Shared Forests Can Guide Parsing

François Barthélemy,
Pierre Boullier, Philippe Deschamp, Éric Villemonte de la Clergerie

Abstract

This paper describes an experiment we have made in order to illustrate a promising method to speed up parsing. A first tabular parser, built from the syntactic skeleton of a unification grammar, parses a sentence and produces a shared forest. This forest is used to guide and speed-up a second tabular parser which performs the unification checks.

Keywords: Guiding, Parsing, Shared Forest, TAG, Tabulation

Introduction

When, during a nondeterministic process, a nondeterministic choice occurs, classical solutions are to explore all possible ways, either in parallel, or one after the other using a backtrack mechanism. In both cases, the nondeterministic process may be assisted by another process to which it asks his way. This assistant may be either a guide or an oracle. An *oracle* will indicate all the good ways that will eventually lead to success, and only those good ways, while a *guide* will indicate all the good ways but may also indicate some wrong ways. In other words, an oracle is a perfect guide [6], and the worst guide indicates all possible ways.

Given two problems P_1 and P_2 and their respective solutions S_1 and S_2, if they are such that $S_1 \supset S_2$, any algorithm which solves P_1 is a candidate guide for nondeterministic algorithms solving P_2. Of course, supplementary conditions have to be fulfilled to be a guide. The first one deals with relative efficiency; it trivially assumes that problem P_1 can be solved more efficiently than problem P_2. Another crucial condition deals with synchronization: the locations from where the guided process asks its way must be meaningful to the guide and, reciprocally, the directions given by the guide must make sense to the guided process. That is, at contact points, guided processes and guides must talk a common language.

In this paper, we report on an experiment with a feature-augmented tree-adjoining grammar [TAG] parsing algorithm. A feature-augmented TAG is processed by a unification-based parser which is guided by a unification-free TAG parser. Their common language is the shared forest output by the latter.

F. Barthélemy works at CNAM and INRIA. E-mail: `barthe@cnam.fr`
The other authors work at INRIA. E-mail: `Pierre.Boullier@inria.fr`

Shared forests are easily built by tabular parsers, and it is well known that they may represent a vast amount of alternative trees in a very efficient way. For instance, when analyzing an input sentence of length n with a context-free parser, the corresponding shared forest has space complexity $O(n^3)$, and may represent an exponential (or even unbounded) number of trees. Moreover, using a shared forest is even more interesting if subsequent processings do not necessitate to unfold all alternatives it covers, producing each and every possible tree in turn. Traditionally, forests can be considered from two different viewpoints, leading to different ways of exploiting them. They can be considered as an AND-OR graph which may be traversed in order to perform some treatment (for instance, check unification constraints). However, in that case, ensuring traversals that do not amount to unfold all alternatives is not always an easy task. Alternatively, forests can be viewed and implemented as grammars, generally context-free grammars, whose productions are the basis for subsequent processing. We claim that this latter viewpoint is better adapted for guiding.

This paper is organized as follows. Section 1 provides some background. Section 2 describes the construction of guides by range concatenation grammar parsers and Section 3 shows how those guides are used by unification-based parsers compiled by the DyALog system. Section 4 provides commented results.

1 Background

The formalism of tree-adjoining grammars [5] is particularly suitable to describe many linguistic phenomena. The rewrite rules of a TAG are given by a set of *elementary trees* partitioned into *initial trees* and *auxiliary trees*. Nodes are labeled by terminals or non-terminals. Each auxiliary tree β has a distinguished leaf, called its *foot* and labeled as the root node of β.

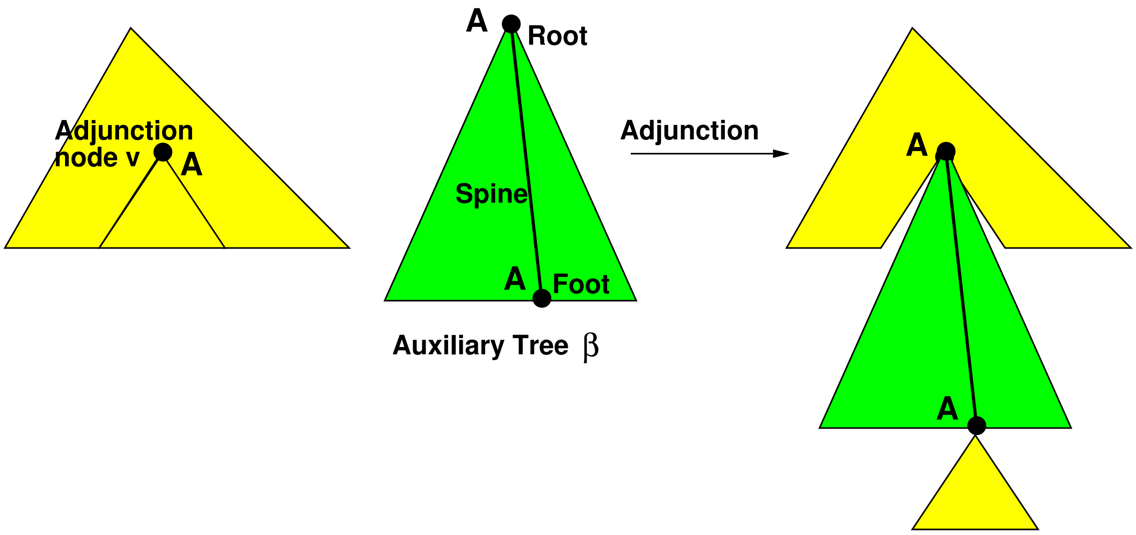


Figure 1: Adjunction

Two operations may be used to derive trees from elementary trees. The first one, called *substitution*, replaces a leaf node labeled by a non-terminal A by an initial tree α whose root is also labeled by A. The second operation, called *adjunction*, is illustrated by Figure 1. An auxiliary tree β whose root is labeled by A may be adjoined at any node ν labeled by A. The subtree rooted at ν is grafted to the foot of β.

Figure 2 shows the elementary trees used to parse sentence (1).

(1) Yves donne un joli livre à Sabine
Yves gives a nice book to Sabine

The output of a TAG parser is usually a shared forest, that may be either a parse forest or a derivation forest. In the case of the above sentence, which is unambiguous, these forests reduce to single trees. Figure 3 shows the single parse tree for that sentence. Figure 4 shows the single corresponding derivation tree, which indicates the different substitutions that have been performed, as well as the adjunction of tree `a(joli)` at node labeled N.

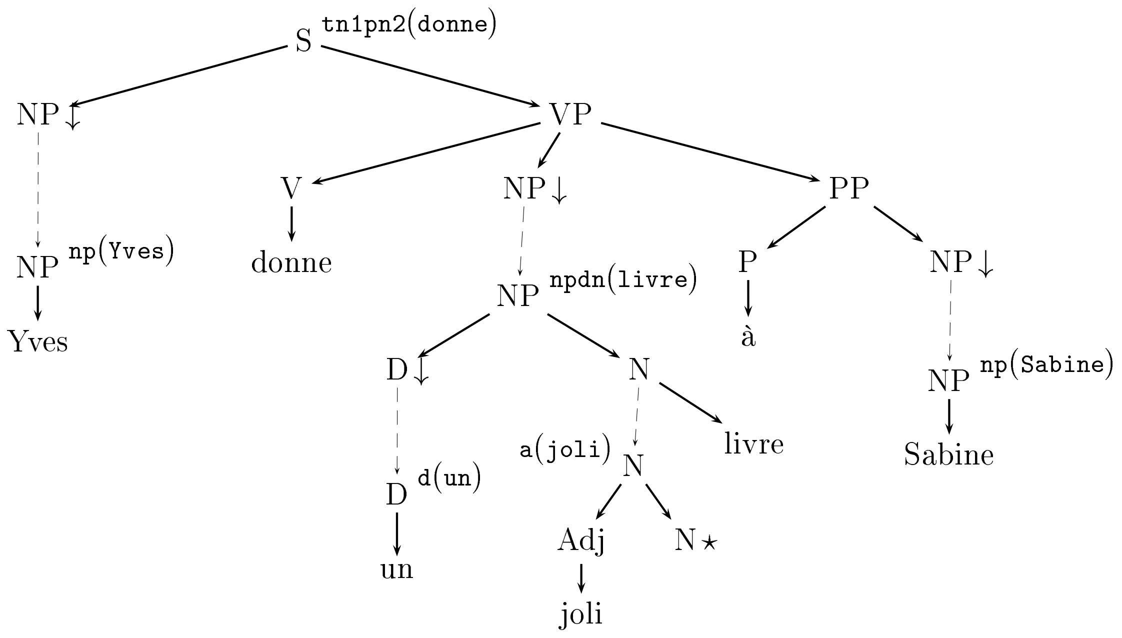


Figure 2: Elementary trees used to parse "Yves donne un joli livre à Sabine"

Our experiment has been carried out with a feature-augmented TAG from the XTAG distribution [10]. It is a small grammar for French, with 50 tree schemata, 117 lemmata and 345 lexical tokens.

In the XTAG description format, elementary trees are derived from tree schemata where a distinguished *anchor node* and possibly some *co-anchor nodes* may be filled by lexical tokens as specified by lemma and lexical entries. The nodes may also be decorated with a pair of first-order terms (*top* and *bottom* arguments). These arguments may be used, for instance, to handle agreement or enforce semantic re-

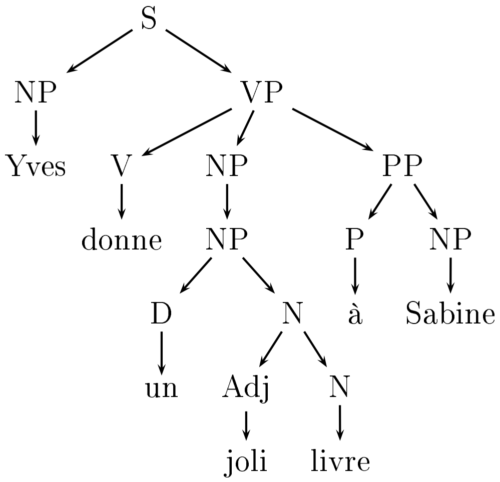


Figure 3: Parse tree for "Yves donne un joli livre à Sabine"

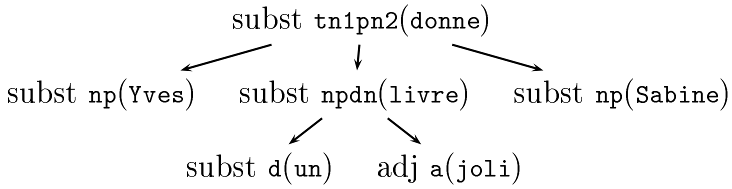
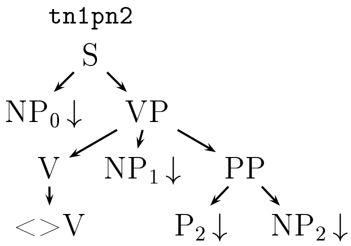


Figure 4: Derivation tree for "Yves donne un joli livre à Sabine"

strictions. Figure 5 shows the basic elements (tree schema `tn1pn2`, lemma `\DONNER\` and lexical token `donne`) used to build the tree `tn1pn2(donne)` corresponding to the pattern (2).

(2) *quelqu'un* **donne** *quelque chose* **à** *quelqu'un*
somebody **gives** *something* **to** *somebody*

The lemma part states that the subject NP_0 and the prepositional complement NP_2 must both be human and that NP_2 is introduced by the preposition *à*.




```
\DONNER\,V: tn1pn2[p_2=à]
      {NP_0.t:restr=+hum,
       NP_2.t:restr=+hum}
donne: \DONNER\, V
      {mode=ind,num=sing}
```

Figure 5: Tree schema

In most TAG parsers, tree schemata are first instantiated by source lexical items (inflected forms), then parsing is performed using only these lexicalized trees. Our approach is instead to pre-compile all tree schemata of the whole grammar, independently of any source text or even of any lexicon. Indeed, during this phase, the tree schemata are instantiated into actual elementary trees, by labeling lexical anchor nodes with terminal symbols. To this end, we have chosen to use as a basis the name of the trees, thus in figure 5, `tn1pn2` is associated with anchor `<>V`. During the parsing phase, co-anchors such as P_2 are processed by substitution (marked by $\downarrow$) and co-anchoring is enforced by constraints checking.

2 Building the guide

The formalism of range concatenation grammars [RCG] (see [4]) is a syntactic framework which can boast both power and efficiency. RCG is powerful, since it strictly includes the class of mildly context-sensitive formalisms, while staying computationally tractable since the sentences defined by its grammars can be parsed in polynomial time. In [2] and [3], we have shown how any TAG or set-local multicomponent TAG can be translated into an equivalent RCG. For example, in our experiment, the 16^{th} initial tree schema `npdn`, whose anchor is instantiated by `livre` in Figure 2, is translated into the RCG rewrite rule:

%% Rcg 16 : Tree npdn
np (Lnp Sd Ln "npdn" Rn Rnp) —> np (Lnp, Rnp) d (Sd) n (Ln, Rn).

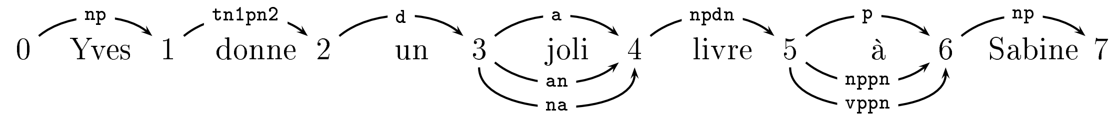
that only handles adjunctions, substitutions and anchors (but neither feature unification nor co-anchoring constraints). This one-to-one translation produces a rewrite rule called a *clause* which defines in its LHS a unary *predicate* `np` whose argument is a string of terminal symbols such as `"npdn"`, and of variables whose names start by an uppercase letter. In its RHS, there are binary predicate calls which handle possible adjunctions at both nodes NP and N, and a unary predicate call that handles substitutions at node D. The values denoted by terminals, variables or arguments are called ranges. A *range* is an ordered pair of integers which denotes an occurrence of a substring in a source text. The concatenation operation on consecutive ranges is denoted by the concatenation operation on strings. The output of a RCG parser is a context-free [CF] shared forest [7, 8, 11] whose productions are instantiated clauses in which each argument is instantiated by a range (see Figure 6). Since some lexical items such as the adjective `joli` or the preposition `à` may anchor or co-anchor several trees (here `"a"`, `"an"` and `"na"` for `joli` and `"p"`, `"nppn"` and `"vppn"` for `à`), our RCG parser has been modified in order to handle simple lattice inputs.

3 Using a guide

Concurrently with the RCG parser construction, DyALog compiles the feature-augmented TAG to produce a tabular parser, *via* an intermediate 2-stack automaton [12]. Features are handled using unification and co-anchoring constraints are handled by escaping to Prolog predicates [1].

The guided execution starts by looking for the axiom production. For instance, in Figure 6, looking at the body of production r_1, the parser knows that it should

%%Sentence:



%%Guide (CF shared forest output by the RCG parser):

```
 1: axiom(0..7) —> s(0..7).                                              %% r1
26:      s(0..7) —> s(0..0, 7..7), np(0..1), vp(1..1, 7..7),              %% r2
                    v(1..1, 2..2), np(2..5), pp(5..5, 7..7), p(5..6), np(6..7).
 4:     np(0..1) —> np(0..0, 1..1).                                      %% r3
16:     np(2..5) —> np(2..2, 5..5), d(2..3), n(3..4, 5..5).              %% r4
 5:      d(2..3) —> d(2..2, 3..3).                                       %% r5
13: n(3..4, 5..5) —> n(3..3, 5..5), a(3..3, 4..4).                       %% r6
 3:      p(5..6) —> p(5..5, 6..6).                                       %% r7
 4:     np(6..7) —> np(6..6, 7..7).                                      %% r8
```

Figure 6: RCG guide for "Yves donne un joli livre à Sabine"

try a substitution for `s` between positions 0 and 7. Still from the guide, the parser discovers that the tree numbered 26 (`tn1pn2`), associated with production r_2, is the only one that should be tried for this substitution. The RHS of r_2 is then traversed recursively from left to right and the information conveyed by each literal item is used to guide the actions performed at each node of `tn1pn2`. More formally, a tree α' numbered n' is selected to process some node ν in a tree α traversed with some production r only if the guide has a production r' for n' whose head equals the literal in r associated with ν. The tree α' is then traversed using r'.

To achieve this, the original compilation process (used by DyALog to build non-guided parsers) has been slightly altered for guided parsers. Essentially, the processing of substitution or adjunction is now preceded by an escape to a Prolog predicate looking for productions (correctly instantiated for the position variables) in the guide. To know to which node is associated a literal in a production body, mapping rules such as the following one (for tree `npdn`) are used to install the run-time calls to the guide.

```
:- rcg_guide (npdn, (16: np (L..R) —>
     np (Lnp..FLnp, FRnp..Rnp), d (Ld..Rd), n (Ln..FLn, FRn..Rn)),
     [[root,L,R], [np,Lnp,Rnp,FLnp,FRnp], [0,Ld,Rd], [n,Ln,Rn,FLn,FRn]]).
```

For instance, in the previous mapping rule, the entry [np,Lnp,Rnp,FLnp,FRnp] states that the position variables (Lnp, Rnp) and (FLnp, FRnp) are to be used for an adjunction on node `np`. Note that no guiding is used for scanning and anchor nodes.

For sentence (1), the RCG pass eliminates (among many others) all trees not pertinent to verb `donne` as well as tree `"na"` (adjective after noun) for `joli`. It also sets the left and right positions of the different constituents. The DyALog parser checks (a) that both the subject NP_0 and the prepositional complement NP_2 of *donne* are human (`restr=+hum`) and (b) that the preposition P_2 is *à*. Sentence (3) is accepted by the RCG pass and rejected by the DyALog pass:

(3) *Yves donne un livre vers Sabine
Yves gives a book towards Sabine

On the other hand, the RCG parser quickly discovers that sentence (4) is not grammatical (w.r.t. our grammar) and produces a void guide. Thus, the DyALog parser learns immediately that there is nothing to do.

(4) *Yves donne un livre
Yves gives a book

4 Results

On a small set of 14 sentences, we have run three parsers, namely a non-guided DyALog parser, a RCG parser, and a RCG-guided DyALog parser. Table 1 lists the execution times (in seconds and measured on a PC Pentium-II 450MHz). These figures speak for themselves and demonstrate the efficiency of guiding.

Sentence	**DyALog**	**RCG**	**Guided**
Jean aime Marie	1.32	0.00	0.46
Yves montre des fleurs à Sharon	2.19	0.00	0.43
Kathy regarde un livre avec Stéphane	1.73	0.00	0.49
un ami américain de Yves qui visite la France boit du bon vin	3.23	0.01	0.49
la jolie fille pense que Andrew est le grand ami de son chien	2.53	0.01	0.47
Marie comprend que Kathy dort bien chez Andrew	1.91	0.00	0.50
Anne voit les fleurs que Yves cherche sur la table	2.77	0.00	0.49
Jean pense que Béatrice perd les pédales	1.84	0.02	0.50
le petit chat de Sabine court vite	1.49	0.00	0.47
Andrew court parfois les filles	1.29	0.00	0.45
la fille de Marie aime bien les fleurs	1.88	0.01	0.47
qui est Bonnie	1.32	0.00	0.48
qui dit que Yves a un chien	1.75	0.00	0.45
un enfant intelligent de Marie comprend lentement que sa mère est bête	2.62	0.00	0.49

Table 1: Comparison of different parsers

Nevertheless, we still have to be careful. Indeed, we have also done further experiments with the set (5) of sentences, indexed by n and whose degree of ambiguity is exponential in n:

(5) Yves aime (la fille de)^n Sabine
Yves loves (the daughter of)^n Sabine

In Table (2), the two rightmost columns give respectively, for each value of n, the size of the guide and the number of trees in the shared forest. These results show that guiding gets worse than none when n increases beyond 12! We are currently investigating this example, but we believe that this behaviour suggests that the guide could be used more efficiently. Besides, the size of the guide passed between the two parsers should be taken into account, as some time is needed to emit and read it. Guiding presents some subtleties which have to be investigated further.

n	**DyALog**	**RCG**	**Guided**	**Prod**	**Trees**
1	1.67	0.00	0.45	17	2
2	2.16	0.01	0.46	30	5
3	2.89	0.00	0.51	50	14
4	3.41	0.01	0.61	80	42
5	3.97	0.01	0.70	124	132
6	5.09	0.01	1.09	187	429
7	5.72	0.02	1.52	275	1430
8	6.58	0.04	2.30	395	4862
9	7.87	0.04	3.22	555	16796
10	8.86	0.06	4.92	764	58786
11	9.87	0.08	7.50	1032	208012
12	11.09	0.10	11.12	1370	742900
13	13.12	0.13	15.98	1790	2674440
14	14.59	0.16	21.63	2305	9694845
15	16.16	0.22	31.60	2929	35357670

Table 2: Exponential degree of ambiguity

Conclusion

We would like to point out how easy it is to transform a non guided process into a guided one, even if some care has to be taken, and the results confirm that it is worthwhile. Of course, this notion of guide is not new and has been used many times, even if not identified as such. Let us cite finite state transducer guided by deterministic finite state automaton (DFSA) or CF language parsing guided by DFSA. Definite clause grammars are based upon a CF skeleton and thus their execution may be guided by a CF parser. We would also mention that this approach is generic and can be adapted in many situations (not only to form a sequence of parsers). For instance, [9] describes a similar idea: because a direct evaluation of recursive aggregate predicates may loop even using tabulation, the author derives a simplified version of these predicates used to build a guide (by tabulation) and precedes each call to a recursive aggregate predicate by a call to the guide.

References

[1] Miguel Alonso Pardo, Djamé Seddah, and Éric Villemonte de la Clergerie. Practical aspects in compiling tabular TAG parsers. In *Proceedings of the 5^{th} International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 27–32, Université Paris 7, Jussieu, Paris, France, May 2000.

[2] Pierre Boullier. A generalization of mildly context-sensitive formalisms. In *Proceedings of the 4^{th} International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4)*, pages 17–20, University of Pennsylvania, Philadelphia, PA, USA, Aug. 1998.

[3] Pierre Boullier. On TAG parsing *and* On multicomponent TAG parsing. In *6ème conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, pages 75–84 and pages 321–326, Cargèse, France, Jul. 1999. See also *Research Report n° 3668*, online at `http://www.inria.fr/RRRT/RR-3668.html`, INRIA, France, Apr. 1999, 39 pages.

[4] Pierre Boullier. Range concatenation grammars. In *Proceedings of the 6^{th} International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, Feb. 2000. See also *Research Report n° 3342*, online at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA, France, Jan. 1998, 41 pages.

[5] Aravind K. Joshi. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia, 1987.

[6] Martin Kay. Guides and oracles for linear-time parsing. In *Proceedings of the 6^{th} International Workshop on Parsing Technologies (IWPT2000)*, pages 6–9, Feb. 2000.

[7] Bernard Lang. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current issues in Parsing Technology*, chapter 11. Kluwer Academic Publishers, 1991. Also appeared in the Proceedings of International Workshop on Parsing Technologies – IWPT89.

[8] Bernard Lang. Recognition can be harder than parsing. *Computational Intelligence*, 10, 1994.

[9] Alexandre Lefèbvre. Towards an efficient evaluation of recursive aggregates in deductive databases. In ICOT, editor, *Proceedings of FGCS'92*, pages 915–925, 1992.

[10] The XTAG Research Group. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA, USA, Mar. 1995.

[11] K. Vijay-Shanker and David J. Weir. The use of shared forest in tree adjoining grammar parsing. In *Proceedings of the 6th Conference of the European Chapter of ACL*, pages 384–393. EACL, 1993.

[12] Éric Villemonte de la Clergerie and Miguel Alonso Pardo. A tabular interpretation of a class of 2-stack automata. In *Proceedings of ACL/COLING'98*, Aug. 1998. online at `http://atoll.inria.fr`.