Indexing Methods for Efficient Parsing

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

Abstract

This paper presents recent developments of an indexing technique aimed at improving parsing times. Although several methods exist today that serve this purpose, most of them rely on statistical data collected during lengthy training phases. Our goal is to obtain a reliable method that exhibits an optimal efficiency/cost ratio, without lengthy training processes. We focus here on static analysis of the grammar, a method that has unworthily received less attention in the last few years in computational linguistics. The paper is organized as follows: first, the parsing and indexing problem are introduced, followed by a description of the general indexing strategy for chart parsing; second, a detailed overview and performance analysis of the indexing technique used for typed-feature structure grammars is presented; finally, conclusions and future work are outlined.

1 Introduction

One of the major obstacles in developing efficient parsers for natural language grammars is the slow parsing time. As recent years witnessed an increase in the use of unification-based grammars (UBG) or large-scale context-free grammars, the need for improving the recognition (parsing) times is more stringent.

Our approach is based on the observation that CFG or UBG parsing has to deal with large amount of data and/or data with complex structure, which leads to slower processing times. This problem is similar to that of the retrieval/updating process in databases, and for this area, it was solved by indexing. This similarity leads to the assumption that the same solution could be applied to parsing.

For chart-parsing techniques, 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. For large-scale CFGs, one position in the chart could contain a large amount of categories; for UBGs, this amount is usually smaller, but the unification process itself is very costly. Thus, as mentioned in (Carpenter, 1995), an indexing method that reduces the number of unifications is much needed.

1.1 Our Goal

Most of the research aimed at improving parsing times uses statistical methods that require training. As mentioned in (Malouf et al., 2000), during grammar development, the time spent for the entire edit-test-debug cycle is important, therefore a method needing considerable time for gathering statistical data could burden the development process. Our goal is to find better indexing methods that are time-efficient for the entire grammar development cycle.

Current techniques (such as quick-check, (Malouf et al., 2000)) reduce the parsing times by means of filtering unnecessary unifications. Using an index presents the advantage of a more organized, yet flexible, approach. Indexing methods are widely used in databases (Elmasri and Navathe, 2000) and automated reasoning (Ramakrishnan et al., 2001).

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 implementational aspects (Wintner and Francez, 1999), or propose approaches similar to indexing for typed-feature structures (TFS) retrieval (Ninomiya et al., 2002). An automaton-based indexing for generation is proposed in (Penn and Popescu, 1997), while (Penn, 1999b) improves the efficiency by re-

ordering of feature encoding. A method (similar to the one introduced in Section 7) 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.

2 Preliminaries

The indexing method proposed here can be applied to any chart-based parser. We chose for illustration the EFD parser implemented in Prolog (an extensive presentation of EFD can be found in (Penn, 1999c)). 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 (n is the number of words in the input sentence), each entry i holding edges that have their right margin at position i.

2.1 TFS Encoding

To ensure that unification is carried through internal Prolog unification, we encoded descriptions as Prolog terms for parsing TFSGs. From the existing methods that efficiently encode TFS into Prolog terms ((Mellish, 1988), (Gerdemann, 1995), (Penn, 1999a)), we used embedded Prolog lists to represent feature structures. As shown in (Penn, 1999a), 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 (SICS, 2001).

3 Chart Parsing with Indexing

In order to close a 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.

3.1 General Indexing Strategy

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

3.2 Index Building

The passive edges added to the chart represent rules' mothers. Each time a rule is closed, its mother is added to the chart according to the indexing scheme. The indexing scheme selects the hash entries where the mother [1] is inserted. For each mother $\mathcal{M}$, the indexing scheme is a list containing the index keys of daughters that are possible candidates to a successful unification with $\mathcal{M}$. The indexing scheme is re-built only when the grammar changes, thus sparing important compiling time.

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. Each entry in the chart has a hash associated with it. When passive edges are added to the chart, they are inserted into one or several hash entries. For an edge representing a mother $\mathcal{M}$, the list of hash entries where it will be added is given by the indexing scheme for $\mathcal{M}$.

3.3 Using the Index

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 that have the appropriate key, thus reducing the time needed for traversing the chart. The index keys can be computed offline (when daughters are indexed by their position, see Section 7) or during parsing (as in Sections 4, 6).

4 Indexing for CFG Chart Parsing

4.1 Indexing Method

The first indexing method presented in this paper is aimed at improving the parsing times for CFGs. The index key for each daughter is daughter's category itself. In order to find the edges that match a specific daughter, the search take place only in the hash entry associated with that daughter's category. This increases to 100% the ratio of successful unifications (Table 1 illustrates the significance of this gain by presenting the successful unification rate for non-indexing parser).

Number of rules	Successful unifications	Failed unifications	Success rate (%)
124	104	1,766	5.56
473	968	51,216	1.85
736	2,904	189,528	1.51
1369	7,152	714,202	0.99
3196	25,416	3,574,138	0.71

Table 1: Successful unification rate for non-indexing parser (for the CFGs from Section 4.2.)

4.2 Experiments for CFG indexing

Several experiments were carried to determine the actual run-times of the EFD and indexed EFD parsers for CFGs.

[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.

Nine CFGs with atomic categories were built from the Wall Street Journal (Penn Tree Bank release 2) annotated parse trees, by constructing a rule from each sub-tree of every parse tree, and removing the duplicates.

For all experiments we chose a test set of 5 sentences (with lengths of 15, 14, 15, 13, and 18 words) such that each grammar will parse successfully all sentences and each word has only one lexical use in all 5 parses. The number of rules varied from 124 to 3196.

Figure 1 shows that even for a smaller number of rules, the indexed parser outperforms the non-indexed version. As the number of rules increases, the need for indexing becomes more stringent. Although unification costs are small for atomic CFGs, using an indexing method is well justified.

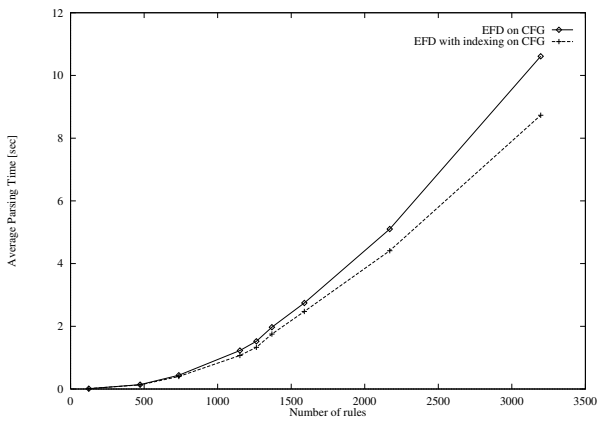

Figure 1: Parsing times for EFD and EFD-indexing applied to CFGs with atomic categories

The performance measurements for all CFG experiments (as well as for TFSG experiments presented later) were carried 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.

5 Typed-Feature Structure Indexing

Compared to CFG parsing, for TFSGs the amount of attempted unifications is much smaller (usually UBGs have fewer rules than CFGs), but the unification itself is very costly. Again, indexing could be the key to efficient parsing by reducing the number of unifications while retrieving categories from the chart.

The major difference between indexing for CFGs and for TFSGs lies in the nature of the categories used: CFGs are mostly associated with the use of atomic categories, while TFSGs employs complex-structure categories (typed-feature structures). This difference makes indexing more difficult for typed-feature structure parsers, since the extraction of an index key from each category is not a trivial process anymore. The following sections describe the solution chosen for indexing typed-feature structure parsers.

5.1 Statistical and Non-Statistical Indexing

In our quest for improving the parsing times for TFSGs, we took two different approaches to indexing. The first approach uses statistical measurements carried on a corpus of training sentences to determine the most appropriate indexing scheme. The second approach relies on *a priori* analysis of the grammar rules, and no training is required.

5.2 Experimental Resources

For both statistical and non-statistical indexing schemes, a simplified version of the MERGE grammar was used. MERGE is the adaptation for TRALE (Meurers and Penn, 2002) of the English Resource Grammar (CSLI, 2002). The simplified version has 13 rules with 2 daughters each and 4 unary rules, and 136 lexical entries. The type hierarchy contains 1157 types, with 144 features introduced. The features are encoded as Prolog terms (lists of length 13) according to their feature-graph colouring.

For performance measurements, we used a test set containing 40 sentences of lengths from 2 to 9 words [3] (5 sentences for each length). For training the statistical indexing scheme we use an additional corpus of 60 sentences.

6 Statistical Indexing for TFS

6.1 Path Indexing

Our statistical approach to indexing has its roots in the automaton-based indexing from (Penn and Popescu, 1997), used in generation, but adapted to indexed edge retrieval. The solution we chose is similar to the quick-check vector presented in (Malouf et al., 2000). When parsing sentences in the training corpus, the parser is modified in order to record, for each unification between two feature structures that failed, the feature path that caused the unification failure. The path causing most of the unification failures across all training corpus will be refered to as the indexing path. The type value at the end of the indexing path is used as an index key.

6.1.1 Index Building

The indexing scheme used for adding edges to the chart during parsing is a slightly modified version of the general scheme presented in Section 3.2. Each edge is associated with an index key. For our statistical indexing, we used the type at the end of an edge's indexing path as the index key for that edge.

[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).

An edge describing a rule's mother $\mathcal{M}$ is added to the indexed chart at all positions indicated by the keys in the list $L(\mathcal{M})$. Since types are used as index keys, this list is defined as $L(\mathcal{M}) = \{t \mid t \sqcup k_M\} \cup \{\bot\}$, where k_M is the index key for $\mathcal{M}$, $\bot$ is the unique most general type, and $\sqcup$ is the type unification.

6.1.2 Using the Index

The retrieval of edges from the indexed chart is accomplished as described in Section 3.3. The index key for each daughter is the type value at the end of the indexing path. In case the indexed path is not specified for a given daughter, the type $\bot$ is used for the key. Hence, searching for a matching edge in the entry described by $\bot$ is identical to using a non-indexed chart parsing.

6.2 Path Indexing with Quick Check

The path indexing scheme presented above makes use of a single feature path that causes most of the failed unifications over a corpus of sentences. Since each of the paths causing unification failures represents relatively small percentages of the total failures (the first two paths account for only 18.6% and 17.2%, respectively), we decided to use the first two paths in a mixed approach: the type at the end of the first path was still used as an index key, while the traversal of edges in a hash entry was accompanied by a quick-check along the second path.

6.3 Performance

Four parsers were tested: the non-indexed EFD parser, the path-indexed parser (using one path), the non-indexed EFD parser using quick-check, and the combination of path indexing and quick-checking. The results are presented in Table 2.

Words per sentence	Non-indexed EFD	Path-indexed EFD	EFD with quick-check	Path-indexed + quick-check EFD
2	0.9	0.9	1.0	0.9
3	4.0	4.4	3.9	4.4
4	15.5	16.4	14.9	16.0
5	46.2	46.9	44.2	46.5
6	103.8	102.5	98.1	100.8
7	184.8	186.9	176.0	180.7
8	311.4	313.5	301.0	295.3
9	594.6	562.7	554.7	551.7

Table 2: Average parsing times [msec] for statistical indexing, using the converted MERGE grammar.

Although the number of unifications dropped almost to 18% for the combination of path indexing and quick-check, the difference in parsing times is not as significant. This is due to the costs of maintaining the index: simple path indexing is constantly slower than quick-check. Path indexing combined with quick-check outperforms quick-check for sentences longer than 7 words.

7 Non-Statistical Indexing for TFS

Statistical indexing and quick-check 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 has to be re-run. Hence, an indexing scheme that does not need training is needed.

The indexing scheme presented in this section reduces the number of hash entries used, thus reducing the cost of manipulating the index. The index key for each daughter is represented by its position (rule number and daughter position in the rule), therefore the time spent in computing the index key during parsing is practically eliminated.

7.1 Index Building

The structure of the index is determined at compile-time (or can be constructed off-line and saved for further uses if parsing is done with the same grammar). 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 $L(Mother) = \{(R_i, D_j) \mid \text{daughters that can match } Mother\}$ 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 CFGs, the list $L(Mother)$ would contain only the daughters that are guaranteed to match with a specific $Mother$ (thus creating a "perfect" index). For UBGs, it is not possible to determine the exact list of matches, 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 the list $L(Mother)$ has a length between that of a "perfect" indexing scheme and that of using no index at all. Indeed, 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 by its simplicity, 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. Compared to the path indexing scheme (Section 6.1) where the number of entries could reach 1157 (total number of types), in this case the number is limited to 30 (total number of daughters).

7.2 Using the Index

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, each daughter is looked up in the chart for a matching edge. 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).

8 Using Statistical Measures to Improve Non-Statistical Indexing for TFS

Although the statistical and non-statistical indexing techniques can be merged in several ways into a single method, the cost of maintaining a complicated indexing scheme overshadows the benefits. An experiment that combined all indexing techniques presented in this paper produced parsing times almost four times longer than the slowest non-statistical indexing. However, as shown in the following paragraphs, the statistical information about paths causing unification failures can be used to improve the efficiency of indexing.

8.1 Encoding Re-ordering

The unification of feature structures is accomplished by means of Prolog term unifications, as described in Section 2.1. This means that the unification of features encoded on the first position in their lists will take place before the unification of features that are encoded at the end of the lists.

During the training phase presented in Section 6, we observed that features causing most of the unification failures are not placed at the beginning of the list in their encodings. Therefore, we re-arranged the placement of encoded features according to their probability to cause unification failures.

9 Performance

Similar to the experiments carried for statistical indexing, the experimental resources presented in Section 5.2 were also used for the indexing method introduced in Section 7. Figure 2 and Figure 3 present the comparison between the original EFD parser and the same parser with indexing (results from statistical indexing experiments are also presented here in order to illustrate the differences between all methods). For sentences having more than 4 words, the indexed parser outperforms both the EFD parser and the best statistical indexing method. Figures 2 and 3 also present the parsing times for the new feature encoding described in Section 8.1.

10 Conclusions

In this paper, we presented an indexing method that uses a hash to index categories during chart parsing. This method works for both context-free grammars with atomic categories, and typed feature structure grammars. The index keys rely on compatibility relations between rules' mothers and daughters statically determined before parsing. Other techniques (like statistic-based quick-check tests or feature re-ordering) can be combined in order to improve the parsing time. Overall, static analysis of grammar rules that index daughters by their position proved to be an efficient method that eliminates the training needed by statistical indexing techniques. Statistical data can improve this method especially by means of feature re-ordering.

11 Future Work

Future work will focus on improving the indexing techniques analyzed in this paper. Possible areas of investigation are substitution tree indexing (Graf, 1995) for non-statistical methods, or restructuring decision trees (Utgoff et al., 1997), while trying to maintain index operation costs at a minimum. Performance profiling combined with software and database engineering techniques will be used to determine the optimum trade-off between indexing efficiency and implementation cost.

Since non-statistical indexing proved to be an efficient solution, our main focus will be on improving the static analysis. Type signature and appropriateness specification will be used to identify both the paths prone to cause unification failures and the paths that lead to successful unifications. Empirical techniques (such as unifying partial representations for TFS ordered by their probability of causing unification failures) will be used, along with a more efficient feature encoding that allows for earlier detection of unification failures.

Acknowledgements

The author wishes to thank Professor Gerald Penn for his restless support during this work, and the anonymous reviewers for their valuable comments.

References

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

CSLI. 2002. CSLI Lingo. http://lingo.stanford.edu/csli.

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

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

P. Graf. 1995. Substitution tree indexing. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*.

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*.

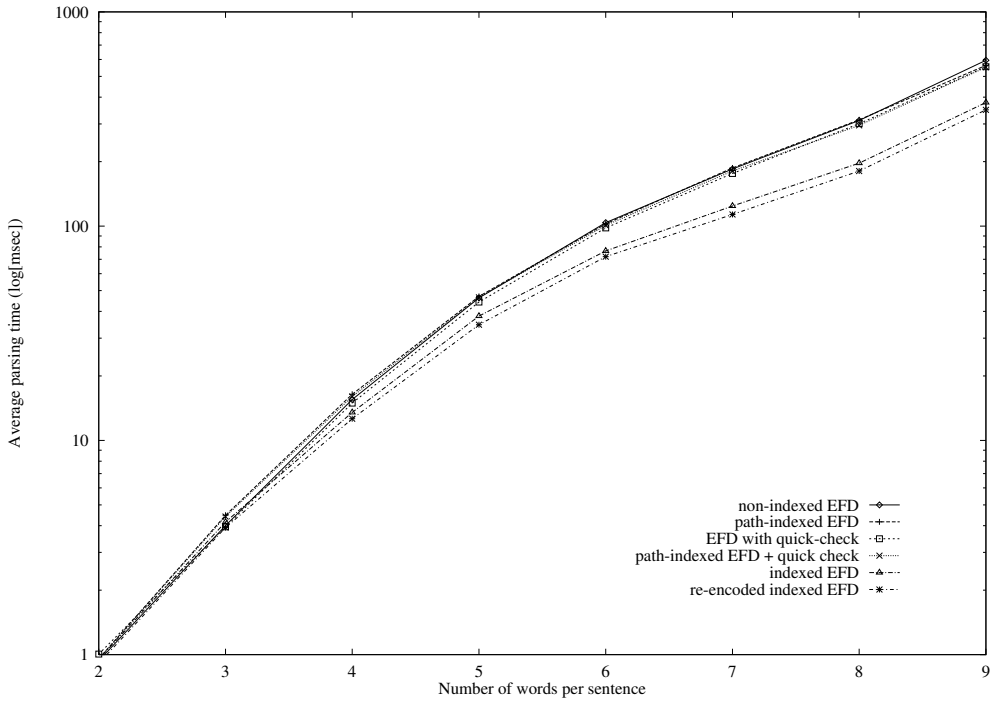

Figure 2: Average parsing times for all indexing methods, using the converted MERGE grammar.

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

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

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

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*.

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*.

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

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

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

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

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

P. Utgoff, N. Berkman, and J. Clouse. 1997. Decision tree induction based on efficient tree restructuring. *Machine Learning Journal*, 29(1).

S. Wintner and N. Francez. 1999. Efficient implementation of unification-based grammars. *Journal of Language and Computation*, 1(1).

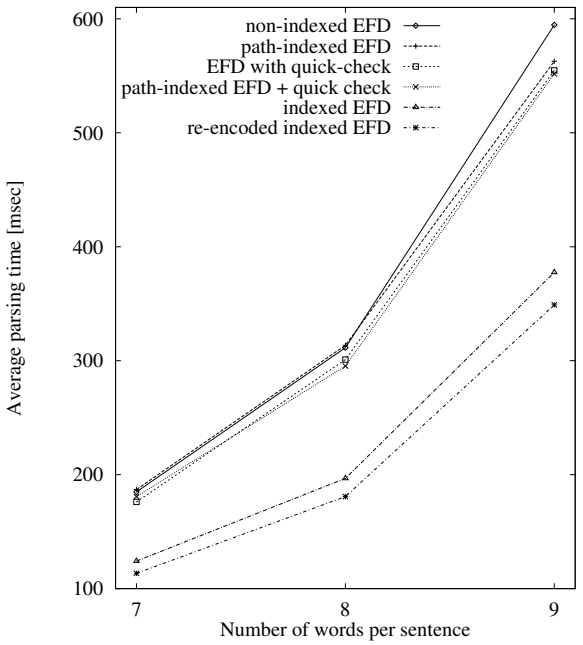

Figure 3: Average parsing times for all indexing methods, using the converted MERGE grammar – detailed view.