Linguistic and Computational Advantages of Bidirectional Bottom-Up Parsing with Top-Down Predictions

José F. Quesada
CICA
Sevilla
Tel: +34-5-462.3811
Fax: +34-5-462.4506
josefran@cica.es

J. Gabriel Amores
Departamento de Lengua Inglesa
Universidad de Sevilla
Tel: +34-5-455.1549
Fax: +34-5-455.1516
gaby@fing.us.es

Abstract

This paper compares two parsing strategies: bidirectional bottom-up parsing with top-down predictions (BBP) and standard chart parsing. We demonstrate that BBP is superior to classical chart parsers from a linguistic and computational points of view. The efficiency of BBP results from two factors: first, top-down predictions bring about an algorithmic improvement, and, second, the memory model, the data structures and the programming techniques incorporate notable computational improvements.

Keywords: Parsing

1 Introduction

This paper compares two parsing strategies: bidirectional bottom-up parsing with top-down predictions (BBP) [17] and standard chart parsing [12]. We will try to demonstrate that BBP is superior to classical chart parsers from a linguistic and computational points of view. The BBP parser has been implemented as part of Episteme [18], a unification-based Machine Translation system developed in C. The paper is organized as follows. Section 2 reviews some background concepts regarding chart parsing and BBP. Section 3 compares both models using two different kinds of grammar comprising phenomena of frequent occurence in natural languages. We will show that both grammars pose serious problems for chart parsers. Finally, section 4 offers some concluding remarks.

2 Chart Parsing and BBP

2.1 Chart Parsing

A chart may be described roughly as a set of trees. Each tree is actually a directed acyclic graph (DAG), in which the leaves correspond to the words in the input sentence and each of the internal nodes corresponds to a syntactic structure of the grammar. This would be a *static* definition, mainly describing the result of parsing with a chart.

On the contrary, a *dynamic* definition of the parsing process distinguishes basically between nodes and arcs. Nodes keep the role described above, and arcs correspond to the application of a rule of the grammar over a set of nodes. An arc is *inactive* when

it has obtained all the nodes required in the right-hand side of the production which generated it. Otherwise, the arc will be *active*.

Let's illustrate these notions with an example. Consider the following basic grammar of English.

```
BeginningOfGrammar
    (1:S ->NP VP)
    (2:NP->np)
    (3:NP->det n)
    (4:VP->v)
    (5:VP->v NP)
EndOfGrammar

BeginningOfLexicon
    Peter     { np }
    eats      { v }
    the       { det }
    cakes     { n }
EndOfLexicon

ParseSentence(Peter eats the cakes)
```

First, upon receiving a string of words to be parsed, each of the spaces between words is numerated sequentially:

```
1 Peter 2 eats 3 the 4 cakes 5
```

A node is represented indicating the initial and final positions defining its interval and the symbol associated with it. In turn, an arc must know the limits of its application interval, the rule which generated it and the number of symbols in the righ-hand side of the rule already consumed. Thus, any arc will be represented as a four-tuple:

```
<i,j,rule,pos>
```

The pair (*rule*, *pos*) is usually represented in the literature by means of dotted rules. For example, the arc:

```
<1,2,2,2> = <1,2,NP->np.>
```

represents the application of rule number 2 (`NP -> np`) in the interval 1-2. The arc has obtained all the symbols in the right-hand side and, therefore, is inactive. Now, the following arc:

```
<1,2,1,2> = <1,2,S->NP.VP>
```

is an active arc which has applied rule number 1 (`S -> NP VP`) in the interval 1-2 and it has only consumed an `NP` symbol, so that it expects a `VP` symbol in the interval 2-N.

Obviously, every inactive arc automatically generates a node. In order to facilitate their representation, henceforward we will show inactive arcs between square brackets, indicating the application interval and the node generated, as follows:

```
[1,2,NP]
```

2.2 Analysis Rules in a Chart

Combination Rule. The basic rule of chart parsing is called the *combination rule* (CR), defined as follows: if a chart contains two adjacent nodes of the form:

$$< i, j, \alpha \rightarrow \Delta . B\Omega >$$
$$< j, k, \beta \rightarrow \Psi . >= [j . k, B]$$

then we must generate a new arc with the form:

$$< i, k, \alpha \rightarrow \Delta B . \Omega >$$

The new arc generated will be active or inactive depending on whether Ω is null or not.

Bottom–Up Rule. A bottom-up chart algorithm is obtained applying the CR rule in combination with the *Bottom-Up Rule* (BR): each time a new inactive arc is added to the chart, for example:

$$< i . j . \alpha \rightarrow \Psi . >= [i, j . \alpha]$$

for each rule in the grammar of the form:

$$(\lambda \rightarrow \alpha\Omega)$$

we must add a new active arc of the form:

$$< i, i, \lambda \rightarrow . \alpha\Omega >$$

Top–Down Rule. Finally, a top-down chart algorithm is obtained applying the CR rule in combination with the *Top-Down Rule* (TR): each time a new active arc is added to the chart, for example:

$$< i, j . \alpha \rightarrow \Delta . B\Omega >$$

for each rule in the grammar of the form

$$(B \rightarrow \Psi)$$

we must include a new active node with the following form:

$$< j, j, B \rightarrow \Psi >$$

2.3 Bottom–up Parsing with Top–down predictions

Bottom–up parsing enriched with top–down predictions is one of the most frequent strategies employed in the design of parsing algorithms for context–free languages. Bottom–up parsers have been championed for their efficiency [15, 21, 7, 19], although they can be substantially improved with the incorporation of top–down predictions [4, 10, 6]. From the parsing strategy standpoint, these parsers are simultaneously controlled by the data (the input string being analyzed) and the goal (the grammar).

This double control in the parsing strategy is already present in the Earley algorithm [11], where the *predictor* operator takes on the role of top–down prediction, inserting in

a state the productions applicable according to the status of the parser at that point. Then, the *scanner* operator, in conjunction with the *look-ahead* component, incorporates the bottom-up strategy. A detailed study of Earley's algorithm suggests a top-down parser, enriched with a bottom-up control component.

For LR-based algorithms [1, 2, 8, 22], the *goto* and *action* tables determine a fundamentally bottom-up strategy in the parser. However, they are not exclusively bottom-up, since the stack of states stores the left context of the symbol being analyzed. This information actually behaves as a predicting component which may be used to solve shift-reduce and reduce-reduce conflicts, as Pereira [15] and Shieber [20] suggest.

The combination of bottom-up and top-down strategies is also part of chart-based parsers [12]. According to Martin et al. [13]

> All efficient parsers combine both top-down and bottom-up information in some way (using the chart, for example), so that it is useless to classify these parsers as one type or the other (p. 272)

Fusing bottom-up and top-down components is also very common in the implementation of parsers over distributed or massively parallel systems [3, 9, 14].

2.4 Relations of Coverage, Derivability and Adjacency as a formal model for the application of Top-down predictions

We will first provide an informal or intuitive justification of some relations between the symbols and productions of a grammar which will be used in later sections.

Consider the following grammar:

```
(1:S -> X W)
(2:S -> a Y)
(3:S -> a Z)
(4:S -> X b)
(5:S -> W b)
(6:X -> a)
(7:X -> b)
(8:Y -> b)
(9:W -> b)
```

A naïve chart parser would generate the following initial arcs for the input string *a b*, provided a bidirectional event[1] generation mechanism has been incorporated:

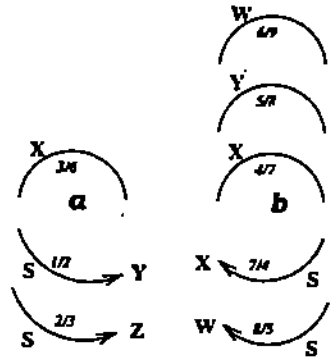

Fig 1 Initial arcs in a bidirectional chart

[1] Roughly speaking, an event is equivalent to an arc in a chart parser.

In the figure, each arc is associated with a pair of numbers indicating the event number (or arc) and the production creating it. All events display information about the symbol generated after they have been applied: that is, the left–hand side symbol of the corresponding production. In addition, active events indicate the symbol expected in the incomplete extreme.

The first type of controlling relation between nodes and events is based on the classical notion of *reachability* in a chart. We will use the term left or right *derivability*, depending on the direction the production has been applied.

Consider, for example, event number 2 in Fig. 1 above. This event is expecting the symbol Z in a position in which we actually find node b. However, a detailed look at the grammar shows that it will never be possible to reach the symbol Z by applying the rules in the grammar over a string starting with b. That is, Z is not a left derivation of b.

Something similar happens to event number 8. This event expects the symbol W in a position where the node a is found. However, W is not a right derivation of a, and therefore, event 8 is dispensable.

On the contrary, events 1 and 7 pass this derivability control, and go ahead.

In addition, we will introduce an *adjacency* control over the closed extremes of an event. Consider events 3 and 4 above. If we would generate the nodes suggested by those events, we would obtain the $X\ X$ string. However, this sequence is ill–formed according to our grammar, since both symbols do not appear immediately adjacent to each other in the right–hand side of any production. On the contrary, the right extreme of event number 3 satisfies this adjacency relation with the left extreme of event number 7, and it should be kept. As for event number 4, it may be eliminated, since no event is related to it through adjacency nor derivability. In turn, event number 5 is in derivability relation to 1, and event number 6 is in adjacency relation to 3.

In sum, the application of derivability and adjacency controls has eliminated three of the 8 initial events, as Fig. 2 shows:

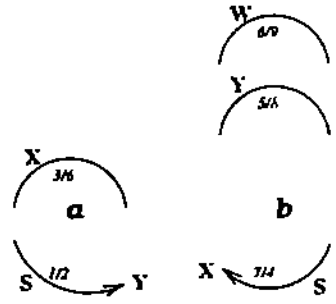

Fig 2 Applying derivability and adjacency relations

In addition to those relations, if we wish to use a bidirectional parsing strategy, it is convenient that every symbol in the grammar knows the events it may trigger; that is, the productions in which this symbol is part of its right–hand side. Storing this information as tables of *coverage* for each symbol during the compilation process will speed up the parsing process.

A more detailed description of these strategies may be found in [16, 18, 17].

2.5 BBP

Basically, BBP may be described as a bidirectional, event–driven parser which incorporates strong top–down predictions over the bottom–up event generation. The main top–down prediction mechanisms are described below.

2.5.1 Analysis of Derivations

These analyses apply on the open or active extremes of events. Intuitively, they try to check whether the expected symbol may be derived from the current symbol.

- Left Derivation Control
  Assume that an event is applying the production
  $$\delta \rightarrow \delta_1 \ldots \delta_n$$
  over the surface
  $$\Gamma\delta_1 \ldots \delta_i\omega\Omega$$
  where $1 \leq i < n$. The event is performing a prediction of the (necessary) symbol δ_{i+1} over the (current) symbol ω. Therefore, we have to check if
  $$\delta_{i+1} \in \mathrm{LDer}(\omega).$$
- Right Derivation Control
  Assume that an event is applying the same production as above, over the surface of analysis
  $$\Gamma\varphi\delta_i \ldots \delta_n\Omega$$
  where $1 < i \leq n$. The event is performing a prediction of the (necessary) symbol δ_{i-1} over the (current) symbol φ. In this case, we must check that
  $$\delta_{i-1} \in \mathrm{RDer}(\varphi).$$

If the derivation control fails, the event may be eliminated.

2.5.2 Analysis of Adjacencies

The closed boundaries of an event (those limits in which the event cannot perform any prediction) must be checked for the possibility of adjacency between the symbol that would be generated after the application of the event, and the adjacent symbols in the surface of analysis. Two situations may be distinguished, depending on whether the analysis is performed on the left or the right extreme of the event.

Assume that a complete event is applying the production

$$\varphi \rightarrow \varphi_1 \ldots \varphi_n$$

over the surface of analysis $\Delta\Gamma\Omega$, so that

$$\Gamma \equiv \varphi_1 \ldots \varphi_n$$

- Left Adjacency Control:
  If Δ is a non–empty string ($\Delta \equiv \ldots\delta$), check whether there is a symbol δ_r in the set $\mathrm{RDer}(\delta)$, and a symbol φ_l in the $\mathrm{LDer}(\varphi)$ such that
  $$\varphi_l \in \mathrm{PAdj}(\delta_r).$$
- Right Adjacency Control:
  If Ω is a non–empty string ($\Omega \equiv \ldots\omega$), check whether there is a symbol ω_l in the set $\mathrm{LDer}(\omega)$, and a symbol φ_r in the set $\mathrm{RDer}(\varphi)$ such that
  $$\omega_l \in \mathrm{PAdj}(\varphi_r).$$

Obviously, if the event is incomplete, it will only check for left or right adjacencies, depending on the direction of the event. If any of the adjacency controls fails, the event is eliminated.

3 Comparing Doxa and Episteme

This section shows the results obtained after comparing the parsing strategies outlined above. We have developed two systems, each implementing a different algorithm [18].

The first one is called Doxa. Doxa may be described as a pure bottom-up chart parser augmented with a unification component. Episteme, the second tool, is a kernel for the development of LFG-based MT systems. The experimental results described below correspond to the comparison of the context-free parsing modules only, without unification. These results will demonstrate that Episteme (BBP) is far more efficient than Doxa (chart). The efficiency of Episteme's parser results from two factors: first, top-down predictions bring about an algorithmic improvement, and, second, the memory model, the data structures and the programming techniques incorporate notable computational improvements.

The experiments will show how the parsers behave under two different situations. The interesting results will be those relative to different situations, rather than absolute results, which largely depend on the specific platform or environment.

- The first experiment takes a straightforward recursive grammar, frequent in any NLP system. Nevertheless, the results are extensible to other linguistic phenomena such as local and non-local dependencies, PP-attachment and coordination, as we have also been able to verify [16].
- The second experiment has a theoretical motivation. We have employed a grammar which puts both parsers in a limit situation due to the massive ambiguity (exponential) which it permits.

3.1 Recursive Grammars

The first grammar includes the three possible types of recursivity: left, right and middle recursion.

```
(1:S -> A BC D)
(2:A -> a)
(3:A -> A a)
(4:BC-> b c)
(5:BC-> b BC c)
(6:D -> d)
(7:D -> d D)
```

This grammar can generate the language $a^n(bc)^m c^l$. The strings analyzed have been of the form:

```
a a ... b b ... c c ... d d ...
```

varying the length of the input string from 50 to 250 words. The following table shows the results obtained. We have considered two possibilities in Episteme: the column `ConfTopDownPredOn` includes top-down predictions (BBP), and `ConfTopDownPredOff` does not, thus behaving like a pure bidirectional chart parser.

L [2]	DOXA			EPISTEME					
				ConfTopDownPredOff			ConfTopDownPredOn		
	T [3]	N [4]	E [5]	T	N	E	T	N	E
50	0.118	363	667	0.133	363	931	0.000	88	249
100	1.467	1400	2349	1.417	1400	3547	0.017	176	499
150	5.866	2963	4817	7.100	2963	7481	0.057	263	749
200	18.349	5300	8499	23.599	5300	13347	0.117	351	999
250	41.823	8063	12717	61.148	8063	20281	0.200	438	1249
1000	-	-	-	-	-	-	3.750	1751	4999

The following conclusions may be drawn from these results.

1. The number of nodes created by Doxa and Episteme, without top–down predictions, is the same. However, the number of events is higher in Episteme, due to the bidirectional strategy. Since the number of events to be processed is higher, the time involved in analysis is also greater. Therefore, it is not advisable to use Episteme without predictions.
2. When Episteme includes top–down predictions, there is a substantial reduction in the number of events and nodes created. Namely, Episteme creates around 80 and 90% less structures than Doxa.
3. In addition, the growing model clearly favors Episteme. The number of nodes and events grows linearly in relation to the size of the string, for this grammar.
4. Doxa and Episteme in `ConfTopDownPredOff` mode are not capable of analyzing a string of 1,000 words, while Episteme in `ConfTopDownPredOn` mode can do it in just three seconds.

3.2 Exponentially Ambiguous Grammars

Consider the following grammar:

```
(1: X -> X X)
```

Episteme cannot apply any type of top–down prediction to this grammar, since all derivations and adjacencies controls would succeed.

The strings analysed, have been of the form:

```
x x x ...
```

varying the length of the input between 5 and 25 words. The following tables show the results. From these results we may draw the following conclusions.

1. The application or not of top–down predictions has not reduced the number of nodes and events, but it has not resulted in a computational overhead either, since the data structures employed restrict the scope of application of top–down controls considerably.
2. Comparing Episteme and Doxa shows the limitations of a naïve implementation such as Doxa's, in which we neglected issues pertaining the memory model, storage, retrieval and comparison of character strings, data structures, etc.

[2] Length of input string (number of words).
[3] Time, in seconds.
[4] Number of nodes generated.
[5] Number of events generated.

L [6]	DOXA		
	T [7]	N [8]	E [9]
5	0.000	39	83
10	128.1	9901	198812
15	0.3 Years	0.3 M [10]	4 M
20	31000 Years	20 M	1162 M
25	3000 M Years [11]	2000 M	282000 M

L	EPISTEME					
	ConfTopDownPredOff			ConfTopDownPredOn		
	T	N	E	T	N	E
5	0.000	15	50	0.000	15	50
10	0.017	55	275	0.017	55	275
15	0.117	120	800	0.100	120	800
20	0.483	210	1750	0.450	210	1750
25	1.500	325	3250	1.467	325	3250

4 Conclusions

In this paper we have demonstrated that Bidirectional Bottom–Up Parsing is superior to standard chart parsing even in cases where Top–Down Predictions do not pose any additional advantage. The efficiency of the parser results from a better algorithm but also from the implementation of computational strategies which reduce the overhead of string comparison, memory allocation and data structure manipulation.

References

[1] Aho, A.V. & J.D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling, Vol. I: Parsing*. Englewood Cliffs, N.J.: Prentice Hall.

[2] Aho, A.V., R. Sethi, R. & J. D. Ullman. 1985. *Compilers: Principles, Techniques and Tools*. Reading, Massachussets: Addison–Wesley.

[3] Alblas, H., R. den Akker, P.P. Luttighuis & K. Sikkel. 1994. A Bibliography on parallel parsing. *SIGPLAN Notices* **29** (1), 54-65.

[4] Andrews, N.A., & J.C. Brown. 1993. A high–speed natural language parser. *AISB Quarterly*, **86**, 12-19.

[5] Bolc, L. (ed.) 1987. *Natural Language Parsing Systems*. Heidelberg: Springer–Verlag.

[6] Carroll, J. 1994. Relating Complexity to Practical Performance in Parsing with Wide–Coverage Unification Grammars. CMP-LG e-print archive cmp-lg/9405033.

[7] Carter, D. 1990. Efficient Disjunctive Unification for Bottom–Up Parsing. *COLING-90*, 70-75.

[6] Length of input string (number of words)
[7] Time, in seconds.
[8] Number of nodes generated.
[9] Number of events generated.
[10] Million (estimated).
[11] Million Years (estimated).

[8] Chapman, N. P. 1987. *LR Parsing. Theory and Practice*. Cambridge: Cambridge University Press.

[9] Chung, M. & D. Moldovan 1994. Applying Parallel Processing to Natural-Language Processing. *IEEE Expert*. February 1994. 36-44.

[10] Dowding, J., R. Moore, F. Andry & D. Moran, D. 1994. Interleaving Syntax and Semantics in an Efficient Bottom-Up Parser. *Proceedings of the 32nd AnnualMeeting of the ACL*, 110-116.

[11] Earley, J. 1970. An Efficient context-free Parsing Algorithm. *Communications of the ACM*. 13(2), 94-102.

[12] Kay, M. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. *CSL-80-12 Xerox Palo Alto Research Center*.

[13] Martin, W.A., K.W. Church & R. S. Patil. 1987. Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results. In [5]. 267-328.

[14] Nurkkala, T. & V. Kumar. 1994. The performance of a highly unstructured parallel algorithm on the KSR1. *Proceedings of the Scalable High-Performance Computing Conference*. 215-220.

[15] Pereira, F.C.N. 1985. A Structure-Sharing Representation for Unification-Based Grammar Formalisms. *Proceedings of 23rd Meeting of the ACL*. 137-144.

[16] Quesada, J.F. 1996. Un Modelo Robusto y Eficiente para el Análisis Sintáctico de Lenguajes Naturales mediante Arboles Múltiples Virtuales. *Procesamiento del Lenguaje Natural*. 19. 14-29.

[17] Quesada, J.F. Forthcoming. Bidirectional and Event-Driven Parsing with Multi-Virtual Trees.

[18] Quesada, J.F. & G. Amores. Forthcoming. *C for Natural Language Processing*. London: UCL Press.

[19] Shann, P. 1991. Experiments with GLR and Chart Parsing. In [22]. 17-34.

[20] Shieber, S. M. 1983. Sentence disambiguation by a shift-reduce parsing technique. *IJCAI-83*. 699-703.

[21] Tomita, M. 1987. An Efficient Augmented Context-Free Parsing Algorithm. *Computational Linguistics* 13 (1-2). 31-46.

[22] Tomita, M. (ed.) 1991. *Generalized LR Parsing*. London: Kluwer Academic Publishers.