INDEXING METHODS FOR EFFICIENT PARSING WITH TYPED FEATURE STRUCTURE GRAMMARS

COSMIN MUNTEANU
Department of Computer Science, University of Toronto
10 King's College Rd., Toronto, M5S 3G4, Canada
E-mail: mcosmin@cs.toronto.edu

Recent years have witnessed an increase in the use of unification-based grammars, especially of typed feature structure grammars (TFSGs). A major obstacle in developing efficient parsers for unification-based grammars (UBGs) is the slow parsing time caused by the large amount of data and the complex structure used to represent grammatical entities. With the broadening coverage of such grammars, their size and complexity increases, rendering the need for improved parsing techniques more acute. Although several methods exist today that exhibits significant improvements in parsing times, most of them rely on statistical data collected during training phases. Our goal is to obtain an indexing method that produces improvements comparable to those of statistical methods, but without lengthy training processes. In this paper, we present an indexing technique based on static analysis of the grammar rules, a method that has received little attention in the last few years in computational linguistics. This method has the advantage of not requiring training phases, and, as experimental results show, it offers significant improvements of parsing times for typed feature structure (TFS) grammars.

Key words: Typed feature structures, parsing, indexing, static analysis.

1. INTRODUCTION

Developing efficient parsers is one of the long-time goals of research in natural language processing. A particular area of grammar development in strong need of improvements in parsing times is that of typed feature structure grammars (TFSGs). With respect to parsing times, much simpler grammar formalisms such as context-free grammars (CFGs) face the same problem of slow parsing time when the size of the grammar increases significantly. While TFSGs are small compared to large-scale CFGs (in terms of the number of rules), the problematic parsing time is generated by the complex structure required to represent the categories in the grammar rules. For example, in HPSGs [Pollard and Sag1994] covering the English language, one category could incorporate thousands of features (while in CFGs, the categories are atomic).

For chart parsers, one of the most time-consuming operations is the retrieval of categories from the chart. This is a look-up process: the retrieved category should match a daughter description from the grammar. The size of the typed feature structures [Carpenter1992], combined with their complex structure, result in slow unification (matching) times, which leads to slow parsing times. Therefore, while retrieving categories from the chart, failing unifications should be avoided. As mentioned in [Carpenter1995], an indexing method that reduces the number of unifications is needed. Using indexing methods for searching categories in the chart is also motivated by the successful use of indexing in the retrieval/updating process in databases.

1.1. Motivation

Most of the research aimed at improving parsing times uses statistical methods that require training. During grammar development, the time spent for the entire edit-test-debug cycle is important [Malouf et al.2000]. Therefore, a method that requires considerable time for gathering statistical data could burden the development process. Indexing methods that are time efficient for the entire grammar development cycle are a needed alternative.

Current techniques (such as quick-check, [Malouf et al.2000]) reduce the parsing times by filtering unnecessary unifications. Widely used in databases [Elmasri and Navathe2000] and automated reasoning [Ramakrishnan et al.2001], indexing presents the advantage of a more organized, yet flexible, approach. Compared to simple filtering, an indexing structure allows for searches based on multiple criteria. It also provides support for complex queries that are not limited to membership checking [Manolopoulos et al.1999]. By using indexing in parsing with UBGs, we can share the same index structure for different types of UBGs, while changing only the search criteria.

1.2. Related Work

An empirical method that addresses the efficiency issue is quick-check [Malouf et al.2000], a method that relies on statistical data collected through training. Other techniques are focused on implementation aspects [Carpenter and Qu1995], or propose approaches similar to indexing for typed feature structure (TFS) retrieval [Ninomiya et al.2002]. An automaton-based indexing for generation is proposed in [Penn and Popescu1997], while [Penn1999b] improves the efficiency by re-ordering of feature encoding. A method (similar to the one introduced in Section 3) that uses pre-compiled rule filters is presented in [Kiefer et al.1999], although the authors did not focus on the indexing potential of the static analysis of mother-daughter relations, nor present the indexing in a large experimental context.

1.3. Preliminaries – Chart Parsing with Indexing

The indexing method proposed here can be applied to any chart-based parser. We chose the EFD-based parser implemented in Prolog for illustration (An extensive presentation of EFD – empty-first-daughter – can be found in [Penn1999c] and [Penn and Munteanu2003]). EFD is a bottom-up, right-to-left parser, that needs no active edges. It uses a chart to store the passive edges. Edges are added to the chart as the result of closing (completing) grammar rules. The chart contains $n-1$ entries (where n is the number of words in the input sentence), each entry i holding edges that have their right margin at position i.

In order to close a rule (close constituents in the chart under a grammar rule), all the rules' daughters should be found in the chart as edges. Looking for a matching edge for a daughter is accomplished by attempting unifications with edges stored in the chart, resulting in many failed unifications.

General Indexing Strategy. The purpose of indexing is to reduce the amount of failed unifications when searching for an edge in the chart. Each edge (edge's category or description) in the chart has an associated index key which uniquely identifies sets of categories that can match with that edge's category. When closing a rule, the chart parsing algorithm looks up edges matching a specific daughter in the chart. Instead of visiting all edges in the chart, the daughter's index key selects a restricted number of edges for traversal, thus reducing the number of unification attempts.

Managing the Index. The passive edges added to the chart represent rules' mothers. Each time a rule is closed, that rule's mother is added to the chart according to its indexing scheme $L(\mathcal{M})$, which selects the hash entries where the mother [1] is inserted. $L(\mathcal{M})$ is a list containing the index keys of daughters that are possible candidates for a successful unification with $\mathcal{M}$. The indexing scheme is re-built only when the grammar changes, thus sparing important compiling time. Each category (daughter) is associated with a unique index key. During parsing, a specific daughter is searched for in the chart by visiting only the list of edges having appropriate keys. The index keys can be computed off-line (when daughters are indexed by their position, see Section 3) or during parsing (as in Sections 4 and 5). In our experiments, the index is represented as a hash[2], where the hash function applied to a daughter is equivalent to the daughter's index key.

[1] Through the rest of the paper, we will also use the shorter term *mother* to denote *rule's mother*. [2] Future work might also take into consideration other dynamic data structures as a support for indexing.

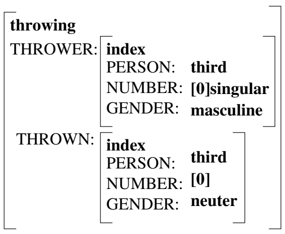

FIGURE 1. A typed feature structure. Features are written with uppercases, while types are written with bold-face lowercases. TFSs can be recursive and can have structure sharing (represented here by the tag **[0]**).

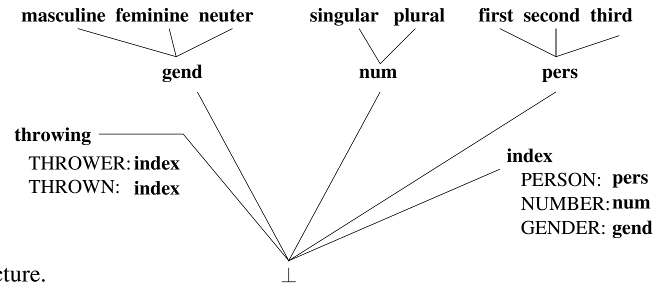

FIGURE 2. A type hierarchy. Appropriateness is specified for a type by listing the type restrictions that apply for the values of features introduced by that type.

2. TYPED-FEATURE STRUCTURE INDEXING

Typed Feature Structures are used in Computational Linguistics (and other areas such as Knowledge Representation and Automated Reasoning) to model and represent information in terms of features (attributes) and their values. In this short overview of TFS (and through the rest of the paper), the formalism and notations introduced in [Carpenter1992] will be used.

A TFS (Figure 1) consists of a type drawn from a type (inheritance) hierarchy and a set of feature/value pairs appropriate for the type. A type hierarchy (Figure 2) is an arrangement of types into sets that form a partial order with respect to the inclusion of sets. Two types are said to unify if they have a least upper bound in the type hierarchy. When designing a TFS grammar or any other application using TFSs, a type signature must be specified. The type signature is the type hierarchy together with a set of appropriateness specifications. The appropriateness specifies which types are allowed for each feature. The appropriateness also requires features to take values (i.e. all features defined in a signature are required to be defined and to have an appropriate type value in a TFS). Additionally, appropriateness, through a condition named unique feature introduction, states that every feature must be introduced by a type (an introducer) that bears that feature, and all subtypes of that introducer will bear all of its features.

Type constraints are another way of placing limitations on the type values of features. They can specify arbitrary restrictions on the types and on the features inside a TFS. For example, a type constraint can require that all structures of a certain type have their features taking type values that are more restricted than those specified by the appropriateness.

An operation of particular interest for the present paper is the unification of two TFSs, defined as a TFS that represents “neither more nor less information than is contained in the TFSs being unified” [Carpenter1992]. Computationally, unification is a very expensive process, since the TFSs to be unified have to be traversed, and the unification between individual types has to be checked.

2.1. Indexing Strategy for Typed Feature Structures

Although the amount of attempted unifications is not very large in TFSGs (usually TFSGs have significantly fewer rules than simpler grammars, such as CFGs), the unification itself is very costly. The major problem in indexing TFSGs lies in the complex nature of the categories used. This makes indexing difficult for TFSG parsers, since the extraction of an index key from each category is not a trivial process.

In this paper, we experiment with two indexing strategies. The first (introduced in Section 3) is based on a static analysis of grammar rules, where the index keys are computed by matching daughters and mothers before the parsing starts. The second method (presented in Section 4) performs a deeper exploration inside daughters and mothers-to-be-edges by defining the index key as a set of types (feature values) extracted from certain paths (indexing paths) during parsing. The decision to select feature paths as indexing paths is a combination of statistical methods and static analysis.

2.2. Experimental Resources

An initial version of the MERGE grammar was used for evaluating the performance of indexing. MERGE is the adaptation for TRALE [Meurers and Penn2002] of the English Resource Grammar [CSLI2002]. The simplified version has 13 rules with 2 daughters each, 4 unary rules, and 136 lexical entries. The type hierarchy contains 1157 types, with 144 introduced features. The features are encoded as Prolog terms according to their feature-graph colouring. For performance measurements, we used a test set containing 29 sentences of lengths from 2 to 16 words[3]. For training the statistical indexing scheme we use a corpus of 90 sentences.

Two versions of MERGE were employed during the experimental evaluation. The first version uses an extended encoding of Prolog terms that allows the representation of type constraints, while the second one simply ignores the constraints (Therefore, since many unification failures will be caused earlier by the type constraints, the times for parsing the first version are faster then for the second).

To ensure that unification is carried through internal Prolog unification, we encoded the descriptions of TFSs as Prolog terms. From the existing methods that efficiently encode TFSs ([Mellish1988], [Gerdemann1995], [Penn1999a]), we used embedded Prolog lists to represent feature structures. As shown in [Penn1999a], if the feature graph is N-colourable, the least number of argument positions in a flat encoding is N. Types were encoded using the attributed variables from SICSTus [SICS2001].

The performance was timed on a Sun Workstation with an UltraSparc v.9 processor at 440 MHz and with 1024 MB of memory. The parser was implemented in SICStus 3.8.6 for Solaris 8.

3. NON-STATISTICAL INDEXING FOR TFS

Statistical methods such as quick-check [Malouf et al.2000] have a major disadvantage if they are used during grammar development cycles. If the grammar suffers important changes, or the sentences to be parsed are not similar to those from training, the training phase must be re-run. Hence, an indexing scheme that does not need training is desirable.

The indexing scheme presented in this section[4] is computed off-line, without parsing a training corpus. The index key for each daughter is represented by its position (rule number and daughter position in the rule), therefore no time is spent during parsing for computing the index keys.

3.1. Building and Using the Index

The structure of the index can be determined at compile-time or off-line. The first step is to create the list containing the descriptions of all rules' mothers in the grammar. Then, for each mother description, a list $L(Mother) = \{(R_i, D_j) |$ daughters that can match $Mother\}$ is created, where each element of the list L represents the rule number R_i and daughter position D_j (inside rule R_i) of a category that can match with $Mother$.

For UBGs it is not possible to determine the exact list of matches between daughters and mothers, since the content of a daughter can change during parsing. However, it is possible to rule out before parsing the daughters that are incompatible (with respect to unification) with a certain $Mother$, hence $L(Mother)$ has a length between that of a "perfect" indexing scheme and that of using no index at

[3] The coverage of our version of the MERGE grammar is quite limited, therefore the test sentences are rather short (which is, however, a common characteristic of TFSGs compared to CFGs). [4] Part of this work appeared in a preliminary version of this paper as [Munteanu2003]. A slightly different TFS encoding was used, with a limited capability of representing the categories in the MERGE grammar.

all. In fact, for the 17 mothers in the MERGE grammar, the number of matching daughters statically determined before parsing ranges from 30 (the total number of daughters in the grammar) to 2. This compromise pays off with its simplicity, which is reflected in the time spent managing the index.

During run-time, each time an edge (representing a rule's mother) is added to the chart, its category *Cat* is inserted into the corresponding hash entries associated with the positions (R_i, D_j) from the list $L(Cat)$. The entry associated to the key (R_i, D_j) will contain only categories that can possibly unify with the daughter at position (R_i, D_j) in the grammar.

Using a positional index key for each daughter presents the advantage of not needing an indexing (hash) function during parsing. When a rule is extended during parsing, a matching edge for each daughter is looked up in the chart. The position of the daughter (R_i, D_j) acts as the index key, and matching edges are searched only in the list indicated by the key (R_i, D_j).

3.2. Performance

The indexing method was evaluated using two versions of the MERGE grammar. The first version (Figure 3) used the Prolog encoding described in Section 2.2, while the second (Figure 4) used an extended encoding that allows for the representation of type constraints. For both versions, indexing through static analysis outperformed the non-indexed parser with an average of 12% (the best improvement being 14%).

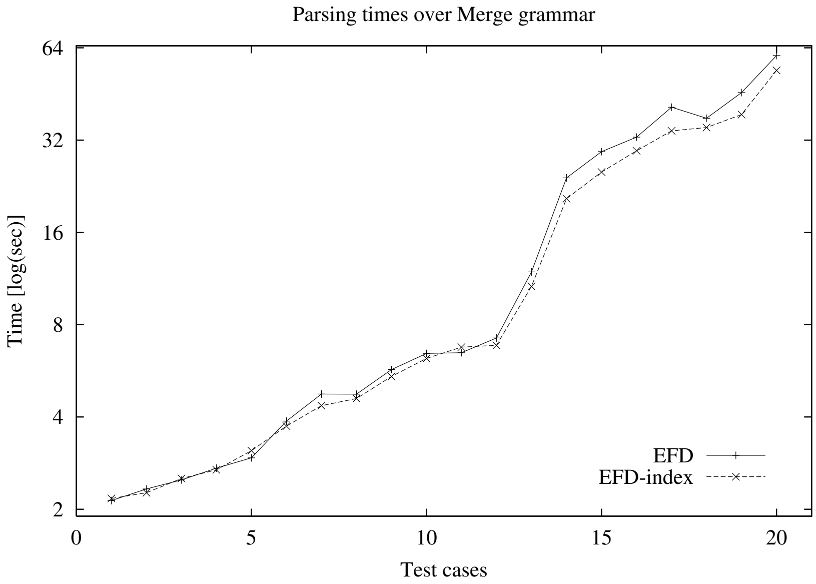

FIGURE 3. Parsing times for EFD and EFD-indexing applied to the MERGE grammar.

4. PATH INDEXING

The second indexing method presented here emphasizes the consideration that categories in UBGs have a dynamic content: feature values inside a daughter can be shared between daughters of the same rule. Therefore, the shared values inside a daughter D_i can be changed as the result of a unification between an edge in the chart and a daughter $D_j, (j = 1 \ldots i-1)$ that lies to the left of D_i inside the same grammar rule. This observation explains why the first method (presented in Section 3) does not rule out all failed unifications that occur during parsing. By extracting some of these

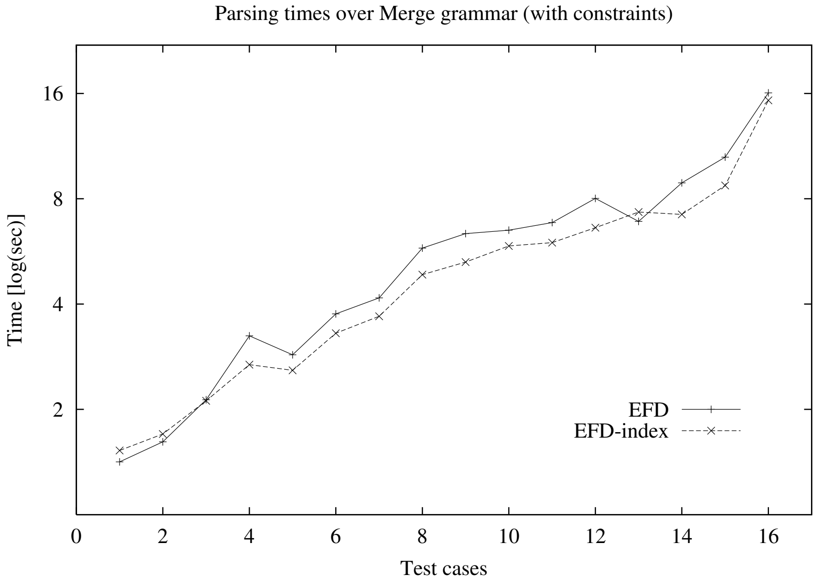

FIGURE 4. EFD and EFD-indexing parsing times for the constraint-version of MERGE.

values before the daughter D_i is matched with an edge E, and matching only these values with values of same features from E, a possible failed unification could be avoided.

4.1. Index Management

The path indexing proposed here [5] relies on data collected about failed unifications caused by paths shared between a daughter and the daughters positioned to its left inside the rule. When parsing sentences in the training corpus, the parser is modified to record, for every failed unification between two feature structures, those feature paths which cause the failure. Only the shared feature paths are candidates for selection, and each of these feature paths is assigned a score based on its probability to cause unification failure. The most probable paths for each daughter are then retained as indexing paths. For a daughter, its index key is defined by the type values at the end of its indexing paths[6]

The path indexing described above can be used without a training phase. However, during our initial experiments, the cost of extracting the values for all shared feature paths negatively affected the performance of the parser. This was avoided by limiting the index keys to the type values from a reduced set of paths. Since the indexing paths are not the same for each daughter, path indexing incorporates the same discriminating power as the non-statistical indexing presented in Section 3.

4.2. Inserting into the Chart

The indexing scheme used for adding edges to the chart during parsing is a slightly modified version of the general scheme presented in Section 1.3. Instead of adding an edge to several entries, each edge is added only once. This is done to compensate for the fact that due to the dynamic content of the index key, the edge's key has to be extracted each time an edge is added to the chart.

[5] A previous version, based entirely on statistical measurements, is presented in [Munteanu2003]. [6] This method shares its underlying principle with quick-check[Malouf et al.2000] – using type values extracted from categories to predict failed unifications. Quick-check uses statistical measurements to determine the method of extracting these values, and does not use them inside an index structure (they are employed as a filter – the *quick-check vector*). Our method uses indexing paths that statistically extract the types from a non-statistically determined set.

4.3. Retrieving Edges

The retrieval of edges from the indexed chart is accomplished in a manner similar to that described in Section 1.3. However, instead of searching matching edges for a daughter in a single entry, the chart is traversed and each edge's index key is first matched (unified) with the daughter's key.

4.4. Performance

To evaluate the path indexing method, the version of MERGE without constraints was used (future work will take into consideration the extension of path indexing to include constraints). The improvements in parsing times (Figure 5) for path indexing over those for the non-indexed parser have an average of 9%, with a maximum of 15%. Although this indexing method incorporates static analysis by having different path indexing for each daughter, it performs more slowly than the simple static analysis. Even if the number of failed unifications is further reduced by path indexing (for the simple static analysis, for a 7-word sentence with a total of 50 successful unifications, the number of failed unifications is reduced from 532 to 401, while path indexing brings it down to 373), the cost of extracting the index key while traversing the chart offsets the benefits of this method[7].

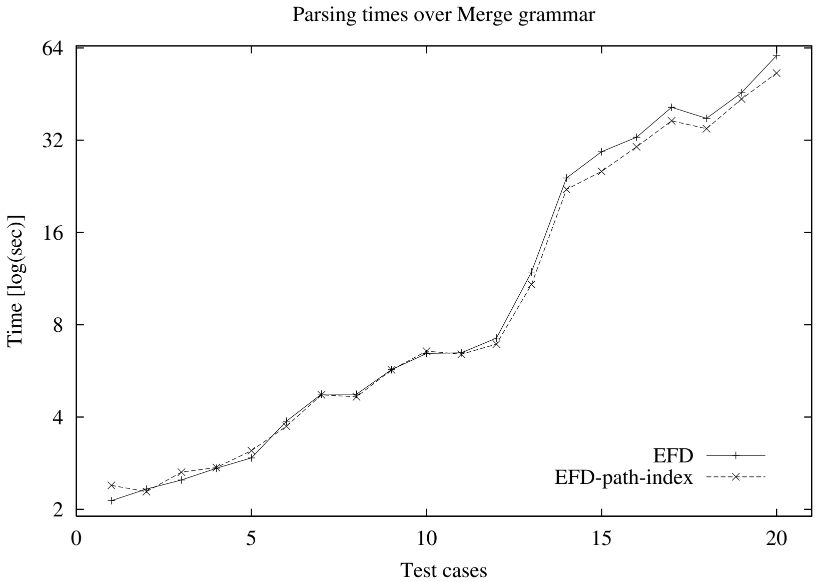

FIGURE 5. Parsing times for EFD and EFD-path-indexing applied to the MERGE grammar.

5. EXPERIMENTAL EVALUATION FOR NON-TFSGS

5.1. Indexing for Alvey Chart Parsing

The English grammar developed within the Alvey Natural Language Tools [Grover et al.1993] is a wide-coverage morphosyntactic and semantic analyzer, based on a formalism similar to that of Generalized Phrase Grammar [Gazdar et al.1985]. For our experiments, we used John Carroll's Prolog port of the Alvey English grammar.

[7] For similar reasons, quick-check was not included in this experimental evaluation. The number of failure-causing paths has to be relatively small in order to successfully employ quick-check [Malouf et al.2000]. In our experiments, this number was large, and increasing the size of the quick-check vector caused slower parsing times. Reducing the size of this vector resulted in parses where no failed unifications were avoided.

Indexing Method. The indexing scheme used for Alvey has the same general structure presented in Section 1.3. The Prolog implementation of Alvey has feature structures encoded as terms. Since there is no typing in Alvey, and the feature structures are small (compared to MERGE), the index key associated with each daughter and each edge is the functor of the Prolog term encoding the category. This is not a "perfect" indexing. However, due to the small category size (and consequently, the small unification cost), a simpler indexing scheme is preferred to one difficult to manage.

Experimental Results. The evaluation set consisted of all test sentences included in the Prolog port of Alvey. Figure 6 presents the parsing times for the longer ones (with lengths of up to 31 words). Even if the indexing scheme is very simple, on average the indexed parser performs 4% better than the non-indexed EFD parser, while for longer sentences the improvement is 7%.

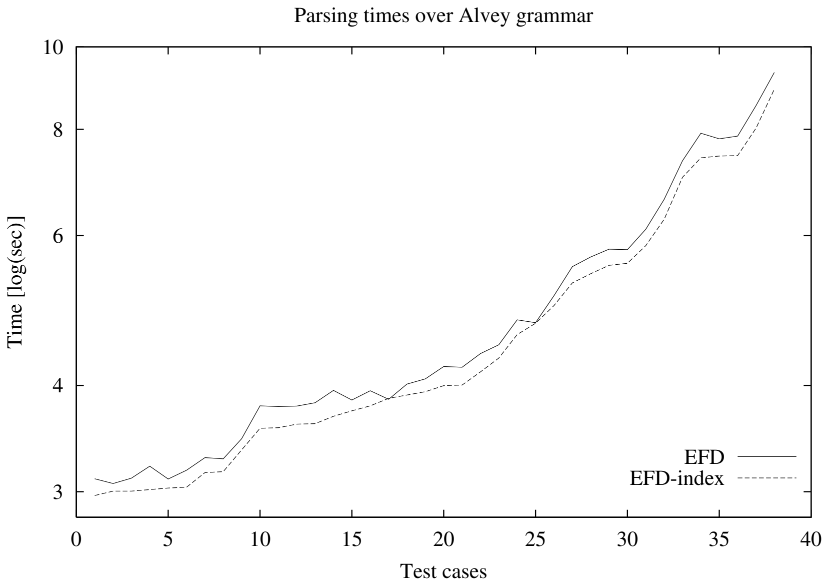

FIGURE 6. Parsing times for EFD and EFD-indexing applied to the Alvey grammar.

5.2. Indexing for CFG Chart Parsing

CFGs are not considered UBGs, because they have atomic categories, and the process of unifying atomic entities is merely a simple string matching. However, as is shown in this Section, the same indexing methods used for TFSGs can be successfully applied to CFGs, with the results demonstrating significant improvement, especially for large-scale CFGs.

Indexing Method. The index key for each daughter is the daughter's category itself. The indexing scheme $L(Mother)$ contains only the daughters that are guaranteed to match with a specific $Mother$ (thus creating a "perfect" index). This increases to 100% the ratio of successful unifications, representing a significant gain in terms of parsing times: for the smallest CFG tested (124 rules), 1870 unifications were counted, of which only 104 (5.56%) were successful; for the largest CFG (20999 rules), there were 849,068,197 unifications, with only 1,863,523 of them (0.21%) successful.

Evaluation. Nine CFGs with atomic categories were built from the Wall Street Journal (Penn Tree Bank r. 2) annotated parse trees, by constructing a rule from each sub-tree of every parse tree, and removing the duplicates. The test set contained 5 sentences of lengths between 13 and 18 words.

The number of rules varied from 124 to 20999. Figure 7 shows that even for smaller number of rules, the indexed parser outperforms the non-indexed version. Although unification costs are small for atomic CFGs, using an indexing method is well justified. The difference in improvements from Penn Tree Bank CFG to MERGE TFSG is explained by the large branching factor of the CFG rules.

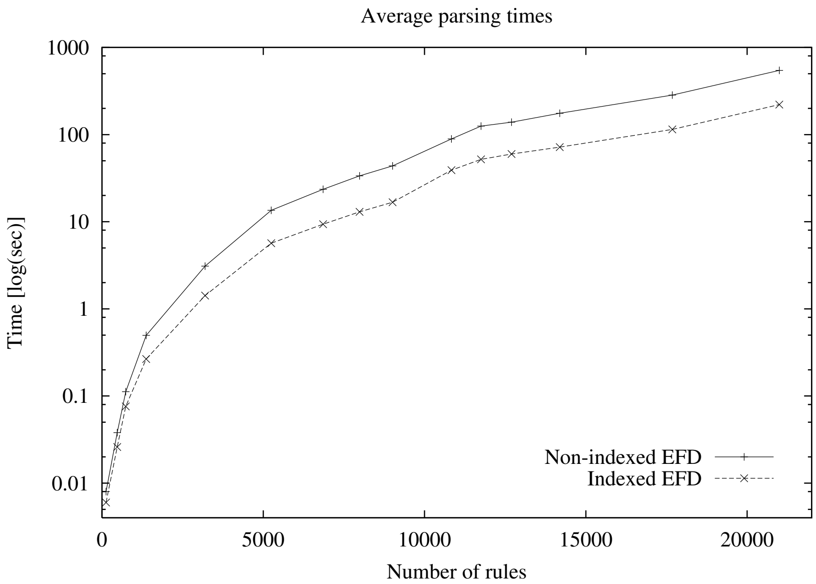

FIGURE 7. Parsing times for EFD and EFD-indexing applied to CFGs with atomic categories.

6. CONCLUSIONS

In this paper, we presented an indexing method that uses a hash to index categories during chart parsing. This method is suitable for several classes of unification-based grammars. The index keys are determined statically and are based on *a priori* analysis of grammar rules. Although a non-statistical method, indexing through the static analysis of grammar rules can be combined with methods based on statistical evidence about mother-daughter unifications. The experimental evaluation carried over several unification-based grammars demonstrates that this indexing technique improves the parsing time. A major advantage of such indexing methods is the elimination of the lengthy training processes needed by statistical methods.

7. FUTURE WORK

Future work will focus on improving the indexing techniques analyzed in this paper. In particular, a better implementation for path indexing will be designed to include types extracted from more paths into the index key. Type signature and appropriateness specification will be used instead of statistical measurements to restrict the dimension of keys in path indexing.

Other areas of investigation include a more efficient feature encoding that allows for different types of signatures and faster unification times. Currently under research is a representation/encoding of TFSs that will allow for earlier detection of unification failures.

ACKNOWLEDGEMENTS

The author wishes to thank Professor Gerald Penn for his strong encouragement and support.

REFERENCES

[Carpenter and Qu1995] B. Carpenter and Y. Qu. 1995. An abstract machine for attribute value logics. In *Proceedings of the 4th International Workshop on Parsing Technologies*, Prague, The Czech Republic.

[Carpenter1992] B. Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press.

[Carpenter1995] B. Carpenter. 1995. Compiling CFG parsers in Prolog. http://www.colloquial.com/carp/Publications.

[CSLI2002] CSLI. 2002. CSLI Lingo. http://lingo.stanford.edu/csli.

[Elmasri and Navathe2000] R. Elmasri and S. Navathe. 2000. *Fundamentals of database systems*. Addison-Wesley.

[Gazdar et al.1985] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.

[Gerdemann1995] D. Gerdemann. 1995. Term encoding of typed feature structures. In *Proceedings of the Fourth International Workshop on Parsing Technologies*.

[Grover et al.1993] C. Grover, J. Carroll, and E. Briscoe. 1993. The alvey natural language tools grammar (4th release). Technical Report 284, Cambridge University, UK.

[Kiefer et al.1999] B. Kiefer, H.U. Krieger, J. Carroll, and R. Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the ACL*.

[Malouf et al.2000] R. Malouf, J. Carrol, and A. Copestake. 2000. Efficient feature structure operations without compilation. *Natural Language Engineering Journal*, 1(1).

[Manolopoulos et al.1999] Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras. 1999. *Advanced Database Indexing*. Kluwer Academin Publishers.

[Mellish1988] C. Mellish. 1988. Implementing systemic classification by unification. *Computational Linguistics*, 14(1).

[Meurers and Penn2002] D. Meurers and G. Penn, 2002. *Trale Milca Environment v. 2.1.4*. http://ling.ohio-state.edu/˜dm.

[Munteanu2003] C. Munteanu. 2003. Indexing methods for efficient parsing. In *Proceedings of the Student Research Workshop at the Joint 3rd International Conference on Human Language Technology / 3rd Meeting of the North American Chapter of the ACL*, Edmonton, Canada.

[Ninomiya et al.2002] T. Ninomiya, T. Makino, and J. Tsujii. 2002. An indexing scheme for typed feature structures. In *Proceedings of the 19th International Conference on Computational Linguistics*.

[Penn and Munteanu2003] G. Penn and C. Munteanu. 2003. A tabulation-based parsing method that reduces copying. In *Proceedings of the 41st Annual Meeting of the ACL*, Sapporo, Japan.

[Penn and Popescu1997] G. Penn and O. Popescu. 1997. Head-driven generation and indexing in ALE. In *ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering*.

[Penn1999a] G. Penn. 1999a. An optimised Prolog encoding of typed feature structures. In *Arbeitspapiere des SFB 340*, number 138.

[Penn1999b] G. Penn. 1999b. Optimising don't-care non-determinism with statistical information. In *Arbeitspapiere des SFB 340*, number 140.

[Penn1999c] G. Penn. 1999c. A parsing algorithm to reduce copying in Prolog. In *Arbeitspapiere des SFB 340*, number 137.

[Pollard and Sag1994] C. Pollard and I. Sag. 1994. *Head-driven Phrase Structure Grammar*. The University of Chicago Press.

[Ramakrishnan et al.2001] I.V. Ramakrishnan, R. Sekar, and A. Voronkov. 2001. Term indexing. In *Handbook of Automated Reasoning*, volume II, chapter 26. Elsevier Science.

[SICS2001] SICS. 2001. SICStus Prolog. http://www.sics.se/sicstus.