeled, directed edges of G. E_G is a set of two-tuples from $V_{NT} \times (V_{NT} \cup V_T)$. If L_G is non-empty, there exists an assignment of edge-labels el which is a total function $el : E_G \rightarrow L_G$ which need be neither surjective nor injective.[3]
5. O_G is a bijective function $O_G : V_T \rightarrow \{1, 2, \ldots, |V_T|\}$ which orders the terminal nodes. That the function is bijective guarantees that all terminal nodes can be ordered totally by O_G.
6. $R_G \in V_{NT}$ is the single root node of G, and has no incoming edges.

G is a graph with the following characteristics:

G1: G is a DAG with exactly one root node R_G.

G2: All nodes $v \in ((V_{NT} \cup V_T) \setminus R_G)$ have exactly one incoming edge in E_G.

G3: All nonterminals $v \in V_{NT}$ must have at least one outgoing edge. That is, $\forall v \in V_{NT} \exists v' \in (V_{NT} \cup V_T) : \langle v, v' \rangle \in E_G$.[4]

Thus syntax graphs are not strict trees in the traditional sense, since crossing edges are not prohibited. Nevertheless, syntax graphs are not arbitrary DAGs, since by **G2**, every node has at most one parent, and in this respect they do resemble trees.

This brief reformulation does not do justice to the full description available in (Lezius 02a) and (König & Lezius 03). For more information on the syntax graph formalism, see the cited publications.

4 Mapping syntax graphs to EMdF

TIGERSearch was developed specifically for use with the TIGERCorpus (Brants & Hansen 02), though it is applicable to other corpora as well (Lezius 02a, p. 136). In order to compare TIGERSearch with Emdros, we had to import a corpus available for TIGERSearch into Emdros. The TIGERCorpus was chosen because it represents the primary example of a TIGERSearch database, and because it has a reasonably large size, furnishing a basis for speed-comparisons.

We have developed an algorithm to transform any database encoded in the syntax graph formalism into an EMdF database. This section describes the algorithm. First, we give some definitions, after which we show the four algorithms involved.

Definition A1: For any syntax graph G, Obj_G is the set of EMdF objects which G gives rise to, and IDD_G is the set of id_d's of the objects in Obj_G. Note, however, that IDD_G may be defined before Obj_G, since there is no causality in the direction from Obj_G to IDD_G; in fact it is the other way around in the algorithms below.

[2] The latter restriction is not mentioned by (Lezius 02a) directly on page 103 where this is defined, but is inferred from the rest of the dissertation.

[3] This is where our reformulation differs in meaning from (Lezius 02a). We think our formalization is slightly clearer than Lezius', but we may, of course, have misunderstood something.

[4] Again, my reformulation differs slightly from Lezius' formulation, due to my reinterpretation of E_G.

Definition A2: For any syntax graph G, NOB_G is a bijective function from syntax graph nodes in G to Obj_G. That is, $NOB_G : (V_{NT} \cup V_T) \rightarrow Obj_G$.

Definition A3: For any syntax graph G and $v \in (V_{NT} \cup V_T)$, $parent(v)$ is the parent node of v if v is not R_G, or $\emptyset$ if v is R_G.

Definition A4: For any syntax graph G and its concomitant Obj_G, id_d_G is a bijective function $id\_d_G : (V_{NT} \cup V_T) \rightarrow IDD_G$ with the definition $id\_d(v) ::= NOB_G(v).\text{self}$. Note, however, that this definition only holds *after* the algorithms have all been applied; in fact id_d_G is defined by construction rather than by the given intensional, after-the-fact definition.

With this apparatus, we can define four algorithms which use each other. Algorithm 0 merely creates an empty object with a unique EMdF id_d corresponding to each node in a syntax graph G. Algorithm 1 adds monads to all objects corresponding to a nonterminal (i.e., all syntax-level nodes). Algorithm 2 constructs a set of EMdF objects for a given syntax graph G, and uses Algorithm 0 and 1. Algorithm 3 constructs an EMdF database from a set $\mathcal{G}$ of syntax graphs, and uses Algorithm 2

Algorithm 0: *Purpose*: Create empty objects in Obj_G and assign id_ds to each object and to the id_d_G function and IDD_G.
Input: A syntax graph G and a starting id_d d.
Output: A four-tuple consisting of the function id_d_G, the set IDD_G, the set Obj_G, the set NOB_G and an ending id_d d_e.

1. let $id\_d_G := \emptyset$, and let $Obj_G := \emptyset$
2. For all nodes $v \in (V_{NT} \cup V_T)$ (the ordering does not matter, so long as each node is treated only once):
   (a) let $id\_d_G(v) := d$
   (b) Create an EMdF object O_d being an empty set of monads and let O_d.self := d
   (c) let $Obj_G := Obj_G \cup \{O_d\}$
   (d) let $IDD_G := IDD_G \cup \{d\}$
   (e) let $NOB_G := NOB_G \cup \langle v, O_d \rangle$
   (f) let $d := d + 1$
3. Return $\langle id\_d_G, IDD_G, Obj_G, NOB_G, d \rangle$.

Algorithm 1: *Purpose*: To add monads to all objects corresponding to a non-terminal.
Input: A non-terminal p, the set IDD_G, and the set Obj_G.
Output: Nothing, but Obj_G is changed. (Obj_G is call-by-value here, so it is changed as a side-effect and not returned.)

1. Let $Ch := \{c | parent(c) = p\}$ (all immediate children of p.
2. For all $c \in Ch$:
   (a) If $c \in V_T$: Let $IDD_G(parent(c)) := IDD_G(parent(c)) \cup IDD_G(c)$ (Add terminals' monad-set to parent.)
   (b) Else:
      i. Call ourselves recursively with the parameters $langlec, IDD_G, Obj_G \rangle$.
      ii. Let $IDD_G(parent(c)) := IDD_G(parent(c)) \cup IDD_G(c)$ (Add c's monad-set to parent.)

Algorithm 2: *Purpose*: To construct a set of EMdF objects from a syntax graph G.
Input: A syntax graph G, a starting id_d d, and a starting monad m.
Output: A three-tuple consisting of a set of EMdF objects Obj_G, an incremented id_d d_e and an ending monad m_e.

1. Call Algorithm 0 on $\langle G, d \rangle$ to obtain $\langle id\_d_G, IDD_G, Obj_G, NOB_G, d_e \rangle$.
2. For all terminals $t \in V_T$:
   (a) let $O_t := NOB_G(t) \cup \{m_t\}$ where $m_t = O_G(t) + m - 1$. (Remember that an object is a set of monads, so we are adding a singleton monad set here.)
   (b) Let $O_t.\text{parent} := id\_d_G(parent(t))$ if t is not R_G, and NIL if t is R_G.
   (c) Assign other features of O_t according to the feature-record F in $t = \langle \nu, F \rangle$.[5]
   (d) if L_G is non-empty, let $O_t.\text{edge} := el(\langle parent(t), t \rangle)$
3. Call Algorithm 1 with the parameters $\langle R_G, IDD_G, Obj_G \rangle$. This assigns monad sets to all objects.
4. For all v in V_{NT}:
   (a) Let $O_v := Obj_G(v)$.
   (b) Let $O_v.\text{parent} := id\_d_G(parent(v_v))$ if v is not R_G, and NIL if v is R_G.
   (c) Assign other features of O_v according to the feature-record F in $v = \langle \nu, F \rangle$.
   (d) if L_G is non-empty, let $O_t.\text{edge} := el(\langle parent(t), t \rangle)$
5. Return $\langle Obj_G, d, m_t \rangle$ where $m_t \equiv O_G(v_t) + m - 1$ where v_t is the rightmost terminal node, i.e., $\exists v_t \in V_T : \forall v_j \in V_T : v_j \neq v_t \Rightarrow O_G(v_t) > O_G(v_j)$

Algorithm 3: *Purpose*: To construct a set of EMdF objects from a universe of syntax graphs $\mathcal{G}$.
Input: A set of syntax graphs $\mathcal{G}$, a starting id_d d, and a starting monad m.
Output: A two-tuple consisting of an incremented id_d d_e and an ending monad m_e.

[5] It is assumed, though the formalisation does not say so, that the feature-records of all V_T in all $G \in \mathcal{G}$ have the same "signature", i.e., have the same set of feature-names that are assigned a value in each F in each $v \in V_T$. A similar assumption is made for the signatures of all feature-records of all V_{NT}. This is certainly the case with the TIGERCorpus. Therefore, the object type `Terminal` is well-defined with respect to its features. Similarly for the object type `Nonterminal` used below.

```
Q1. Find sentences that include the word 'saw'.
Q2. Find sentences that do not include the word 'saw'.
Q3. Find noun phrases whose rightmost child is a noun.
Q4. Find verb phrases that contain a verb immediately
    followed by a noun phrase that is immediately
    followed by a prepositional phrase.
Q5. Find the first common ancestor of sequences of a
    noun phrase followed by a verb phrase.
Q6. Not relevant to TIGER Corpus.
Q7. Find a noun phrase dominated by a verb phrase.
    Return the subtree dominated by that noun phrase.
```

Figure 3: The test queries from (Lai & Bird 04), Fig. 1.

```
Q1  #s:[cat="S"] & #l:[word="sehen"] & #s >* #l
Q2* #s:[cat="S"] & #l:[word="sehen"] & #s !>* #l
Q3  #n1:[cat="NP"] & #n2:[pos="NN"]  & (#n1 >@r #n2)
Q4  #vp:[cat="VP"] & #v:[pos="VVFIN"] & #np:[cat="NP"]
    & #pp:[cat="PP"]& #vp >* #v & #vp >* #np
    & #vp >* #pp & #v >@r #vr & #np >@l #npl
    & #vr .1 #npl & #np >@r #npr & #pp >@l #ppl
    & #npr .1 #ppl
Q5* #vp:[cat="VP"] & #np:[cat="NP"] & (#np .* #vp)
    & (#x >* #vp) & (#x >* #np)
Q7* #vp:[cat="VP"] & #np:[cat="NP"] & (#vp >* #np)
```

Figure 4: The test queries of Figure 3 attempted implemented in TIGERSearch. Adapted from (Lai & Bird 04), Fig. 4. The queries marked with a * may not produce the correct results.

1. For all graphs G in $\mathcal{G}$ (if an ordering is intended, i.e., this is not a quotation corpus, then that order should be applied; otherwise, the order is undefined):
   (a) Let $\langle Obj_G, d_e, m_e \rangle$ be the result of calling Algorithm 2 on $\langle G, d, m \rangle$
   (b) Add Obj_G to the EMdF database.
   (c) Let $d := d_e$ and let $m := m_e + 1$
2. Return $\langle d, m \rangle$

5 Comparing TIGERSearch and Emdros

Using a variant of this algorithm, we have imported the TIGERCorpus into Emdros. This gives us a common basis for comparing TIGERSearch and Emdros.

The paper (Lai & Bird 04) sets out to specify some requirements on corpus query systems for treebanks that the authors perceive to be essential. Among other criteria, Lai and Bird set up a set of standard queries which are reproduced in Figure 3.

Lai and Bird show how some of the queries can be expressed in TIGERSearch, though they find that not all queries can be expressed. I have attempted to reformulate Lai and Bird's TIGERSearch queries in therms of the TIGERCorpus (see Figure 4).

Query Q2 cannot be formulated correctly in TIGERSearch. This is because what is being negated is the *existence* of the word "sehen", and in TIGERSearch, all nodes are implicitly existentially quantified. Negated existence would require a forall-quantification, as mentioned e.g. in (König & Lezius 03).

Query Q5 is probably not expressible in TIGERSearch, and the given query fails to find the *first* common ancestor only. The currect syntax graphs are returned, but with a number of subgraphs which are not rooted in the first common ancestor.

```
Q1  [Sentence [Word surface="sehen"] ]
Q2  [Sentence NOTEXIST [Word surface="sehen"] ]
Q3  [Phrase tag="NP" [Word last postag="NN"] ]
Q4  [Phrase tag="VP"
       [Word postag="VVFIN"]!
       [Phrase tag="NP"]!
       [Phrase tag="PP"]
    ]
Q5* [Phrase
       [Phrase tag="NP"][Phrase tag="VP"]
    ]
Q7* [Phrase tag="VP"  [Phrase tag="NP"]  ]
```

Figure 5: Emdros queries for Q1-Q7

```
Find all NPs which is a subject, inside of which there
is a relative clause whose parent is the NP.  Inside
the relative clause, there must be a phrase p2, inside
of which there must be a word which is a cardinal.  At
the end of the relative clause must be a finite verb
whose parent is the same as that of p2.  No PP may
intervene between p2 and the verb.

[Phrase as p1 tag="NP" AND edge="SB"
   [Phrase edge="RC" and parent=p1.self
      [Phrase as p2  [Word postag="CARD"] ]
      ..
      NOTEXIST [Phrase tag="PP"]
      ..
      [Word last postag="VVFIN"
            AND parent=p2.parent]
   ]
]
```

Figure 6: Emdros query for Q8

Query Q7 again finds the correct syntax graphs, but fails to retrieve exactly the subtree dominated by the NP. In TIGERSearch, what parts of a matched syntax-graph to retrieve is, in a sense, an irrelevant question, since the main result is the syntax graph itself. Thus the assumption of Lai and Bird that only parts of the matched tree is returned does not hold for TIGERSearch.

Emdros fares slightly better as regards functionality, as can be seen in Figure 5. Query Q2 is correctly expressed in Emdros using the NOTEXIST operator at object-level, which gives Emdros a slight edge over TIGERSearch in this comparison. However, queries Q5 and Q7 fail to give correct results on Emdros as they did on TIGERSearch. Query Q5 fails because, while it returns the correct syntax graphs, it fails to find only the first common ancestor. This is the same situation as with TIGERSearch. As in TIGERSearch, the requirement to find the "first common ancestor" is difficult to express in Emdros. Query Q7 fails because Emdros, like TIGERSearch, was not designed to retrieve subgraphs as part of the query results – subgraphs are to be retrieved later, e.g., for viewing purposes. Like TIGERSearch, Emdros returns the correct syntax graphs, and thus works as designed.

Query Q8 can be seen in Figure 6 along with the Emdros equivalent. It cannot be expressed in TIGERSearch because of the negated existence-operator on the intervening PP.

The queries were all timed, except for Q2 and Q6, which were not expressible in either or both of the corpos query systems. The hardware was an AMD Athlon 64 3200+ with

Query	Emdros	TIGERSearch
Q1	0.199; 0.202; 0.179	0.5; 0.3; 0.3
Q3	1.575; 1.584; 1.527	10.1; 9.9; 9.9
Q4	1.604; 1.585; 1.615	9.9; 9.9; 9.9
Q5	3.449; 3.319; 3.494	5.5; 6.6; 5.5
Q7	0.856; 0.932; 0.862	1.1; 1.1; 1.1
Q8	3.877; 3.934; 4.022	N/A

Table 1: Execution times in seconds

1GB of RAM and a 7200RPM harddrive running Linux Fedora Core 4. Three measurements were taken for each query. In the case of TIGERSearch, the timings reported by the program's status bar were used. For Emdros, the standard Unix command `time` was used. The results can be seen in Table 1.

As can be seen, Emdros is faster than TIGERSearch on every query that they can both handle. (Lezius 02a) mentions that the complexity is exponential in the number of query terms. It is very difficult to assess the complexity of an Emdros query, since it depends on a handful of factors such as the number of query items, the number of objects that match each query item, and the number of possible combinations of these.

Probably Emdros is faster in part because it takes a different algorithmic approach to query resolution that TIGERSearch: Instead of using proof-theory, it uses a more linear approach of first retrieving all possible object-"hits", then iteratively walking the query, combining the objects in monad-order as appropriate. Part of the speed increase may stem from its being written in C++ rather than Java, but for queries such as Q3 and Q4, the algorithm rather than the language seems to be the decisive factor, since such a large difference in execution time, relative to the other increases, cannot be accounted for by language differences alone.

6 Conclusion

In this paper, we have compared two corpus query systems, namely TIGERSearch on the one hand and our own Emdros on the other. We have briefly introduced the EMdF model underlying Emdros. The EMdF model is based on the MdF model described in (Doedens 94). We have also given a reformalization of the syntax graph formalism underlying TIGERSearch, based on the presentation given in (Lezius 02a). We have then presented an algorithm for converting the syntax graph formalism into the EMdF model.

Having done this, we have compared the two corpus query systems with respect to query functionality and speed. The queries were mostly culled from the literature. It was found that Emdros was able to handle all the test queries that TIGERSearch was able to handle, in addition to a few that TIGERSearch was not able to express. The latter involved the negation of the existence of an object; it is a limitation in the current TIGERSearch that all objects are implicitly existentially quantified, which means that negating the existence of an object is not possible. Negation at the feature-level is, however, possible in both corpus query systems. In both systems, the semantics of feature-level negation is the same as the $\neg$ operator in First Order Logic.

Finally, the test queries which both systems were able to handle were executed on the same machine over the same corpus, namely the TIGERCorpus, and it was found that Emdros was faster than TIGERSearch on every query, and that the algorithm of Emdros seems to scale better than that of TIGERSearch.

References

(Bird *et al.* 00) Steven Bird, Peter Buneman, and Tan Wang-Chiew. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 807–814. European Language Resources Association, Paris, 2000. http://arxiv.org/abs/cs/0007023 Access Online August 2004.

(Bird *et al.* 05) Steven Bird, Yi Chen, Susan Davidson, Haejoong Lee, and Yifeng Zheng. Extending XPath to support linguistic queries. In *Proceedings of Programming Language Technologies for XML (PLANX) Long Beach, California. January 2005.*, pages 35–46, 2005.

(Brants & Hansen 02) Sabine Brants and Silvia Hansen. Developments in the TIGER annotation scheme and their realization in the corpus. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002), Las Palmas, Spain, May 2002*, pages 1643–1649, 2002. http://www.ims.uni-stuttgart.de/projekte/TIGER/paper/lrec2002-brants-hansen.pdf Access Online August 2004.

(Cassidy & Bird 00) Steve Cassidy and Steven Bird. Querying databases of annotated speech. In M.E. Orlowska, editor, *Database Technologies: Proceedings of the Eleventh Australasian Database Conference, volume 22 of Australian Computer Science Communications, Canberra, Australia*, pages 12–20. IEEE Computer Society, 2000. http://arxiv.org/abs/cs/0204026, Access Online August 2004.

(Doedens 94) Christianus Franciscus Joannes Doedens. *Text Databases: One Database Model and Several Retrieval Languages*. Number 14 in Language and Computers. Editions Rodopi, Amsterdam and Atlanta, GA., 1994.

(Heid *et al.* 04) U. Heid, H. Voormann, J-T Milde, U. Gut, K. Erk, and S. Pado. Querying both time-aligned and hierarchical corpora with NXT Search. In *Fourth Language Resources and Evaluation Conference, Lisbon, Portugal, May 2004*, 2004.

(König & Lezius 03) Esther König and Wolfgang Lezius. The TIGER language. a description language for syntax graphs. formal definition. Technical report, Institut für Maschinelle Sprachverarbeitung (IMS), University of Stuttgart, Germany, April 22 2003.

(Lai & Bird 04) Catherine Lai and Steven Bird. Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop, December 2004*, pages 139–146, 2004.

(Lezius 02a) Wolfgang Lezius. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Unpublished PhD thesis, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart, December 2002. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4. http://www.ims.uni-stuttgart.de/projekte/corplex/paper/lezius/diss/, Access Online August 2004.

(Lezius 02b) Wolfgang. Lezius. TIGERSearch – ein Suchwerkzeug für Baumbanken. In Stephan Busemann, editor, *Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2002), Saarbrücken*, pages 107–114, 2002.

(McCawley 82) James D. McCawley. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106, 1982.

(Mengel & Lezius 00) Andreas Mengel and Wolfgang Lezius. An XML-based encoding format for syntactically analyzed corpora. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000), Athens, Greece, 31 May – 2 June 2000*, pages 121–126, 2000.

(Mengel 99) Andreas Mengel. MATE deliverable D3.1 – specification of coding workbench: 3.8 improved query language (Q4M). Technical report, Institut für Maschinelle Sprachverarbeitung, Stuttgart, 18. November, 1999. http://www.ims.uni-stuttgart.de/projekte/mate/q4m/.

(Petersen 04) Ulrik Petersen. Emdros — a text database engine for analyzed or annotated text. In *Proceedings of COLING 2004, held August 23-27 in Geneva*. International Commitee on Computational Linguistics, 2004. http://www.hum.aau.dk/~ulrikp/pdf/petersen-emdros-COLING-2004.pdf, Access online August 2004.

(Rohde 04) Douglas L. T. Rohde. Tgrep2 user manual, version 1.12. Available online http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf. Access Online April 2005, 2004.

(Voormann & Lezius 02) Holger Voormann and Wolfgang Lezius. TIGERin - Grafische Eingabe von Benutzeranfragen für ein Baumbank-Anfragewerkzeug. In Stephan Busemann, editor, *Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2002)*, pages 231–234, Saarbrücken, 2002.