Querying Both Parallel And Treebank Corpora: Evaluation Of A Corpus Query System

Ulrik Petersen

Department of Communication and Psychology
University of Aalborg, Kroghstræde 3
9220 Aalborg East, Denmark
ulrikp@hum.aau.dk

Abstract

The last decade has seen a large increase in the number of available corpus query systems. Some of these are optimized for a particular kind of linguistic annotation (e.g., time-aligned, treebank, word-oriented, etc.). In this paper, we report on our own corpus query system, called Emdros. Emdros is very generic, and can be applied to almost any kind of linguistic annotation using almost any linguistic theory. We describe Emdros and its query language, showing some of the benefits that linguists can derive from using Emdros for their corpora. We then describe the underlying database model of Emdros, and show how two corpora can be imported into the system. One of the two is a parallel corpus of Hungarian and English (the Hunglish corpus), while the other is a treebank of German (the TIGER Corpus). In order to evaluate the performance of Emdros, we then run some performance tests. It is shown that Emdros has extremely good performance on "small" corpora (less than 1 million words), and that it scales well to corpora of many millions of words.

1. Introduction

The last decade has seen a large increase in the number of available corpus query systems. Systems such as TGrep2 (Rohde, 2005), Emu (Cassidy and Harrington, 2001), TIGERSearch (Lezius, 2002a; Lezius, 2002b), NXT Search (Heid et al., 2004), Viqtoria, Xaira, Emdros, and others have been implemented during this time. Often, these corpus query systems will specialize in one or two kinds of corpora, such as time-aligned, treebank, parallel, or word-oriented corpora; others are optimized for a particular size of corpus.

The value of a corpus query system lies in its two-fold ability to store and retrieve corpora — both the text and its linguistic annotation. The query capability is important for researchers in both theoretical and computational linguistics. Theoretical linguists might be enabled to answer theoretical questions and back up their claims with actual usage rather than introspective intuitions about language. Computational linguists are given a repository in which to store their data in the short- or long-term, and are also being given query capabilities which might help them, e.g., test the accuracy of a parser or pull up a list of all words with specific properties.

In this paper, we present our own Corpus Query System, called Emdros[1]. Emdros is very generic, and can be applied to almost any kind of linguistic annotation from almost any linguistic theory. We show that when applied to parallel corpora, many millions of words are easily supported with quick execution times. When applied to treebanks, Emdros performs extremely well for "small" corpora (less than 1 million words; see (Petersen, 2005)), but performance is also good for "large" corpora (many millions of words).

The rest of the paper is laid out as follows. First, we briefly describe Emdros and the benefits a researcher might reap from using the software. Second, we describe the EMdF database model underlying Emdros. This sets the stage, then, for describing how the EMdF model has been applied to two corpora, namely the Hunglish corpus (Varga et al., 2005), and the TIGER Corpus (Brants and Hansen, 2002; Brants et al., 1999). We then describe some experiments used to evaluate the speed of Emdros based on these two corpora, followed by the results of the experiments and an evaluation of the results. Finally, we conclude the paper.

2. Benefits of Emdros

In this section, we briefly describe some of the characteristics and features of Emdros, as well as describing some of the query language of Emdros.

Emdros has a four-layer architecture (see Fig. 1): At the bottom, a relational DBMS lays the foundation, with backends for PostgreSQL, MySQL, and SQLite currently implemented. On top of that, a layer implementing the EMdF database model is found. The EMdF model is a particular model of text which lends itself extremely well to linguistic annotation, and is described in more detail in the next section. On top of the EMdF layer, a layer implementing the MQL query language is found. MQL is a "full access language", featuring statements for create/retrieve/update/delete on the full range of the data types made available in the EMdF model. The EMdF model and the MQL query language are descendants of the MdF model and the QL query language described in (Doedens, 1994).

On top of the MQL layer, any number of linguistic applications can be built. For example, the standard Emdros distribution comes with: a) a generic graphical query application; b) importers from Penn Treebank and NeGRA format (with more importers to come); c) exporters to Annotation Graph XML format and MQL; d) a console application for accessing the features of MQL from the command-line; e) a graphical "chunking-tool" for exemplifying how to use Emdros; f) and a number of toy applications showing linguistic use, among other tools.

Emdros has been deployed successfully in a number of research projects, e.g., at the Free University of Amster-

[1]See `http://emdros.org/` and (Petersen, 2004; Petersen, 2005; Petersen, 2006 to appear)

2447

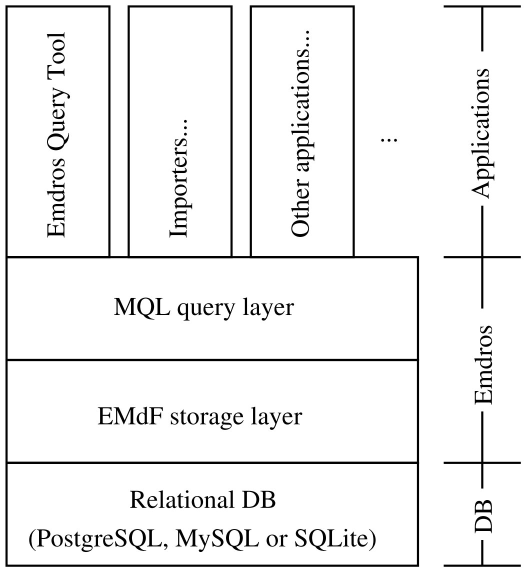

Figure 1: Emdros architecture

dam (for a database of Hebrew), and at the *Institut de Recerche en Informatique de Toulouse* (for a concordancer-application), among others. Two companies have licensed Emdros for inclusion in their software products, one of which is Logos Research Systems, using Emdros to query a number of Biblical Greek and Hebrew databases.

Emdros runs on Windows, Mac OS X, Linux, FreeBSD, NetBSD, Sun Solaris, and other operating systems, and has been implemented in a portable subset of C++. Language bindings are available for Java, Perl, Python, Ruby, and PHP. It is being made available under the GNU General Public License, but other licensing can be negotiated with the author.

The retrieval-capabilities of the MQL query language are particularly powerful, and can be very useful to linguists. Examples are given in Fig. 2 and Fig. 3.

MQL is centered around "blocks" enclosed in "[square brackets]". There are three kinds of blocks: *Object blocks* (which match objects in the database); *Gap blocks* (which match "gaps" in the database, e.g., embedded relative clauses); and *power blocks* (which match "arbitrary stretches of monads"). The examples given in this paper only use object blocks; for more examples, please see the website.

The overruling principle of MQL is: "The structure of the query mirrors the structure of the objects found", i.e., there is an isomorphism between the structure of the query and the inter-object structure of the objects found. This is with respect to two key principles of text, both of which are very familiar to linguists, namely "sequence" and "embedding". For example, query Q1 in Fig. 3 simply finds "Root" objects (i.e., "Sentence" objects) *embedded within which* there is a "Token" object whose attribute "surface" is equal to "sehen".

Similarly, query Q4 finds "Nonterminal" objects of type "NP" embedded within which we find: a) first a token of type "VVFIN", then b) a Nonterminal of type "NP", and then c) a Nonterminal of type "PP". The fact that these three are placed after each other implies (because of the overruling principle of MQL) that the objects found must occur in that sequence.

Query Q9 shows how to use references between objects — the surrounding NP Nonterminal is labelled "`AS p1`", which effectively gives the object a name which can be used further down in the query. This is used in query Q9 to ensure that the NP is the immediate parent of the objects found embedded inside of it (the automatically generated "self" attribute of any object gives the ID of that object).

Query Q3 and Q9 show the "first" and "last" keywords — meaning that the object that bears such a designation must be either "first" or "last" in its surrounding context.

Queries Q2 and Q8 show the "NOTEXIST" operator. As it is currently implemented, the NOTEXIST operator means that the following object must not exist in the surrounding context from the point at which it is found on to the end of the surrounding context. For example, in query Q8, once the Token of type "NN" has been found, there must not exist a Token of type "ADJA" or type "ADJD" after the "NN" token, up to the end of the surrounding NP. Note that this is *existential* negation at *object-level* ($\neg\exists$) — not negation of *equality* at the object *attribute* level ($\neq$).

Various attribute-comparison operators are available, including "=", "<>" (inequality)", "<", ">", "<=", ">=", `IN` a list, regular expressions "`~`", and negated regular expressions "`!~`", among others. Queries H1-H4 in Fig. 2 illustrate the regular expression operator "`~`" for simple queries. These examples, however do not show the full range of capabilities in MQL. For example, Kleene Star is not shown, nor is the `OR` operator between strings of objects shown. The latter supports searches for permutations of positions of objects using one query rather than several queries. MQL is able to handle queries of any complexity, and the queries shown here are all on the low end of the scale of complexity which MQL can handle. For more information, consult either the documentation on the website[2] or (Petersen, 2004; Petersen, 2005; Petersen, 2006 to appear).

3. The EMdF model

The EMdF (Extended MdF) model derives from the MdF (Monads dot Features) model described in (Doedens, 1994). There are four basic concepts in the EMdF model, which all derive from Doedens' work: Monad, Object, Object Type, and Feature. A monad is simply an integer — no more, no less. An object *is* a set of monads, and *belongs to* an Object Type. The Object Type groups objects with similar characteristics, e.g., Words, Phrases, Clauses, Sentences, Documents, etc. The model is generic in that it does not dictate *what* Object Types to instantiate in any database schema. Thus the database designer is free to design their linguistic database in ways that fit the particular linguistic problems at hand. The Object Type of an Object determines what *features* (or attributes) it has. Thus a database designer might choose to let the "Word" object type have features called "surface", "part_of_speech", "lemma", "gloss", etc. Or the database designer might choose to let the "Phrase" object type have features called "phrase_type", "function", "parent", etc.

[2] `http://emdros.org`

```
H1:  [Sentence english ~ " is "]
H2:  [Sentence english ~ " is " AND
        english ~ " was "
     ]
H3:  [Sentence english ~ " is " AND
        english ~ " necessary "
     ]
H4:  [Sentence english ~ " [Ii]s "
        AND english ~ " [Ww]as "
     ]
```

Figure 2: Queries on the Hunglish corpus

The backbone of the database is the string of monads (i.e., the integers: 1,2,3,... etc.). As mentioned, an object *is* a set of monads. The set is completely arbitrary, in that it need not be contiguous, but can have arbitrarily many "gaps". This supports things like embedded clauses with a surrounding clause of which it is not a part, discontiguous phrases, or other discontiguous elements.
Thus far, we have described the MdF model. The Extended MdF (EMdF) model that Emdros implements adds some additional concepts.
First, each object has an *id_d*, which is simply a database-widely unique integer that uniquely identifies the object.
Second, the datatypes that a feature can take on includes: strings, integers, id_ds, and enumerations (sets of labels), along with lists of integers, lists of id_ds, and lists of enumerations.
Third, an object type can be declared to be one of three *range-classes*. The range-classes are: a) "WITH SINGLE MONAD OBJECTS", b) "WITH SINGLE RANGE OBJECTS", and c) "WITH MULTIPLE RANGE OBJECTS". The "SINGLE MONAD" range-class is for object types that will only ever have objects that consist of a single monad, e.g., Word-object types. The "SINGLE RANGE" range-class is for object types that will only ever have contiguous objects, never objects with gaps. Finally, the "MULTIPLE RANGE" range-class is for object types that will have objects that *may* (but need not) have gaps in them. These range-classes are used for optimizations in the way the data is stored, and can lead to large performance gains when used properly.
In the next section, we show how we have applied the EMdF model to the design of two Emdros databases for two corpora.

4. Application

For the purposes of this evaluation, two corpora have been imported into Emdros. One is the Hunglish corpus (Varga et al., 2005), while the other is the TIGER Corpus (Brants and Hansen, 2002; Brants et al., 1999).
The TIGER Corpus has been imported from its instantiation in the Penn Treebank format, rather than its native NeGRA format. That is, the secondary edges have been left out, leaving only "normal" tree edges and labels. Coreference labels have, however, been imported.
Each root tree gets imported into an object of type "Root". This has been declared "WITH SINGLE RANGE OBJECTS".

```
Q1:  [Root
         [Token surface="sehen"]
     ]
Q2:  [Root
         NOTEXIST [Token surface="sehen"]
     ]
Q3:  [Nonterminal mytype="NP"
         [Token last mytype="NP"]
     ]
Q4:  [Nonterminal mytype="VP"
         [Token mytype="VVFIN"]!
         [Nonterminal mytype="NP"]!
         [Nonterminal mytype="PP"]
     ]
Q8:  [Nonterminal mytype="NP"
         [Token mytype="NN"]
         NOTEXIST [Token mytype="ADJA"
              OR mytype="ADJD"]
     ]
Q9:  [Nonterminal AS p1 mytype="NP"
         [Token FIRST mytype="ART"
              AND parent = p1.self
         ]
         [Token mytype="ADJA"
              AND parent = p1.self
         ]
         [Token LAST mytype="NN"
              AND parent = p1.self
         ]
     ]
```

Figure 3: Queries on the TIGER Corpus

Likewise, each Nonterminal (whether it be an S or a Phrase) gets imported into an object of type "Nonterminal". This object type has the features "mytype" (for the edge label, such as "NP"), function (for the function, such as "SUBJ"), and "coref" (a list of id_ds pointing to coreferent nodes), as well as a "parent" feature (pointing to the id_d of the parent).
Finally, each terminal (whether it be a word or punctuation) is imported as an object of type "Token". This object type has the same features as the "Nonterminal" object type, with the addition of a "surface" feature of type STRING, showing the surface text of the token. The "Token" object type has been declared "WITH SINGLE MONAD OBJECTS".
The Hunglish corpus has been imported in a very simple manner: Each sentence has been imported as a single object, belonging to the object type "Sentence". This object type has only two features: "English" and "Hungarian", both of which are of type "STRING". For each sentence, punctuation has been stripped, and each word surrounded by a space on both sides. This makes for easy searching using regular expressions. Since there is no syntactic markup for the Hunglish corpus, having only sentence-boundaries, it seemed natural to gather all words into a single string rather than splitting them out into separate objects. As it turns out, this leads to a huge increase in performance, simply because there are fewer rows to query in the backend. Each object occupies exactly one monad in the monad-stream, and so the object type has been declared "WITH

1000 Tokens	H1	H2	H3	H4
16531	6.53	7.625	7.74	6.2
33063	13.345	16.095	16.085	11.91
49595	21.08	23.705	23.58	18.565
66127	26.99	30.49	32.375	24.785
82659	33.98	42.485	40.245	31.275

Table 1: Average times in seconds for SQLite on the Hunglish corpus

1000 Tokens	Q1	Q2	Q3	Q4	Q8	Q9
712	0.47	0.80	1.91	1.17	3.37	2.40
2849	1.80	3.00	7.54	4.39	12.55	9.03
8547	5.37	9.16	22.97	12.75	36.56	27.64
17095	11.09	17.56	45.52	26.77	77.66	54.48
25643	16.97	26.83	72.64	43.68	117.72	84.76
34191	25.62	36.52	105.63	71.35	175.80	129.78

Table 2: Average times in seconds for SQLite on the TIGER corpus

SINGLE MONAD OBJECTS".

5. Experiments

In order to test the scalability of Emdros, both corpora have been concatenated a number of times: The Hunglish corpus has been concatenated so as to yield the corpus 1-5 times (i.e., with 0-4 concatenation operations), while the TIGER Corpus has been concatenated so as to yield the corpus 4, 12, 24, 36, and 48 times. There are 712,332 tokens and 337,881 syntactic objects on top in the TIGER corpus, yielding 34.19 million tokens and 16.22 million syntactic objects in the case where the corpus has been concatenated 47 times. For the Hunglish corpus, there are 852,334 sentences in two languages totalling 16,531,968 tokens. For the case where the corpus has been concatenated 4 times, this yields 81.09 million tokens and 4.26 million sentences.
A number of queries have been run on either corpus. They are shown in Fig. 2 for the Hunglish corpus and in Fig. 3 for the TIGER Corpus. For the TIGER Corpus, queries Q1-Q4 have been adapted from (Lai and Bird, 2004).
The performance of Emdros has been tested by running all queries in sequence, twice in a row each (i.e., Q1, Q1, Q2, Q2, etc.). The queries have been run twice so as to guard against bias from other system processes. This has been done on a Linux workstation running Fedora Core 4 with 3GB of RAM, a 7200 RPM ATA-100 harddrive, and an AMD Athlon64 3200+ processor. The queries have been run against each of the concatenated databases.
For each database, a number of queries have been run against the database before speed measurements have taken place, in order to prime any file system caches and thus get uniform results.[3] In a production environment, the databases would not be queried "cold", but would be at least partially cached in memory, thus this step ensures production-like conditions.

6. Results

The results of the experiments can be seen in Figures 4–5. Fig. 4 shows the time for queries H1-H4 added together on

[3]The queries used for "priming" were: H1 for the Hunglish corpus; and Q2, Q4, and Q8 for the TIGER Corpus.

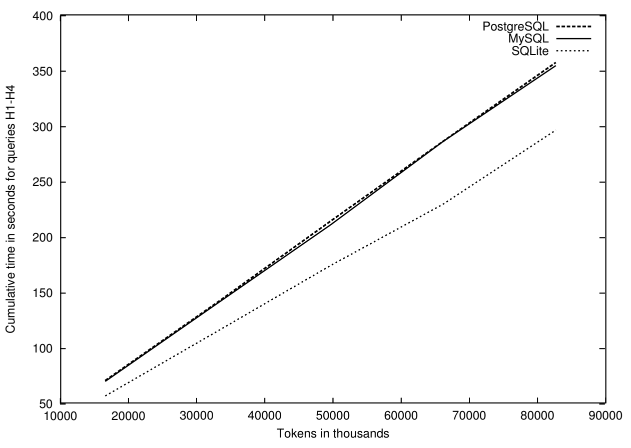

Figure 4: Times for all queries added together on the Hunglish corpus

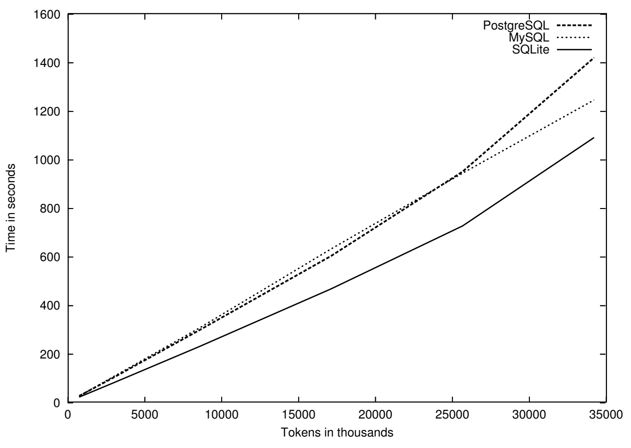

Figure 5: Times for all queries added together on the TIGER corpus

the Hunglish corpus. Fig. 5 shows the same for the queries on the TIGER Corpus. Figures 6, 7, and 8 show the times of the individual queries on the TIGER Corpus for SQLite, MySQL, and PostgreSQL respectively. The average times for each query can be seen for SQLite on the Hunglish corpus in Table 1, and for SQLite on the TIGER Corpus in Table 2. The distribution of times is similar for PostgreSQL and MySQL, and so these times are not shown as tables, only as graphs.

7. Evaluation

As can be seen from Table 2, Emdros performs extremely well on the single instance of the TIGER corpus (712×10^3 words), running the most complex query, Q8, in less than 3.5 seconds. This is typical of Emdros' performance on "small" corpora of less than a million words. For further details, please see (Petersen, 2005).
As can be seen from a comparison of Table 1 and Table 2, the query times for the Hunglish corpus are significantly lower per token queried than for the TIGER corpus. This is because of the differences in the way the EMdF databases for the two corpora have been designed: The Hunglish corpus has been gathered into far fewer RDBMS rows than the

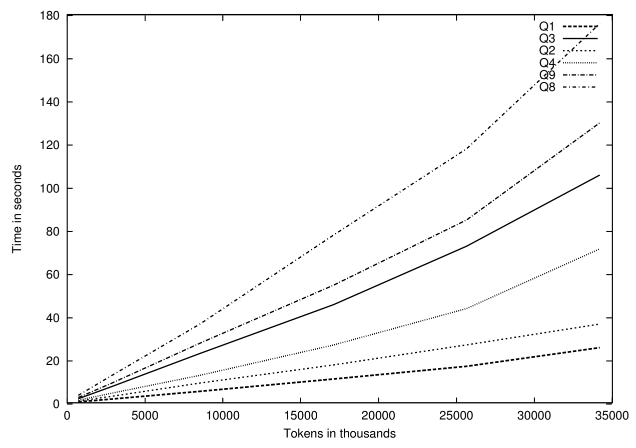

Figure 6: TIGER SQLite execution times

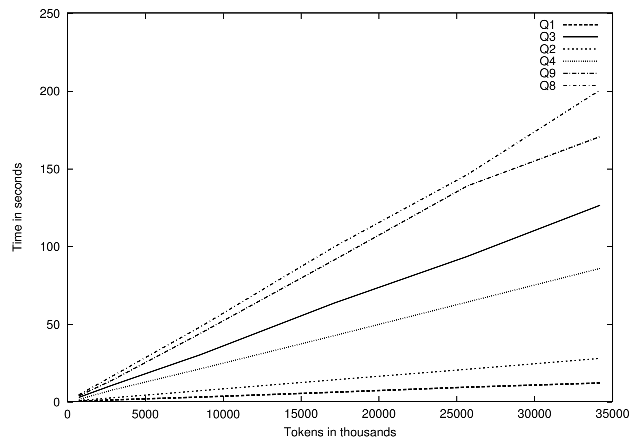

Figure 7: TIGER MySQL execution times

TIGER Corpus, in that each sentence becomes one row as is the case for the Hunglish corpus, rather than one token becoming one row as is the case for the TIGER corpus. In addition, there is no linguistic information associated with each word in the Hunglish corpus. These two factors mean that the storage overhead per token is significantly less for the Hunglish corpus. This is the reason for the dramatical difference in query times between the two corpora.

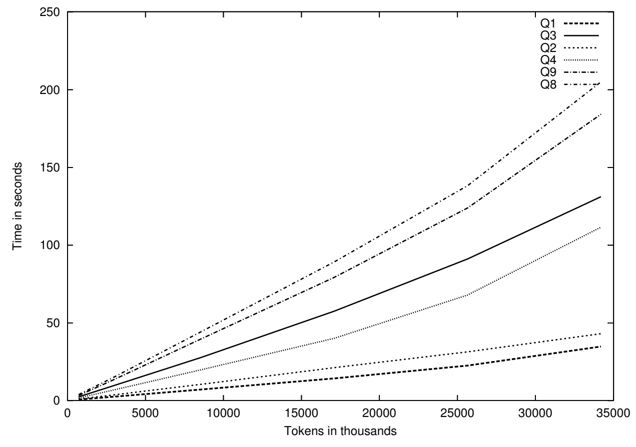

Figure 8: TIGER PostgreSQL execution times

It will be noted, however, that the TIGER corpus, because it is a treebank, supports significantly more advanced queries than the Hunglish corpus. Also, query Q1 on the TIGER corpus is only marginally more advanced than query H1 on the Hunglish corpus, in that both queries query for the existence of a single word, the only difference being that query Q1 also retrieves the structurally enclosing Root (i.e., Sentence) object. Moreover, if we extrapolate the SQLite query time for query Q1 linearly (see Table 2) up to the size of the biggest concatenation of the Hunglish corpus (82 million), we get an execution time of $25.62 \times \frac{82659}{34191} = 61.93$, which is only roughly twice the execution time of H1 (33.98).[4] Thus the added complexity of the TIGER corpus only lowers performance by a factor of roughly 2, while adding many complex query capabilities, as exemplified by query Q9.

As can be seen from Fig. 4, which shows the times of all queries added together for the Hunglish corpus, performance on the Hunglish corpus is very linear in the number of tokens queried.

The same is almost true for the TIGER corpus, as can be seen from Fig. 5, which shows the times of all queries added together for the TIGER corpus. However, here the curves suffer a bend after 25 million tokens — at least on PostgreSQL and SQLite, while MySQL stays linear even up to 34 million words. It is our estimation that tuning PostgreSQL's memory usage, and increasing the amount of RAM available to SQLite, would change this picture back to linear for these two databases, even beyond 25 million tokens queried.

As can be seen from Fig. 6, which shows the time taken for individual queries on SQLite, it is the case that the curve suffers a bend on all queries after 25 million tokens queried. The same is true for PostgreSQL, as can be seen from Fig. 8. On MySQL, however, all queries are linear even beyond 25 million, except for query Q9, which strangely shows better-than-linear performance after 25 million words, as can be seen in Fig. 7. We have no explanation for this phenomenon at this point.

It is curious that query Q8 is uniformly slower than query Q9 across the three backend databases, even though query Q8 is less complex than query Q9 in the number of query terms. This is probably because query Q8 finds more than 12.58 times the number of "hits" than query Q9[5], and so has to do more memory-house-keeping, as well as dumping more results afterwards.

8. Conclusion and further work

Corpus query systems are of great value to the Language Resources community. In this paper, we have presented our own corpus query system, called Emdros, and have described its architecture, its MQL query language, and its underlying EMdF database model. We have then shown how one can apply the EMdF database model to two kinds

[4]As Fig. 6 shows, we are not completely justified in extrapolating linearly, since query Q1 (as well as the other queries) show a small but significant non-linear bend in the curve after 25 million words queried. However, this bend is very small for query Q1.

[5]3,843,312 for Q8 vs. 305,472 for Q9 on the 34 million-word corpus.

of corpora, one being a parallel corpus (the Hunglish corpus) and the other being a treebank (the TIGER corpus).

We have then described some experiments on the two corpora, in which we have measured the execution time of Emdros against the two corpora on a number of queries. The corpora have been concatenated a number of times so as to get more data to query. This has resulted in databases of different sizes, up to 82 million words for the Hunglish corpus and up to 34 million tokens for the TIGER corpus. The execution times have been plotted as graphs, which have been shown, and selected times have been shown as tables.

We have then discussed and evaluated the results. It has been shown that execution time is linear in the number of tokens queried for the Hunglish corpus, and nearly linear for the TIGER Corpus. It has also been shown that execution times are extremely good for "small" corpora of less than a million words, while execution time remains good for "large" corpora of many millions of words.

We plan to extend Emdros in variuos ways. For example: Adding importers for more corpus formats; Adding an `AND` operator between strings of object blocks; Adding automatically generated permutations of blocks; Adding support for Kleene Star on groups of blocks rather than single blocks; Extending the underlying EMdF model to scale even better; Adding ngram support directly into the underlying EMdF model; Adding lists of strings as a feature-type; Adding caching features which would support web-based applications better; and adding a graphical management tool in addition to the existing graphical query tool.

The good execution times, coupled with a query language that is easy to read, easy to learn, and easy to understand while supporting very complex queries, makes Emdros a good choice as a tool for researchers working with linguistic corpora.

9. References

Galia Angelova, Kalina Bontcheva, Ruslan Mitkov, Nicolas Nicolov, and Nikolai Nikolov, editors. 2005. *International Conference Recent Advances in Natural Language Processing 2005, Proceedings, Borovets, Bulgaria, 21-23 September 2005*, Shoumen, Bulgaria. INCOMA Ltd. ISBN 954-91743-3-6.

Sabine Brants and Silvia Hansen. 2002. Developments in the TIGER annotation scheme and their realization in the corpus I. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002), Las Palmas, Spain, May 2002*, pages 1643–1649. ELRA, European Language Resources Association.

Thorsten Brants, Wojciech Skut, and Hans Uszkoreit. 1999. Syntactic annotation of a German newspaper corpus. In *Proceedings of the ATALA Treebank Workshop*, pages 69–76, Paris, France.

Steve Cassidy and Jonathan Harrington. 2001. Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1,2):61–77.

Crist-Jan Doedens. 1994. *Text Databases: One Database Model and Several Retrieval Languages*. Number 14 in Language and Computers. Editions Rodopi Amsterdam, Amsterdam and Atlanta, GA. ISBN 90-5183-729-1.

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

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

Wolfgang Lezius. 2002a. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Ph.D. thesis, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart, December. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4.

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

Ulrik Petersen. 2004. Emdros — a text database engine for analyzed or annotated text. In *Proceedings of COLING 2004,* 20th *International Conference on Computational Linguistics, August* 23rd *to* 27th*, 2004, Geneva*, pages 1190–1193. International Commitee on Computational Linguistics. http://emdros.org/petersen-emdros-COLING-2004.pdf.

Ulrik Petersen. 2005. Evaluating corpus query systems on functionality and speed: Tigersearch and emdros. In Angelova et al. (Angelova et al., 2005), pages 387–391. ISBN 954-91743-3-6.

Ulrik Petersen. 2006; to appear. Principles, implementation strategies, and evaluation of a corpus query system. In *Proceedings of the FSMNLP 2005 workshop*, Lecture Notes in Artifical Intelligence, Berlin, Heidelberg, New York. Springer Verlag. Accepted for publication.

Douglas L. T. Rohde. 2005. Tgrep2 user manual, version 1.15. Available online http://tedlab.mit.edu/˜dr/Tgrep2/tgrep2.pdf.

Dániel Varga, Peter Hálacsy, András Kornai, Viktor Nagy, Lázló Németh, and Viktor Trón. 2005. Parallel corpora for medium density languages. In Angelova et al. (Angelova et al., 2005), pages 590–596. ISBN 954-91743-3-6.