EFFICIENCY IN LARGE-SCALE PARSING SYSTEMS

PROCEEDINGS

a workshop held at Coling 2000
the 18th International Conference on Computational Linguistics
Luxembourg, 5 August 2000

Organisation

Organisers

John Carroll

Robert C. Moore

Stephan Oepen

Programme Committee

To obtain additional copies of these proceedings contact:

Dr. John Carroll
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH, UK

john.carroll@cogs.susx.ac.uk

Contents

Invited Talk

Papers

Efficient Large-Scale Parsing — a Survey

John Carroll
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH, UK
johnca@cogs.susx.ac.uk

Stephan Oepen
Computational Linguistics
Saarland University
66041 Saarbrücken, Germany
oe@coli.uni-sb.de

Abstract

We survey work on the empirical assessment and comparison of the efficiency of large-scale parsing systems. We focus on (1) grammars and data used to assess parser efficiency; (2) methods and tools for empirical assessment of parser efficiency; and (3) comparisons of the efficiency of different large-scale parsing systems.

1 Background

Interest in large-scale, grammar-based parsing has recently seen a large increase, in response to the complexities of language-based application tasks such as speech-to-speech translation, like rules, each category containing on average around 30 nodes. Associated with the grammar is a test suite, originally written by the grammarian to monitor coverage dursome sort of analysis; and degree of ambiguity of a grammar in terms of the 'parse base', the expected number of parses for a given input length (Carroll, Briscoe, & Sanfilippo, 1998). Work on quantifying parse correctness has used various measures of structural consistency with respect to constituent structure annotations of a corpus (e.g. exact match, crossing brackets, tree similarity, and others—see Black et al., 1991, Black, Garside, & Leech, 1993, Grisham, Macleod, & Sterling, 1992, and Briscoe & Carroll, 1993); recently, more general schemes have been advocated that deploy functor – argument (dependency) relations as an abstraction over different phrase structure analyses that a parser may assign (Lin, 1995; Lehmann et al., 1996; Carroll et al., 1998). The Penn Treebank and the SUSANNE corpus are well-established resources for the evaluation of parser accuracy.

In a sharp contrast, there is little existing methodology, let alone established reference data or software tools, for the evaluation and contrastive comparison of parser efficiency. Although most grammar development environments and large-scale parsing systems supply facilities to batch-process a test corpus and record the results produced by the system, these are typically restricted to processing a flat, unstructured input file (listing test sentences, one per line) and outputting a small number of processing results to a log file.[2] Additionally, no metrics exist that allow the comparison of parser efficiency across different grammars and sets of reference data. We therefore note a striking methodological and technological deficit in the area of precise and systematic assessment of grammar and parser behaviour.

Recently though, a new methodology, termed *competence & performance profiling* (Oepen & Flickinger, 1998; Oepen & Carroll, 2000), has been proposed that aims to fill this gap. Profiles are rich, precise, and structured snapshots of parser competence (coverage and correctness) and performance (efficiency), where the production, maintenance, and inspection of profiles is supported by a specialised software package called [incr tsdb()].[3] Profiles are stored in a relational database that serves as the basis for flexible report generation, visualisation, data analysis via basic descriptive statistics, and of course comparison to other profiles. The [incr tsdb()] package has so far been interfaced with some eight unification-based grammar development and/or parsing systems, and has served as the 'clearing house' in a multi-site collaborative effort on parser benchmarking (Flickinger, Oepen, Tsujii, & Uszkoreit, 2000), resulting in useful feedback to all participating groups.

4 Efficiency Comparisons

Many parsing algorithms suitable for NL grammars have been proposed over the years, their proponents often arguing that the number of computational steps are minimised with respect to alternative, competing algorithms. However, such arguments can only be made in the case of very closely related algorithms; qualitatively different computations can only reliably be compared empirically. So, for example, generalised LR parsing was put forward as an improvement over Earley-style parsing (Tomita, 1987), with a justification made by running implementations of the two types of parser on a medium-sized CF grammar with attribute-value augmentations. However, comparisons of this type have to be done with care. The coding of different strategies must use exactly equivalent techniques, and to be able to make any general claims, the grammar(s) used must be large enough to fully stress the algorithms. In particular, with grammars admitting less ambiguity, parse time is likely to increase more slowly with increasing input length, and also with smaller grammars rule application can be constrained tightly with relatively simple predictive techniques. In fact, a more recent evaluation (Moore, 2000) using a number of large-scale CF grammars has shown conclusively that generalised LR parsing is less efficient than certain left-corner parsing strate-

[2]Some (Meta-)Systems like PLEUK (Calder, 1993) and HDrug (van Noord & Bouma, 1997) that facilitate the exploration of multiple descriptive formalisms and processing strategies come with slightly more sophisticated benchmarking facilities and visualisation tools. However, they still largely operate on monolithic, unannotated input data sets, restrict accounting of system results to a small number of parameters (e.g. number of analyses, overall processing time, memory consumption, possibly the total number of chart edges), and only offer a limited, predefined choice of analysis techniques.

[3]See '`http://www.coli.uni-sb.de/itsdb/`' for the (draft) [incr tsdb()] user manual, pronunciation guidelines, and instructions on obtaining and installing the package.

gies.

Moore and Dowding (1991) document a process of refining a unification-based (purely bottom-up) CKY parser (forming part of a speech understanding system) by incorporating top-down information to prevent it hypothesising constituents bottom-up that could not form part of a complete analysis, given the portions of rules already partially instantiated. An important step was reducing the spurious prediction of gaps by means of grammar transformations. The refinement process was guided throughout by empirical measurements of parser throughput on a test corpus.

Improvements in efficiency can be gained by specialising a general-purpose grammar to a particular corpus. Samuelsson and Rayner (1991) describe a machine learning technique that is applied to the CLE grammar to produce a version of the grammar that parses ATIS corpus sentences much faster than the original grammar. In general there are more rules in the specialised grammar than in the original, but they are more specific and can thus be applied more efficiently.

Maxwell and Kaplan (1993) investigate the interaction between parsing with the CF backbone component of a grammar and the resolution of functional constraints, using a precursor of the English ParGram grammar. A number of parsing strategies are evaluated, in combination with two different unifiers, on a small set of test sentences. There is a wide gap between the best and worst performing technique; the differences can be justified intuitively, but not with any formal analyses of computational complexity.

Carroll (1994) discusses the throughput of three quite distinct unification-based parsing algorithms running with the ANLT grammar. T

A number of other empirically-driven research efforts into efficient parsing are described in the same journal special issue (Flickinger et al., 2000). These include grammar-writing techniques for improved parser efficiency, new efficient algorithms for feature structure operations, fast pre-unification filtering, and techniques for the extraction of CF grammars and abstract machine compilation for HPSGs.

5 Conclusions

Recent interest in large-scale, grammar-based parsing (in response to the demands of complex language-based application tasks) has led to renewed efforts to develop wide-coverage, general-purpose grammars, and associated research efforts into efficient parsing with these grammars. Some initial progress has been made towards precise empirical assessment of parser efficiency. However, more work is needed on methods, standard reference grammars and test data to facilitate improved comparability.

Acknowledgements

The first author is supported by a UK EPSRC Advanced Fellowship, and the second by the *Deutsche Forschungsgemeinschaft* Collaborative Research Division *Resource-Adaptive Cognitive Processes* (SFB 378) project B4 (PERFORM).

References

Alshawi, H. (Ed.). (1992). *The Core Language Engine.* Cambridge, MA: MIT Press.

Black, E., Abney, S., Flickenger, D. P., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., & Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of E

Lehmann, S., Oepen, S., Regnier-Prost, S., Netter, K., Lux, V., Klein, J., Falkedal, K., Fouvry, F., Estival, D., Dauphin, E., Compagnion, H., Baur, J., Balkan, L., & Arnold, D. (1996). TSNLP — Test Suites for Natural Language Processing. In *Proceedings of the 16th International Conference on Computational Linguistics* (pp. 711 – 716). Kopenhagen, Denmark.

Lin, D. (1995). A dependency-based method for evaluating broad-coverage parsers. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 1420 – 1425). Montreal, Canada.

Maxwell III, J. T., & Kaplan, R. M. (1993). The interface between phrasal and functional constraints. *Computational Linguistics*, *19 (4)*, 571 – 590.

Moore, R. (2000). Improved left-corner chart parsing for large context-free grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies* (pp. 171–182). Trento, Italy.

Moore, R., & Dowding, J. (1991). Efficient bottom-up parsing. In *DARPA Speech and Natural Language Workshop* (pp. 200–203). Asilomar, CA.

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics*, *23 (3)*, 425 – 456.

van Noord, G., & Bouma, G. (1997). Hdrug. A flexible and extendible development environment for natural language processing. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering* (pp. 91 – 98). Madrid, Spain.

Oepen, S., & Carroll, J. (2000). Performance profiling for parser engineering. *Natural Language Engineering*, *6 (1) (Special Issue on Efficient Processing with HPSG)*, 81 – 97.

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language*, *12 (4) (Special Issue on Evaluation)*, 411 – 436.

Oepen, S., Netter, K., & Klein, J. (1997). TSNLP — Test Suites for Natural Language Processing. In J. Nerbonne (Ed.), *Linguistic Databases* (pp. 13 – 36). Stanford, CA: CSLI Publications.

Samuelsson, C., & Rayner, M. (1991). Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 609 – 615). Sydney, Australia.

Sarkar, A. (2000). Practical experiments in parsing using tree adjoining grammars. In *5th international workshop on tree adjoining grammars and related formalisms* (pp. 193 – 198). Paris France.

Sleator, D., & Temperley, D. (1993). Parsing English with a link grammar. In *Proceedings of the 3rd International Workshop on Parsing Technologies* (pp. 277–292). Tilburg, The Netherlands.

Tomita, M. (1987). An efficient augmented-context-free parsing algorithm. *Computational Linguistics*, *13 (1)*, 31–46.

XTAG Group (1995). *A lexicalized tree adjoining grammar for english* (Tech. Rep. No. IRCS Report 95-03). The Institute for Research in Cognitive Science, University of Pennsylvania.

even with exact CF representation. Since this eliminates Kiefer and Krieger’s annotation with values of *relevant paths*, large rule numbers are more tolerable.

Our parsing speed-ups are comparable with Kentaro Torisawa’s speed-ups of between 47 and 4 in C++ with array of a *rest*

3 Modification of ALE rules

Figure 1 is the *head-subject-complement* schema in the ALE formalism. Figure 2 shows sections of the single Prolog clause containing 57 goals in 61 lines including the head, produced when ALE co number in the *comps* list of the sub-constituent unifying with the head daughter *D1*. Goal 61 creates the edge of the new phrase and corresponds to the mother *M*. Goal 20, and lines 46 to 49 forming one goal, are invocations of ALE procedures, corresponding to *P1*, and *P3* to *P6*. Apart fro

The extra argument 7 of rulejcb is a list of sub-constituent TFSs after constraint application. *Z-bot* is the new TFS. *Edge_countA* is an initial count of 1 o

Figure 5: Application of Head-Complement Schema to an Edge Sequence

5 Treatment of Semantic Structures

In a constituent, the co-indexing of paths in a copy of the semantic sub-structure of a semantic head was explained in (7) to (10). Fi where each p_k argument corresponds to a 3-tuple, ordered as *Indn* nodes are encountered in a TFS traversal. *Path_no* fields appear in the same order in an asserted clause r_n(*cat, Paths*) where *cat* is T_l allocated to the lexeme. During constituent construction in CFG parsing, *Daught_no* of the sub-constituent is known, and *Rulenam* complement is constrained, so these verbs differ from verbs that take np as a subject. A linguistic reason is that the semantic structure of a verb depends on its word morphology as in Figure 5, whilst in an np this dependency applies only to RESTR and not to INDEX.

An automatic mechanism to generate our

Time as a Measure of Parsing Efficiency

Robert C. Moore
Microsoft Research
One Microsoft Way
Redmond, Washington 98052, USA
bobmoore@microsoft.com

Abstract

Charniak and his colleagues have proposed implementation-independent metrics as a way of comparing the efficiency of parsing algorithms implemented on different platforms, in different languages, and with different degrees of "incidental optimization". We argue that there are easily immaginable circumstances in which their proposed metrics would mask significant differences in efficiency; we point out that their data do not, in fact, support the usability of such metrics for comparing the efficiency of different algorithms; and we analyze data for a

have non-zero probability, nor are they even guaranteed to explore all analyses that might turn out to have the highest probability.

- They both prune partial analyses on the basis of a figure of merit that can be viewed as based on an heuristic estimate of the expected probability that the full model would assign to extensions of a given partial analysis.

Within this framework, let us consider some of the ways that the number of events considered could fail to correlate with parse time in an essential way; that is, not based on what Roark and Charniak refer to as "incidental optimizations". First, the full probability models might take vastly different amounts of time to comficiency. (Arguably, this would be the same as the edges-popped-of-the-agenda metric, when parser is allowed to run to exhaustion.)

To evaluate the suitability of using total number of edges in the chart as an efficiency metric, we will present a selection of results from our CFG parsing experiments, including edge statistics not previously published. Results for five parsing algorithms on three different grammars are included. The parsing algorithms consist of two variants of left-corner parsing (LC_1+BUPM and LC_2+BUPM), an Earley/Graham-Harrison-Ruzzo parser (E/GHR), a Cocke-Kasami-Younger parser (CKY), and a generalized LR parser without look-ahead (GRL(0)). (The identifiers for these parsers are the ones used in our earlier report.) It should be mentioned that all these parsers represent the best of several implementations of the general approach, and all parsers are implemented using similar techniques and data structures wherever possible. Furthermore, all algorithms are implemented in the same language (Perl5) on the same platform (Windows 2000, 550 MHz Pentium III). Thus we believe that the performance differences are genuinely representative of inherent differences in the algorithms, and not just irrelevant implementation details.

The grammars used are independently motivated by analyses of natural-language corpora or actual applications of natural language processing. The CT grammar was compiled into a CFG from a task-specific unification grammar written for CommandTalk (Moore et al., 1997), a spoken-language interface to a military simulation system. The ATIS grammar was extracted from an internally generated treebank of the DARPA ATIS3 training sentences. The PT grammar is Charniak's PCFG grammar extracted from the Penn Treebank, with the probabilities omitted. The most significant variation among the grammars is the degree of ambiguity of the test sets associated with each grammar. The CT test set has 5.4 parses/sentence with the CT grammar, the ATIS test set has 940 parses/sentence with the ATIS grammar, and the PT test set has 7.2×10^{27} parses/sentence with the PT grammar.

Table 1 shows the results of applying these five parsers to the three grammars and their associated test sets. The first column gives the average number of chart edges per sentence, including both complete and incomplete edges (where incomplete edges are generated). For the GLR(0) parser, this is equivalent to the number of edges in the graph-structured stack used by most implementations of GLR parsing. The second column gives the average number of seconds per sentence to parse exhaustively. This includes only time to populate the chart, and does not include time to extract parses. The final column compares the second column to the first, to derive an average number of milliseconds per chart edge. The more constant this number is across the different parsers and grammars, the better total edges in the chart will be as a measure of parser efficiency.

If we look at the last column in detail, we see that total number of chart edges generated does have some crude validity as a measure of parsing efficiency; since the majority of the test cases fall around 0.1 milliseconds per edge. However, the variation is fairly large. The two left-corner parsers make a particularly interesting comparison, because they differ only in a single detail, and produce exactly the same edges. In these parsers incomplete edges are subjected to two tests before being added to the chart. The mother of an incomplete edge has to be a possible left corner of the next daughter required by some previous incomplete edge at the appropriate position in the input; furthermore, the next daughter of the incomplete edge being tested has to have the next token in the input as a possible left corner. These tests are independent, so they can be performed in either order. In LC_1+BUPM the check on the mother is performed first, and in LC_2+BUPM the check on the next daughter is performed first. These results show that performing the check on the mother first is 14% to 68% slower than performing the check on the next daughter first. This is a substantial difference that cannot be detected looking only at the edges added to the chart.

There are several places in the data where the numbers of chart edges strongly, but incorrectly, predict which of two parsers should be faster on a given grammar. For example, the LC_1+BUPM parser generates only about half as many edges as the E/GHR parser with the ATIS grammar, but is nevertheless 35% slower.

Grammar	Parser	Edges/sent	Sec/sent	msec/edge
CT	LC_1+BUPM	165.3	0.0219	0.132
	LC_2+BUPM	165.3	0.0191	0.116
	E/GHR	283.0	0.0448	0.158
	CKY	1598.2	0.1540	0.096
	GLR(0)	159.0	0.0214	0.135
ATIS	LC_1+BUPM	673.4	0.119	0.177
	LC_2+BUPM	673.4	0.071	0.105
	E/GHR	1276.6	0.088	0.069
	CKY	537.3	0.078	0.145
	GLR(0)	1282.5	0.143	0.112
PT	LC_1+BUPM	6675.4	1.14	0.171
	LC_2+BUPM	6675.4	0.90	0.135
	E/GHR	11143.9	0.92	0.083
	CKY	5785.6	1.70	0.294
	GLR(0)			

cult; but for a given class, this approach should make cross platform comparisons straightforward. For example, for CFGs, a particular variant of CKY could be chosen, and implemented as efficiently as possible in C, Lisp, Prolog, Perl(!), and any other language considered relevant. The source code for implementations would need to be provided, so that the claim to be the best possible implementation of the algorithm in the language could be examined, and improvements made, without changing the

Confidence	

Measuring efficiency in high-accuracy, broad-coverage statistical parsing*

Brian Roark **Eugene Charniak**
Brown Laboratory for Linguistic Information Processing (BLLIP), and
Cognitive and Linguistic Sciences Computer Science
Box 1978 Box 1910
Brown University Brown University
Providence, RI 02912 Providence, RI 02912
`brian-roark@brown.edu` `ec@cs.brown.edu`

Abstract

Very little attention has been paid to the comparison of efficiency between high accuracy statistical parsers. This paper proposes one machine-independent metric that is general enough to allow comparisons across very different parsing architectures. This metric, which we call "events considered", measures the number of "events", however they are defined for a particular parser, for which a probability must be cal cu

competitor measures, such as time or total heap operations, which can be improved through optimization techniques that do not change the search space. This is not to say that these techniques do not have a great deal of value; simply that, for comparisons between approaches to statistical parsing, the implementations of which may or may not have carried out the to be conditioned, $e_1 \ldots e_n$ the n conditioning events used in the model, and $\hat{P}$ the empirically observed conditional probability. Then the following is a recursive definition of the interpolated probability:

$$P(e_0|e_1 \ldots e_n) = \lambda_n(e_1 \ldots e_n)\hat{P}(e_0|e_1 \ldots e_n) + (1-\lambda_n(e_1 \ldots e_n))P(e_0|e_1 \ldots e_{n-1})$$

This has been shown to be very effective in circumstances where sparse data requires smoothing to avoid assigning a probability of zero to a large number of possible events that happen not to have been observed in the training data with the n conditioning events.

Using such a model[2], the time to calculate a particular conditional probability can be significant. There are a variety of techniques that can be used to speed this up, such as precompilation or caching. These techniques can have a fairly large effect on the time of computation, but they contribute little to a comparison between pruning techniques or issues of search. More generally, optimization and lack of it is something that can obscure algorithm similarities or differences, over and above differences in machine or platform. Researchers whose interest lies in improving parser accuracy might not care to improve the efficiency once it reaches an acceptable level. This should not bar us from trying to compare their techniques with regards to efficiency.

Another such example contrasts our metric with one that measures total heap operations. Depending on the pruning method, mpthrit r st n i that h t betw ible ev t l h . T t t 6l o
tryi mizad l n s
p th t an s ab dt t 6 sn t t pots p p u6 p s t ymmigt eo s pn ts

section 23: 2416 sentences of length ≤ 100 Average length: 23.46 words/sentence			
Times past first parse	Avg. Prec/Rec	Events Considered†	Time in seconds†
21	89.7	212,014	26.7
13	89.6	107,221	14.0
7.5	89.1	48,606	6.7
2.5	86.8	9,621	1.5
2	85.6	6,826	1.1

†per sentence

Table 1: Results from the EC parser at different initial parameter values

constituents proposed by the parser. Recall is the number of correct constituents divided by the number of constituents in the actual parse. Labelled precision and recall counts only non-part-of-speech non-terminal constituents. The two numbers are generally quite close, and are averaged to give a single composite score.

2.1 EC parser

The EC parser first prunes the search space by building a chart containing only the most likely edges. Each new edge is assigned a figure-of-merit (FOM) and pushed onto a heap. The FOM is the product of the probability of the constituent given the simple PCFG and the boundary statistics. Edges that are popped from the heap are put into the chart, and standard chart building occurs, with new edges being pushed onto the heap. This process continues until a complete parse is found; hence this is a best-first approach. Of course, the chart building does not necessarily need to stop when the first parse is found; it can continue until some stopping criterion is met. The criterion that was used in the trials that will be reported here is a multiple of the number of edges that were present in the chart when the first parse was found. Thus, if the parameter is 1, the parser stops when the first parse is found; if the parameter is 10, the parser stops when the number of edges in the chart is ten times the number that were in the chart when [illegible]

section 23: 2416 sentences of length ≤ 100 Average length: 23.46 words/sentence				
Base Beam Factor	Avg. Prec/Rec	Events Considered†	Time in seconds†	Pct. failed
10^{-12}	85.9	265,509	7.6	1.3
10^{-11}	85.7	164,127	4.3	1.7
10^{-10}	85.3	100,439	2.7	2.2
10^{-8}	84.3	36,861	0.9	3.8
10^{-6}	81.8	13,512	0.4	7.1

†per sentence

Table 2: Results from the BR parser at different initial parameter values

cessful" analyses are discarded. This is a beam-search, and the criterion by which it is judged that "enough" analyses have succeeded can be either narrow (i.e. stopping early) or wide (i.e. stopping late). The unpruned parse with the highest probability that successfully covers the entire input string is evaluated for accuracy.

The beam parameter in the trials that will be reported here, is called the base beam factor, and it works as follows. Let β be the base beam factor, and let $\tilde{p}$ be the probability of be t þ t þ t o s þ s m t is þ t t 6 h t

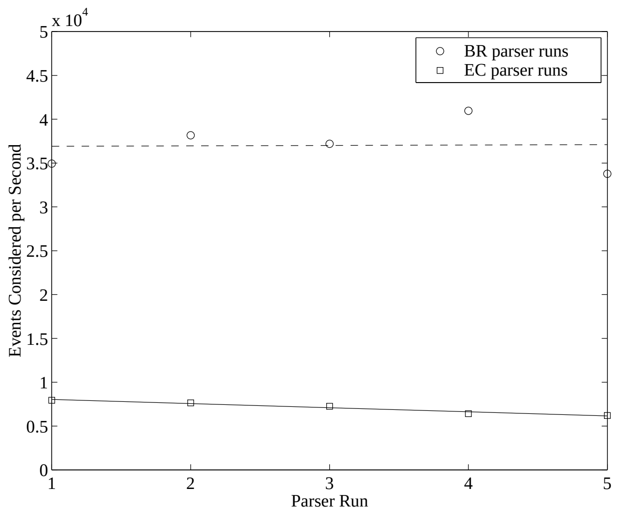


Figure 1: Events Considered per Second for each parser run, with a linear fit

processor, nor how fast each individual processor was, time is a particularly poor point of comparison.

In order for our metric to be useful, however, it should be highly correlated with time. Figure 1 shows the number of events considered divided by the total parse time for each of the five runs reported for each parser. While there is some noise between each of the runs, this ratio is relatively constant across the runs, as shown by the linear fit, indicating a very high correlation between the number of events considered and the total time. Figure 2 plots the edges considered versu

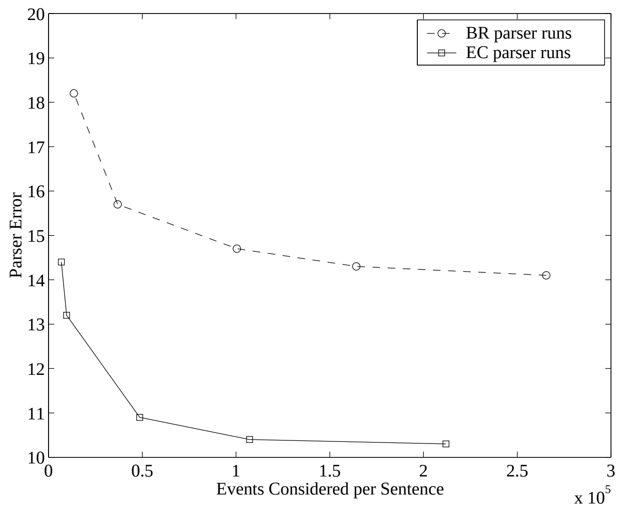


Figure 3: Reduction in parser error as the number of events considered increases

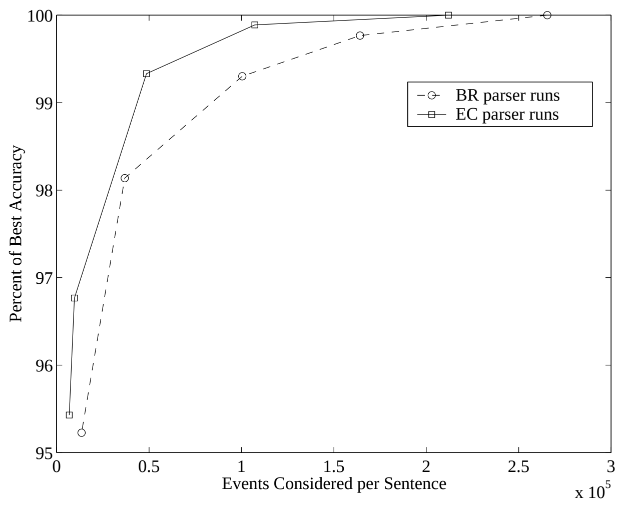


Figure 4: Convergence to highest accuracy parse as number of events considered increase

stituents once for every parse within which they occur. Particularly useless constituents will be thrown out once by the EC parser, but perhaps many times by the BR parser.

This difference in efficiency is tangible, but it is relatively small. What would be problematic in this domain would be orders of magnitude differences, which we don't get here.

4 Conclusion

We have presented in this paper a very general, machine- and implementation-independent metric that can be used to compare the efficiency of quite different statistical parsers. To illustrate its usefulness, we compared the performance of two parsers that follow different strategies in arriving at their parses, and which on the surface would appear to be very difficult to compare with respect to efficiency. Despite this, the two algorithms seem to require a fairly similar number of events considered to squeeze the most accuracy out of their respective models. Furthermore, the decrease in events considered in both cases was accompanied by a more-or-less proportional decrease in time. This data confirmed our intuitions that the two algorithms are roughly similar in terms of efficiency. It also lends support to consideration of this metric as a legitimate, machine and implementation independent measure of statistical parser efficiency.

In practice, the scores on this measure could be reported alongside of the standard PARSEVAL accuracy measures (Black et al., 1991), as an indicator of the amount of work required to arrive at the parse. What is this likely to mean to researchers in high accuracy, broad-coverage statistical parsing? Unlike accuracy measures, whose fluctuations of a few tenths of percent are attended to with interest, such an efficiency score is likely to be attended to only if there is an order of magnitude difference. On the other hand, if two parsers have very similar performance in accuracy, the relative efficiency of one over the other may recommend its use.

When can this metric be used to compare parsers? We would contend that it can be used whenever measures such as precision and recall can be used, i.e. same training and testing corpora. If the parser is working in an entirely different search space, such as with a dependency grammar, or when the training or testing portions of the corpus are different, s t t c

References

Black, E., S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *DARPA Speech and Natural Language Workshop*, pages 306–311.

Blaheta, D. and E. Charniak. 1999. Automatic compensation for parser figure-of-merit flaws. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 513–518.

Caraballo, S. and E. Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Charniak, E. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Menlo Park. AAAI Press/MIT Press.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1* e h s R *D* R 4 *W 1* e *S*i e *mp* λ R

Some Experiments on Indicators of Parsing Complexity for Lexicalized Grammars

Anoop Sarkar, Fei Xia and Aravind Joshi
Dept. of Computer and Information Science
University of Pennsylvania
200 South 33rd Street,
Philadelphia, PA 19104-6389, USA
{anoop,fxia,joshi}@linc.cis.upenn.edu

Abstract

In this paper, we identify syntactic lexical ambiguity and sentence complexity as factors that contribute to parsing complexity in fully lexicalized grammar formalisms such as Lexicalized Tree Adjoining Grammars. We also report on experiments that explore the effects of these factors on parsing complexity. We discuss how these constraints can be exploited in improving efficiency of parsers for such grammar formalisms.

1 Introduction

The time taken by a parser to produce derivations for input sentences is typically associated with the length of those sentences. The longer the sentence, the more time the parser is expected to take. How-
[illegible]
affected by several factors. A common experience is that parsing algorithms differ in the number of
[illegible]
paper, we explore some of these constraints from the perspective of lexicalized grammars and explore how these constraints might be exploited to improve parser efficiency.

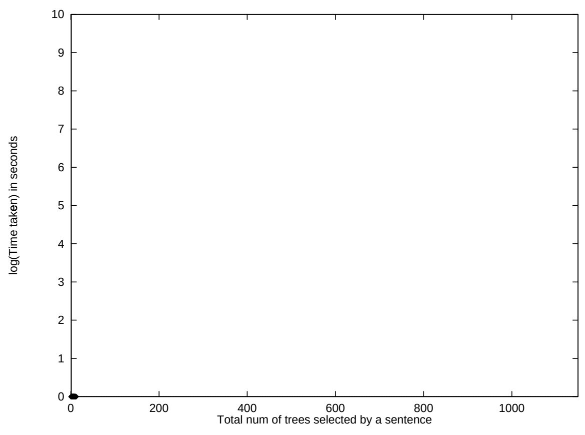
We concentrate on the problem of parsing using *fully* lexicalized grammars by looking at parsers for Lexicalized Tree Adjoining Grammar (LTAG). By a fully lexicalized grammar we mean a grammar in which there are one or more syntactic structures associated with each lexical item. In the case of LTAG each structure is a tree (or, in general, a directed acyclic graph). For each structure there is an explicit structural slot for each of the arguments of the lexical item. The various advantages of defining a lexicalized grammar formalism in this way are discussed in (Joshi and Schabes, 1991).

An example LTAG is shown in Figure 1. To parse the sentence *Ms. Haag plays Elianti* the parser has to combine the trees selected by each word in the sentence by using the operations of substitution and adjunction (the two composition operations in LTAG) producing a valid derivation for the sen-
tence. [illegible] a dif for

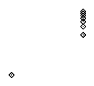
Notice that as a consequence 98 0 Td(Aeof)Tj -21570 Td((3.0399 0 Td(ef)Tj 8.153teru2o0 Td(of)Tj Tj -197.159 -c6

NP u
NNP n ◇
NNP m ◇
NP n ⁕

Treebank


log(Time taken) in seconds
Total num of trees selected by a sentence
0
1
2
3
4
5
6
7
8
9
10
0
200
400
600
800
1000

each other. In the context of lexicalized tree-adjoining grammar (and in other lexical frameworks, perhaps with some modifications) the complexity of syntactic and semantic processing is related to the number of predicate-argument structures being computed for a given sentence.



Large Scale Parsing of Czech

Pavel Smrž and **Aleš Horák**
Faculty of Informatics, Masaryk University Brno
Botanick´

generates the possible rewritings of the non-terminal `N`. The resulting terms are then subject to standard constraints and intersegment insertion. In some cases, one needs to force a certain constituent to be the first non-terminal on the right hand side. The construct `first(N)` ensures that `N` is firmly tied to the beginning and can neither be preceded by an intersegment nor any other construct. In the above example, the `relclause` is transformed to CF rules starting with `relprongr` followed by the right hand sides of the non-terminal `clause` with possible intersegments filled in.

In the current version, we have added two generative constructs and the possibility to define rule templates to simplify the creation and maintenance of the grammar. The first construct is formed by a set of `%list_*` expressions, which automatically produce new rules for a list of the given non-terminals either simply concatenated or separated by comma and co-ordinative conjunctions:

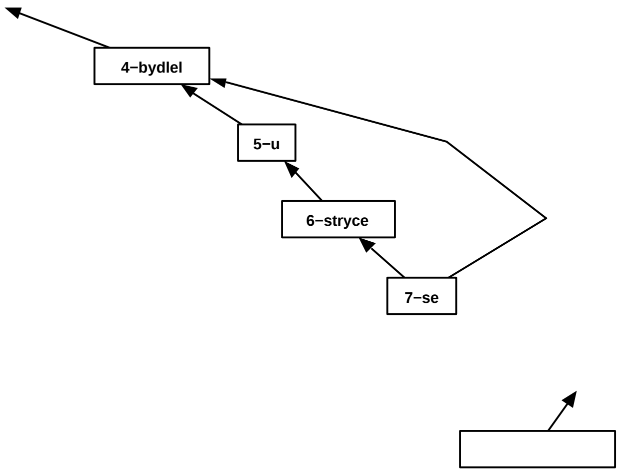
```
/* (nesmim) zapomenout udelat -
   to forget to do */
%list_nocoord vi_list
vi_list -> VI

%list_nocoord_case_number_gender modif
/* velky cerveny -
   big red */
modif -> adjp

/* krute a drsne -
   cruelly and roughly */
%list_coord adv_list
adv_list -> ADV

%list_coord_case_number_gender np
/* krasny pes -
   beautiful dog */
np -> left_modif np
...
```

The endings `*_case`, `*_number_gender` and `*_case_number_gender` denote the kinds of agreement between [illegible] clause.

is therefore $2k$, where k is the number of edges in the resulting chart.

The number of chart edges that are involved in the appropriate output derivation structure

Sent #	# of words	G2		G3		# edges
		# edges	time	# edges		


4−bydlel
5−u
6−stryce
7−se

Demos

Cross-Platform, Cross-Grammar Comparison — Can it be Done?

Ulrich Callmeier and **Stephan Oepen**
Saarland University
Computational Linguistics
{uc | oe}@coli.uni-sb.de

(see 'http://www.coli.uni-sb.de/itsdb/')

Abstract

This software demonstration reviews recent improvements in comparing large-scale unification-based parsing systems, both across different platforms and multiple grammars. Over the past few years significant progress was accomplished in efficient processing with wide-coverage HPSG grammars. A large number of engineering improvements in current systems were achieved through collaboration of multiple research centers and mutual exchange of experience, encoding techniques, algorithms, and pieces of software.

We argue for an approach to grammar and system engineering that makes systematic experimentation and the precise empirical study of system properties a focal point in development. Adapting the profiling metaphor familiar from software engineering to constraint-based grammars and parsers enables developers to maintain an accurate record of system evolution, identify grammar and system deficiencies quickly, and compare to earlier versions, among analytically varied configurations, or between different systems. We demonstrate a suite of integrated software packages facilitating this approach, which are publicly available both separately and together.

The [incr tsdb()] profiling environment (Oepen & Carroll, 2000) integrates empirical assessment and systematic progress evaluation into the development cycle for grammars and processing systems; it enables developers to obtain an accurate snapshot of current system behaviour (a profile) with minimal effort. Profiles can then be analysed and visualized at variable granularity, reflecting various aspects of system competence and performance, and compared to earlier results. Since the [incr tsdb()] package has been integrated with some eight processing platforms by now, it has greatly facilitated cross-fertilization between various research groups and implementations.

PET is a platform for experimentation with processing techniques and the implementation of efficient processors for unification-based grammars (Callmeier, 2000). It synthesizes a range of techniques for efficient processing from earlier systems into a modular C++ implementation, supplying building blocks (such as various unifiers) from which a large number of experimental setups can be configured. A parser built from PET components can be used as a time- and memory-efficient run-time system for grammars developed in the LKB system distributed by CSLI Stanford (Copestake & Flickinger, 2000). In daily grammar development it allows frequent, rapid regression tests.

We emphasize in this demonstration the crucial importance of experimental system comparison, eclectic engineering, and incremental optimization. Only through the careful analysis of a large number of interacting system parameters can one establish reliable points of comparison across different parsers and multiple grammars simultaneously.

References

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 99 – 108.

Copestake, A., & Flickinger, D. (2000). An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second Linguistic Resources and Evaluation onference* (pp. 591 – 600). Athens, Greece.

Oepen, S., & Carroll, J. (2000). Performance profiling for parser engineering. *Natural Language Engineering, 6 (1) (Special Issue on Efficient Processing with HPSG)*, 81 – 97.

Tools for Large-Scale Parser Development

Natural Language Processing Group
Microsoft Research
One Microsoft Way
Redmond WA 98052 USA

1. Introduction

We demonstrate the tool set available to linguistic developers in our NLP lab, with a particular emphasis on the tools for incremental regression testing and creation of regression suites. These tools are currently under use in the daily development of broad-coverage language analysis systems for 7 languages (Chinese, English, French, German, Japanese, Korean and Spanish). The system is modular, with the parsing engine and debugging environments shared by all languages. Linguistic rules are written in a proprietary language (called *G*) whose features are uniquely suited to linguistic tasks (Heidorn, in press). The engine underlying the system, as well as the user interface for linguistic developers, is unicode-enabled thus supporting both European and non-Indo-European languages.

2. Tools for regression testing

The purpose of this class of tools is to build regression suites, which is a collection of what we call *master files*. The master files take the form of stored output trees, and keep a record of the state of development atatpor

lp

tey