Guided Parsing of Range Concatenation Languages

François Barthélemy, Pierre Boullier, Philippe Deschamp and **Éric de la Clergerie**
INRIA-Rocquencourt
Domaine de Voluceau
B.P. 105
78153 Le Chesnay Cedex, France
{Francois.Barthelemy Pierre.Boullier
Philippe.Deschamp Eric.De_La_Clergerie}@inria.fr

Abstract

The theoretical study of the range concatenation grammar [RCG] formalism has revealed many attractive properties which may be used in NLP. In particular, range concatenation languages [RCL] can be parsed in polynomial time and many classical grammatical formalisms can be translated into equivalent RCGs without increasing their worst-case parsing time complexity. For example, after translation into an equivalent RCG, any tree adjoining grammar can be parsed in $\mathcal{O}(n^6)$ time. In this paper, we study a parsing technique whose purpose is to improve the practical efficiency of RCL parsers. The non-deterministic parsing choices of the main parser for a language L are directed by a *guide* which uses the shared derivation forest output by a prior RCL parser for a suitable superset of L. The results of a practical evaluation of this method on a wide coverage English grammar are given.

1 Introduction

Usually, during a nondeterministic process, when a nondeterministic choice occurs, one explores all possible ways, either in parallel or one after the other, using a backtracking mechanism. In both cases, the nondeterministic process may be assisted by another process to which it asks its way. This assistant may be either a guide or an oracle. An *oracle* always indicates all the good ways that will eventually lead to success, and those good ways only, while a *guide* will indicate all the good ways but may also indicate some wrong ways. In other words, an oracle is a perfect guide (Kay, 2000), and the worst guide indicates all possible ways. Given two problems P_1 and P_2 and their respective solutions S_1 and S_2, if they are such that $S_1 \supset S_2$, any algorithm which solves P_1 is a candidate guide for nondeterministic algorithms solving P_2. Obviously, supplementary conditions have to be fulfilled for P_1 to be a guide. The first one deals with relative efficiency: it assumes that problem P_1 can be solved more efficiently than problem P_2. Of course, parsers are privileged candidates to be guided. In this paper we apply this technique to the parsing of a subset of RCLs that are the languages defined by RCGs. The syntactic formalism of RCGs is powerful while staying computationally tractable. Indeed, the positive version of RCGs [PRCGs] defines positive RCLs [PRCLs] that exactly cover the class *PTIME* of languages recognizable in deterministic polynomial time. For example, any mildly context-sensitive language is a PRCL.

In Section 2, we present the definitions of PRCGs and PRCLs. Then, in Section 3, we design an algorithm which transforms any PRCL L into another PRCL L_1, $L \subseteq L_1$ such that the (theoretical) parse time for L_1 is less than or equal to the parse time for L: the parser for L will be guided by the parser for L_1. Last, in Section 4, we relate some experiments with a wide coverage tree-adjoining grammar [TAG] for English.

2 Positive Range Concatenation Grammars

This section only presents the basics of RCGs, more details can be found in (Boullier, 2000b).

A *positive range concatenation grammar* [PRCG] $G = (N, T, V, P, S)$ is a 5-tuple where N is a finite set of *nonterminal symbols* (also called *predicate names*), T and V are finite, disjoint sets of *terminal symbols* and *variable symbols* respectively, $S \in N$ is the *start predicate name*, and P is a finite set of *clauses*

$$\psi_0 \rightarrow \psi_1 \dots \psi_m$$

where $m \geq 0$ and each of $\psi_0, \psi_1, \dots, \psi_m$ is a *predicate* of the form

$$A(\alpha_1, \dots, \alpha_i, \dots, \alpha_p)$$

where $p \geq 1$ is its *arity*, $A \in N$, and each of $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, is an *argument*.

Each occurrence of a predicate in the LHS (resp. RHS) of a clause is a predicate *definition* (resp. *call*). Clauses which define predicate name A are called A-clauses. Each predicate name $A \in N$ has a fixed arity whose value is *arity*(A). By definition *arity*$(S) = 1$. The *arity* of an A-clause is *arity*(A), and the *arity* k of a grammar (we have a k-PRCG) is the maximum arity of its clauses. The *size* of a clause $c = A_0(\dots) \rightarrow \dots A_i(\dots) \dots A_m(\dots)$ is the integer $|c| = \sum_{i=0}^{m}$ *arity*(A_i) and the *size* of G is $|G| = \sum_{c \in P} |c|$.

For a given string $w = a_1 \dots a_n \in T^*$, a pair of integers (i, j) s.t. $0 \leq i \leq j \leq n$ is called a *range*, and is denoted $\langle i..j \rangle_w$: i is its *lower bound*, j is its *upper bound* and $j - i$ is its *size*. For a given w, the set of all ranges is noted $\mathcal{R}_w$. In fact, $\langle i..j \rangle_w$ denotes the occurrence of the string $a_{i+1} \dots a_j$ in w. Two ranges $\langle i..j \rangle_w$ and $\langle k..l \rangle_w$ can be concatenated iff the two bounds j and k are equal, the result is the range $\langle i..l \rangle_w$. Variable occurrences or more generally strings in $(T \cup V)^*$ can be instantiated to ranges. However, an occurrence of the terminal t can be instantiated to the range $\langle j-1..j \rangle_w$ iff $t = a_j$. That is, in a clause, several occurrences of the same terminal may well be instantiated to different ranges while several occurrences of the same variable can only be instantiated to the same range. Of course, the concatenation on strings matches the concatenation on ranges.

We say that $A(\rho_1, \dots, \rho_p)$ is an *instantiation* of the predicate $A(\alpha_1, \dots, \alpha_p)$ iff $\rho_i \in \mathcal{R}_w, 1 \leq i \leq p$ and each symbol (terminal or variable) of α_i, $1 \leq i \leq p$ is instantiated to a range in $\mathcal{R}_w$ s.t. α_i is instantiated to ρ_i. If, in a clause, all predicates are instantiated, we have an *instantiated clause*.

A binary relation *derive*, denoted $\underset{G,w}{\Rightarrow}$, is defined on strings of instantiated predicates. If $\Gamma_1 \, \gamma \, \Gamma_2$ is a string of instantiated predicates and if γ is the LHS of some instantiated clause $\gamma \rightarrow \Gamma$, then we have $\Gamma_1 \, \gamma \, \Gamma_2 \underset{G,w}{\Rightarrow} \Gamma_1 \Gamma \Gamma_2$.

An input string $w \in T^*$, $|w| = n$ is a sentence iff the empty string (of instantiated predicates) can be derived from $S(\langle 0..n \rangle_w)$, the instantiation of the start predicate on the whole source text. Such a sequence of instantiated predicates is called a *complete derivation*. $\mathcal{L}(G)$, the PRCL defined by a PRCG G, is the set of all its sentences.

For a given sentence w, as in the context-free [CF] case, a single complete derivation can be represented by a *parse tree* and the (unbounded) set of complete derivations by a finite structure, the *parse forest*. All possible derivation strategies (i.e., top-down, bottom-up, ...) are encompassed within both parse trees and parse forests.

A clause is:

- *combinatorial* if at least one argument of its RHS predicates does not consist of a single variable;
- *bottom-up erasing* (resp. *top-down erasing*) if there is at least one variable occurring in its RHS (resp. LHS) which does not appear in its LHS (resp. RHS);
- *erasing* if there exists a variable appearing only in its LHS or only in its RHS;
- *linear* if none of its variables occurs twice in its LHS or twice in its RHS;
- *simple* if it is non-combinatorial, non-erasing and linear.

These definitions extend naturally from clause to set of clauses (i.e., grammar).

In this paper we will not consider negative RCGs, since the guide construction algorithm

presented is Section 3 is not valid for this class. Thus, in the sequel, we shall assume that RCGs are PRCGs.

In (Boullier, 2000b) is presented a parsing algorithm which, for any RCG G and any input string of length n, produces a parse forest in $\mathcal{O}(|G|n^d)$ time. The exponent d, called *degree* of G, is the maximum number of free (independent) bounds in a clause. For a non-bottom-up-erasing RCG, d is less than or equal to the maximum value, for all clauses, of the sum $p_c + v_c$ where, for a clause c, p_c is its arity and v_c is the number of (different) variables in its LHS predicate.

3 PRCG to 1-PRCG Transformation Algorithm

The purpose of this section is to present a transformation algorithm which takes as input any PRCG G and generates as output a 1-PRCG G_1, such that $L = \mathcal{L}(G) \subseteq L_1 = \mathcal{L}(G_1)$.

Let $G = (N, T, V, P, S)$ be the initial PRCG and let $G_1 = (N_1, T_1, V_1, P_1, S_1)$ be the generated 1-PRCG. Informally, to each p-ary predicate name A we shall associate p unary predicate names A^i, each corresponding to one argument of A. We define

$$N_1 = \cup_{A \in N}\{A^i \mid A \in N, 1 \leq i \leq arity(A)\}$$

and $T_1 = T$, $V_1 = V$, $S_1 = S^1$ and the set of clauses P_1 is generated in the way described below.

We say that two strings α and β, on some alphabet, *share a common substring*, and we write $\mathcal{S}(\alpha, \beta)$, iff either α, or β or both are empty or, if $\alpha = uvw$ and $\beta = xvy$, we have $|v| \geq 1$.

For any clause $c = \psi_0 \rightarrow \psi_1 \ldots \psi_j \ldots \psi_m$ in P, such that $\psi_j = A_j(\alpha_j^1, \ldots, \alpha_j^{m_j}), 0 \leq j \leq m, m_j = arity(A_j)$, we generate the set of m_0 clauses $\mathcal{K}_c = \{c^1, \ldots, c^{m_0}\}$ in the following way. The clause $c^k, 1 \leq k \leq m_0$ has the form $A_0^k(\alpha_0^k) \rightarrow \Psi^k$ where the RHS Ψ^k is constructed from the ψ_j's as follows. A predicate call $A_j^i(\alpha_j^i)$ is in Ψ^k iff the arguments α_j^i and α_0^k share a common substring (i.e., we have $\mathcal{S}(\alpha_0^k, \alpha_j^i)$).

As an example, the following set of clauses, in which X, Y and Z are variables and a and b are terminal symbols, defines the 3-copy language $\{www \mid w \in \{a, b\}^*\}$ which is not a CF language [CFL] and even lies beyond the formal power of TAGs.

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X, Y, Z) \\ A(aX, aY, aZ) & \rightarrow & A(X, Y, Z) \\ A(bX, bY, bZ) & \rightarrow & A(X, Y, Z) \\ A(\varepsilon, \varepsilon, \varepsilon) & \rightarrow & \varepsilon \end{array}$$

This PRCG is transformed by the above algorithm into a 1-PRCG whose clause set is

$$\begin{array}{lcl} S^1(XYZ) & \rightarrow & A^1(X)\ A^2(Y)\ A^3(Z) \\ A^1(aX) & \rightarrow & A^1(X) \\ A^2(aY) & \rightarrow & A^2(Y) \\ A^3(aZ) & \rightarrow & A^3(Z) \\ A^1(bX) & \rightarrow & A^1(X) \\ A^2(bY) & \rightarrow & A^2(Y) \\ A^3(bZ) & \rightarrow & A^3(Z) \\ A^1(\varepsilon) & \rightarrow & \varepsilon \\ A^2(\varepsilon) & \rightarrow & \varepsilon \\ A^3(\varepsilon) & \rightarrow & \varepsilon \end{array}$$

It is not difficult to show that $L \subseteq L_1$.

This transformation algorithm works for any PRCG. Moreover, if we restrict ourselves to the class of PRCGs that are non-combinatorial and non-bottom-up-erasing, it is easy to check that the constructed 1-PRCG is also non-combinatorial and non-bottom-up-erasing. It has been shown in (Boullier, 2000a) that non-combinatorial and non-bottom-up-erasing 1-RCLs can be parsed in cubic time after a simple grammatical transformation. In order to reach this cubic parse time, we assume in the sequel that any RCG at hand is a non-combinatorial and non-bottom-up-erasing PRCG.

However, even if this cubic time transformation is not performed, we can show that the (theoretical) throughput of the parser for L_1 cannot be less than the throughput of the parser for L. In other words, if we consider the parsers for L and L_1 and if we recall the end of Section 2, it is easy to show that the degrees, say d and d_1, of their polynomial parse times are such that $d_1 \leq d$. The equality is reached iff the maximum value d in G is produced by a unary clause which is kept unchanged by our transformation algorithm.

The starting RCG G is called the *initial grammar* and it defines the *initial language* L. The corresponding 1-PRCG G_1 constructed by our transformation algorithm is called the *guiding grammar* and its language L_1 is the *guiding language*.

If the algorithm to reach a cubic parse time is applied to the guiding grammar G_1, we get an equivalent *n^3-guiding grammar* (it also defines L_1). The various RCL parsers associated with these grammars are respectively called *initial parser*, *guiding parser* and *n^3-guiding parser*. The output of a (n^3-) guiding parser is called a *(n^3-) guiding structure*. The term *guide* is used for the process which, with the help of a guiding structure, answers 'yes' or 'no' to any question asked by the *guided* process. In our case, the guided processes are the RCL parsers for L called *guided parser* and *n^3-guided parser*.

4 Parsing with a Guide

Parsing with a guide proceeds as follows. The guided process is split in two phases. First, the source text is parsed by the guiding parser which builds the guiding structure. Of course, if the source text is parsed by the n^3-guiding parser, the n^3-guiding structure is then translated into a guiding structure, as if the source text had been parsed by the guiding parser. Second, the guided parser proper is launched, asking the guide to help (some of) its nondeterministic choices.

Our current implementation of RCL parsers is like a (cached) recursive descent parser in which the nonterminal calls are replaced by instantiated predicate calls. Assume that, at some place in an RCL parser, $A(\rho_1, \rho_2)$ is an instantiated predicate call. In a corresponding guided parser, this call can be guarded by a call to a guide, with A, ρ_1 and ρ_2 as parameters, that will check that both $A^1(\rho_1)$ and $A^2(\rho_2)$ are instantiated predicates in the guiding structure. Of course, various actions in a guided parser can be guarded by guide calls, but the guide can only answer questions that, in some sense, have been registered into the guiding structure. The guiding structure may thus contain more or less complete information, leading to several guide *levels*.

For example, one of the simplest levels one may think of, is to only register in the guiding structure the (numbers of the) clauses of the guiding grammar for which at least one instantiation occurs in their parse forest. In such a case, during the second phase, when the guided parser tries to instantiate some clause c of G, it can call the guide to know whether or not c can be valid. The guide will answer 'yes' iff the guiding structure contains the set $\mathcal{K}_c$ of clauses in G_1 generated from c by the transformation algorithm.

At the opposite, we can register in the guiding structure the full parse forest output by the guiding parser. This parse forest is, for a given sentence, the set of all instantiated clauses of the guiding grammar that are used in all complete derivations. During the second phase, when the guided parser has instantiated some clause c of the initial grammar, it builds the set of the corresponding instantiations of all clauses in $\mathcal{K}_c$ and asks the guide to check that this set is a subset of the guiding structure.

During our experiment, several guide levels have been considered, however, the results in Section 5 are reported with a restricted guiding structure which only contains the set of all (valid) clause numbers and for each clause the set of its LHS instantiated predicates.

The goal of a guided parser is to speed up a parsing process. However, it is clear that the theoretical parse time complexity is not improved by this technique and even that some practical parse time will get worse. For example, this is the case for the above 3-copy language. In that case, it is not difficult to check that the guiding language L_1 is T^*, and that the guide will always answer 'yes' to any question asked by the guided parser. Thus the time taken by the guiding parser and by the guide itself is simply wasted. Of course, a guide that always answer 'yes' is not a good one and we should note that this case may happen, even when the guiding language is not T^*. Thus, from a practical point of view the question is simply "will the time spent in the guiding parser and in the guide be at least recouped by the guided parser?" Clearly, in the general case, no definite answer can be brought to such a question, since the total parse time may depend not only on the input grammar, the (quality of) the guiding grammar (e.g., is L_1 not a too "large" superset of L), the guide level, but also it may depend on the parsed sentence itself. Thus, in our opinion, only the results of practical experiments may *globally* decide if using a guided parser is worthwhile .

Another potential problem may come from the size of the guiding grammar itself. In particular, experiments with regular approximation of

CFLs related in (Nederhof, 2000) show that most reported methods are not practical for large CF grammars, because of the high costs of obtaining the minimal DFSA.

In our case, it can easily be shown that the increase in size of the guiding grammars is bounded by a constant factor and thus seems *a priori* acceptable from a practical point of view.

The next section depicts the practical experiments we have performed to validate our approach.

5 Experiments with an English Grammar

In order to compare a (normal) RCL parser and its guided versions, we looked for an existing wide-coverage grammar. We chose the grammar for English designed for the XTAG system (XTAG, 1995), because it both is freely available and seems rather mature. Of course, that grammar uses the TAG formalism.[1] Thus, we first had to transform that English TAG into an equivalent RCG. To perform this task, we implemented the algorithm described in (Boullier, 1998) (see also (Boullier, 1999)), which allows to transform any TAG into an equivalent simple PRCG.[2]

However, Boullier's algorithm was designed for pure TAGs, while the structures used in the XTAG system are not trees, but rather tree schemata, grouped into linguistically pertinent tree families, which have to be instantiated by inflected forms for each given input sentence. That important difference stems from the radical difference in approaches between "classical" TAG parsing and "usual" RCL parsing. In the former, through lexicalization, the input sentence allows the selection of tree schemata which are then instantiated on the corresponding inflected forms, thus the TAG is not really part of the parser. While in the latter, the (non-lexicalized) grammar is precompiled into an optimized automaton.[3]

Since the instantiation of all tree schemata by the complete dictionary is impracticable, we designed a two-step process. For example, from the sentence "George loved himself .", a *lexer* first produces the sequence "George {`n-n nxn-n nn-n`} loved {`tnx0vnx1-v tnx0vnx1s2-v tnx0vs1-v`} himself {`tnx0n1-n nxn-n`} . {`spu-punct spus-punct`}", and, in a second phase, this sequence is used as actual input to our parsers. The names between braces are *pre-terminals*. We assume that each terminal leaf l of every elementary tree schema τ has been labeled by a pre-terminal name of the form $t = f\text{-}c[\text{-}i]$ where f is the family of τ, c is the category of l (verb, noun, ...) and i is an optional occurrence index.[4]

Thus, the association George "{`n-n nxn-n nn-n`}" means that the inflected form "George" is a noun (suffix `-n`) that can occur in all trees of the "`n`", "`nxn`" or "`nn`" families (everywhere a terminal leaf of category noun occurs).

Since, in this two-step process, the inputs are not sequences of terminal symbols but instead simple DAG structures, as the one depicted in Figure 1, we have accordingly implemented in our RCG system the ability to handle inputs that are simple DAGs of tokens.[5]

In Section 3, we have seen that the language L_1 defined by a guiding grammar G_1 for some RCG G, is a superset of L, the language defined by G. If G is a simple PRCG, G_1 is a simple 1-PRCG, and thus L_1 is a CFL (see (Boullier, 2000a)). In other words, in the case of TAGs, our transformation algorithm approximates the initial tree-adjoining language by a CFL, and the steps of CF parsing performed by the guiding parser can well be understood in terms of TAG parsing.

The original algorithm in (Boullier, 1998) performs a one-to-one mapping between elementary trees and clauses, initial trees generate simple unary clauses while auxiliary trees generate simple binary clauses. Our transformation algorithm leaves unary clauses unchanged (simple unary clauses are in fact CF productions). For binary A-clauses, our algorithm generates two clauses,

[1] We assume here that the reader has at least some cursory notions of this formalism. An introduction to TAG can be found in (Joshi, 1987).

[2] We first stripped the original TAG of its feature structures in order to get a pure featureless TAG.

[3] The advantages of this approach might be balanced by the size of the automaton, but we shall see later on that it can be made to stay reasonable, at least in the case at hand.

[4] The usage of f as component of t is due to the fact that in the XTAG syntactic dictionary, lemmas are associated with tree family names.

[5] This is done rather easily for linear RCGs. The processing of non-linear RCGs with lattices as input is outside the scope of this paper.

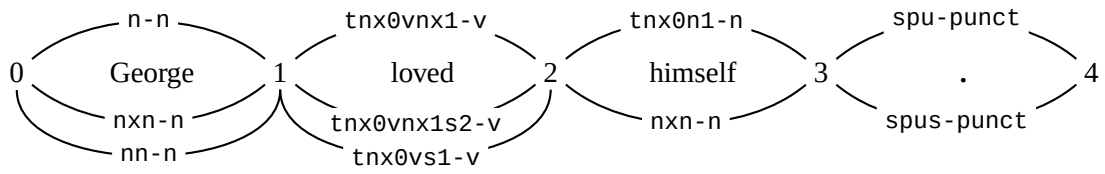

Figure 1: Actual source text as a simple DAG structure

an A^1-clause which corresponds to the part of the auxiliary tree to the left of the spine and an A^2-clause for the part to the right of the spine. Both are CF clauses that the guiding parser calls independently. Therefore, for a TAG, the associated guiding parser performs substitutions as would a TAG parser, while each adjunction is replaced by two independent substitutions, such that there is no guarantee that any couple of A^1-tree and A^2-tree can glue together to form a valid (adjoinable) A-tree. In fact, guiding parsers perform some kind of (deep-grammar based) shallow parsing.

For our experiments, we first transformed the English XTAG into an equivalent simple PRCG: the initial grammar G. Then, using the algorithms of Section 3, we built, from G, the corresponding guiding grammar G_1, and from G_1 the n^3-guiding grammar. Table 1 gives some information on these grammars.[6]

RCG	initial	guiding	n^3-guiding
$\|N\|$	22	33	4 204
$\|T\|$	476	476	476
$\|P\|$	1 144	1 696	5 554
$\|G\|$	15 578	15 618	17 722
degree	27	27	3

Table 1: RCGs $G = (N, T, V, P, S)$ facts

For our experiments, we have used a test suite distributed with the XTAG system. It contains 31 sentences ranging from 4 to 17 words, with an average length of 8. All measures have been performed on a 800 MHz Pentium III with 640 MB of memory, running Linux. All parsers have been compiled with gcc without any optimization flag.

We have first compared the total time taken to produce the guiding structures, both by the n^3-guiding parser and by the guiding parser (see Table 2). On this sample set, the n^{27}-guiding parser is twice as fast as the n^3-guiding parser. We guess that, on such short sentences, the benefit yielded by the lowest degree has not yet offset the time needed to handle a much greater number of clauses. To validate this guess, we have tried longer sentences. With a 35-word sentence we have noted that the n^3-guiding parser is almost six times faster than the n^{27}-guiding parser and besides we have verified that the even crossing point seems to occur for sentences of around 16–20 words.

parser	guiding	n^3-guiding
sample set	0.990	1.870
35-word sent.	30.560	5.210

Table 2: Guiding parsers times (sec)

parser	load module
initial	3.063
guided	8.374
n^3-guided	14.530

Table 3: RCL parser sizes (MB)

parser	sample set	35-word sent.
initial	5.810	3 679.570
guided	1.580	63.570
n^3-guided	2.440	49.150
XTAG	4 282.870	>5 days

Table 4: Parse times (sec)

[6] Note that the worst-case parse time for both the initial and the guiding parsers is $\mathcal{O}(n^{27})$. As explained in Section 3, this identical polynomial degrees $d = d_1 = 27$ comes from an untransformed unary clause which itself is the result of the translation of an initial tree.

The sizes of these RCL parsers (load modules) are in Table 3 while their parse times are in Table 4.[7] We have also noted in the last line, for reference, the times of the latest XTAG parser (February 2001),[8] on our sample set and on the 35-word sentence.[9]

6 Guiding Parser as Tree Filter

In (Sarkar, 2000), there is some evidence to indicate that in LTAG parsing the number of trees selected by the words in a sentence (a measure of the syntactic lexical ambiguity of the sentence) is a better predictor of complexity than the number of words in the sentence. Thus, the accuracy of the tree selection process may be crucial for parsing speeds. In this section, we wish to briefly compare the tree selections performed, on the one hand by the words in a sentence and, on the other hand, by a guiding parser. Such filters can be used, for example, as pre-processors in classical [L]TAG parsing. With a guiding parser as tree filter, a tree (i.e., a clause) is kept, not because it has been selected by a word in the input sentence, but because an instantiation of that clause belongs to the guiding structure.

The recall of both filters is 100%, since all pertinent trees are necessarily selected by the input words and present in the guiding structure. On the other hand, for the tree selection by the words in a sentence, the precision measured on our sample set is 15.6% on the average, while it reaches 100% for the guiding parser (i.e., each and every selected tree is in the final parse forest).

7 Conclusion

The experiment related in this paper shows that some kind of guiding technique has to be considered when one wants to increase parsing efficiency. With a wide coverage English TAG, on a small sample set of short sentences, a guided parser is on the average three times faster than its non-guided counterpart, while, for longer sentences, more than one order of magnitude may be expected.

However, the guided parser speed is very sensitive to the level of the guide, which must be chosen very carefully since potential benefits may be overcome by the time taken by the guiding structure book-keeping procedures.

Of course, the filtering principle related in this paper is not novel (see for example (Lakshmanan and Yim, 1991) for deductive databases) but, if we consider the various attempts of guided parsing reported in the literature, ours is one of the very few examples in which important savings are noted. One reason for that seems to be the extreme simplicity of the interface between the guiding and the guided process: the guide only performs a direct access into the guiding structure. Moreover, this guiding structure is (part of) the usual parse forest output by the guiding parser, without any transduction (see for example in (Nederhof, 1998) how a FSA can guide a CF parser).

As already noted by many authors (see for example (Carroll, 1994)), the choice of a (parsing) algorithm, as far as its throughput is concerned, cannot rely only on its theoretical complexity but must also take into account practical experiments. Complexity analysis gives worst-case upper bounds which may well not be reached, and which implies constants that may have a preponderant effect on the typical size ranges of the application.

We have also noted that guiding parsers can be used in classical TAG parsers, as efficient and (very) accurate tree selectors. More generally, we are currently investigating the possibility to use guiding parsers as shallow parsers.

[7] The time taken by the lexer phase is linear in the length of the input sentences and is negligible.

[8] It implements a chart-based head-corner parsing algorithm for lexicalized TAGs, see (Sarkar, 2000). This parser can be run in two phases, the second one being devoted to the evaluation of the features structures on the parse forest built during the first phase. Of course, the times reported in that paper are only those of the first pass. Moreover, the various parameters have been set so that the resulting parse trees and ours are similar. Almost half the sample sentences give identical results in both that system and ours. For the other half, it seems that the differences come from the way the co-anchoring problem is handled in both systems. To be fair, it must be noted that the time taken to output a complete parse forest is not included in the parse times reported for our parsers. Outputing those parse forests, similar to Sarkar's ones, takes one second on the whole sample set and 80 seconds for the 35-word sentence (there are more than 3 600 000 instantiated clauses in the parse forest of that last sentence).

[9] Considering the last line of Table 2, one can notice that the times taken by the guided phases of the guided parser and the n^3-guided parser are noticeably different, when they should be the same. This anomaly, not present on the sample set, is currently under investigation.

The above results also show that (guided) RCL parsing is a valuable alternative to classical (lexicalized) TAG parsers since we have exhibited parse time savings of several orders of magnitude over the most recent XTAG parser. These savings even allow to consider the parsing of medium size sentences with the English XTAG.

The global parse time for TAGs might also be further improved using the transformation described in (Boullier, 1999) which, starting from any TAG, constructs an equivalent RCG that can be parsed in $\mathcal{O}(n^6)$. However, this improvement is not definite, since, on typical input sentences, the increase in size of the resulting grammar may well ruin the expected practical benefits, as in the case of the n^3-guiding parser processing short sentences.

We must also note that a (guided) parser may also be used as a guide for a unification-based parser in which feature terms are evaluated (see the experiment related in (Barthélemy et al., 2000)).

Although the related practical experiments have been conducted on a TAG, this guide technique is not dedicated to TAGs, and the speed of all PRCL parsers may be thus increased. This pertains in particular to the parsing of all languages whose grammars can be translated into equivalent PRCGs — MC-TAGs, LCFRS, ...

References

F. Barthélemy, P. Boullier, Ph. Deschamp, and É. de la Clergerie. 2000. Shared forests can guide parsing. In *Proceedings of the Second Workshop on Tabulation in Parsing and Deduction (TAPD'2000)*, University of Vigo, Spain, September.

P. Boullier. 1998. A generalization of mildly context-sensitive formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 17–20, University of Pennsylvania, Philadelphia, PA, August.

P. Boullier. 1999. On tag parsing. In $6^{ème}$ *conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, pages 75–84, Cargèse, Corse, France, July. See also *Research Report N° 3668* at `http://www.inria.fr/RRRT/RR-3668.html`, INRIA-Rocquencourt, France, Apr. 1999, 39 pages.

P. Boullier. 2000a. A cubic time extension of context-free grammars. *Grammars*, 3(2/3):111–131.

P. Boullier. 2000b. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 53–64, Trento, Italy, February.

John Carroll. 1994. Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32th Annual Meeting of the Association for Computational Linguistics (ACL'94)*, pages 287–294, New Mexico State University at Las Cruces, New Mexico, June.

A. K. Joshi. 1987. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–114. John Benjamins, Amsterdam.

M. Kay. 2000. Guides and oracles for linear-time parsing. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 6–9, Trento, Italy, February.

V.S. Lakshmanan and C.H. Yim. 1991. Can filters do magic for deductive databases? In *3rd UK Annual Conference on Logic Programming*, pages 174–189, Edinburgh, April. Springer Verlag.

M.-J. Nederhof. 1998. Context-free parsing through regular approximation. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, Ankara, Turkey, June–July.

M.-J. Nederhof. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44.

A. Sarkar. 2000. Practical experiments in parsing using tree adjoining grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 193–198, University of Paris 7, Jussieu, Paris, France, May.

the research group XTAG. 1995. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA, USA, March.