# GUIDED EARLEY PARSING

**Pierre Boullier**
INRIA-Rocquencourt
BP 105
78153 Le Chesnay Cedex, France
Pierre.Boullier@inria.fr

**Abstract**

In this paper, we present a method which may speed up Earley parsers in practice. A first pass called a *guiding parser* builds an intermediate structure called a *guide* which is used by a second pass, an Earley parser, called a *guided parser* whose Predictor phase is slightly modified in such a way that it selects an initial item only if this item is in the guide. This approach is validated by practical experiments performed on a large test set with an English context-free grammar.

## 1 Introduction

The investigation of approaches that allow practical parsing speed-ups is an important topic of research, especially in real-word applications based upon large-scale grammars. It seems difficult to find a technique that would improve the throughput of context-free (CF) parsers, due to the huge amount of research that has already been performed by the parsing community. First, we must stress that we are not looking for a new CF parsing strategy with a better theoretical parse-time complexity, but will only propose a technique which, in some cases (i.e., for certain grammars or for certain input sentences) improves practical parse-times.

Though the results reported in [12] are not very encouraging, we decided to investigate the usage of guiding techniques in CF parsing.

When a nondeterministic choice occurs during a nondeterministic process, one usually 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 which directs its way. This assistant may be either a guide or an oracle. An *oracle* always indicates all the right ways that will eventually lead to success, and only these ways, whereas a *guide* will indicate all the good ways but may also indicate some wrong ways. In other words, an oracle is a perfect guide.

Of course, nondeterministic parsers are the most likely candidates to be guided. Parsing with a guide proceeds as follows. The guided process is split into two phases. The first *guiding parser* builds a structure called the *guide*, while the second phase, the *guided parser* proper, looks into the guide in order to help (some of) its nondeterministic choices.

In this paper, we will see how a (slightly modified) general CF parser can in practice be speeded up using guiding techniques.

We choose to guide Earley type parsers [9], the guide being called during its *Predictor* phase: an initial Earley item $[A \rightarrow .\alpha, i]$ is added to a given table $T_i$, for a given source index $i$, only

if that initial item is in the guide.[1]

For a given guided parser, the choice of a guide only fixes the output of the corresponding guiding parser, still leaving a degree of freedom in the choice of this guiding parser.

In this paper we will define several guiding parsers, and for each of them we will address two issues: on the one hand, we will quantify its quality (i.e., how far from an oracle the corresponding guide is) and, on the other hand, we will compare the total run-times of the pair guiding parser/guided recognizer to the run-time of a standard (non guided) Earley recognizer. These measures will be performed on a large test set parsed by a guided Earley parser for a large context-free grammar (CFG) extracted from a wide coverage English tree-adjoining grammar.

Unless otherwise stated, for a CFG $(N, T, P, S)$, terminal symbols in $T$ are denoted by lower case letters such as $a, t, \ldots$, nonterminal symbols in $N$ are denoted by upper case letters such as $A, B, \ldots$, while strings of symbols in $(N \cup T)^*$ are denoted by Greek letters such as $\alpha, \Gamma$.

## 2 The Oracle

In this Section, we define both oracle and guides, together with their precisions.

Let $G = (N, T, P, S)$ be a CFG and let $w = a_1 \ldots a_n$ be an input string of length $n \geq 0$. We say that a production $A \rightarrow \alpha \in P$ is *useful at* $i$ for some $w$ or equivalently that an (initial) Earley item $[A \rightarrow .\alpha, i]$ is *useful* for some $w$, if and only if there is a complete derivation of the form $S \underset{G}{\overset{*}{\Rightarrow}} \Gamma_1 A \Gamma_2 \underset{G}{\Rightarrow} \Gamma_1 \alpha \Gamma_2 \underset{G}{\overset{*}{\Rightarrow}} w_1 w_2 = w$ in which $\Gamma_1 \underset{G}{\overset{*}{\Rightarrow}} w_1 = a_1 \ldots a_i$ and $\alpha \Gamma_2 \underset{G}{\overset{*}{\Rightarrow}} w_2 = a_{i+1} \ldots a_n$. In other words, in the parse forest for $w$, there is a subtree rooted at $A \rightarrow \alpha$ which spans a substring of $w$ whose leftmost position in $w$ is at index $i$.

As suggested in the introduction, for a given string $w$, we define a *guide* $\mathcal{G}^w$ as a set of initial Earley items which contains the set of all useful items for $w$. During the Predictor phase of a guided Earley recognizer, an initial Earley item $[A \rightarrow .\alpha, i]$ not in $\mathcal{G}^w$, will not be added to the table $T_i$. The guide $\mathcal{G}^w$ which is defined as the set of all useful items for $w$ is called *oracle* and is noted $\mathcal{O}^w$. The *size* $|\mathcal{G}^w|$ of a guide $\mathcal{G}^w$ is its cardinality.

As usual, the *precision score* (precision for short) of a guide (w.r.t. the oracle) is, for a given $w$, defined by the quotient $\frac{|\mathcal{O}^w \cap \mathcal{G}^w|}{|\mathcal{G}^w|}$. However, since by definition we have $\mathcal{O}^w \subseteq \mathcal{G}^w$, this precision reduces to $\frac{|\mathcal{O}^w|}{|\mathcal{G}^w|}$.

By definition of a guide, its *recall score* is 100%, since all useful items are in $\mathcal{G}^w$.

For a test set $\mathcal{S}$, the *average precision* of a guide is defined by $\frac{\sum_{w \in \mathcal{S}} |\mathcal{O}^w|}{\sum_{w \in \mathcal{S}} |\mathcal{G}^w|}$.

Therefore, in order to evaluate the average precision of the various guides that we will define below, we first need to measure the number $|\mathcal{O}^w|$, for each input string $w$ in some test set $\mathcal{S}$. This can be done very easily by using a complete Earley parser and extracting from each parse forest for $w$ the number $|\mathcal{O}^w|$.

We can remark that, in the light of this presentation, it is also possible to measure the precision of the Predictor phase of a standard Earley recognizer: for a given $w$, we only have

[1] The filtering out of productions at their very beginning in the Predictor phase seems more efficient than their elimination, after complete recognition, in the Completer phase. This remark also explains why we choose Earley parsers rather than GLR parsers, since, in GLR parsing, the only place where a call to a guide can be inserted seems to be at reduction time, when the production has been validated. Note that the Scanner phase could also have been a judicious action to be guided. Moreover, the technique reported here could be applied to left-corner (chart) parsers as well (descriptions and practical experiments of left-corner parsers can be found in [13]).

to count the number of initial Earley items that are produced during its Predictor phases.

# 3 The Guides

## 3.1 Lexicalization Guiding

We say that a production of a CFG is *lexicalized* if it contains a terminal symbol. A CFG is *lexicalized* if all its productions are lexicalized. In a lexicalized grammar terminal symbols are called *anchors*. By extension, we will use the word anchor as a synonym of terminal symbol. Lexicalized formalisms are popular in NL processing because the rewriting rules in which an anchor appears give all its valid syntactic constructions. As an example, tree-adjoining grammars (TAGs) are very often lexicalized TAGs (LTAGs) (See [10] for an introduction).

From a computational point of view, some parsing algorithms may take advantage of lexicalized grammars: each word of a given text first selects in a (usually) huge grammar only the matching rules. This filtering process generally results in a much smaller grammar which may become more manageable. In order to further improve the efficiency of parsers for lexicalized formalisms, Joshi and Srinivas have introduced in [11] the idea of supertagging: for each word, given its context in a sentence, its appropriate descriptions are selected. However, since this selection is usually based upon statistical distributions, parser correctness is not preserved. Regarding this issue (see [7]), we also present some correctness-preserving supertagging approaches which rely heavily upon our guiding techniques. When a parsing technique, more or less directly, uses the input grammar, as is the case in Earley-type parsers, the lexicalized filtering process or even the supertagging approach may both be considered as guiding parsers: their output guides are smaller grammars. However, this is only a particular case of guiding since the knowledge of a smaller set of productions may well be of little interest for some parsing techniques which need other kinds of guiding information. As an example, we can find in [2] the informations needed by guided RCG parsers for tree-adjoining languages.

In this Section, we see how it is possible to use the lexicalized productions of a (non-lexicalized) CFG in a guided parser.

For a CFG $(N, T, P, S)$ and a string $w = a_1 \dots a_n$, the set of *selected* productions $P_w$ is the subset of $P$ defined by $P_w = \{A \to \sigma \mid A \to \sigma \in P \text{ and } \sigma \in N^*\} \cup \{A \to \sigma_0 t_1 \sigma_1 \dots t_j \sigma_j \mid A \to \sigma_0 t_1 \sigma_1 \dots t_j \sigma_j \in P \text{ and } \sigma_k \in N^*, 0 \leq k \leq j \text{ and } t_h \in T, 1 \leq h \leq j \text{ and } \exists (i_1, \dots, i_j), 1 \leq i_1 < \dots < i_j \leq n \text{ such that } \forall h, 1 \leq h \leq j, a_{i_h} = t_h\}$. In other words, for a given input $w$, a production is selected either if it is not lexicalized or if all its anchors appear in $w$ in the correct order.

The processor which, from an input $w$ and a CFG $(N, T, P, S)$, builds the set of selected productions $P_w$ and outputs the set $\mathcal{G}^w = \{[A \to .\alpha, i] \mid 0 \leq i \leq |w| \text{ and } A \to \alpha \in P_w\}$ is a *guiding parser* which is called Lex1 in our experiment.

Note that in Lex1, the selected productions are independent of any source index. Of course, as the guiding parser proceeds from left to right, the length of the suffix of the text not yet processed decreases, that is the number of productions which, at some source index, are able to start also decreases. This phenomenon may be taken into account to improve the guide's precision. Therefore, for a given source index $i, 0 \leq i \leq n$, the set of *index selected* productions $P_w^i$ is defined by $P_w^i = \{A \to \sigma \mid A \to \sigma \in P \text{ and } \sigma \in N^*\} \cup \{A \to \sigma_0 t_1 \sigma_1 \dots t_j \sigma_j \mid A \to$

$\sigma_0 t_1 \sigma_1 \ldots t_j \sigma_j \in P$ and $\sigma_k \in N^*, 0 \leq k \leq j$ and $t_h \in T, 1 \leq h \leq j$ and $\exists (i_1, \ldots, i_j), i+1 \leq i_1 < \ldots < i_j \leq n$ such that $\forall h, 1 \leq h \leq j, a_{i_h} = t_h\}$. In other words, for a given input, a production is selected at $i$ either if it is not lexicalized or if all its anchors appear in the suffix $a_{i+1} \ldots a_n$ of this input in the correct order.

This improved guiding parser is called Lex2 in our experiment.

## 3.2 Regular Guiding

In [12], Martin Kay challenged the parsing community "*to construct a weakened version of a particular grammar that will serve as an effective guide*". In this paper, we take up this challenge.

As a *weakened* version, we decided to design regular supersets of context-free languages (CFLs), mainly because sentences of regular languages can be recognized in linear time.[2]

The design of regular supersets (or more generally regular approximations) of CFLs is not new (see [15] for example) and even their usage in CF parsing (see [14]) has been attempted. These approximations of CFLs mainly rely on two methods. A transformation occurs either at the grammar level or at the pushdown automaton (PDA) level. In the first case, CFGs are transformed by exploiting some necessary condition for a CFG to define a regular language.[3] In the second case, a PDA is first built from the CFG, and then that PDA is approximated by a finite-state automaton (FA). However, for a given real size CFG, the associated regular grammar or the associated FA are often too big, or the total number of operations performed by a pair guiding parser/guided parser seems to exceed the number of operations performed by a standard parser.

Despite these rather discouraging results, we decided to investigate the FA method.[4] The superset approximation process which transforms a PDA into an FA always consists in reducing an infinite number of parse-stacks into a finite one, by defining an equivalence relation on parse-stacks which induces a finite partition. Each element of that partition is then mapped to a state of the FA. For example, we could say that two parse-stacks are equivalent if and only if they share their $h$ top-most elements for a given integer $h$.[5] Of course, the precision of the PDA increases with $h$, but, for a large grammar (say more than 10 000 symbol occurrences), only the value $h = 1$ is practicable. For large CFGs, the challenge is thus to define regular supersets by means of FAs that are both manageable and whose number of operations is favorably counterbalanced by the number of operations saved in the guided parser.

Let $G = (N', T, P', S'), N' = N \cup \{S'\}, S' \notin N, P' = P \cup \{S' \rightarrow S\}$ be an augmented CFG. Let $I_G = \{A \rightarrow \alpha.\beta \mid A \rightarrow \alpha\beta \in P'\}$ be its set of *items* (dotted rules). A *complete item* has the form $A \rightarrow \alpha.$, a *nonterminal item* has the form $A \rightarrow \alpha.B\beta$, while a *terminal item* has the form $A \rightarrow \alpha.t\beta$.

Let $I$ and $J$ be two subsets of $I_G$, close-reduce$(I, J)$ is defined as the smallest set $K$ such that $J \subseteq K \subseteq I_G$ by:

1. Initially $K = J$;

[2] Of course, some other linear run-time supersets could also be studied.

[3] Recall that it is undecidable whether the language defined by a CFG is regular or not.

[4] Though the grammar transformation method can be more easily tuned, it seems that it produces less precise approximations.

[5] If the PDA is an LR automaton, for $h = 1$, we have the method described in [1], while for $h > 1$, we have the methods of [4] and [3].

2. **closure**: If $A \to \alpha.B\gamma \in K$, then add $B \to .\beta$ to $K$ for each $B \to \beta \in P$;

3. **reduce**: If $A \to \alpha. \in K$, and if there exists $B \to \beta.A\gamma \in (I \cup K)$ then add $B \to \beta A.\gamma$ to $K$;

4. Steps 2) and 3) are repeated until no new item is added to $K$.

From $G = (N', T, P', S')$, we define a deterministic FA $\mathcal{A}_G = (Q, T, \delta, q_0, F)$ as follows. Its set of states is $Q = \{[I, J] \mid I, J \in 2^{I_G}\}$. For a state $q = [I, J]$, its first component $I$ is a set of items called its *control* set, while its second component $J$ is called its *active* set. The initial state $q_0$ is the pair $[I_0, I_0]$ with $I_0 = \text{close-reduce}(\emptyset, \{S' \to .S\})$. The set of final states is $F = \{[I, J] \mid S' \to S. \in J\}$. Let $[I, J]$ be a state in $Q$, for each $a \in T$, we write, for the (deterministic) transition function $\delta$, $\delta([I, J], a) = [I', J']$ where $I' = I \cup J_a$, $J' = J_a$ and $J_a = \text{close-reduce}(I, \{A \to \alpha a.\beta \mid A \to \alpha.a\beta \in J\})$.

Informally, we see that in each state $[I, J]$, the active set $J$ is responsible for the terminal transitions[6] while the intent behind the control set $I$ is to play the role of a *degenerated parse-stack*. If we follow a path from the initial state, we see that the control sets of the traversed states do not decrease. In fact, they (mainly) record the nonterminal items that have already been seen since the initial state. These nonterminal items are used during the reduce phase (see step 3) to perform nonterminal $A$-transitions only on recorded $A$-items. By analogy with the notion of PDA, and by reference to the control and active sets, $\mathcal{A}_G$ is called a *set automaton* (SA).

As usual, we define a *configuration* as an element of $Q \times T^*$ and a binary relation between configurations, noted $\underset{\mathcal{A}_G}{\vdash}$ by $(q, tx) \underset{\mathcal{A}_G}{\vdash} (q', x)$, if and only if $\delta(q, t) = q'$. Let $w = a_1 \dots a_n$ be a terminal string and let $w_i = a_{i+1} \dots a_n$ be its suffixes (we have $w_0 = w$ and $w_n = \varepsilon$). The *initial configuration* is noted $c_0$ and is the pair $(q_0, w_0)$. For each $i$, $0 < i < n$, we define $c_i$ as a configuration of the form $(q_i, w_i)$ such that $c_0 \underset{\mathcal{A}_G}{\vdash} c_1 \dots \underset{\mathcal{A}_G}{\vdash} c_i$ (that is $q_i = \delta(q_{i-1}, a_i)$). The *final configuration* $c_n$ has the form $(q_n, w_n)$, with the supplementary condition that $q_n$ must be a final state in $F$. We say that $w$ is *recognized* by $\mathcal{A}_G$ if and only if $c_n$ exists (we have $c_0 \underset{\mathcal{A}_G}{\vdash} c_1 \dots \underset{\mathcal{A}_G}{\vdash} c_n$).

It can be shown that $\mathcal{L}(\mathcal{A}_G)$, the language *accepted* by $\mathcal{A}_G$ (i.e., the set of all its recognized strings) is a regular superset of $\mathcal{L}(G)$.[7] More precisely, if $c_i = ([I, J], w_i)$, if some initial item $A \to .\alpha$ is an element of $J$ and if its RHS $\alpha$ can derive the substring $a_{i+1} \dots a_j$ of $w$ starting at index $i$ (i.e., $\alpha \overset{*}{\underset{G}{\Rightarrow}} a_{i+1} \dots a_j$), then we have $c_j = ([I', J'], w_j)$ and the complete item $A \to \alpha.$ is an element of $J'$. Conversely, if a complete item $A \to \alpha.$ is an element of $J'$ for some configuration $c_j = ([I', J'], w_j)$, there exists a configuration $c_i = ([I, J], w_i)$, $i \leq j$ (i.e., $c_0 \overset{*}{\underset{\mathcal{A}_G}{\vdash}} c_i \overset{*}{\underset{\mathcal{A}_G}{\vdash}} c_j$) such that $J$ contains the initial item $A \to .\alpha$, however, in this case, the substring $a_{i+1} \dots a_j$ may well not be derived from $\alpha$ in $G$.

Thus, an SA may detect both the beginning of a production and its complete recognition. However, when the beginning of a production $A \to .\alpha$ is detected in some configuration $c_i$, we are not sure that there exists a configuration $c_j$, $i \leq j$, in which that production will

[6] Note how fresh leftmost terminals (i.e., $A \to \alpha.t\beta, \alpha \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$) are introduced by the close-reduce operation.

[7] Moreover, if $q_0$, the initial state of $\mathcal{A}_G$, has the form $q_0 = [I_G, I_0]$ (instead of $q_0 = [I_0, I_0]$), $\mathcal{L}(\mathcal{A}_G)$ is the language generated by the superset approximation based on recursive transition networks described in [15]. This is also the case for the language accepted by the FA defined in [1], if we replace its underlying LR automaton by a left-corner automaton. If $\mathcal{A}_G^I$ is the SA whose initial state $q_0$ is the pair $[I, I_0]$, it can be shown that $\mathcal{L}(\mathcal{A}_G^{I_1}) \subseteq \mathcal{L}(\mathcal{A}_G^{I_2})$ if $I_1 \subseteq I_2$. Thus, $\mathcal{A}_G^{I_0}$ (i.e., $\mathcal{A}_G$) is the most accurate SA.

be completed. Conversely, when a production $A \rightarrow \alpha.$ is completed in some $c_j$, we are sure that its recognition has started in some $c_i$, $i \leq j$. Therefore, even if we are not sure that $\alpha$ derives $a_{i+1} \ldots a_j$ in $G$, we can state that, with $\mathcal{A}_G$, the detection of complete items is more precise than the detection of initial items. In other words, as defined, SAs are more suited to detecting complete items than to detecting initial items while our guiding strategy needs (only) the detection of initial items. In order to detect initial items with the same accuracy as complete items, we define *mirror* SAs.

If $G = (N', T, P', S')$ is an augmented CFG, we define $\widetilde{G} = (N', T, \widetilde{P'}, S')$, its *mirror* grammar in which we have $\widetilde{P'} = \{A \rightarrow X_p \ldots X_1 \mid A \rightarrow X_1 \ldots X_p \in P'\}$. Of course, $\mathcal{L}(\widetilde{G})$ is the mirror language of $\mathcal{L}(G)$: $\mathcal{L}(\widetilde{G}) = \{a_n \ldots a_1 \mid a_1 \ldots a_n \in \mathcal{L}(G)\}$. Let $\mathcal{A}_{\widetilde{G}}$ be the (mirror) SA associated with $\widetilde{G}$. Of course, the mirror automaton $\mathcal{A}_{\widetilde{G}}$ must be interpreted with the mirror input string $\widetilde{w} = a_n \ldots a_1$, or equivalently with $w$, scanned from right to left. If a configuration $c_i = ([I, J], \widetilde{w}_i), c_0 \overset{*}{\underset{\mathcal{A}_{\widetilde{G}}}{\vdash}} c_i$ of $\mathcal{A}_{\widetilde{G}}$ is such that $J$ contains a complete item of the form $A \rightarrow \widetilde{\alpha}.$, this means that a recognition of the production $A \rightarrow \alpha$ may start at the index $n - i$ and, moreover, that this production will eventually be completed. In other words, in the configuration $c_i$ of $\mathcal{A}_{\widetilde{G}}$, we can place the Earley item $[A \rightarrow .\alpha, n - i]$ into the guide.

Due to our own experience in building regular covers for large CFGs, we believe that the combinatorial explosion in the number of their states will prohibit the construction of large SAs, or at least the construction of their (minimal) deterministic versions. Thus, we advocate constructing, at guiding time, only the sub-automaton, connected to the initial state $q_0$ and consistent with a given input string $w$. Therefore, if $w$ is recognized, exactly $n + 1$ states of $\mathcal{A}_{\widetilde{G}}$ are constructed.

It is possible to benefit from the fact that an SA is built from a CFG $G$ at run-time in building this SA only for a sub-grammar of $G$. More precisely, instead of using $G = (N', T, P', S')$, we can benefit from the lexicalized productions of $G$ by using the sub-grammar $G_w = (N', T, P'', S'), P'' = P_w \cup \{S' \rightarrow S\}$, where $P_w$ is the set of selected productions defined in Section 3.1, and $w$ is the input string. It is even possible to use, for each configuration $c_i$, a different set of productions $P_w^i$, which is the set of index selected productions of Section 3.1.

This guiding parser with a mirror SA, using for each configuration $c_i$, a set of index selected productions, is called SA in the experiment.

In fact, a guiding parser may itself be a guided parser. For example, a finite transducer which uses the mirror SA $\mathcal{A}_{\widetilde{G}}$ may be guided by a finite transducer which uses $\mathcal{A}_G$. In such a schema, two guides are constructed. A first guide, which is built by a finite transducer based upon $\mathcal{A}_G$, and a second guide which is built by a finite transducer based upon $\mathcal{A}_{\widetilde{G}}$. The first guide contains useful Earley items of the form $[A \rightarrow .\alpha, i]$ which are produced when the (guiding) finite transducer based upon $\mathcal{A}_G$ is in the configuration $c_i = ([I, J], w_i)$ and if $J$ contains the initial item $A \rightarrow .\alpha$. The guided version of the finite transducer based upon $\mathcal{A}_{\widetilde{G}}$ works as its non-guided version, except that the useful Earley item $[A \rightarrow .\alpha, i]$ is put into the second guide only if it also an element of the first guide. Though the detailed results of this two phase guiding schema are not reported in the experiment section of this paper, we got the following results: as expected, the run-times of the guiding phases are roughly doubled, the average precision is improved from 77.9% to 79.2%, while the global average run-time of the associated guided Earley recognizers increases from 33ms to 35ms.

# 4 Experiments with an English Grammar

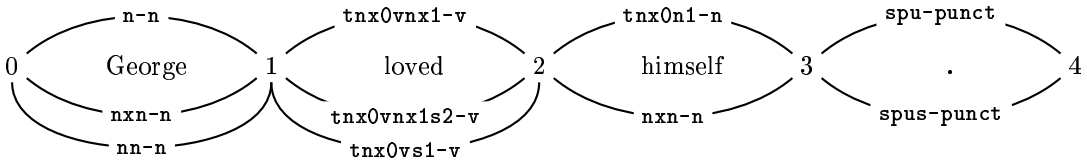


Figure 1: Actual source text as a simple DAG structure

The measures presented in this section have been taken on a 1.2GHz AMD Athlon PC running Linux. All parsers are written in C and have been compiled with gcc without any optimization flag.

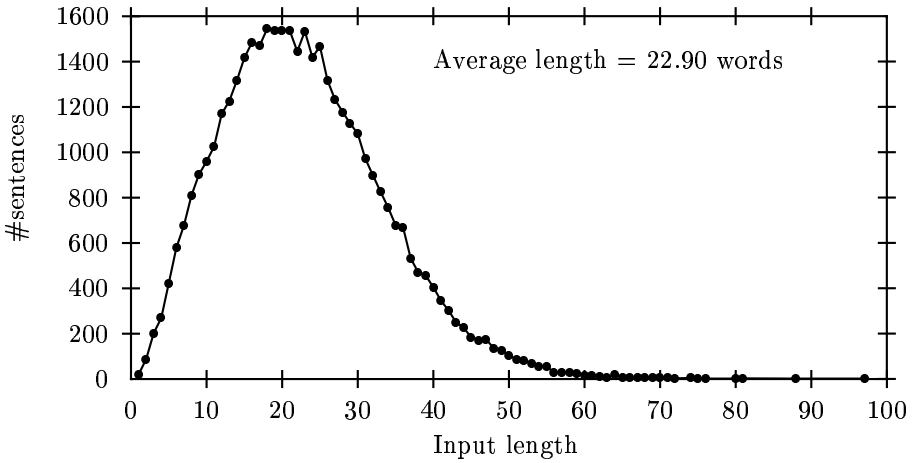


Figure 2: Distribution of the number of sentences by their lengths

Our experiment is based upon a wide-coverage English grammar designed for the XTAG system [16]. Of course, that grammar uses the TAG formalism. In the first stage, the English XTAG is automatically transformed into a CFG.[8] Let $G$ be this generated CFG,[9] the sizes of its components are reported in Table 1.[10]

| $\|N\|$ | $\|T\|$ | $\|P\|$ | $\|G\|$ | max RHS #nt |
|---|---|---|---|---|
| 33 | 476 | 1 131 | 17 129 | 26 |

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

[8]Following [6], the English TAG, stripped of its feature structures, is first transformed into an equivalent simple positive range concatenation grammar (PRCG). Then, following [2], this simple PRCG is transformed into a simple 1-PRCG which defines a CF superset. This simple 1-PRCG is in turn transformed into an equivalent CFG (See [5]), upon which all our experiments are based. In [7], we can find an algorithm which directly transforms a TAG into a CFG that defines a superset language.

[9]Though XTAG is an LTAG, the CFG $G$ is not lexicalized.

[10]The elements in $T$ are POS tags such as `n-n`, `nxn-n`, ... (see below). The *size* $|G|$ of a CFG $G$ is the number $\sum_{A \to \alpha \in P} |A\alpha|$. In order to give an idea of the complexity of this grammar, the last column gives the number of non-terminal symbols which occur in the RHS of the longest production. Such an impressively large number is directly related to the number of adjunction and substitution nodes in the largest elementary tree of the original TAG.

The association between the words (inflected forms) of an input sentence and the terminal symbols (anchors) of $G$ is the task performed by a *lexer* by means of dictionary searches. A dictionary search performs a one-to-many mapping between an inflected form and its associated anchors. For example, from the sentence "George loved himself .", the lexer 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`}", which is the actual input to all our processors.[11] Of course, all our processors have been prepared to accept such simple DAG structures as input (See Figure 1).

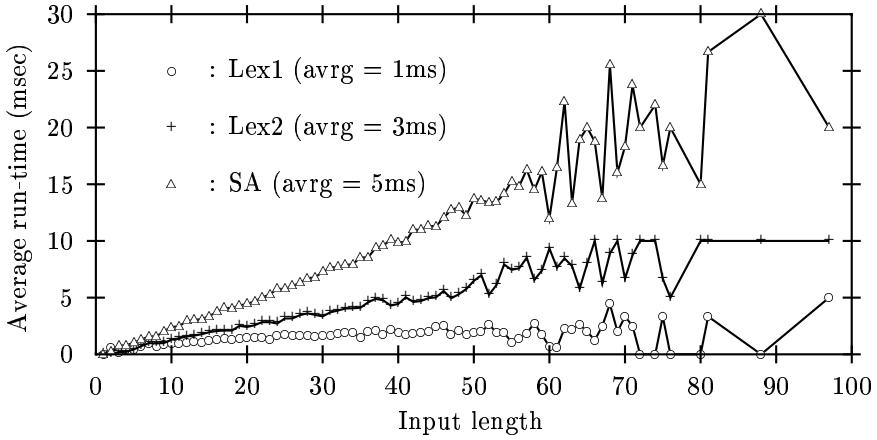


Figure 3: Guiding parsers average run-times

For our test set, we have used sentences extracted from the Wall-Street Journal. We used 42 253 sentences for a total length of 1M words. On this 1M word test set, the lexer phase runs in a little more than 10 seconds. However, 964 of these sentences are extra-grammatical w.r.t. $G$, and thus had to be ignored, leaving 41 289 sentences for a total of 945 772 words. In this test set, the lengths of individual sentences show great variations: with an average length of almost 23 words per sentence, there are single word sentences while the longest one contains 158 words! However, in order to smooth the results, all unique sentences of any given length have been ignored.[12] The distribution of our test set is shown in Figure 2.

As previously explained, a single input word usually selects several anchors. In all, the 944 609 words select 10 417 451 anchors (that is more than 11 anchors per word on average). In other words, when a sentence of $n$ words is processed, in fact, this corresponds to a linear input of about $11 * n$ anchors.

| | Oracle | Predictor | Lex1 | Lex2 | SA |
|---|---|---|---|---|---|
| $\|\mathcal{G}^w\|$ | 77 526 680 | 919 198 693 | 199 219 402 | 136 517 224 | 99 582 822 |
| Precision (%) | 100 | 8.4 | 38.9 | 56.8 | 77.9 |

Table 2: Average precisions of various guides

Recall that the purpose of our experiment is twofold: on the one hand we want to speed up

[11] The names between braces are anchors whose structure has the form `f-c` where the prefix `f` is a *family* name, reminiscent of the XTAG organization in family trees, while the suffix `t` is a category (verb, noun,...).

[12] This puts aside twelve (long) sentences of length 73, 78, 79, 83, 84, 92, 93, 94, 95, 106, 128 and 158.

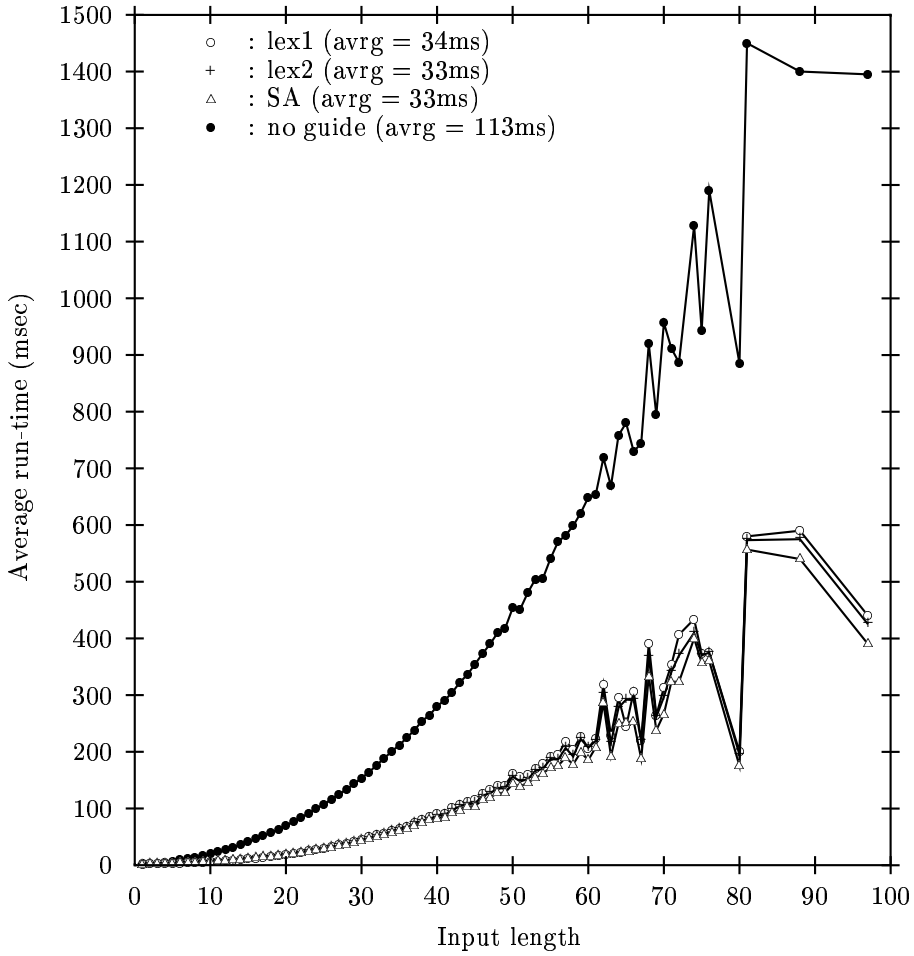


Figure 4: Guided recognizers average run-times

Earley type CF parsers and, on the other hand, we want to quantify the quality of the chosen guides, in other words how far they are from the ideal guide, the oracle. We can note that the best guides w.r.t. the second criteria will not necessarily be the ones which will get the best results w.r.t. the first criteria. The speed-up of a guided parser w.r.t. its non guided version will clearly be a trade-off between the precision of the guide and the time taken to build it. It is clear that the theoretical parse-time complexity is not improved by this technique and even that some practical parse-times can get worse.

The throughputs of the three guiding parsers Lex1, Lex2 and SA are shown in Figure 3, while the throughput of each associated guided Earley recognizer is shown in Figure 4, together with the results of a non-guided Earley recognizer.

On our test set, our experiment shows that, with any of the three methods, a guided Earley recognizer is, on average, more than three times faster than its non-guided counterpart. We can also note that only a small part (3 to 15%) of the total guided parse-time is spent, on average, in each guiding phase, and that costly guiding phases produce higher savings during their guided phase. However, for each of the three methods, it seems that the time saved during the guided phase is almost counterbalanced by the run-time of the guiding phase, at least within the input length ranges of our test set. This explains why, rather surprisingly, the three guided Earley

recognizers give close results.[13] An interesting question pointed out by one referee is to know at which break point it becomes more convenient to pre-process the input sentence with a guiding parser rather than to perform a direct parsing? Looking at the curves, it appears that this break point is very low. In fact, by examining the corresponding data, it seems that, even for very short sentences of a single word, the parse-times of all the guided recognizers outperform the standard recognizer.

The average precisions of the different guides are shown in Table 2. For the set of grammatical sentences, the line $|\mathcal{G}^w|$ gives the total number of useful items stored in the different guides. The first column is for the oracle, while the second column is for the Predictor phase of a standard Earley recognizer.

# 5 Conclusion

As already noted by many authors (see for example [8]), the choice of a parsing algorithm, as far as its throughput is concerned, cannot only rely on its asymptotical complexity but must also take into account practical experiments. Complexity analysis gives worst-case upper bounds which may well not be reached, and implies constants that may have a preponderant effect on the typical size ranges of the application. If we consider general CF parsing algorithms that have a practical implementation, all methods exhibit a theoretical cubic parse-time complexity, for appropriate grammar forms. However, their practical behaviours differ greatly from one another. For example, for two popular classes, GLR and Earley,[14] it seems that GLR-based methods can be chosen when the CFG at hand exhibits a rather low rate of ambiguity, while Earley-based methods are better for very ambiguous CFGs. Since the large-scale grammars upon which real-word NL applications are based are very often highly ambiguous, we have decided to speed up Earley-type parsers by a guiding technique. Although such an attempt is not novel, previously reported results were not very encouraging (see [14] and [12]).

In this paper, on the one hand, we have quantified the quality of the guided Predictor phase of an Earley recognizer by defining its precision. The quantity of work performed by an Earley parser decreases as the guided Predictor phase precision increases. In this paper, three guide construction methods have been experimented. These experiments were performed on a large, highly ambiguous[15] wide coverage English CFG, with a large sample set of sentences of unrestricted length. We can note that the Predictor phase of a Earley recognizer guided by a set automaton guiding phase exhibits a precision which is more than nine times better than that of the Predictor phase of a standard Earley recognizer.

On the other hand we have shown that some kind of guiding technique has to be considered

[13] Of course, we must also consider that the scale of the $y$-axis does not allow a clear distinction between the three curves.

[14] One referee has noted that "*Earley and GLR are not the most commonly used CF parsing algorithms for NLP, and they are certainly not the best. Often the algorithm of choice is (tabular) left-corner parsing, with or without top-down filtering*". As a matter of fact, in [13], Moore claims that "*LC parsing outperforms several other major approaches to context-free parsing, including some previously claimed to be the best general context-free parsing method. We conclude that our improved form of LC parsing may now be the leading contender for that title*". Recall that the purpose of this paper is not to compare guided Earley parsers with other general CF parsing methods, but rather to show how guiding techniques may improve a given parsing method, without making any other changes. Nevertheless, such a comparison naturally arises. So we have built guided Earley parsers for the three (huge) CFGs defined in [13]. Preliminary results on the available test sets show that the speed of our guided parsers could be favourably compared with Moore's.

[15] In fact our test CFG exhibits many cycles.

when one wants to increase parsing efficiency. Though this technique can be implemented with very few modifications, we have seen that a guided Earley recognizer can run more than three times faster that its standard counterpart.

Finally, we can note that this study also opens up a way to applications which require both accurate and efficient supersets of CFLs.

# References


[1] Theodore P. Baker. Extending lookahead for LR parsers. *Journal of Computer and System Sciences*, 22:243–259, 1981.

[2] François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric de la Clergerie. Guided parsing of range concatenation languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, pages 42–49, University of Toulouse, France, July 2001.

[3] Manuel E. Bermudez and Karl M. Schimpf. Practical arbitrary lookahead LR parsing. *Journal of Computer and System Sciences*, 41:230–250, 1990.

[4] Pierre Boullier. *Contribution à la construction automatique d'analyseurs lexicographiques et syntaxiques*. PhD thesis, Université d'Orléans, France, 1984. Thèse d'État (in French).

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

[6] Pierre Boullier. On TAG parsing. *Traitement Automatique des Langues (T.A.L.)*, 41(3):759–793, 2000. Issued June 2001.

[7] Pierre Boullier. Supertagging: A non-statistical parsing-based approach. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 2003)*, Nancy, France, April 2003.

[8] John Carroll. 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 1994.

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

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

[11] Aravind K. Joshi and Bangalore Srinivas. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING'94)*, Kyoto, Japan, 1994.

[12] Martin Kay. 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 2000.

[13] Robert C. Moore. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 171–182, Trento, Italy, February 2000. Revised version at `http://www.cogs.susx.ac.uk/lab/nlp/carroll/cfg-resources/iwpt2000-rev2.ps`.

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

[15] Mark-Jan Nederhof. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1):17–44, 2000.

[16] The research group XTAG. 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 1995.