Are Very Large Context-Free Grammars Tractable?

Pierre Boullier & Benoît Sagot
INRIA-Rocquencourt
Domaine de Voluceau, Rocquencourt BP 105
78153 Le Chesnay Cedex, France
{Pierre.Boullier,Benoit.Sagot}@inria.fr

Abstract

In this paper, we present a method which, in practice, allows to use parsers for languages defined by very large context-free grammars (over a million symbol occurrences). The idea is to split the parsing process in two passes. A first pass computes a sub-grammar which is a specialized part of the large grammar selected by the input text and various filtering strategies. The second pass is a traditional parser which works with the sub-grammar and the input text. This approach is validated by practical experiments performed on a Earley-like parser running on a test set with two large context-free grammars.

1 Introduction

More and more often, in real-word natural language processing (NLP) applications based upon grammars, these grammars are no more written by hand but are automatically generated, this has several consequences. This paper will consider one of these consequences: the generated grammars may be very large. Indeed, we aim to deal with grammars that have, say, over a million symbol occurrences and several hundred thousands rules. Traditional parsers are not usually prepared to handle them, either because these grammars are simply too big (the parser's internal structures blow up) or the time spent to analyze a sentence becomes prohibitive.

This paper will concentrate on context-free grammars (CFG) and their associated parsers. However, virtually all Tree Adjoining Grammars (TAG, see e.g., (Schabes et al., 1988)) used in NLP applications can (almost) be seen as lexicalized Tree Insertion Grammars (TIG), which can be converted into strongly equivalent CFGs (Schabes and Waters, 1995). Hence, the parsing techniques and tools described here can be applied to most TAGs used for NLP, with, in the worst case, a light over-generation which can be easily and efficiently eliminated in a complementary pass. This is indeed what we have achieved with a TAG automatically extracted from (Villemonte de La Clergerie, 2005)'s large-coverage factorized French TAG, as we will see in Section 4. Even (some kinds of) non CFGs may benefit from the ideas described in this paper.

The reason why the run-time of context-free (CF) parsers for large CFGs is damaged relies on a theoretical result. A well-known result is that CF parsers may reach a worst-case running time of $\mathcal{O}(|G| \times n^3)$ where $|G|$ is the *size* of the CFG and n is the *length* of the source text.[1] In typical NLP applications which mainly work at the sentence level, the length of a sentence does not often go beyond a value of say 100, while its average length is around 20-30 words.[2] In these conditions, the size of the grammar, despite its linear impact on the complexity, may be the prevailing factor: in (Joshi, 1997), the author remarks that "the real limiting factor in practice is the size of the grammar".

The idea developed in this paper is to split the parsing process in two passes. A first pass called *filtering* pass computes a sub-grammar which is the

[1] These two notions will be defined precisely later on.
[2] At least for French, English and similar languages.

Proceedings of the 10th Conference on Parsing Technologies, pages 94–105,
Prague, Czech Republic, June 2007.

sub-part of the large input grammar selected by the input sentence and various filtering strategies. The second pass is a traditional parser which works with the sub-grammar and the input sentence. The purpose is to find a filtering strategy which, in typical practical situations, minimizes on the average the total run-time of the filtering pass followed by the parser pass.

A filtering pass may be seen as a (filtering) function that uses the input sentence to select a sub-grammar out of a large input CFG. Our hope, using such a filter, is that the time saved by the parser pass which uses a (smaller) sub-grammar will not totally be used by the filter pass to generate this sub-grammar.

It must be clear that this method cannot improve the worst-case parse-time because there exists grammars for which the sub-grammar selected by the filtering pass is the input grammar itself. In such a case, the filtering pass is simply a waste of time. Our purpose in this paper is to argue that this technique may profit from typical grammars used in NLP. To do that we put aside the theoretical view point and we will consider instead the average behaviour of our processors.

More precisely we will study on two large NL CFGs the behaviour of our filtering strategies on a set of test sentences. The purpose being to choose the *best* filtering strategy, if any. By best, we mean the one which, on the average, minimizes the total run-time of both the filtering pass followed by the parsing pass.

Useful formal notions and notations are recalled in Section 2. The filtering strategies are presented in Section 3 while the associated experiments are reported in Section 4. This paper ends with some concluding remarks in Section 5.

2 Preliminaries

2.1 Context-free grammars

A CFG G is a quadruple (N, T, P, S) where N is a non-empty finite set of *nonterminal symbols*, T is a finite set of *terminal symbols*, P is a finite set of (context-free rewriting) *rules* (or *productions*) and S is a distinguished nonterminal symbol called the *axiom*. The sets N and T are disjoint and $V = N \cup T$ is the *vocabulary*. The rules in P have the form $A \rightarrow \alpha$, with $A \in N$ and $\alpha \in V^*$.

For a given string $\alpha \in V^*$, its size (length) is noted $|\alpha|$. As an example, for the input string $w = a_1 \cdots a_n, a_i \in T$, we have $|w| = n$. The empty string is denoted ε and we have $|\varepsilon| = 0$. The size $|G|$ of a CFG G is defined by $|G| = \sum_{A \rightarrow \alpha \in P} |A\alpha|$.

For G, on strings of V^*, we define the binary relation *derive*, noted $\Rightarrow$, by $\gamma_1 A \gamma_2 \overset{A \rightarrow \alpha}{\underset{G}{\Rightarrow}} \gamma_1 \alpha \gamma_2$ if $A \rightarrow \alpha \in P$ and $\gamma_1, \gamma_2 \in V^*$. The subscript G or even the superscript $A \rightarrow \alpha$ may be omitted. As usual, its transitive (resp. reflexive transitive) closure is noted $\overset{+}{\underset{G}{\Rightarrow}}$ (resp. $\overset{*}{\underset{G}{\Rightarrow}}$). We call *derivation* any sequence of the form $\gamma_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \gamma_2$. A *complete derivation* is a derivation which starts with the axiom and ends with a terminal string w. In that case we have $S \overset{*}{\underset{G}{\Rightarrow}} \gamma \overset{*}{\underset{G}{\Rightarrow}} w$, and γ is a *sentential form*.

The *string language* defined (generated, recognized) by G is the set of all the terminal strings that are derived from the axiom: $\mathcal{L}(G) = \{w \mid S \overset{+}{\underset{G}{\Rightarrow}} w, w \in T^*\}$. We say that a CFG is empty iff its language is empty.

A nonterminal symbol A is *nullable* iff it can derive the empty string (i.e., $A \overset{+}{\underset{G}{\Rightarrow}} \varepsilon$). A CFG is *$\varepsilon$-free* iff its nonterminal symbols are non-nullable.

A CFG is *reduced* iff every symbol of every production is a symbol of at least one complete derivation. A reduced grammar is empty iff its production set is empty ($P = \emptyset$). We say that a non-empty reduced grammar is in *canonical form* iff its vocabulary only contains symbols that appear in the productions of P.[3,4]

Two CFGs G and G' are *weakly equivalent* iff they generate the same string language. They are *strongly equivalent* iff they generate the same set of structural descriptions (i.e., parse trees). It is a well known result (See Section 3.2) that every CFG G can be transformed in time linear w.r.t. $|G|$ into a strongly equivalent (canonical) reduced CFG G'.

For a given input string $w \in T^*$, we define its

[3] We may say that the canonical form of the empty reduced grammar is $(\{S\}, \emptyset, \emptyset, S)$ though the axiom S does not appear in any production.

[4] Note that the pair (P, S) completely defines a reduced CFG $G = (N, T, P, S)$ in canonical form since we have $N = \{X_0 \mid X_0 \rightarrow \alpha \in P\} \cup \{S\}$, $T = \{X_i \mid X_0 \rightarrow X_1 \cdots X_p \in P \wedge 1 \leq i \leq p\} - N$. Thus, in the sequel, we often note simply $G = (P, S)$ grammars in canonical form.

ranges as the set $R^w = \{[i..j] \mid 1 \leq i \leq j \leq |w|+1\}$. If $w = w_1tw_3 \in T^*$ is a terminal string, and if $t \in T \cup \{\varepsilon\}$ is a (terminal or empty) symbol, the *instantiation* of t in w is the triple noted $t[i..j]$ where $[i..j]$ is a range with $i = |w_1|+1$ and $j = i + |t|$. More generally, the *instantiation* of the terminal string w_2 in $w_1w_2w_3$ is noted $w_2[i..j]$ with $i = |w_1|+1$ and $j = i + |w_2|$. Obviously, the instantiation of w itself is then $w[1..1+|w|]$.

Let us consider an input string $w = w_1w_2w_3$ and a CFG G. If we have a complete derivation $d = S \overset{*}{\underset{G}{\Rightarrow}} w_1Aw_3 \overset{A\to\alpha}{\underset{G}{\Rightarrow}} w_1\alpha w_3 \overset{*}{\underset{G}{\Rightarrow}} w_1w_2w_3$, we see that A derives w_2 (we have $A \overset{+}{\underset{G}{\Rightarrow}} w_2$). Moreover, in this complete derivation, we also know a range in R^w, namely $[i..j]$, which covers the substring w_2 which is derived by A ($i = |w_1|+1$ and $j = i + |w_2|$). This is represented by the *instantiated nonterminal symbol* $A[i..j]$. In fact, each symbol which appears in a complete derivation may be transformed into its instantiated counterpart. We thus talk of instantiated productions or (complete) instantiated derivations. For a given input text w, and a CFG G, let P_G^w be the set of instantiated productions that appears in all complete instantiated derivations.[5] The pair $(P_G^w, S[1..|w|+1])$ is the *(reduced) shared parse forest* in canonical form.[6]

2.2 Finite-state automata

A *finite-state automaton* (FSA) is the 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a non empty finite set of *states*, Σ is a finite set of *terminal symbols*, δ is the transition relation $\delta = \{(q_i, t, q_j) | q_i, q_j \in Q \wedge t \in T \cup \{\varepsilon\}\}$, q_0 is a distinguished element of Q called the *initial state* and F is a subset of Q whose elements are called *final states*. The size of A is defined by $|A| = |\delta|$.

As usual, we define both a *configuration* as an element of $Q \times T^*$ and *derive* a binary relation between configurations, noted $\underset{A}{\vdash}$ by $(q, tx) \underset{A}{\vdash} (q', x)$, iff $(q, t, q') \in \delta$. If $w'w'' \in T^*$, we call *derivation* any sequence of the form $(q', w'w'') \underset{A}{\vdash} \cdots \underset{A}{\vdash} (q'', w'')$. If $w \in T^*$, the *initial configuration* is noted c_0 and is the pair (q_0, w). A *final configuration* is noted c_f and has the form (q_f, ε) with $q_f \in F$. A *complete derivation* is a derivation which starts with c_0 and ends in a final configuration c_f. In that case we have $c_0 \overset{*}{\underset{A}{\vdash}} c_f$.

The *language* $\mathcal{L}(A)$ *defined* (*generated*, *recognized*) by the FSA A is the set of all terminal strings w for which there exists a complete derivation. We say that an FSA is empty iff its language is empty. Two FSAs A and A' are *equivalent* iff they defined the same language.

An FSA is *ε-free* iff its transition relation has the form $\delta = \{(q_i, t, q_j) | q_i, q_j \in Q, t \in \Sigma\}$, except perhaps for a distinguished transition, the *ε-transition* which has the form (q_0, ε, q_f), $q_f \in F$ and allows the empty string ε to be in $\mathcal{L}(A)$. Every FSA can be transformed into an equivalent ε-free FSA.

An FSA $A = (Q, \Sigma, \delta, q_0, F)$ is *reduced* iff every element of δ appears in a complete derivation. A reduced FSA is empty iff we have $\delta = \emptyset$. We say that a non-empty reduced FSA is in *canonical form* iff its set of states Q and its set of terminal symbols Σ only contain elements that appear in the transition relation δ.[7] It is a well known result that every FSA A can be transformed in time linear with $|A|$ into an equivalent (canonical) reduced FSA A'.

2.3 Input strings and input DAGs

In many NLP applications[8] the source text cannot be considered as a single string of terminal symbols but rather as a finite set of terminal strings. These sets are finite languages which can be defined by particular FSAs. These particular type of FSAs are called *directed-acyclic graphs* (DAGs). In a DAG $w = (Q, \Sigma, \delta, q_0, F)$, the initial state q_0 is 1 and we assume that there is a single final state f ($F = \{f\}$), Q is a finite subset of the positive integers less than or equal to f: $Q = \{i | 1 \leq i \leq f\}$, Σ is the set of terminal symbols. For the transition relation δ, we

[5] For example, in the previous complete derivation d, let the right-hand side α be the (vocabulary) string $X_1 \cdots X_k \cdots X_p$ in which each symbol X_k derives the terminal string $x_k \in T^*$ (we have $X_k \overset{*}{\underset{G}{\Rightarrow}} x_k$ and $w_2 = x_1 \cdots x_k \cdots x_p$), then the instantiated production $A[i_0..i_p] \to X_1[i_0..i_1] \cdots X_k[i_{k-1}..i_k] \cdots X_p[i_{p-1}..i_p]$ with $i_0 = |w_1|+1$, $i_1 = i_0 + |x_1|, \ldots, i_k = i_{k-1} + |x_k| \ldots$ and $i_p = i_0 + |w_2|$ is an element of P_G^w.

[6] The popular notion of shared forests mainly comes from (Billot and Lang, 1989).

[7] We may say that the canonical form of the empty reduced FSA is $(\{q_0\}, \emptyset, \emptyset, q_0, \emptyset)$ though the initial state q_0 does not appear in any transition.

[8] Speech processing, lexical ambiguity representation, …

require that its elements (i, t, j) are such that $i < j$ (there are no loops in a DAG). Without loss of generality, we will assume that DAGs are ε-free reduced FSAs in canonical form and that any DAG w is noted by a triple (Σ, δ, f) since its initial state is always 1 and its set of states is $\{i \mid 1 \leq i \leq f\}$.

For a given CFG G, the recognition of an input DAG w is equivalent to the emptiness of its intersection with G. This problem can be solved in time linear in $|G|$ and cubic in $|Q|$ the number of states of w.

If the input text is a DAG, the previous notions of range, instantiations and parse forest easily generalize: the indices i and j which in the string case locate the positions of substrings are changed in the DAG case into DAG states. For example if $A[i_0..i_p] \to X_1[i_0..i_1] \cdots X_p[i_{p-1}..i_p]$ is an instantiated production of the parse forest for $G = (N, T, P, S)$ and $w = (\Sigma, \delta, f)$, we have $A \to X_1 \cdots X_p \in P$ and there is a path in the input DAG from state i_0 to state i_p via states $i_1, \ldots, i_{p-1}$.

Of course, any nonempty terminal string $w \in T^+$, may be viewed as a DAG (Σ, δ, f) where $\Sigma = \{t \mid w = w_1 t w_2 \wedge t \in T\}$, $\delta = \{(i, t, i+1) \mid w = w_1 t w_2 \wedge t \in T \wedge i = 1 + |w_1|\}$ and $f = 1 + |w|$. If the input string w is the empty string ε, the associated DAG is (Σ, δ, f) where $\Sigma = \emptyset$, $\delta = \{(1, \varepsilon, 2)\}$ and $f = 2$. Thus, in the sequel, we will assume that the inputs of our parsers are not strings but DAGs. As a consequence the size (or *length*) of a sentence is the size of its DAG (i.e., its number of transitions).

3 Filtering Strategies

3.1 Gold Strategy

Let $G = (N, T, P, S)$ be a CFG, $w = (\Sigma, \delta, f)$ be an input DAG of size $n = |\delta|$ and $\langle F_w \rangle = (\langle P_w \rangle, S[1..f])$ be the reduced output parse forest in canonical form. From $\langle P_w \rangle$, it is possible to extract a set of (reduced) uninstantiated productions $P_w^g = \{A \to X_1 \cdots X_p \mid A[i_0..i_p] \to X_1[i_0..i_1] X_2[i_1..i_2] \cdots X_p[i_{p-1}..i_p] \in \langle P_w \rangle\}$, which, together with the axiom S, defines a new reduced CFG $G_w^g = (P_w^g, S)$ in canonical form. This grammar is called the *gold* grammar of G for w, hence the superscript g. Now, if we use G_w^g to reparse the same input DAG w, we will get the same output forest $\langle F_w \rangle$. But in that case, we are sure that every production in P_w^g is used in at least one complete derivation. Now, if this process is viewed as a filtering strategy that computes a filtering function as introduced in Section 1, it is clear that this strategy is *size-optimal* in the sense that P_w^g is of minimal size, we call it the *gold* strategy and the associated gold filtering function is noted g. Since we do not want that a filtering strategy looses parses, the result $G_w^f = (P_w^f, S)$ of any filtering function f must be such that, for every sentence w, P_w^f is a superset of P_w^g. In other words the *recall score* of any filtering function f must be of 100%. We can note that the parsing pass which generates G_w^g may be led by any filtering strategy f.

As usual, the *precision score* (precision for short) of a filtering strategy f (w.r.t. the gold case) is, for a given w, defined by the quotient $\frac{|P_w^g|}{|P_w^f|}$ which expresses the number of useful productions selected by f on w (for some G).

However, it is clear that we are interested in strategies that are *time-optimal* and size-optimal strategies are not necessarily also time-optimal: the time taken at filtering-time to get a smaller grammar will not necessarily be won back at parse-time.

For a given CFG G, an input DAG w and a filtering strategy c, we only have to plot the times taken by the filtering pass and by the parsing pass to make some estimations on their average (median, decile) parse times and then to decide which is the winner. However, it may well happens that a strategy which has not received the award (with the sample of CFGs and the test sets tried) would be the winner in another context!

All the following filtering strategies exhibit necessary conditions that any production must hold in order to be in a parse.

3.2 The *make-a-reduced-grammar* Algorithm

An algorithm which takes as input any CFG $G = (N, T, P, S)$ and generates as output a strongly equivalent *reduced* CFG G' and which runs in $\mathcal{O}(|G|)$ can be found in many text books (See (Hopcroft and Ullman, 1979) for example).

So as to eliminate from all our intermediate subgrammars all useless productions, each filtering strategy will end by a call to such an algorithm named *make-a-reduced-grammar*.

The *make-a-reduced-grammar* algorithm works as follows. It first finds all *productive*[9] symbols. Afterwards it finds all *reachable*[10] symbols. A symbol is *useful* (otherwise *useless*) if it is both productive and reachable. A production $A \to X_1 \cdots X_p$ is *useful* (otherwise *useless*) iff all its symbols are useful. A last scan over the grammar erases all useless production and leaves the reduced form. The *canonical form* is reached in only retaining in the nonterminal and terminal sets of the sub-grammar the symbols which occur in the (useful) production set.

3.3 Basic Filtering Strategy: b-filter

The basic filtering strategy (b-filter for short) which is described in this section will always be tried the first. Thus, its input is the couple (G, w) where $G = (N, T, P, S)$ is the large initial CFG and the input sentence w is a reduced DAG in canonical form $w = (\Sigma, \delta, f)$ of size n. It generates a reduced CFG in canonical form noted $G^b = (P^b, S)$ in which the references to both G and w are assumed. Besides this b-filter, we will examine in Sections 3.4 and 3.5 two others filtering strategies named a and d. These filters will always have as input a couple (G^c, w) where $G^c = (P^c, S)$ is a reduced CFG in canonical form which has already been filtered by a previous sequence of strategies noted c. They generate a reduced CFG in canonical form noted $G^{cf} = (P^{cf}, S)$ with $f = a$ or $f = d$ respectively. Of course it may happens that G^{cf} is identical to G^c if the f-filter is not effective. A filtering strategy or a combination of filtering strategies may be applied several times and lead to a filtered grammar of the form say G^{ba^2da} in which the sequence ba^2da explicits the order in which the filtering strategies have been performed. We may even repeatedly apply a until a fixed point is reached before applying d, and thus get something of the form $G^{ba^\infty d}$.

The idea behind the b-filter is very simple and has largely been used in lexicalized formalisms parsing, in particular in LTAG (Schabes et al., 1988) parsing. The filter rejects productions of P which contain terminal symbols that do not occur in Σ (i.e., that are not terminal symbols of the DAG w) and thus takes

[9] $X \in V$ is productive iff we have $X \overset{*}{\underset{G}{\Rightarrow}} w, w \in T^*$.

[10] $X \in V$ is reachable iff we have $S \overset{*}{\underset{G}{\Rightarrow}} w_1 X w_2, w_1 w_2 \in T^*$.

$$
\begin{array}{lcll}
S & \to & AB & (1) \\
S & \to & BA & (2) \\
A & \to & a & (3) \\
A & \to & ab & (4) \\
B & \to & b & (5) \\
B & \to & bc & (6)
\end{array}
$$

Table 1: A simple grammar

$\mathcal{O}(|G|)$ time if we assume that the access to the elements of the terminal set Σ is performed in constant time. *Unlexicalized* productions whose right-hand sides are in N^* are kept. It also rejects productions in which several terminal symbol occurs, in an order which is not compatible with the linear order of the input.

Consider for example the set of productions shown in Table 1 and assume that the source text is the terminal string ab. It is clear that the b-filter will erase production 6 since c is not in the source text.

The execution of the b-filter produces a (non-reduced) CFG G' such that $|G'| \leq |G|$. However, it may be the case that some productions of G' are useless, it will thus be the task of the *make-a-reduced-grammar* algorithm to transform G' into its reduced canonical form G^b in time $\mathcal{O}(|G'|)$. The worst-case total running time of the whole b-filter pass is thus $\mathcal{O}(|G| \times n)$.

We can remark that, after the execution of the b-filter, the set of terminal symbols of G^b is a subset of $T \cap \Sigma$.

3.4 Adjacent Filtering Strategy: a-filter

As explained before, we assume that the input to the adjacent filtering strategy (a-filter for short) described in this section is a couple (G^c, w) where $G^c = (N^c, T^c, P^c, S)$ is a reduced CFG in canonical form. However, the a-filter would also work for a non-reduced CFG. As usual, we define the symbols of G^c as the elements of the vocabulary $V^c = N^c \cup T^c$.

The idea is to erase productions that cannot be part of any parses for w in using an adjacency criteria: if two symbols are adjacent in a rule, they must derive terminal symbols that are also adjacent in w. To give a (very) simple practical idea of what we mean by adjacency criteria, let us consider again the source string ab and the grammar defined in Table 1 in which the last production has already been erased by the b-filter.

The fact that the B-production ends with a b and that the A-productions all start with an a, implies that production 2 is in a complete parse only if the source text is such that b is immediately followed by a. Since it is not the case, production 2 can be erased.

More generally, consider a production of the form $A \to \cdots XY \cdots$. If for each couple $(a, b) \in T^2$ in which a is a terminal symbol that can terminate (the terminal strings generated by) X and b is a terminal symbol that can lead (the terminal strings generated by) Y, there is no transition on b that can follow a transition on a in the DAG w, it is clear that the production $A \to \cdots XY \cdots$ can be safely erased.

Now assume that we have the following (left) derivation $Y \overset{*}{\Rightarrow} Y_1\beta_1 \overset{*}{\Rightarrow} Y_i\beta_i \cdots \beta_1 \overset{*}{\Rightarrow} \cdots \overset{Y_{p-1} \to \alpha_p Y_p \beta_p}{\Rightarrow} \alpha_p Y_p \beta_p \cdots \beta_1 \overset{*}{\Rightarrow} Y_p \beta_p \cdots \beta_1$, with $\alpha_p \overset{*}{\Rightarrow} \varepsilon$. If for each couple (a, b') in which a has the previous definition and b' is a terminal symbol that can lead (the terminal strings generated by) Y_p, there is no transition on b' that can follow a transition on a in the DAG w, the production $Y_{p-1} \to \alpha_p Y_p \beta_p$ can be erased if it is not valid in another context.

Moreover, consider a (right) derivation of the form $X \overset{*}{\Rightarrow} \alpha_1 X_1 \overset{*}{\Rightarrow} \alpha_1 \cdots \alpha_i X_i \overset{*}{\Rightarrow} \cdots \overset{X_{p-1} \to \alpha_p X_p \beta_p}{\Rightarrow} \alpha_1 \cdots \alpha_p X_p \beta_p \overset{*}{\Rightarrow} \alpha_1 \cdots \alpha_p X_p$, with $\beta_p \overset{*}{\Rightarrow} \varepsilon$. If for each couple (a', b) in which b has the previous definition and a' is a terminal symbol that can terminate (the terminal strings generated by) X_p, there is no transition on b that can follow a transition on a' in the DAG w, the production $X_{p-1} \to \alpha_p X_p \beta_p$ can be erased if it is not valid in another context.

In order to formalize these notions we define several binary relations together with their (reflexive) transitive closure.

Within a CFG $G = (N, T, P, S)$, we first define *left-corner* noted $\llcorner$. Left-corner (Nederhof, 1993; Moore, 2000), hereafter *LC*, is a well-known relation since many parsing strategies are based upon it. We say that X is in the LC of A and we write $A \llcorner X$ iff $(A, X) \in \{(B, Y) \mid B \to \alpha Y \beta \in P \wedge \alpha \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. We can write $A \underset{A \to \alpha X \beta}{\llcorner} X$ to enforce how the couple (A, X) may be produced.

For its dual relation, *right-corner*, noted $\lrcorner$, we say that X is in the right corner of A and we write $X \lrcorner A$ iff $(X, A) \in \{(Y, B) \mid B \to \alpha Y \beta \in P \wedge \beta \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. We can write $X \underset{A \to \alpha X \beta}{\lrcorner} A$ to enforce how the couple (X, A) may be produced.

We also define the *first* (resp. *last*) relation noted $\hookrightarrow_t$ (resp. $\hookleftarrow_t$) by $\hookrightarrow_t = \{(X, t) \mid X \in V \wedge t \in T \wedge X \underset{G}{\overset{*}{\Rightarrow}} tx \wedge x \in T^*\}$ (resp. $\hookleftarrow_t = \{(X, t) \mid X \in V \wedge t \in T \wedge X \underset{G}{\overset{*}{\Rightarrow}} xt \wedge x \in T^*\}$).

We define the *adjacent* ternary relation on $V \times N^* \times V$ noted $\leftrightarrow$ and we write $X \overset{\sigma}{\leftrightarrow} Y$ iff $(X, \sigma, Y) \in \{(U, \beta, V) \mid A \to \alpha U \beta V \gamma \in P \wedge \beta \underset{G}{\overset{*}{\Rightarrow}} \varepsilon\}$. This means that X and Y occur in that order in the right-hand side of some production and are separated by a nullable string σ. Note that X or Y may or may not be nullable.

On the input DAG $w = (\Sigma, \delta, f)$, we define the *immediately precede* relation noted $<$ and we write $a < b$ for $a, b \in \Sigma$ iff $w_1 a b w_3 \in \mathcal{L}(w), w_1, w_3 \in \Sigma^*$.

We also define the *precede* relation noted $\ll$ and we write $a \ll b$ for $a, b \in \Sigma$ iff $w_1 a w_2 b w_3 \in \mathcal{L}(w), w_1, w_2, w_3 \in \Sigma^*$.We can note that $\ll$ is not the transitive closure of $<$.[11]

For each production $A \to \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ in P^c and for each symbol pairs (X_0, X_p) of non-nullable symbols s.t. $X_1 \cdots X_{p-1} \underset{G^c}{\overset{*}{\Rightarrow}} \varepsilon$, we compute two sets A_1 and A_2 of couples $(a, b), a, b \in T^c$ defined by $A_1 = \cup_{0 < i \leq p} = \{(a, b) \mid a \hookleftarrow_t X_0 \overset{X_1 \cdots X_{i-1}}{\leftrightarrow} X_i \hookrightarrow_t b\}$ and $A_2 = \cup_{0 \leq i < p} = \{(a, b) \mid a \hookleftarrow_t X_i \overset{X_{i+1} \cdots X_{p-1}}{\leftrightarrow} X_p \hookrightarrow_t b\}$. Any

[11] Consider the source string $bcab$ for which we have $a \overset{+}{<} c$, but not $a \ll c$.

pair (a, b) of A_1 is such that the terminal symbol a may terminate a phrase of X_0 while the terminal symbol b may lead a phrase of $X_1 \cdots X_p$. Since X_0 and X_p are not nullable, A_1 is not empty. If none of its elements (a, b) is such that $a < b$, the production $A \rightarrow \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ is useless and can be erased. Analogously, any pair (a, b) of A_2 is such that the terminal symbol a may terminate a phrase of $X_0 X_1 \cdots X_{p-1}$ while the terminal symbol b may lead a phrase of X_p. Since X_0 and X_p are not nullable, A_2 is not empty. If none of its elements (a, b) is such that $a < b$, the production $A \rightarrow \alpha X_0 X_1 \cdots X_{p-1} X_p \gamma$ is useless and can be erased. Of course if $X_1 \cdots X_{p-1} = \varepsilon$, we have $A_1 = A_2$.[12]

The previous method has checked some adjacent properties inside the right-hand sides of productions. The following will perform some analogous checks but at the beginning and at the end of the right-hand sides of productions.

Let us go back to Table 1 to illustrate our purpose. Recall that, with source text ab, productions 6 and 2 have already been erased. Consider production 4 whose left-hand side is an A, the terminal string ab that it generates ends by b. If we look for the occurrences of A in the right-hand sides of the (remaining) productions, we only find production 1 which indicates that A is followed by B. Since the phrases of B all start with b (See production 5) and since in the source text b does not immediately follow another b, production 4 can be erased.

In order to check that the input sentence w starts and ends by valid terminal symbols, we augment the adjacent relation with two elements $(\$, \varepsilon, S)$ and $(S, \varepsilon, \$)$ where \$ is a new terminal symbol which is supposed to start and to end every sentence.[13]

Let $Z \rightarrow \alpha U \beta$ be a production in P^c in which U is non-nullable and $\alpha \overset{*}{\underset{G_c}{\Rightarrow}} \varepsilon$. If X is a non-nullable symbol, we compute the set $L = \{(a, b) \mid a \hookleftarrow_t X \overset{\sigma}{\leftrightarrow} Y \overset{*}{\llcorner} Z \underset{Z \rightarrow \alpha U \beta}{\llcorner} U \hookrightarrow_t b\}$. Since G^c is reduced and since $\$ < S$, we are sure that the set $X \overset{\sigma}{\leftrightarrow} Y \overset{*}{\llcorner} Z$ is non-empty, thus L is also non-empty.[14]

We can associate with each couple $(a, b) \in L$ at least one (left) derivation of the form $X\sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 \sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 Y \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 Z \gamma_2 \overset{Z \rightarrow \alpha U \beta}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 \alpha U \beta \gamma_2 \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 w_4 U \beta \gamma_2 \overset{*}{\underset{G^c}{\Rightarrow}} w_0 a w_1 w_2 w_3 w_4 w_5 b \gamma_1 \beta \gamma_2$ in which $w_1 w_2 w_3 w_4 w_5 \in T^{c*}$. These derivations contains all possible usages of the production $Z \rightarrow \alpha U \beta$ in a parse. If for every couple $(a, b) \in L$, the statement $a \ll b$ does not hold, we can conclude that the production $Z \rightarrow \alpha U \beta$ is not used in any parse and can thus be deleted.

Analogously, we can check that the order of terminal symbols is compatible with both a production and its right grammatical context.

Let $Z \rightarrow \alpha U \beta$ be a production in P^c in which U is non-nullable and $\beta \overset{*}{\underset{G_c}{\Rightarrow}} \varepsilon$. If Y is a non-nullable symbol, we compute the set $R = \{(a, b) \mid a \hookleftarrow_t U \underset{Z \rightarrow \alpha U \beta}{\lrcorner} Z \overset{*}{\lrcorner} X \overset{\sigma}{\leftrightarrow} Y \hookrightarrow_t b\}$. Since G^c is reduced and since $S < \$$, we are sure that the set $Z \overset{*}{\lrcorner} X \overset{\sigma}{\leftrightarrow} Y$ is non-empty, thus R is also non-empty.[14]

To each couple $(a, b) \in R$ we can associate at least one (right) derivation of the form $X\sigma Y \overset{*}{\underset{G^c}{\Rightarrow}} X \sigma w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} X w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 Z w_3 w_2 w_1 b w_0 \overset{Z \rightarrow \alpha U \beta}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha U \beta w_3 w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha U w_4 w_3 w_2 w_1 b w_0 \overset{*}{\underset{G^c}{\Rightarrow}} \gamma_1 \alpha \gamma_2 a w_5 w_4 w_3 w_2 w_1 b w_0$ in which $w_5 w_4 w_3 w_2 w_1 \in T^{c*}$. These derivations contains all possible usages of the production $Z \rightarrow \alpha U \beta$ in a partial parse. If for every couple $(a, b) \in L$, the statement $a \ll b$ does not hold, we can conclude that the production $Z \rightarrow \alpha U \beta$ is not used in any parse and can thus be deleted.

Now, a call to the *make-a-reduced-grammar* algorithm produces a reduced CFG in canonical form named $G^{ca} = (N^{ca}, T^{ca}, P^{ca}, S)$.

[12] It can be shown that the previous check can be performed on (G^c, w) in worst-case time $\mathcal{O}(|G^c| \times |\Sigma|^3)$ (recall that $|\Sigma| \leq n$). This time reduces to $\mathcal{O}(|G^c| \times |\Sigma|^2)$ if the input sentence is not a DAG but a string.

[13] This is equivalent to assume the existence in the grammar of a *super-production* whose right-hand side has the form $\$S\$$.

[14] This statement does not hold any more if we exclude from P^c the productions that have been previously erased during the current a-filter. In that case, an empty set indicates that the production $Z \rightarrow \alpha U \beta$ can be erased.

3.5 Dynamic Set Automaton Filtering Strategy: d-filter

In (Boullier, 2003) the author has presented a method that takes a CFG G and computes a FSA that defines a regular superset of $\mathcal{L}(G)$. However his method would produce intractable gigantic FSAs. Thus he uses his method to dynamically compute the FSA at parse time on a given source text. Based on experimental results, he shows that his method called *dynamic set automaton* (DSA) is tractable. He uses it to *guide* an Earley parser (See (Earley, 1970)) and shows improvements over the non guided version. The DSA method can directly be used as a filtering strategy since the states of the underlying FSA are in fact sets of *items*. For a CFG $G = (N, T, P, S)$, an item (or dotted production) is an element of $\{[A \to \alpha.\beta] \mid A \to \alpha\beta \in P\}$. A *complete* item has the form $[A \to \gamma.]$, it indicates that the production $A \to \gamma$ has been, in some sense, recognized. Thus, the complete items of the DSA states gives the set of productions selected by the DSA. This selection can be further refined if we also use the mirror DSA which processes the source text from right to left and if we only select complete items that both belong to the DSA and to its mirror.

Thus, if we assume that the input to the DSA filtering strategy (d-filter) is a couple (G^c, w) where $G^c = (P^c, S)$ is a reduced CFG in canonical form, we will eventually get a set of productions which is a subset of P^c. If it is a strict subset, we then apply the *make-a-reduced-grammar* algorithm which produces a reduced CFG in canonical form named $G^{cd} = (P^{cd}, S)$.

The Section 4 will give measures that may help to compare the practical merits of the a and d-filtering strategies.

4 Experiments

The measures presented in this section have been taken on a 1.7GHz AMD Athlon PC with 1.5 Gb of RAM running Linux. All parsers are written in C and have been compiled with gcc 2.96 with the *O2* optimization flag.

4.1 Grammars and corpus

We have performed experiments with two large grammars described below. The first one is an automatically generated CFG, the other one is the CFG equivalent of a TIG automatically extracted from a factorized TAG.

The first grammar, named $G^{T>N}$, is a variant of the CFG backbone of a large-coverage LFG grammar for French used in the French LFG parser described in (Boullier and Sagot, 2005). In this variant, the set T of terminal symbols is the whole set of French inflected forms present in the Le*fff*, a large-coverage syntactic lexicon for French (Sagot et al., 2006). This leads to as many as 407,863 different terminal symbols and 520,711 lexicalized productions (hence, the average number of categories — which are here non-terminal symbols — for an inflected form is 1.27). Moreover, this CFG entails a non-neglectible amount of syntactic constraints (including over-generating sub-categorization frame checking), which implies as many as $|P_u| = 19,028$ non-lexicalized productions. All in all, $G^{T>N}$ has 539,739 productions.

The second grammar, named G^{TIG}, is a CFG which represents a TIG. To achieve this, we applied (Boullier, 2000)'s algorithm on the unfolded version of (Villemonte de La Clergerie, 2005)'s factorized TAG. The number of productions in G^{TIG} is comparable to that of $G^{T>N}$. However, these two grammars are completely different. First, G^{TIG} has much less terminal and non-terminal symbols than $G^{T>N}$. This means that the basic filter may be less efficient on G^{TIG} than on $G^{T>N}$. Second, the size of G^{TIG} is enormous (more than 10 times that of $G^{T>N}$), which shows that right-hand sides of G^{TIG}'s productions are huge (the average number of right-hand side symbols is more than 24). This may increase the usefulness of a- and d-filtering strategies.

Global quantitative data about these grammars is shown in Table 2.

Both grammars, as evoked in the introduction, have not been written by hand. On the contrary, they are automatically generated from a more abstract and more compact level (a meta-level over LFG for $G^{T>N}$, and a metagrammar for G^{TIG}). These grammars are not artificial grammars set up only for this experiment. On the contrary, they are automatically generated huge real-life CFGs that are variants of grammars used in real NLP applications.

Our test suite is a set of 3093 French journalistic sentences. These sentences are the *general_lemonde*

G	$\|N\|$	$\|T\|$	$\|P\|$	$\|P_u\|$	$\|G\|$
$G^{T>N}$	7,862	407,863	539,739	19,028	1,123,062
G^{TIG}	448	173	493,408	4,338	12,455,767

Table 2: Sizes of the grammars $G^{T>N}$ and G^{TIG} used in our experiments

part of the EASy parsing evaluation campaign corpus. Raw sentences have been turned into DAGs of inflected forms known by both grammar/lexicon couples.[15] This step has been achieved by the pre-syntactic processing chain SxPipe (Sagot and Boullier, 2005). They are all recognized by both grammars.[16] The resulting DAGs have a median size of 28 and an average size of 31.7.

Before entering into details, let us give here the first important result of these experiments: it was actually possible to build parsers out of $G^{T>N}$ and G^{TIG} and to parse efficiently with the resulting parsers (we shall detail later on efficiency results). Given the fact that we are dealing with grammars whose sizes are respectively over 1,000,000 and over 12,000,000, this is in itself a very satisfying result.

4.2 Precision results

Let us recall informally that the precision of a filtering strategy is the proportion of productions in the resulting sub-grammar that are in the gold grammar, i.e., that have effectively instantiated counterparts in the final parse forest.

We have applied different strategies so as to compare their precisions. The results on $G^{T>N}$ and G^{TIG} are summed up in Table 3. These results give several valuable results. First, as we expected, the basic b-filter drastically reduces the size of the grammar. The result is even better on $G^{T>N}$ thanks to its large number of terminal symbols. Second, both the adjacency a-filter and the DSA d-filter efficiently reduce the size of the grammar: on $G^{T>N}$, the a-filter eliminates 20% of the productions they receive as input, a bit less for the d-filter. Indeed, the a-filter performs better than the d-filter introduced in (Boullier, 2003), at least as precision is concerned. We shall see later that this is still the case on global parsing times. However, applying the d-filter after the a-filter still removes a non-neglectible amount of productions:[17] each technique is able to eliminate productions that are kept by the other one. The result of these filters is suprisingly good: in average, after all filters, only approx. 20% of the productions that have been kept will not be successfully instantiated in the final parse forest. Third, the adjacency filter can be used in its one-pass mode, since almost all the benefit from the full (fix-point) mode is already reached after the first application. This is practically a very valuable result, since the one-pass mode is obviously faster than the full mode.

Strategy	Average precision $G^{T>N}$	G^{TIG}
no filter	0.04%	0.03%
b	62.87%	39.43%
bd	74.53%	66.56%
ba	77.31%	66.94%
ba^{∞}	77.48%	67.48%
bad	80.27%	77.16%
$ba^{\infty}d$	80.30%	77.41%
gold	*100%*	*100%*

Table 3: Average precision of six different filtering strategies on our test corpus with $G^{T>N}$ and G^{TIG}.

However, all these filters do require computing time, and it is necessary to evaluate not only the precision of these filters, but also their execution time as well as the influence they have on the global (including filtering) parsing time .

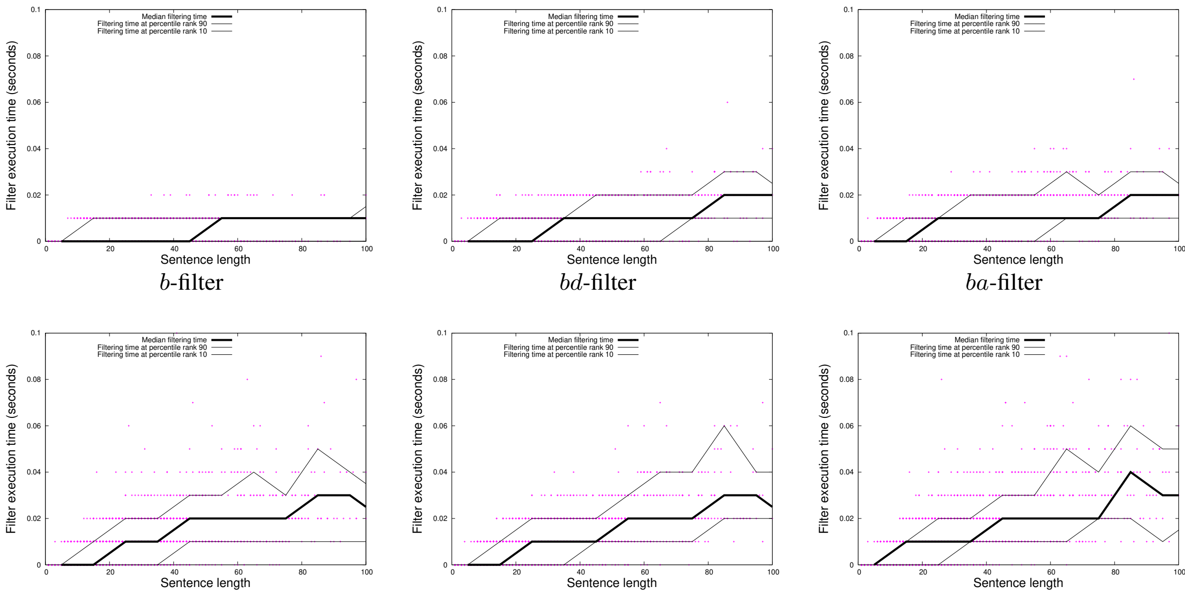
4.3 Parsing time and best filter

Filter execution times for the six filtering strategies introduced in Table 3 are illustrated for $G^{T>N}$ in Figure 1. These graphics show three extremely valuable pieces of information. First, filtering times are extremely low: the average filtering time for the slowest filter ($ba^{\infty}d$, i.e., basic plus full adjacency plus DSA) on 40-word sentences is around 20 ms. Second, on small sentences, filtering times are virtually zero. This is important, since it means that there

[15] As seen above, inflected forms are directly terminal symbols of $G^{T>N}$, while G^{TIG} uses a *lexicon* to map these inflected forms into its own terminal symbols, thereby possibly introducing lexical ambiguity.

[16] Approx. 15% of the original set of sentences were not recognized, and required error recovery techniques; we decided to discard them for this experiment.

[17] Although not reported here, applying the a before d leads to the same conclusion.



b-filter   *bd*-filter   *ba*-filter

ba^∞-filter   *bad*-filter $ba^\infty d$-filter

Figure 1: Filtering times for six different strategies with $G^{T>N}$

is almost no fixed cost to pay when we use these filters (let us recall that without any filter, building efficient parsers for such a huge grammar is highly problematic). Third, all these filters, at least when used with $G^{T>N}$, are executed in a time which is *linear* w.r.t. the size of the input sentence (i.e., the size of the input DAG).

The results on G^{TIG} lead to the same conclusions, with one exception: with this extremely huge grammar with so long right-hand sides, the basic filter is not as fast as on $G^{T>N}$ (and not as precise, as we will see below, which slows down the *make-a-reduced-grammar* algorithm since it is applied on a larger filtered grammars). For example, the median execution time for the basic filter on sentences whose size is approximately 40 is 0.25 seconds, to be compared with the 0.00 seconds reached on $G^{T>N}$ (this zero value means a median time strictly lower than 0.01 seconds, which is the granularity of our time measurments).

Figure 2 and 3 show the global (filtering+parsing) execution time for the 6 different filters. We only show median times computed on classes of sentences of length $10i$ to $10(i+1)-1$ and plotted with a centered x-coordinate $(10(i+1/2))$, but results with other percentiles or average times on the same classes draw the same overall picture.

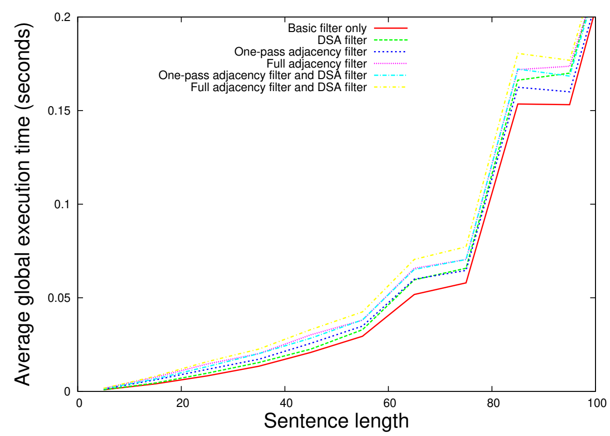


Figure 2: Global (filtering+parsing) times for six different strategies with $G^{T>N}$

One can see that the results are completely different, showing a strong dependency on the characteristics of the grammar. In the case of $G^{T>N}$, the huge number of terminal symbols and the reasonable average size of right-hand sides of productions, the basic filtering strategy is the best strategy: although it is fast because relatively simple, it reduces the grammar extremely efficiently (it has a 60.56% precision, to be compared with the precision of the void filter which is 0.04%). Hence, for $G^{T>N}$, our only result

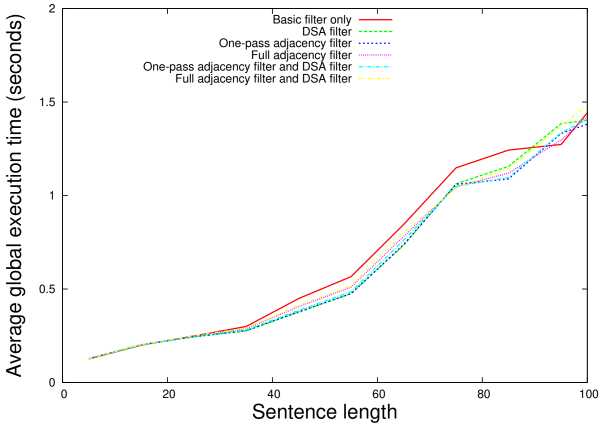


Figure 3: Global (filtering+parsing) times for six different strategies with G^{TIG}

is that this basic filter does allow us to build an efficient parser (the most efficient one), but that refined additionnal filtering strategies are not useful.

The picture is completely different with G^{TIG}. Contrary to $G^{T>N}$, this grammar has comparatively very few terminal and non-terminal symbols, and very long right-hand sides. These two facts lead to a lower precision of the basic filter (39.43%), which keeps many more productions when applied on G^{TIG} than when applied on $G^{T>N}$, and leads, when applied alone, to the less efficient parser. This gives to the adjacency filter much more opportunity to improve the global execution time. However, the complexity of the grammar makes the construction of the DSA filter relatively costly despite its precision, leading to the following conclusion: on G^{TIG} (and probably on any grammar with similar characteristics), the best filtering strategy is the one-pass adjacency strategy. In particular, this leads to an improvement over the work of (Boullier, 2003) which only introduced the DSA filter. Incidentally, the extreme size of G^{TIG} leads to much higher parsing times, approximately 10 times higher than with $G^{T>N}$, which is consistent with the ratio between the sizes of both involved grammars.

5 Conclusion

It is a well known result in optimization techniques that the key to practically improve these processes is to reduce their search space. This is also the case in parsing and in particular in CF parsing.

Many parsers process their inputs from left to right but we can find in the literature other parsing strategies. In particular, in NLP, (van Noord, 1997) and (Satta and Stock, 1994) propose bidirectional algorithms. These parsers have the reputation to have a better efficiency than their left-to-right counterpart. This reputation is not only based upon experimental results (van Noord, 1997) but also upon mathematical arguments in (Nederhof and Satta, 2000). This is specially true when the productions of the CFG strongly depend on lexical information. In that case the parsing search space is reduced because the constraints associated to lexical elements are evaluated as early as possible. We can note that our filtering strategies try to reach the same purpose by a totally different mean: we reduce the parsing search space by eliminating as many productions as possible, including possibly non-lexicalized productions whose irrelevance to parse the current input can not be directly deduced from that input.

We can also remark that our results are not in contradiction with the claims of (Nederhof and Satta, 2000) in which they argue that "Earley algorithm and related standard parsing techniques [...] cannot be directly extended to allow left-to-right and correct-prefix-property parsing in acceptable time bound". First, as already noted in Section 1, our method does not work for any large CFG. In order to work well, the first step of our basic strategy must filter out a great amount of (lexicalized) productions. To do that, it is clear that the set of terminals in the input text must select a small ratio of lexicalized productions. To give a more concrete idea we advocate that the selected productions produce roughly a grammar of *normal* size out of the large grammar. Second, our method as a whole clearly does not process the input text from left-to-right and thus does not enter in the categories studied in (Nederhof and Satta, 2000). Moreover, the authors bring strong evidences that in case of polynomial-time off-line compilation of the grammar, left-to-right parsing cannot be performed in polynomial time, independently of the size of the lexicon. Once again, if our filter pass is viewed as an off-line processing of the large input grammar, our output is not a compilation of the large grammar, but a (compilation of a) smaller grammar, specialized in (some abstractions of) the source text only. In other words their negative results do not necessarily apply to our specific case.

The experiment campaign as been conducted in using an Earley-like parser.[18] We have also successfuly tried the coupling of our filtering strategies with a CYK parser (Kasami, 1967; Younger, 1967) as post-processor. However the coupling with a GLR parser (See (Satta, 1992) for example) is perhaps more problematic since the time taken to build up the underlying nondeterministic LR automaton from the sub-grammar can be prohibitive.

Though no definitive answer can be made to the question asked in the title, we have shown that, in some cases, the answer is certainly *yes*.

References

Sylvie Billot and Bernard Lang. 1989. The structure of shared forests in ambiguous parsing. In *Meeting of the Association for Computational Linguistics*, pages 143–151.

Pierre Boullier and Benoît Sagot. 2005. Efficient and robust LFG parsing: SxLfg. In *Proceedings of IWPT'05*, pages 1–10, Vancouver, Canada.

Pierre Boullier. 2000. On TAG parsing. *Traitement Automatique des Langues (T.A.L.)*, 41(3):759–793.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of IWPT 03*, pages 43–54, Nancy, France.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communication of the ACM*, 13(2):94–102.

Jeffrey D. Hopcroft and John E. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass.

Aravind Joshi. 1997. Parsing techniques. In *Survey of the state of the art in human language technology*, pages 351–356. Cambridge University Press, New York, NY, USA.

Tadao Kasami. 1967. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65–758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts, USA.

Robert C. Moore. 2000. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of IWPT 2000*, pages 171–182, Trento, Italy. Revised version at `http://www.cogs.susx.ac.uk/lab/nlp/carroll/cfg-resources/iwpt2000-rev2.ps`.

Mark-Jan Nederhof and Giorgio Satta. 2000. Left-to-right parsing and bilexical context-free grammars. In *Proceedings of the first conference on North American chapter of the ACL*, pages 272–279, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Mark-Jan Nederhof. 1993. Generalized left-corner parsing. In *Proceedings of the sixth conference on European chapter of the ACL*, pages 305–314, Morristown, NJ, USA. ACL.

Benoît Sagot and Pierre Boullier. 2005. From raw corpus to word lattices: robust pre-parsing processing. In *Proceedings of L&TC 2005*, pages 348–351, Poznań, Poland.

Benoît Sagot, Lionel Clément, Éric Villemonte de La Clergerie, and Pierre Boullier. 2006. The Lefff 2 syntactic lexicon for french: architecture, acquisition, use. In *Proc. of LREC'06*.

Giorgio Satta and Oliviero Stock. 1994. Bidirectional context-free grammar parsing for natural language processing. *Artif. Intell.*, 69(1-2):123–164.

Giorgio Satta. 1992. Review of "generalized lr parsing" by masaru tomita. kluwer academic publishers 1991. *Comput. Linguist.*, 18(3):377–381.

Yves Schabes and Richard C. Waters. 1995. Tree insertion grammar: Cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Comput. Linguist.*, 21(4):479–513.

Yves Schabes, Anne Abeillé, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Comput. Linguist. (COLING'88)*, Budapest, Hungary.

Gertjan van Noord. 1997. An efficient implementation of the head-corner parser. *Comput. Linguist.*, 23(3):425–456.

Éric Villemonte de La Clergerie. 2005. From metagrammars to factorized TAG/TIG parsers. In *Proceedings of IWPT'05*, pages 190–191, Vancouver, Canada.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

[18] Contrarily to classical Earley parsers, its *predictor* phase uses a pre-computed structure which is roughly an LC relation. Note that this feature forces our filters to compute an LC relation on the generated sub-grammar. This also shows that LC parsers may also benefit from our filtering techniques.