The selection of a parsing strategy for an on-line machine translation system in a sublanguage domain. A new practical comparison.

Patrick Shann
University of Geneva (ISSCO) & University of Zurich[1], Switzerland

1. Introduction

The paper reports the results of a practical comparison of different parsing strategies. The research was carried out in the context of a larger project for the development of a machine translation (MT) system for translating avalanche forecast bulletins from German to French. The design of the MT system requires controlled input and no post-editing of the translated texts. The parsing experiment had as a goal to select the most suitable parsing strategy for a parser that allows the composition of the sentences in on-line fashion with mouse and windowing[2]. In order to guarantee correct translation, the input system accepts only words and sentences that are known by their grammar and dictionary and it refuses wrong input. To minimize input errors, the user can select the possible next words with the mouse from different windows, which display the choices at a particular point in the sentence. The sentences are parsed word by word from left to right so that wrong input is detected immediately. After each word, the input device has to predict, with the help of the parser, all the words that can possibly continue the sentence that is being made. For our type of on-line parser, time is critical. The interface window has to be refreshed immediately after each word chosen by the user.

When we looked for a suitable parser, no comparison existed between Tomita's extended LR parser and enhanced chart parsers (top-down filter, rule compiling and lookahead) using different strategies (CKY, LC, BI[3]) apart from Tomita's own comparison with the Earley parser (TD). Furthermore, practical tests (Wirén 1987) are normally performed by using only simple phrase structure grammars and by measuring pure parse time. In our experiment we were interested in real time performance (what is seen by a user). Since the grammar type can heavily influence the overall processing efficiency, we chose to base our experience on three grammar types in the paradigm of context-free parsing (monadic, simple features and unification). Our parsing experiment is a continuation of the work of J.Slocum (1981a) and M.Tomita (1985) on parsing algorithms and parsing strategies. The emphasis of the research lies on the real-world performance of the parsers in connection with different grammar types rather than on the theoretical space and time complexity of the parsers.

2. Description of the parsers

In our experiment, we have compared the Tomita parser with four chart-parsers[4] that have different rule-invocation strategies. In this section we will introduce the different parsing strategies and the improvements that can be made, i.e. top-down filtering, lookahead and rule compilation.

2.1. Chart parsers

Our four chart parsers can be distinguished in the way they define the two basic operations *Combine* and *Propose*[5]. Combine is the procedure that builds new edges in the chart by combining existing ones, Propose is the rule invocation strategy that predicts new edges on the basis of the grammar. In the next chapter we define the basic algorithms. The improvements of the chart parsers are described in the following chapters on top-down filtering, lookahead and rule compilation.

2.1.1. Four different chart parsers: TD, LC, CKY, BI

Let G be a context-free grammar with S as start symbol. We will represent *terminal symbols* by lowercase letters: a, b, c; *nonterminals* by capitals: A, B, C; strings of *terminals or nonterminals*

[1] This research has been supported by a grant from the University of Zurich.

[2] A system with a similar input facility is reported by H.R.Tennant (1983).

[3] S. Steel & A. De Roeck (1987).

[4] We assume basic familiarity with chart parsing and with Tomita's LR parsing algorithm. For further literature on charts see Wirén (1987), for LR parsing Aho & Ullman (1979).

[5] Our Combine is more general than Winograds (1983) since we use a CKY variant with complete edges only.

with Greek letters: α, β, γ; *vertices* by: i, j, k; *edges* [1] as pairs of the rule in dotted notation and their left and right vertices. We will call the first symbol to the right of the dot in an active edge the *required category*. In the following example of an active edge, <A -> B · C D | i,j>, C is the required category, i the left and j the right vertex. TD, LC are implemented in such a way that they use only active edges, CKY only complete edges and BI active and complete edges.

2.1.1.1. Top-down (TD)

This strategy can be considered as Earley-like since it is very similar to Earley's algorithm apart from the fact that it does not use a lookahead. Some authors describe its Combine as the 'fundamental rule' of chart parsing[2].

Combine
Whenever a complete edge E_c <A -> α · | j,k> is added to the chart, combine it with all active edges E_a <B -> β · Cγ | i,j> ending at E_c's starting point j if E_c's category A corresponds to E_a's required category C and build the corresponding new edges <B -> βC · γ | i,k>.

Propose
Whenever an active edge E_a <A -> α · Bβ | i,j> is added to the chart, if its required category B is a nonterminal, for every rule B -> γ in the grammar G that expands E_a's required category B add an empty active edge E_x <B -> · γ | j,j> .

The parse runs top-down and is triggered by the first active edge <S -> · α | 0,0> expanding α with all the rules that have the start symbol S as left-hand side. It proceeds in a strict left-to-right fashion, the next input word is read when all Proposes and Combines up to the current input point have been executed. Opposed to the TD strategy are the two typical bottom-up parsers LC and CKY. Instead of using the rule selecting mechanism for building new hypotheses or active edges on the basis of required categories, the bottom-up parsers trigger the rules from the categories of complete edges.

2.1.1.2. Left-corner (LC)

As a bottom-up technique new edges are proposed on the basis of complete edges. The corresponding grammar rules are triggered if the first symbol of the right-hand side (RHS) of the rule , the 'left-corner', has the same category as the complete edge. LC and TD have the same 'Combine' and expand active edges from left to right.

Propose
Whenever a complete edge E_a <A -> α · | i,j> is added to the chart, for every rule B -> Aβ in the grammar G whose left-corner symbol A has the same category as E_a, add an active edge E_n <B -> A · β | i,j> to the chart.

2.1.1.3. Cocke-Kasami-Younger (CKY)

The second bottom-up parser is a variant of the Cock-Kasami-Younger algorithm. It is similar to CKY in the sense that it is pure bottom-up and combines only complete edges, but the grammar rules are not restricted to Chomsky normal form. To achieve this, Combine works from the right to the left and the rules are proposed on the rightmost symbol of the right-hand side.

Propose
Whenever a complete edge E_a <A -> α · | i,j> is added to the chart, propose all rules B -> βA in the grammar G, whose rightmost symbol is A.

---

[1] Edges correspond to Earley's (1970) 'states' and to 'items' in Aho & Ullman (1977).

[2] H. Thompson (1981). We will describe the two operations in a similar style to Thompson and Wirén (1987).

Combine

Whenever a complete edge E_c <A -> α · | i,j> is added to the chart, for each rule B -> βA that is proposed on A and for each combination of consecutive[1] complete edges starting with E_c and going to the left whose categories satisfy the sequence βA build a new complete edge E_n <B -> βA· | k,j> starting at the vertex k of its left-most edge and ending at the right vertex j of E_c.

2.1.1.4. Bi-directional (BI)

De Roeck (1987) gives the following motivation for bi-directional rule invocation. Form a linguistic point of view, certain phenomena like traces are best analysed top-down whereas others are best discovered from evidence in the string, e.g. in coordination, the conjunction is the best evidence for triggering the rule. But in the two bottom-up chart parsers the rules are triggered by a fixed handle, which is either the left-most or the right-most symbol of the RHS of a rule. In bi-directional chart parsing the linguist can tailor the rule invoking strategy locally by annotating the rules if they are used top-down or bottom-up. For bottom-up rules, one has to indicate which symbol they are triggered on. A rule for coordinating Np's can be annotated for example 'up' on the conjunction: Np -> Np Conj Np {up Conj}. When the complete edge for Conj is added to the chart, this rule will be triggered and it will add an active edge that tries to combine with an NP to the left as well as to the right. The Propose of the bi-directional parser acts accordin.g the the annotation of the rules. In order to avoid duplication Combine has been implemented in such a way that it first combines to the left and only then to the right. We have to expand the dotted rule notation in the sense that a colon marks the beginning of the recognized symbols of an edge and the dot the end of the recognized parts. Symbols to the right of a colon and to the left of a dot have been recognized. Our implementation proposes only to the right. An active edge can be left-active, if it is expecting a symbol to the left.

Propose

Whenever a complete edge E_a <A -> : γ · | i,j> is added to the chart, for every rule B -> αAβ annotated bottom-up on the symbol A, add an active edge E_n <B -> α : A · β | i,j> to the chart.
Whenever an active edge E_a <A -> : α · Bβ | i,j> is added to the chart, if its required category B is a nonterminal, add an empty active edge E_x <B -> · δ | j,j> for each rule in the grammar G that is annotated down and that expands E_a's required category B.

Combine

Whenever a left active edge E_a <A -> α : γ · β | i,j> is added to the chart, for each combination of complete edges starting with E_a and going to the left whose categories satisfy the sequence α build a new active edge E_n <A -> : αγ · β | k,j> starting at the vertex k of its left-most edge and ending at the right vertex j of E_a.
Whenever a complete edge E_c <A -> α · | j,k> is added to the chart, combine it with all active edges E_a <B -> : β · Cγ | i,j> ending at E_c's starting point j if E_c's category A corresponds to E_a's required category C and build the corresponding new edges <B -> : β C · γ | i,k>.

The bi-directional chart parser was included in the tests for verifying the hypothesis if triggering annotations of the rules reduce the search space and improve the overall performance.

2.1.2. Top-down filter (tdf)

In general, bottom-up algorithms have a reduced search space by the fact that they are data-driven. On evidence of complete edges, that are present in the string, they are faster in finding the corresponding rules. They do not have to explore the whole search space of the grammar as the TD parser that is over-productive in active edges. On the other hand, bottom-up parsers have problems in dealing with rules that have common right parts as in the following example: 'CD' is the common right string of both rules A -> BCD and A -> CD. Both rules will fire on a string 'BCD'. Bottom-up chart parsers are over-productive in complete edges that do not attach to phrases on the left. The next two chapters deal with filters to reduce over-production of useless edges: top-down-filtering, a method for bottom-

[1] Two complete edges can be combined to the left if the starting vertex of the first edge corresponds to the ending vertex of the second one.

up parsers to reduce the production of useless complete edges and lookahead, a method to reduce the production of unnecessary active edges, useful for TD, LC and BI.

Top-down-filtering is described like running a top-down parser in parallel with a bottom-up parser[1]. The bottom-up parser proposes new edges while the top-down process checks if they can be derived from the root. The tdf rejects all proposed rules that will generate phrases that can't be attached to the left context. The tdf uses a "reachability relation $\mathfrak{R}$ where A$\mathfrak{R}$B holds if there exists some derivation from A to B such that B is the left-most element in a string derived from A" (Wirén 1987, cf also Pratt 1975). The reachability relation $\mathfrak{R}$ can be precompiled so that the tdf can check in constant time if $\mathfrak{R}$ holds for a new proposed edge.

In the LC parser, the tdf is implemented in the following way: For each nonterminal category A the transitive closure of the categories that are reachable from A are precalculated . At each vertex, the tdf keeps a list of the reachable categories. Vertex 0 is initialised with the list of the categories that are reachable from the root category. For each new active edge E_n, the tdf adds the categories that are reachable from the new required category to the tdf -list of reachable categories at the ending vertex of E_n. In the function Propose, the tdf checks for every proposed rule if its left-hand side category is in the list of the reachable categories of the current vertex. Only rules that pass the tdf lead to the creation of new active edges.

2.1.3. Lookahead (la)

Top-down-filtering cuts down the production of useless complete edges in bottom-up parsing by checking if they can combine with the left context. The lookahead function verifies if a new edge can be attached to the right context. Wirén (1987) reports an experiment where la was used successfully to reduce the over-production of active edges in TD or LC[2]. La is based on the same reachability relation as tdf but is looking to the right. Each time an active edge is proposed, the la function checks if the new required category C_n can reach the preterminal category of the next input word a_{i+1}, that is if $C_n \mathfrak{R} a_{i+1}$ holds. We have tested all our parsers without lookahead.

2.1.4. Rule compilation

The third method for reducing the number of edges in chart parsing is precompiling the grammar rules into decision trees. Assume two rules used by a LC parser, A -> BC and A -> BDE. The two rules have the common left part B and can therefore be merged into a single combined rule with a shared part B: A -> B (C, DE). In parsing, the two rule scan share the common part B which is represented by a single active edge. TD and LC compile the rules by factoring out similar left parts. CKY combines from right to left and does therefore the factoring from the right. BI, based on annotations of single rules, uses both ways of building its rule decision trees. Note that building decision trees for rules is related to the way in which the canonical set of items is built for the construction of LR parsing tables. The first step in making a new canonical LR set is done by taking all the items in a set that have the same category to the right of the dot. Building decision trees from rules also groups them together on the basis of the next category that has to be recognized.

2.2. Tomita's extended LR parser (TOM)

Tomita's Parser (Tomita 1985) is a generalised version of a LR shift-reduce parser. It is based on two data structures: a graph structured stack and a parser forest for representing the result. The graph-structured stack allows nondeterministic parsing of ambiguous grammars with LR shift-reduce technique. Tomita (1988) shows that his graph-structured stack is very similar to the chart in chart parsing. The parse forest allows an efficient representation of the result. While the number of parses can grow exponentially, the parse forest grows polynomially. In order to see which part of the program is responsible for efficiency, we compare two versions of Tomita's parser, one with and one without parse forest.

---

[1] J. Slocum (1981b), M.Kay (1982), Pratt (1975), Wirén (1987).

[2] Earley uses the lookahead in a different way: The lookahead is in his Completer and not in the Predictor, as in Wiréns programms.

2.3. The grammar types

Each parser can be run with three different types of context-free grammars. This is done by adding annotations to the context-free rule skeleton. Whenever all constituents of a context-free rule are found, before the new edge is constructed, the parser calls for a rule-body procedure (Slocum 1981b) that evaluates the annotations of the rule. Each grammar type has a different module for evaluating the rule-body procedure. If the rule-body procedure returns an error because a test has failed, the new edge is discarded.

The first grammar type uses simple phrase structure rules with monadic categories that have no annotations. The second grammar type has annotations that go with simple sets of attribute-value pairs where the values are atomic. These annotations allow testing and assigning features to particular nodes of the context-free rules. The third grammar type is unification based and uses complex features and annotations in the PATR-II style. The three grammar types vary the rule-body procedure overhead (unification being very time consuming) and therefore show a more realistic picture of the behaviour of the parsers in real context.

3. Previous empirical comparisons

In this section we report the results of three practical comparisons of parsers relevant to our experiment: Slocum who compared particularly LC and CKY with top-down filter, Tomita who compared his extended LR parser with Earley's parser and Wirén who compared TD and LC with top-down filter and lookahead. Each of the comparisons gives an incomplete picture. They usually compare two basic strategies with different refinements like top-down filtering etc.

One of the important points for comparisons is stressed by Slocum (1981b): Theoretical calculations about worst case behaviour of algorithms can be quite inaccurate because they often neglect the constant factors that seem to have a dominant effect in practical situations. He writes: "In order to meaningfully describe performance, one must take into account the complete operational context of the natural language processing system, particularly the expenses encountered in storage management and applying rule-body procedures" since a significant portion of the sentence analysis effort may be invested in evaluating the rule-body procedures. To measure performance accurately he suggests including "everything one actually pays for in real computing world: Paging, storage management, building interpretations, rule-body procedure, etc., as well as parse time".

3.1. Slocum: two bottom-up chart parsers, LC vs. CKY

Slocum has conducted two experiments, one at SRI and the second one at LRC, which is more important for us. In the second experiment, he carefully compared two bottom-up chart parsers: LC and CKY enhanced with top-down filtering and early constituent tests[1]. He used the German analysis grammar (~ 500 rules) of the MT system that was under development at the time at LRC and a corpus of 262 sentences going from 1 - 39 words per sentence (15,6 words/sentence average). The rule-body procedures were rather considerable for a parser test but interesting for realistic performance evaluation. They consisted of "the complete analysis procedures for the purpose of subsequent translation which includes the production of a full syntactic and semantic analysis via phrase-structure rules, feature tests and operations, transformations and case frames".

Given his grammar and test sentences Slocum establishes two things:

1) LC with tdf (without early constituent test) performs best, better than CKY (which is the opposite of the common expectation). He comments that a tdf increases the search space, but that the overhead is balanced in practice by the fact that the tdf reduces the number of phrases and therefore particularly the rule-body procedure overhead, which is considerable in his case. "The overhead for filtering in LC is less than that in CKY. This situation is due to the fact that LC maintains a natural left-right ordering of the rule constituents in its internal representation, whereas CKY does not and must therefore compute it at run time."

---

[1] "The early constituent test calls for the parser to evaluate that protion of the rule body-procedure which tests the first constituent, as soon as it is discovered, to determine if it is acceptable" (Slocum 1981b)

2) "The benefits of top-down filtering are dependent on sentence length: in fact filtering is detrimental for shorter sentences. Averaging over all other strategies, the break-even point for top-down filtering occurs at about 7 words."

We conclude this section with a statement from Slocum about filters: "Filtering always increases pure parse time because the parser sees it as pure overhead. The benefits are only observable in overall system performance, due primarily to a significant reduction in the time/space spent evaluating rule-body procedures." This point will be important in our comparisons since we use three different grammar types with rule-body procedures that take increasingly more time.

3.2. Tomita: The Tomita parser vs. Earley's algorithm

Tomita (1985) compared his parser empirically with two versions of the Earley algorithm (E-I and E-II). In our terminology this would correspond to TD and TD+la. While the Tomita parser was producing a parse forest, E-I and E-II were run as recognizers and produced no parse.

In the comparison, four pure context-free phrase-structure grammars were used, consisting of a varying number of rules: G1 8, G2 40, G3 220 and G4 400 rules. These grammars were tested with two sets of sentences, S1: 40 sentences from texts and S2: 13 artificial sentences that have an increasing number of prepositional phrases (1 to 13). These artificial sentences are useful for testing growing sentence ambiguity since the number of parses grows exponentially (Martin et al. 1981).

Tomita's experiment shows that his algorithm works 5 to 10 times faster than Earley's standard algorithm (TD), and 2 to 3 times faster than Earley's improved algorithm (TD+la). He states that this result is due to the pre-compilation of the grammar into an LR table. Tomita summarizes that his algorithm "is significantly faster than Earley's algorithm, in the context of practical natural language processing. . . Its parsing time and space remain tractable when sentence length, sentence ambiguity or grammar size grows in practical applications."

3.3. Wirén: top-down and bottom-up chart parsers, TD vs. LC

Wirén compared in his experiment two basic chart parsers with several improvements, TD versus LC, both with lookahead, LC with top-down filtering[1]. He tested his parsers with grammars G1 to G3 from Tomita, with a reduced number of the two sentence sets, S21 and S2.

The results of his experiments show that the "directed methods" - based on top-down filtering and lookahead - reduce significantly the number of edges and perform better than undirected parsers. Tested independently, the selectivity filter (lookahead in our terminology) turned out to be much more time efficient than top-down filtering that degraded time performance as the grammar grew larger[2]. "The maximally directed strategy - ... with selectivity and top-down filtering - remained the most efficient one throughout all the experiments, both with respect to edges produced and time consumed." It performed better than TD with lookahead.

Putting the results of the three experiments together, we would expect that improved LC performs best amongst chart parsers. Since the Tomita parser has only been compared with TD, we can expect a different result by comparing it with improved bottom-up chart parsers that compile their rules into decision trees (cf. chap. 2.1.4). Tomita and Wirén measure pure parsing time determined by CPU time minus time for garbage collection. Their grammars are pure CF grammars using little rule-body procedure time and it is therefore difficult to predict what the interaction will be between filtering overhead and rule-body procedure and how this will influence overall performance.

4. The comparison

4.1. The parsers

Our main goal was the selection of a suitable parsing strategy for our on-line MT-system. Since our application is time critical, one of the important questions was what combination of parser and rule-

---

[1] LC à la Kilbury has already been used by Slocum. What it comes down to is that new active edges subsume the complete edges that have provoked their proposal. Since we use that variant of LC (cf. 2.1.1.2) coming from Slocum (1981a), we don't distinguish between a standard LC and the Kilbury variant.

[2] Wirén explains this puzzle with implementational reasons.

body procedure is best for our purpose. One of the objectives was to verify if the Tomita parser is as efficient as predicted if it is compared to improved bottom-up chart parsers. Since no comparison existed between all the basic rule invocation strategies for chart parsers, we decided to compare the Tomita parser with four chart parsers. To guarantee the comparability of the chart parsers, we chose Slocum's implementation (1981a) as basic design for all chart parsers. We added two supplementary rule invocation strategies to his bottom-up left-corner (LC) and Cocke-Kasami-Younger strategy (CKY), namely a top-down Earley-like strategy (TD) and a bi-directional strategy (BI). The basic chart parsers were augmented by two enhancements, i.e. top-down filtering and compilation of the rules into decision trees. We took the Tomita parser as described by Tomita (1985) and added a second version without the parse forest representation. Since its LR(0) parsing table has no lookahead, we added no lookahead to the chart parsers.

All the programs are implemented in Allegro Common Lisp and tested on a Macintosh II (MC68020 with 5 MB RAM). As main parameters we compared number of edges, number of rule-body procedure executions and over-all time.

4.2. The grammars and sentences

The first test uses small grammars (22 and 80 rules) together with the same 50 artificial sentences. The monadic grammars are tested with all 9 parsing strategies (TOM +/-parse forest; TD, LC, CKY, BI, the bottom-up parsers +/-tdf), for features and unification grammars we use TOM without parse forest and all the chart parsers. The 50 test sentences are constructed artificially to control parameters like sentences ambiguity, sentences length and three linguistic phenomena, i.e. PP-attachment, relative clauses and coordination. They can be classified into two groups, one where ambiguity grows exponentially with increasing sentence length (PP-attachment and coordination), and a second group, where the sentence length does not influence ambiguity (they have 1 to 3 readings). The sentence length varies from 3 to 24 words. Each grammar type has two small grammars with approximately 25 resp. 80 rules.

The second test compares a reduced number of parsers (TOM, TD, LC, CKY, bottom-up +/-tdf) with a bigger monadic grammar based on the German avalanche corpus that has 750 rules and 300 lexical items. The 50 test sentences were taken from the avalanche corpus, their length varies from 6 to 42 words (average 19 words per sentence).

5. Test-results and discussion

Before we comment, we will give a brief outline of how we present the test-results in appendix 1 and 2. The seven tables in appendix 1 summarize the statistics for each grammar and set of sentences. We give the total number of edges and the total time for each parser over all sentences. The figures for time indicate overall time[1] that includes rule-body procedure etc. The reader should be careful in the interpretation of the timings; these figures are dependent on machine, lisp system and the way in which the algorithms are programmed. Nevertheless, we think that they give an indication of relations. Appendix 2 shows a limited number of diagrams to illustrate the figures graphically.

In appendix 1, each table shows three fields, one for the number of edges and two for timings: 1) the total time for all sentences and 2) the time for 26 sentences with low ambiguity. The second group of test sentences includes relative clauses and coordinations. The number of words per sentence goes from 5 to 23 words (13 average) and they have 1 to 5 readings. Time is measured in milliseconds. The column 'diff' indicates the difference of the parsers from Tomita which is set to 1. In the field 'time all', we added the average time per word (ms/word) in order to have a figure that can easily be compared across the different tests. We have listed the number of edges because this figure is often given as measurement for parser performance. But one can observe that the rankings based on the number of edges and the one based on timing do not correspond. This is due to the particular way in which the chart parsers are implemented. As we have mentioned in chap. 2.1.1, TD and LC keep only active edges in the chart, whereas CKY has only complete edges and BI both. For TOM, we counted the number of shift operations.

Since there is limited space for diagrams, most of them show three parsers: TOM, TD and LC +/-tdf. All the diagrams display the time/word relation for a particular grammar and a sentence set. Diagram 1

[1] Since we have forced a garbage collection before each sentence, the garbage collector does not interfere with the timings.

and 2 show PP-attachment (high ambiguity: a 20 word sentence has 132 parses), diagram 3 the time/word relation for the 750 rule grammar and all the avalanche sentences. Diagram 4 represents the times for LC +/-tdf with the three different grammar types for a set of coordinations in high ambiguity. Diagram 5 shows all parsers with a set of relative clauses that have low ambiguity.

5.1. The chart parsers

Our tests confirm Slocum's and Wirén's data: the left-corner parser (LC) with top-down filtering is overall the most efficient chart parser. It ranks highest among the chart parsers with all grammar types and grammar sizes. The only exceptions are monadic and feature grammars of the size of 80 rules with low ambiguity sentences (see below 5.3.). Earley-like top-down (TD) with the two small grammars is highly overproductive in active edges and therefore a bad choice if it is used without lookahead. Diagram 1 and 2 show how TD is influenced negatively by the grammar size, the grammar in diagram 2 has three times more rules. Strangely enough, in the large grammar (table 3 and diagram 3), TD is converging towards LC as the sentences grow longer. In diagram 3, one can see well its initial overhead of active edges .

The bi-directional chart parser (BI) was included in the tests for verifying the hypothesis if triggering annotations on the rules reduce the search space and improve the overall performance. None of our tests could confirm such a hypothesis. It seems that top-down filtering or lookahead influence performance to a greater extent than linguistic triggering annotations. BI did not perform better with any particular set of test sentences or grammars.

5.2. The Tomita parser and chart parsers

Diagram 1 and 2 show how the Tomita parser (to+) performs best in situations of high ambiguity. Taking the overall timings in table 1 and 2, TD is 4.75 to 6.53 times slower than TOM (and our comparison stops at sentences with 20 words with 132 readings). The situation is less dramatic if we take LC+tdf. Here the difference is 1.67 to 1.9. But, if we take our grammar of 750 rules with its low ambiguity sentences, the gap is much smaller: 1.38 for LC+tdf and 2.15 for TD. A closer look at diagram 1 and 2 shows that TOM without parse forest (to-) is roughly equivalent to LC+tdf (lc+). We therefore think that the major speed gain of TOM comes from its parse forest, which is an efficient way of packing the parse trees. But, this representation could be used with any parser and is not specific of TOM. In diagram 3, TOM and LC+tdf show a constant time difference. Precompiling the grammar rules into a LR parsing table or precompiling them into decision trees does not make a crucial difference, even with very long sentences of up to 42 words and a large grammar of 750 rules.

5.3. Filters, grammar size and rule-body procedures

This chapter tries to address the complex interaction between parsing strategy, grammar size, sentence ambiguity and overheads for top-down filtering and rule-body procedure. There is no standard grammar size. According to the grammar type, the size varies. We estimate that unification grammars, which are highly lexical, might have 50 to 100 rules, grammars with simple features around 500[1], and monadic grammars several thousand rules.

In general, a TD parser is disadvantaged if the grammar has a high branching factor because of its overproduction of active edges (cf. chap. 2.1.3.). Bottom-up parsers suffer from rules with common right factoring in the right-hand side of the rules (cf. chap. 2.1.2.). A grammar might produce different results about TD overproduction or top-down filters according to its branching factor or right factoring. The effect of a top-down filter is not always a good one. We have contradicting results about top-down filtering. In the test with the monadic grammar of 750 rules, the two chart parsers with top-down filter (lc+ and cky+) perform better than their counterparts without filter. Diagram 3 also shows a converging TD and a diverging LC-tdf (lc-) as the sentence length increases. This is due to the high right factoring of that grammar. The opposite result is shown by monadic and feature grammars with 75 rules together with the sample of low ambiguity sentences. In these cases, the overhead from the top-down filter deteriorates the efficiency of the chart parsers with top-down filter. Unfiltered parsers with sentences up to 19 words are faster than the filtered ones. This result is influenced by the nature of the grammar as well as its size since the top-down filter with the small grammars (22 or 30 rules) shows a positive effect.

[1] The Metal German analysis grammar, which is based on simple features, has 500-600 rules.

Another tradeoff is between top-down filter and rule-body procedure. In our tests we compare three different types of rule-body procedures: no annotations in monadic grammars or simple features and unification. Monadic grammars and simple feature grammars have a small rule-body procedure whereas the overhead for unification is considerable (2/3 for unification and 1/3 for pure parsing). Diagram 4 shows optically that the top-down filter has a positive effect as the rule-body procedure grows. With a time consuming rule-body procedure, a top-down filter becomes vital for the overall efficiency. This statement should not be interpreted as a generalization about simple feature grammars versus unification. Our point is independent of a particular grammar type but has to do with the relation between pure parse time and rule-body procedure time.

5.4. Sentence length

As we reported in chap. 3.1., Slocum claims that the benefits of top-down filtering are dependent on the sentence length and that the break-even point for top-down filtering (averaged over LC and CKY) occurs at about 7 words. As we have shown above, the question is more complex and influenced furthermore by the number of parses as well as by the nature and by the size of the grammar (right factoring and branching factor). Some of our tests show clearly that the length of the sentence is not necessarily the main parameter. We believe that no generalization is possible unless all the mentioned factors are taken into account.

5.5. Final choice

The choice of the parsing strategy for our MT-system was guided by the following ideas: Possible candidates for an on-line parser that parses strictly from left to right are TOM, LC+tdf and TD. Given the performance, TD was ruled out. The question of the grammar type was more difficult to solve. The grammar has to predict all the sentences but only the correct ones, no overproduction is allowed. We therefore have to subclassify heavily by using a system of about 100 grammatical and semantic features. The worst cases for an empirical efficiency test are sentences with high ambiguity. Diagram 4 shows the performance of the three grammar types where the 20 word sentence has the highest ambiguity. The average time per word varies heavily according to the grammar type: monadic - 70 ms, features - 160 ms and unification -1267 ms. Unification is slower by a factor of about 20. This factor would be increased by the search for possible next words because it is not a simple matching of categories but a complicated search that has to take into account all the instantiated variables from constituents that have already been found. Given this poor expectation for unification grammar in on-line parsing, we were left with two grammar types, and we opted for simple monadic grammars, rather as a matter of computational simplicity. Together with monadic grammars, we chose the Tomita parser, because it was slightly more performant with the large grammar for the avalanche corpus, and last but not least, because of its elegance. We like the idea of precompiling the grammar into a LR table.

We have come to the conclusion that it is very difficult to test empirically the performance of algorithms or better of programs and to find good generalizations[1]. Nevertheless, we believe that we have shown that the parse forest representation is to a large extent responsible for the good performance of the Tomita parser, and second, that the difference in efficiency between the Tomita parser without the parse forest representation and an enhanced left-corner parser with top-down filtering and compiled rules is small. Two points of empirical research have not been addressed in our tests, which could also help the practitioners of computational linguistics when they have to select their parsing strategies: 1) We have excluded the use of a lookahead. We think that this point needs further investigation (i.e. TOM with an LALR table versus LC+tdf with la). 2) Since the parse forest representation is highly efficient, its benefits in combination with unification grammars need more clarification.

6. Acknowledgement

I would like to thank Anne De Roeck and Tony Lawson for all the theoretical and practical discussions as well as for the contribution of the unification grammar from Anne and the unifyer from Tony. Thanks also to Mike Rosner, Rod Johnson, Dominique Petitpierre and Thomas Russi for their comments and useful hints.

[1] On a different machine with a different lisp system the same programs might behave differently.

7. References

A. Aho and J. Ullman (1979), *Principles of compiler design*, Addison Wesly.

J. Earley (1970), An efficient context-free parsing algorithm, *Communications of the ACM 13(2)*, 94-102.

M. Kay (1982), *Algorithmic schemata and data structures in syntactic processing*, CSL-80-12, Xerox Parc, Palo Alto.

W. Martin, K.Church & R. Patil (1981), Preliminary analysis of a breadth-first parsing algorithm: Theoretical and experimental results, MIT LCS Technical report.

V. Pratt (1975), LINGOL - A progress report, *Proc. 4th IJCAI*, Tbilisi, 422-428.

J. Slocum (1981a), *A practical comparison of parsing strategies for machine translation and other natural language processing purposes*, PhD University of Texas, Austin.

J. Slocum (1981b), A practical comparison of parsing strategies, *Proc. 19th ACL*, Standford.

S. Steel & A. De Roeck (1987), Bi-directional parsing, in: Hallam & Mellish (eds.), *Advances in AI, Proc. of the 1987 AISB Conference*, J. Wiley, London.

H.R. Tennant et al. (1983) , Menu-based natural language understanding, *Proc. 21st ACL*, 151-158.

H. Thompson (1981), Chart parsing and rule schemata in GPSG, *Proc. 19th ACL*, Stanford .

M. Tomita (1985), *An efficient context-free parsing algorithm for natural languages and its applications*, PhD CMU Pittsburg. Also as: *Efficient parsing for natural language. A fast algorithm for practical purposes*. Kluwer, Boston1986.

M. Tomita (1987), An efficient augmented-context-free parsing algorithm, *Computational Linguistics* 13(1/2).

M. Tomita (1988), Graph-structured stack and natural language parsing, *Proc. 26th ACL*, Buffalo.

J. Winograd (1983), *Language as a cognitive process, Syntax*, Addison-Wesley.

M. Wirén (1987), A comparison of rule-invocation strategies in context-free chart parsing, *Proc. 3rd European chapter ACL*, 226-233.

Table 1 Monadic grammar: 22 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6532	5	3.08	24049	2	1.67	38	8000	3	1.07
lc-	13332	8	6.28	41418	7	2.88	66	12301	6	1.64
cky+	3449	2	1.62	33219	4	2.31	53	12901	7	1.72
cky-	6886	6	3.24	34634	5	2.41	55	9599	4	1.28
bi+	6497	4	3.06	44483	8	3.10	71	15850	9	2.11
bi-	9655	7	4.55	36265	6	2.52	58	10033	5	1.34
td	19766	9	9.31	68217	9	4.75	109	12985	8	1.73
tom	2124	1	1.00	14364	1	1.00	23	7498	1	1.00
to-2	3881	3	1.83	25756	3	1.79	41	7940	2	1.06

Table 2 Monadic grammar: 75 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6224	5	3.38	24418	3	1.90	39	8417	6	1.36
lc-	9020	8	4.90	27834	6	2.16	44	7318	5	1.18
cky+	2803	2	1.52	38980	7	3.03	62	16481	7	2.66
cky-	4884	6	2.65	25650	4	1.99	41	6150	1	0.99
bi+	7476	4	4.06	56649	8	4.40	90	20084	8	3.24
bi-	8277	7	4.50	25815	5	2.00	41	6730	4	1.09
td	33028	9	17.95	84130	9	6.53	134	20665	9	3.33
tom	1840	1	1.00	12883	1	1.00	21	6200	2	1.00
to-2	3117	3	1.69	21899	2	1.70	35	6382	3	1.03

Table 3 Monadic grammar: 750 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word
lc+	3693	3	1.94	20132	2	1.28	21
lc-	18411	6	9.67	51132	6	3.26	54
cky+	1923	2	1.01	36715	4	2.34	39
cky-	6662	4	3.50	40785	5	2.60	43
td	16951	5	8.90	33717	3	2.15	35
tom	1904	1	1.00	15684	1	1.00	16

Abbreviations

+	+ top-down filter (tdf)
-	- tdf
tom	Tomita + parse forest
to-2	Tomita - parse forest

Table 4 Feature grammar: 30 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6067	4	2.10	38669	1	0.93	62	11434	2	1.06
lc-	12666	7	4.39	76415	6	1.85	122	18483	8	1.71
cky+	2661	1	0.92	47015	3	1.14	75	15233	4	1.41
cky-	5304	3	1.84	69844	5	1.69	111	16213	5	1.50
bi+	6165	5	2.14	64901	4	1.57	103	17517	7	1.62
bi-	10266	6	3.56	81869	7	1.98	130	14050	3	1.30
td	21669	8	7.52	114548	8	2.77	182	16982	6	1.57
to-2	2883	2	1.00	41368	2	1.00	66	10818	1	1.00

Table 5 Feature grammar: 80 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6232	4	2.02	41265	1	0.96	66	12383	3	1.04
lc-	8963	7	2.91	60867	6	1.42	97	14649	5	1.23
cky+	2831	1	0.92	57884	4	1.35	92	20668	6	1.73
cky-	4871	3	1.58	59248	5	1.38	94	13983	4	1.17
bi+	7459	5	2.42	80217	7	1.87	128	26467	8	2.22
bi-	8198	6	2.66	49901	3	1.16	79	11967	2	1.00
td	32792	8	10.64	135650	8	3.16	216	24985	7	2.10
to-2	3083	2	1.00	42985	2	1.00	68	11917	1	1.00

Table 6 Unification grammar: 30 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	5449	3	1.79	144349	1	0.91	251	23433	1	0.93
lc-	11446	6	3.75	525250	7	3.32	913	74750	8	2.97
cky+	2813	1	0.92	153500	2	0.97	267	27233	3	1.08
cky-	7068	4	2.32	533515	8	3.38	928	73167	7	2.91
bi+	8675	5	2.85	210300	5	1.33	366	29034	5	1.16
bi-	14247	7	4.67	307949	6	1.95	536	37866	6	1.51
td	18795	8	6.16	169684	4	1.07	295	27983	4	1.11
to-2	3049	2	1.00	158032	3	1.00	275	25132	2	1.00

Table 7 Unification grammar: 80 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	5519	3	2.00	108382	1	0.95	188	20531	2	1.03
lc-	12527	7	4.54	272181	8	2.39	473	42549	8	2.14
cky+	2483	1	0.90	122618	3	1.08	213	27603	3	1.39
cky-	5700	4	2.07	268468	7	2.36	467	41485	7	2.09
bi+	6770	5	2.45	135834	4	1.19	236	29918	4	1.51
bi-	12232	6	4.43	189650	6	1.67	330	30217	5	1.52
td	34093	8	12.36	146203	5	1.29	254	31551	6	1.59
to-2	2759	2	1.00	113750	2	1.00	198	19866	1	1.00

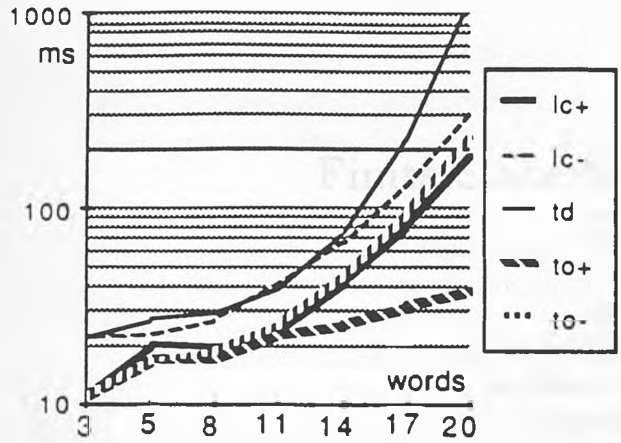

Diagram 1 Monadic-22 PP-attachment

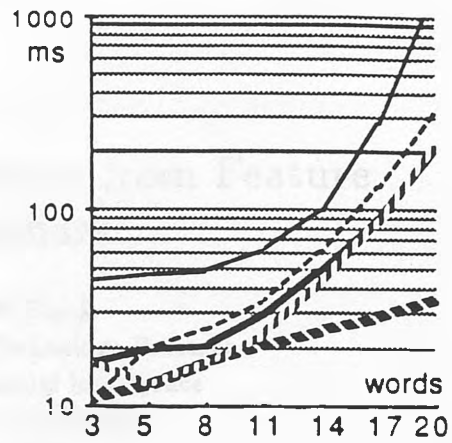

Diagram 2 Monadic-75 PP-attachment

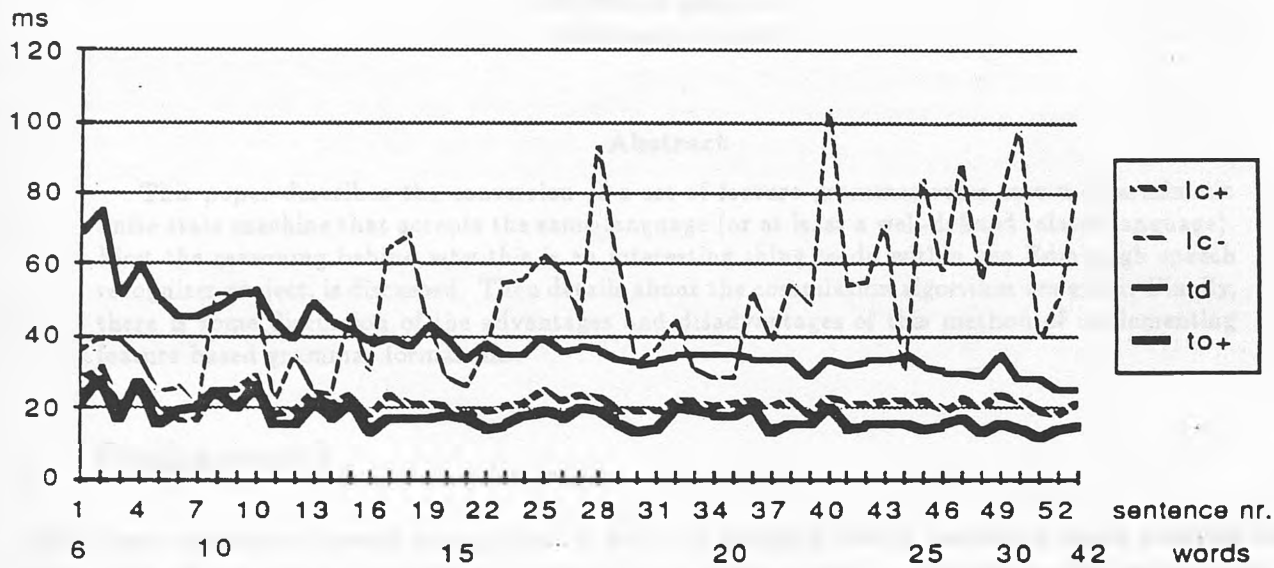

Diagram 3 Monadic-750 (all sentences)

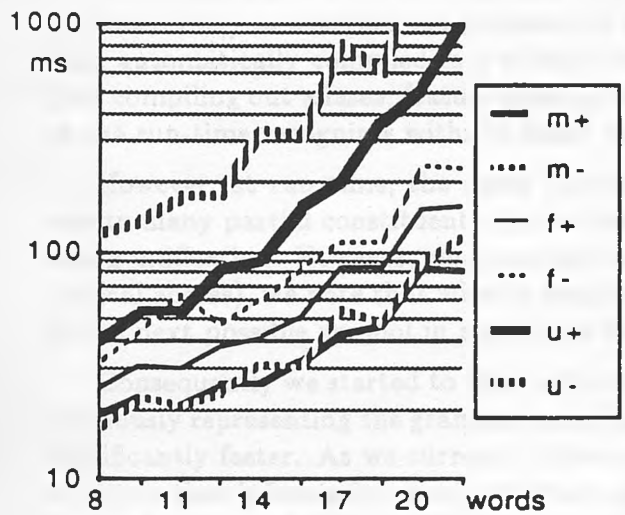

Diagram 4 LC +/-tdf (30 rules)
3 grammar types, coordination
high ambiguity

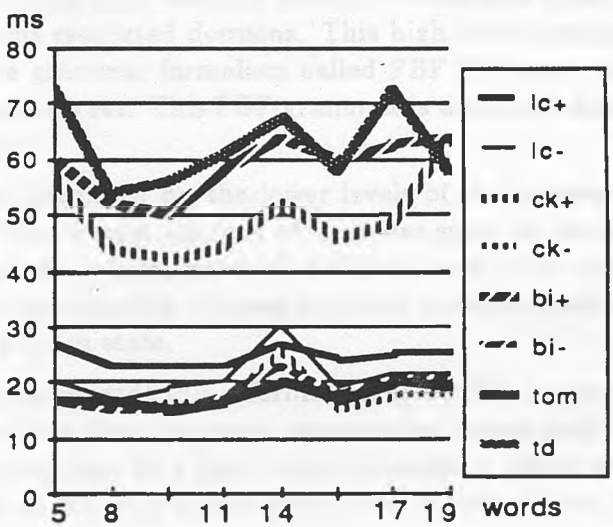

Diagram 5 Monadic-75, relative clauses
low ambiguity