Controlling Bottom-Up Chart Parsers through Text Chunking

Fabio Ciravegna, Alberto Lavelli
Istituto per la Ricerca Scientifica e Tecnologica
Loc. Panté di Povo, I-38050 Trento, Italy
Email:{cirave|lavelli}@irst.itc.it

Abstract

In this paper we propose to use text chunking for controlling a bottom-up parser. As it is well known, during analysis such parsers produce many constituents not contributing to the final solution(s). Most of these constituents are introduced due to the parser inability of checking the input context around them. Preliminary text chunking allows to focus directly on the constituents that seem more likely and to prune the search space in the case some satisfactory solutions are found. Preliminary experiments show that a CYK-like parser controlled through chunking is definitely more efficient than a traditional parser without significantly losing in correctness. Moreover the quality of possible partial results produced by the controlled parser is high. The strategy is particularly suited for tasks like Information Extraction from text (IE) where sentences are often long and complex and it is very difficult to have a complete coverage. Hence, there is a strong necessity of focusing on the most likely solutions; furthermore, in IE the quality of partial results is important.

1 Introduction

Bottom-up parsers spend large amounts of time in building constituents not contributing to the final solution(s). Many of such constituents are produced due to the inability of the parser to take into account the surrounding context. For example, in the sentence `John Smith works in the field of language processing` a standard bottom-up parser builds an S constituent spanning just `John Smith works in the field of language`, a constituent not useful for the final correct parse tree (so as many of the constituents subsumed by it). The impact of such constituents on parser performances is relevant when long sentences are analyzed.

One possible alternative solution is that of adopting strategies that integrate the advantages of both bottom-up and top-down strategies (see [Wiren, 1987] for a comparison of different chart-based parsing strategies). Such strategies, however, do not show to work very efficiently or robustly on long and complex texts, especially when the grammar coverage is not expected to be complete (as is often the case in applicative domains such as Information Extraction from text, henceforth IE).

Chunk Parsing [Abney, 1995] is a technique that mostly allows to overcome the problem of effectively taking into consideration the surrounding context. Chunks are portions of text close to the structures highlighted by psycholinguistic and phonological studies. The chunk parser is designed so to analyze first of all these chunks by using a limited grammar (recognizing constituents such as NPs, PPs, etc.). In a second step it builds clause structures composing chunks by using theta-roles, in a dependency-like fashion. Such strategies render a text largely unambiguous. We think that there are some drawbacks in the original Chunk Parsing proposal:

- it requires the definition of a specifically suited parsing strategy (i.e., a strategy that partially builds constituents and partially dependency structures);
- it requires the use of a non standard grammar, as - for the same reason as above - some of the operations are performed on constituents in a standard way, some others are performed in a non standard way; this obviously prevents the re-use of already existing grammars;
- a deterministic behavior is provided for constituent attachment (e.g. PP attachment) [Abney, 1991]; unfortunately, in analyzing complex sentences effective heuristics for attachment are difficult to be found; very often true ambiguity arises and it is then necessary to deal with it in a proper way.

The idea proposed in this paper is that of providing an agenda-based bottom-up chart parser with the ability of performing its analysis in a chunk based fashion, while maintaining the general characteristics of chart parsing (i.e. the standard algorithm, the ability of coping with ambiguity and the possibility of re-using existing grammars). The proposal fits within a well-established experience of chart-based parsing techniques at IRST [Satta and Stock, 1994] and is a first response to the challenges put to a unification-based approach by the necessity of analyzing real texts.

The parser performs its analysis in consecutive steps and at each step it behaves as follows:

- partial constituents are built by using only the constituents produced during the previous step;
- only some of the possible tasks are activated; the parser chooses among the most promising ones, delaying or pruning the others.

This strategy allows to consider at each time only a limited set of paths in the search space, so to reduce the ambiguity seen by the parser at a given time.

In order to select the constituents to be built (and to be used in the following analysis), the parser control module uses the information provided by an automaton which performs preliminary text chunking. Aim of such preliminary chunking is that of identifying the input areas where certain types of constituents (such as NPs, PPs, etc.) are likely to be found and to divide the input in sub-chunks that are likely to be clauses[1]. The parser attempts first of all to build a (set of) solution(s) that fulfills the expectations selected by the chunking algorithm: at three different steps NPs, clauses and sentences are recognized; if the expectations are fulfilled (e.g. some complete parse trees are found that cover the sentence under analysis), the paths in the search space that are not likely to provide equivalent solutions are pruned out and the found solution(s) are returned.

For example in the case of the sentence `John Smith works in the field of language processing` the idea is that of temporarily hiding the availability of the NP [`language`] while promoting the NP [`language processing`]. During the first step the NPs [`John Smith`], [`field`] and [`language processing`] are recognized. During the following step these NPs are promoted to the prejudice of others such as [`John`], [`Smith`] and [`language`]. Then PPs are recognized and promoted during the clause recognition[2] so that [`in the field`] and [`of language processing`] are promoted. Since just one clause is present in the sentence, the third step (sentence analysis) is not necessary. Given that a complete parse tree is found, at the end of clause analysis the tasks that require the use of constituents spanning [`John`], [`Smith`], [`language`] and [`the field`] are pruned from the agenda.

If a solution has not been found after visiting the paths in the search space recommended by the chunker, different possibilities are envisaged:

- all the remaining paths are visited;
- only a limited number of paths are visited choosing among the most promising ones not yet pursued (i.e. those that are not too far from the chunking expectation);
- the best partial solutions are chosen among those already found by the parser and no additional steps are performed.

The best strategy to be adopted depends on both the characteristics of the corpus under analysis and the requirements set for the parser. For example when the grammar is likely to be more or less complete for the corpus and completeness of the analysis is preferred to efficiency, all alternative solutions can be searched; on the other hand, when the lack of coverage is likely to be frequently experienced and/or efficiency is one of the requirements for the parser (as it is often the case in IE), one of the other two strategies can be selected.

The aim of this paper is to analyze how the three step analysis schema can be introduced in an agenda-based bottom-up chart parser without:

- imposing requirements on the grammar structure or content (so to allow the re-use of preexisting resources);

[1] Throughout the paper we will use the term *clause* not in the usual linguistic sense, but in an informal way that will become clear in the following.

[2] DP and PP recognition is where mostly our approach differs from standard chunk parsing; in the following we will clarify the difference.

- requiring modifications in the general parsing algorithm, except in the agenda manager - a quite limited piece of software - (so to make it applicable to any agenda-based bottom-up chart parsers).

In the following we will:

- formalize the definition of each chunk level (Section 2);
- analyze the impact on the parsing algorithm (Section 3);
- present some preliminary experimental evidence in terms of efficiency and effectiveness of the three step strategy on real world texts (Section 4);
- present some conclusions and related research (Section 5).

2 The Four Levels of Chunks

Abney, starting from a cognitive point of view, identifies three levels of chunks, broadly corresponding to:

- **level 0** words;
- **level 1** non recursive NPs, DPs, PPs and Verb Groups (VGs);
- **level 2** clauses.

We have modified Abney's original proposal, giving a more grammatically oriented definition of chunks; in particular we split the definition of level 1 in two because, as we will see in the following, in a standard parser elements such as DPs and PPs are to be treated differently from NPs and VGs:

- **level 0** *Word-Chunks*: words (ignored in this paper);
- **level 1a** *Head-Chunks*: non recursive NPs and VGs;
- **level 1b** *Mod-Chunks*: non recursive PPs, DPs and Complex VGs (CVGs);
- **level 2** *Clause-Chunks*: clauses.

In the following subsections we will provide a more precise definition of each level for Italian.

2.1 Head-Chunks: NPs and VGs

Definition[3]:

$$\begin{aligned} \text{NP} &\rightarrow (\text{adv}^* \ \text{adj}^+)^* \ \text{noun}^+ \ (\text{adv}^* \ \text{adj}^+)^* \\ \text{VG} &\rightarrow (\text{adv})^* \ \text{verb} \ (\text{adv})^* \end{aligned}$$

Very often constituents built by separately analyzing Head-Chunks are found as they are inside the final parse tree. Consider the following example:

Example 1 `di una società italiana del gruppo XYZ` [4]

In Figure 1 the parse tree for Example 1 is shown: a complete constituent (the NP spanning `società italiana`) recognizes the Head-Chunk highlighted. For this reason Abney's proposal of recognizing such chunks separately still holds also when using a standard parser, because - if chunking is done correctly - the resulting constituents will contribute to the final parse tree.

[3] The regular expressions used in the definitions are just meant to be representative of the set of rules used by the chunker. In the regular expressions the following operators are employed: * (Kleene star), + (with the usual meaning), and ? (for optionality).

[4] Literally, `of a company Italian of-the group XYZ`.

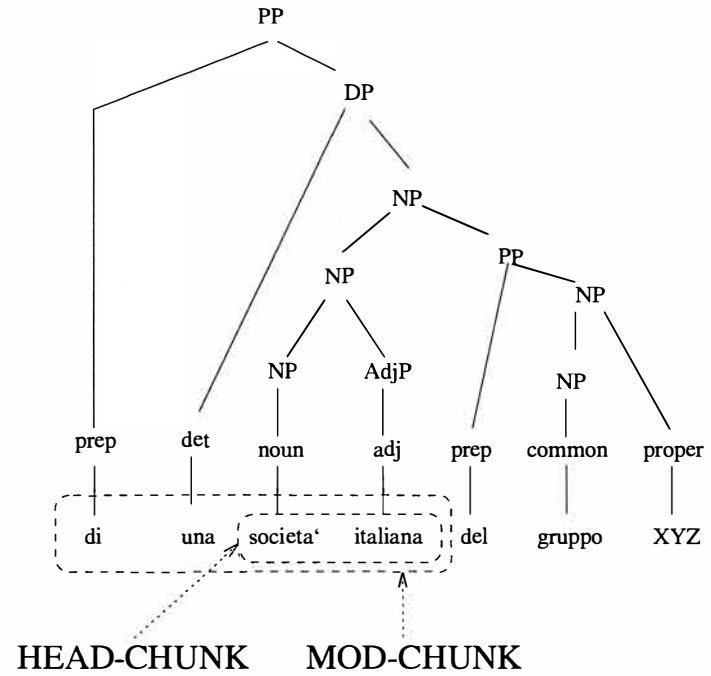

Figure 1: Parse tree (with two chunks highlighted) for Example 1.

2.2 Mod-Chunks: DPs, PPs and Complex Verb Groups

Mod-Chunks contain Head-Chunks:

$$\begin{array}{lcl} \text{DP} & \rightarrow & \text{Det}^{?}\ \text{NP} \\ \text{PP} & \rightarrow & \text{Prep Det}^{?}\ \text{NP} \\ \text{CVG} & \rightarrow & (\text{adv})^{*}\ (\text{auxiliary})^{*}\ \text{VG} \end{array}$$

Often constituents spanning exactly a Mod-Chunk are not found as they are within the final parse tree. In Figure 1 there is no PP covering exactly the Mod-Chunk shown. The PP spanning it also spans other parts of the input not within the same chunk (i.e. the PP spanning `del gruppo XYZ`). What is provided by Mod-Chunks is the coverage of DPs and PPs when they do not include any modifier of the same level. The consequence is that it is not possible to recognize Mod-Chunks separately because the resulting constituents may not contribute to the final tree.

2.3 Clause-Chunks

Clause-Chunks identify the structures of both main and subordinate clauses. Most of the problems in constituent attachments arise within Clause-Chunks. Chunking points are mainly commas, relative pronouns, the Italian complementizer `che` (more or less equivalent to the English complementizer `that`), some types of conjunctions, and some domain-specific cue words. For example, in the sentence:

Example 2 `Corporacion Banesto, holding industriale e di servizi del Banco Espanol de Credito, avrebbe perso nel '93 103,4 miliardi di peseta, circa 1.200 miliardi di lire, dopo l'utile di 5,6 miliardi del '92.` [5]

[5] Literally. `Corporacion Banesto, holding industrial and of services of Banco Espanol de Credito, would-have lost in '93 103.4 billion of pesetas, about 1,200 billion of lire, after the profit of 5.6 billion of '92.`

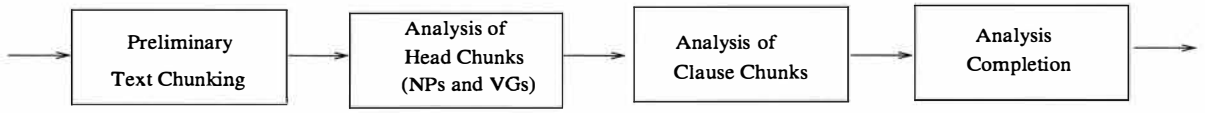

Figure 2: Outline of the overall approach.

chunking after commas produces the following result:

Chunk1: `Corporacion Banesto,`
Chunk2: `holding industriale e di servizi del Banco Espanol de Credito,`
Chunk3: `avrebbe perso nel '93 103,4 miliardi di peseta,`
Chunk4: `circa 1.200 miliardi di lire,`
Chunk5: `dopo l'utile di 5,6 miliardi del '92.`

Chunk2 and Chunk4 identify appositive clauses, whereas Chunk1 and Chunk3 are two parts of the main clause; Chunk5 identifies a PP modifying the main sentence.

2.4 The Chunking Process

The chunking process is performed via a finite state automaton. In case of uncertainty between closing a chunk or extending it with the next word, a greedy strategy is used, i.e. the next element is included in the current chunk. Currently the rules are manually encoded. The chunker relies on the output of a Part of Speech Tagger to solve lexical ambiguity.

3 Parsing, Chunks and Agenda Control

As said in the introduction, our aim is that of defining a control algorithm that allows a bottom-up chart parser to visit the search space so that:

1. *satisfactory solutions* are pursued as firsts;
2. non satisfactory solutions can be pruned out from the search space;
3. the quality of partial solutions is maximized (when a complete solution is not available).

We propose text chunking as a way for providing an estimation of the satisfactoriness of solutions. The analysis is divided in three steps; at each stage satisfactory solutions (when found) are promoted during the following step to the prejudice of other solutions. A recovery action is provided for cases where such solutions are not found. An outline of the three step analysis (plus preliminary chunking) is shown in Figure 2.

Before running the parser, preliminary text chunking is performed via a finite state automaton. The output of this step is a collection of chunking points, i.e. the boundaries for Head-Chunks, Mod-Chunks and Clause-Chunks. Such output is used by the control mechanism of an agenda-based bottom-up chart parser. In agenda-based chart parsers the control relies on activating/delaying tasks in the agenda. A task is the combination of two adjacent edges; each pair of edges represents two (groups of) adjacent elements of the right hand side of a rule. One of the two edges will contain the head of the rule and is defined here as the head edge; the other edge is defined here as modifier edge.

The control algorithm requires the parser make the following steps:

- Step 1: Analysis of Head-Chunks;
- Step 2: Analysis of Clause-Chunks;

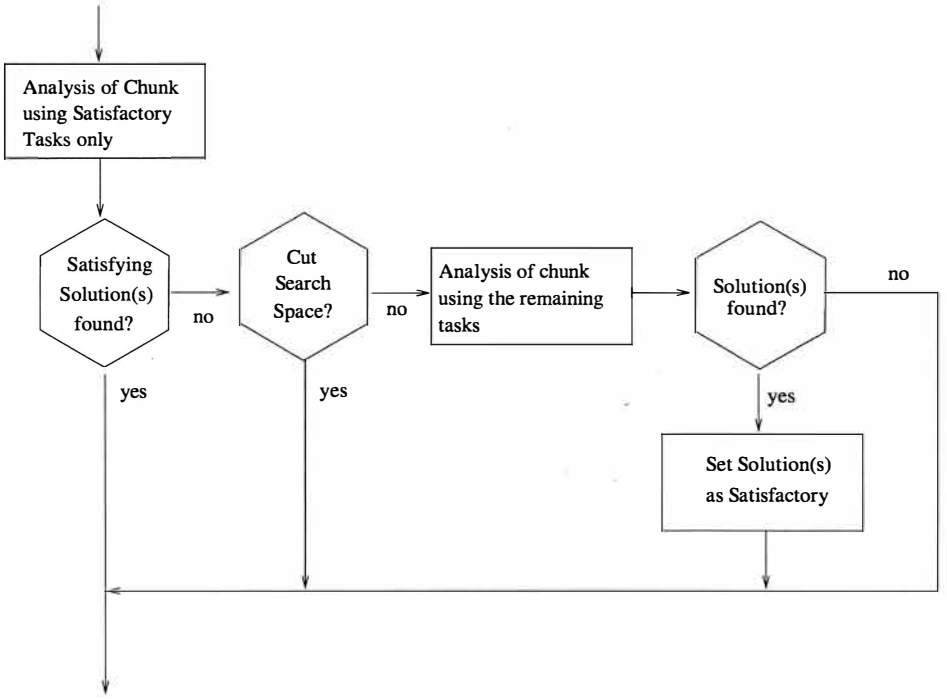

Figure 3: Outline of the algorithm for the analysis of a chunk (of any type).

- Step 3: Text Analysis Completion.

During *Step 1* the parser analyzes each Head-Chunk separately. Tasks that require to go beyond the chunk limits are delayed. If chunking is done correctly, this step produces complete constituents spanning the entire chunk that will be found as they are in the final parse tree. A constituent spanning exactly a Head-Chunk is defined here as *HEAD-CONST*.

During *Step 2* each Clause-Chunk is analyzed separately. The following two rules are used:

RULE0:

head edge is a generic edge (e.g. a preposition, an article, etc.) not spanned by any HEAD-CONST;

modifier edge is at least a HEAD-CONST;

both the edges are within the same Mod-Chunk.

and

RULE1:

head edge is a HEAD-CONST;

modifier edge is MOD-CONST.

We define a MOD-CONST as an edge either built by following RULE0 and spanning a whole Mod-Chunk, or built by following RULE1.

Tasks trying to build MOD-CONSTs are favored in Clause-Chunk recognition; a MOD-CONST spanning a complete Clause-Chunk is defined as a CLAUSE-CONST.

During *Step 3* the completion of text analysis is executed; in this phase the parser executes all the tasks delayed in the previous steps, i.e. it executes those tasks that require to overcome the constraints imposed by any levels of chunks. But the agenda is ordered so as to favor those tasks following:

RULE2:

the head is at least a HEAD-CONST and the modifier is a CLAUSE-CONST.

A constituent built by following RULE2 is also a CLAUSE-CONST. The analysis stops when the agenda is empty.

3.1 Some Considerations

Head-Chunks identify input portions where constituents contributing to the final parse tree can be recognized. After being recognized during Step 1, for the rest of the analysis these constituents are preferred to those subsumed by them. This strategy allows to declare portions of input as non separable units. For example, in:

Example 3 `La Dhl International del network DHL consegue un utile lordo` [6]

a Head-Chunk spans `utile lordo`. Favoring the NP that covers it exactly means favoring the construction of the S constituent spanning the entire sentence, while delaying the production of an S constituent spanning only `La Dhl International del network DHL consegue un utile`.

Mod-Chunks pick the minimum area in which modifiers are found. Minimum area means that chunks identify the portion of the sentence spanned by such constituents if they do not have any modifiers either. This allows delaying the use of constituents spanning sub-chunks of Mod-Chunks as modifiers. For example it delays the use of NPs internal to a pre-verbal PP as verbs subjects. In Example 3 the combination

$$(network\ Dhl)^{modif} + (consegue\ un\ utile)^{head}$$

is delayed because `network DHL` is a sub-chunk of the Mod-Chunk spanning `del network DHL`.

Clause-Chunks help in the complex problem of connecting constituents. In defining Clause-Chunks we start from an intuition about some cue words: let's consider for example commas. In most languages commas do not follow precise rules, nevertheless they are largely used in texts for introducing intonation pauses; very often commas delimit portions within sentences too long to be read as a whole. Instead of trying to capture in the grammar the information brought by commas (quite a difficult task for many languages), we give them a status of preference feature to be used in constituent attachment; i.e. we prefer attachments of CLAUSE-CONSTs rather than attachments of subconstituents of CLAUSE-CONSTs.

In the text about Banesto (Example 2) the use of constituents that entirely span chunks 2, 4 and 5 (i.e. appositives for 2 and 4 and a PP for 5) is favored. As a matter of fact it seems obvious to a normal reader that none of the PPs inside Chunk5 (except the first one) can modify anything outside such chunk: for example `di 5,6 miliardi` cannot modify `perso`. That seems to happen because commas separate the sentence in well defined subparts.

Clause-Chunks are used to capture such separation. By using the limits imposed by Clause-Chunks, most attachments are done without the need of a comparison with other possibilities involving their subconstituents, greatly reducing the problem of attachment when the complexity of the sentence grows.

3.2 A Refinement: the Direction of Analysis

Building Clause-Chunks separately means not taking into account that the constituents internal to the Clause-Chunk under consideration may have as modifiers some of the other CLAUSE-CONSTs. In the text about Banesto (Example 2) Chunk4 modifies the NP spanning `103,4 miliardi` in Chunk3. Building Chunk3 without considering the rest of the sentence means completely recognizing such NP without attaching one of its modifiers. In general this means regularly sending the parser down a garden path when the text is composed by many Clause-Chunks.

In Italian (the language we are working on) modifiers appear mainly to the right of the head; hence we build CLAUSE-CONSTs starting from the rightmost. In this way when considering the constituents internal to a Clause-Chunk, any CLAUSE-CONST to its right is available for attachments. For this reason tasks composing CLAUSE-CONSTs external to the current Clause-Chunk with edges internal to the current Clause-Chunk are also considered during Step 2. In Example 2 Chunk3 is recognized after Chunk4 and Chunk5 and the composition of parts of Chunk3 with Chunk4 and Chunk5 are considered during Step 2.

[6] Literally, `The Dhl International of-the network DHL achieves a profit gross.`

3.3 Further Details

As mentioned, the aim of the control algorithm is first of all that of rapidly focusing on *satisfactory solutions*. Satisfactory solutions are defined as follows:

- for a word: a lexical edge or its projections;
- for a Head-Chunk: a HEAD-CONST;
- for a Clause-Chunk: a CLAUSE-CONST;
- for a sentence: a CLAUSE-CONST spanning the whole sentence.

To focus on satisfactory solutions, during the three steps of analysis tasks are classified as[7]:

- *Exec-Curr-Step*: a task with immediate evaluation; it involves the composition of edges declared as satisfactory solutions for some sub-chunks; the edge that results from the task execution will still span within the current chunk;
- *Delay-Curr-Step*: a task delayed in the current step; a composition involving non satisfactory solutions for some sub-chunks; resulting edge still spans within the current chunk;
- *Exec-Next-Step*: a task to be favored in the next step; composition involving satisfactory solutions for the current chunk or some of its sub-chunks; resulting edge no longer spans within the current chunk;
- *Delay-Next-Step*: a task delayed in the next step; composition involving non satisfactory solutions; resulting edge no longer spans within the chunk.

Table 1 summarizes the task classification algorithm for the three steps of analysis.

Class/Step	Step 1	Step 2	Step 3
Exec-Curr-Step	INT	INT RULE0-1 or INT-EXT RULE2	INT RULE2
Delay-Curr-Step	none	other INT	other tasks
Exec-Next-Step	INT-EXT RULE0-1	EXT RULE2	none
Delay-Next-Step	other INT-EXT	other tasks	none

Table 1: Task classification during analysis: INT tasks involve edges only spanning within the current chunk; INT-EXT tasks compose an edge of the current chunk with one external to it. EXT tasks involve an edge (generated by the execution of INT-EXT tasks) spanning more than the current chunk and an external edge. RULE0-1 tasks respect the conditions imposed by RULE0 or RULE1.

3.4 Agenda Pruning

Until now we have shown how pursuing the satisfactory solutions as firsts. In this paragraph we will see how to reduce the search space. This is done either when at least one satisfactory solution is found or when we are sure that no satisfactory solution can be found for a specific chunk. The general idea is that of pruning the non satisfactory solutions for the chunks for which a satisfactory solution is available. For the other chunks only a limited number of tasks are executed after the satisfactory solution has demonstrated to be not available; the number of tasks activated is a function of the complexity of the current chunk (e.g. the length of the spanned input).

At the end of the analysis of each chunk (of any type) all the Exec-Curr-Step tasks have been activated. The other tasks are then reclassified according to the algorithm shown in Tables 2 (for cases when a satisfactory solution is found) and 3 (for the other cases).

The final algorithm for task classification (inclusive of agenda pruning) is shown in Figure 4.

[7] Such classification is performed when tasks are generated and inserted into the agenda.

	Step 1	Step 2	Step 3
A: Exec-Curr-Step	activated	activated	activated
B: Delay-Curr-Step	removed	removed	removed
C: Exec-Next-Step	→ 2A	→ 3A	none
D: Delay-Next-Step	removed	removed	none

Table 2: Task reclassification at the end of each step when a satisfactory solution was found for the current chunk; → 2A means that before starting Step 2 the tasks contained in that cell are reclassified as Exec-Curr-Step.

	Step 1	Step 2	Step 3
A: Exec-Curr-Step	activated	activated	activated
B: Delay-Curr-Step	activated	activated	activated
C: Exec-Next-Step	none	none	none
D: Delay-Next-Step	→ 2A	→ 3A	none

Table 3: Task reclassification at the end of each step when NO satisfactory solution was found for the current chunk. → 2A means that the current tasks are reclassified during Step 2 as Exec-Curr-Step.

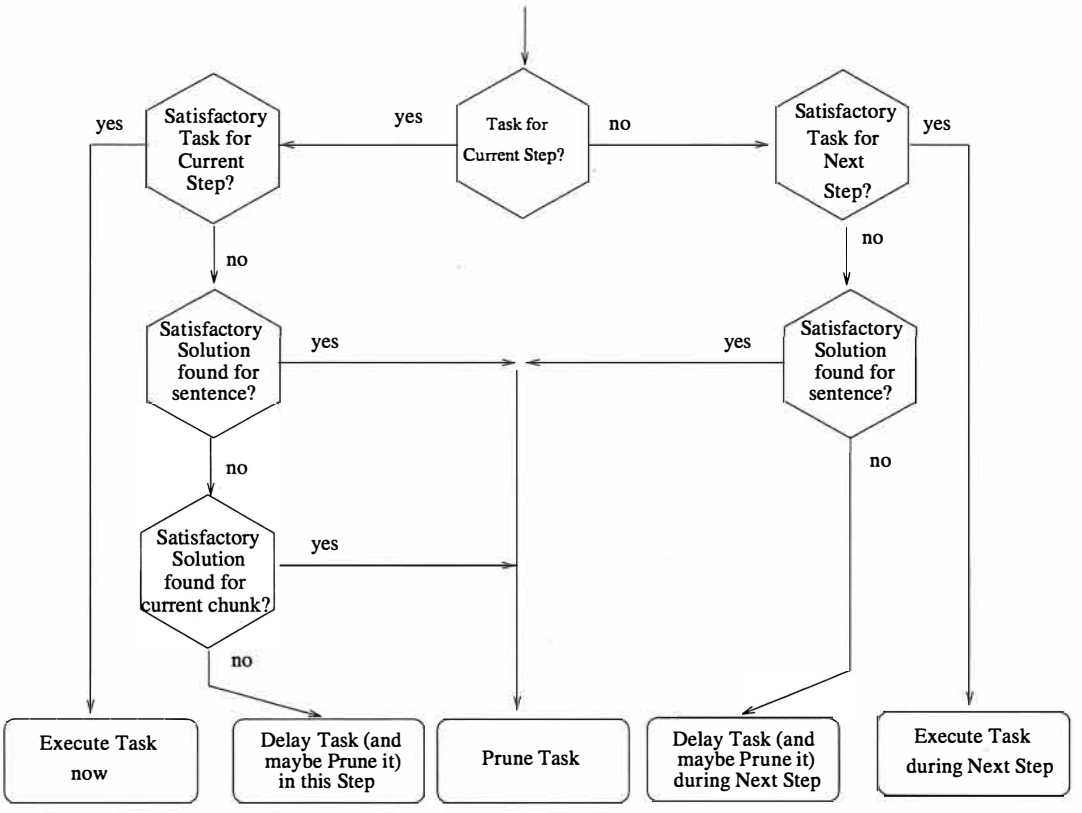

Figure 4: Task Classification Algorithm.

Parsing Strategy	Total edges	average/text	% produced
NON controlled	350,644	2,874	100%
Controlled	110,511	905	32%

Table 4: Edges produced in analyzing 122 sentences.

Parsing Strategy	Global Time	average/text	% Time
NON controlled	06:54:18	00:03:26	100%
Controlled	01:32:25	00:00:45	22%

Table 5: Time consumed in analyzing 122 sentences.

4 Preliminary Experiments

We have carried on some experiments on the effectiveness of the strategy proposed in this paper. Given the lack of corpora for Italian annotated such as in TreeBanks, the amount of data used in such experiments is quite limited, and therefore the results are only preliminary. We plan in the near future to carry on more extensive experiments in order to verify the effectiveness of the approach.

The chunk-based control strategy was added to a CYK-like parser; the parser uses a grammar based on a Typed Features Logic (TFL) [Carpenter, 1992] oriented formalism; the control strategy effectiveness was compared with that of the standard control mechanism in analyzing 122 sentences taken from articles of the Italian financial newspaper "IL SOLE 24 Ore" and news provided by ANSA. The total number of words in the corpus is 2769, an average of 23 words for sentence. Table 4 and 5 show the results of the comparison.

As it may be seen, there is a reduction of about 68% of constituents (edges) generated and of 78% of time consumed. Note that both parsers were constrained to produce less than 10,000 edges per sentence; the non-controlled version reached 21 times such limit, against 4 of the controlled; this means that a completely uncontrolled parser could generate more edges and spend more time than showed and that the effectiveness of the strategy could increase in the tables.

Concerning correctness of the results, this is the most difficult part to be tested given the lack of annotated corpora for Italian. We have manually checked the correctness of the results on 31 sentences among the most complex of our corpus. The controlled version of the parser was able to find 29 correct solutions missing 2. The two errors were due to the pruning strategy based on Clause-Chunks. In both cases the parser missed to produce a complete parse tree and produced 2 meaningful partial parses. The non-controlled parser produced 27 correct solutions out of 31; missing 4 solutions when the upper limit of 10,000 edges was reached. In such cases the quality of the partial parses was kind of random, due to the LIFO strategy in activating the tasks in the agenda: some parts of the input were fully analyzed, other were completely unanalyzed; with a FIFO strategy the partial results could be better, but the number of the missing solutions could increase as the edge limit could be reached for other sentences. The inability of the non-controlled parser to find in some cases the correct solutions due to the upper limit of 10,000 edges is an interesting point. Some other experiments show that reducing the limit to 9,000 the number of missing solutions grows for the non controlled parser; on the contrary, for making the controlled parser losing further solutions it is necessary to bring that limit under 4,000. Further experiments on the 122 sentences show that, establishing for the controlled parser an upper limit of 4,000 edges (instead of 10,000), only 25% of the edges were produced and 12% of the time was consumed, if compared with the non-controlled one (with the limit of 10,000 edges). We are currently investigating the impact of the 4,000 limit on the correctness of the results on all the 122 sentences. Anyway, considering that we use the controlled parser in an application of Information Extraction from text (where there are demanding time and space constraints and complete solutions are not necessarily required) and considering that the analysis of only 10 out of 122 sentences exceeded the limit of 4,000 edges, we are confident that such limit can provide a suitable trade-off between parsing efficiency and effectiveness.

As for the relationship between sentence length and effectiveness of the proposed approach, some qualitative considerations can be drawn. As said above, the sentences were taken from two different sources, the short news of a financial newspaper and a news agency. In the former case, sentences are short (the length is usually less than 25 words), the text grammatical structure is plain and the linguistic phenomena used are very often completely dealt with by the grammar; the improvement introduced by our strategy on these texts is relevant.

As for the latter case, agency news are usually longer (35 words on average) and much more complex from a linguistic point of view: nested clauses (such as nested incidentals) and long PP sequences are frequent; the grammar very often does not completely recognize the sentences. In this case the strategy seems to be insufficient in the part relying on the clause level chunking, as - especially when confronted with nested clauses - the current strategy is too straightforward and needs to be refined. Anyway the cut in the search space is always relevant, especially when the grammar does not cover the sentence.

5 Final Remarks

This paper shows how it is possible to modify Abney's definition of Chunk Parsing so to allow its use within a framework such as standard chart parsing. The proposed approach maintains the advantages of Chunk Parsing (mainly the reduction of ambiguity) avoiding most of its problems such as the use of a mixed constituency/dependency parsing and the complexity of constituent attachment at the clause level. The performances of the chart parser based on preliminary chunking outperformed those of one adopting a traditional LIFO strategy for ordering tasks in the agenda, without significantly losing in correctness. Introducing the control strategy in a pre-existing parser required a limited effort (about one person/month); the chunking grammar accounting for three levels of chunking is composed by about 20 rules in all. The controlled parser is now used within a system for Information Extraction (IE) from economical news: more precisely it is used in FACILE [FACILE-Team, 1996], a 3-year project funded by the European Union within the framework of Language Engineering. The goal of the project is to build a set of tools for categorization of financial texts and information extraction from them. The general aim of an application derived from such tools is message dispatching and routing for financial institutions (banks, trading companies, rating companies). Relevant texts include agency news and newspaper articles.

We think that the chunk-based strategy is particularly suited for Information Extraction from text where there is a strong interest in focusing on the most likely phenomena [Lehnert, 1994] even if it can cost in terms of completeness of the analysis. When the parser focuses on the solution proposed by chunking, actually promotes constituents that are supposed to be likely, given the current sentence. The (possible) analysis of parts of the search space outside the chunking hints is an attempt to look also for alternative solutions when they are needed; limiting the number of steps done when looking for alternative solutions is a way for pursuing such solutions only if they are not too distant from the expectation (i.e. if they are not too unlikely). Moreover the experiments show that the partial results produced by the controlled parser are interesting from a qualitative point of view (and the quality of partial results is another requirement for IE).

The use of chunking for controlling the parser is coherent with the current trends in IE towards the use of finite state models of language for helping in analyzing texts: these models have been used for preprocessing (e.g. in proper name recognition [Vilain and Day, 1996]) or even for the whole IE process [Appelt et al., 1993]. In our approach finite state devices are used for chunking input texts: the results are later on used by the control mechanism of the parser in order to decide how to visit the search space.

Other authors use techniques similar to preliminary chunking for information extraction. McDonald [McDonald, 1992] pre-segments the input that is then analyzed by a bidirectional chart parser adopting a semantic grammar. McDonald's segments broadly correspond to our Mod-Chunks and are used in a similar manner. The problem of the complexity of attachment in long sentences (even when using semantic knowledge for solving attachment problems) is not taken into consideration.

Terminal Substring Parsing is used in TACITUS [Hobbs et al., 1992] for avoiding the complete chart-based analysis of very long sentences; sentences are segmented around commas and other function words (i.e. in a way similar to our Clause-Chunk) and segments are analyzed from right to left. Terminal Substring Parsing is adopted only for very long sentences and not for all the sentences as we do; moreover within the segments the overgeneration of false constituents is not controlled as we do by using the information on Head-Chunks and Mod-Chunks.

References

[Abney, 1991] Abney, S. P. (1991). Parsing by chunks. In Berwick, A. and Tenny, editors, *Principle-Based Parsing*. Kluwer.

[Abney, 1995] Abney, S. P. (1995). Chunks and dependencies: Bringing processing evidence to bear syntax. In *Computational Linguistics and the Foundation of Linguistic Theory*. CSLI.

[Appelt et al., 1993] Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D., and Tyson, M. (1993). FASTUS: A finite-state processor for information extraction from real-world text. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France.

[Carpenter, 1992] Carpenter, B. (1992). *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, Massachusetts.

[FACILE-Team, 1996] FACILE-Team (1996). Facile project summary. description available at http://www2.echo.lu/langeng/en/le1/facile/facile.html.

[Hobbs et al., 1992] Hobbs, J. R., Appelt, D. E., and Tyson, M. (1992). Robust processing of real-world natural-language texts. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Trento, Italy.

[Lehnert, 1994] Lehnert, W. (1994). Cognition, computer and car bombs: How Yale prepared me for the 90's. In Schank and Langer, editors, *Beliefs, Reasoning and Decision Making: Psycho-logic in Honor of Bob Abelson*, pages 143–173. Lawrence Erlbaum Associates.

[McDonald, 1992] McDonald, D. D. (1992). Robust partial-parsing through incremental, multi-algorithm processing. In Jacobs, P. S., editor, *Text-Based Intelligent Systems*. Lawrence Erlbaum Associates.

[Satta and Stock, 1994] Satta, G. and Stock, O. (1994). Bi-Directional Context-Free Grammar Parsing for Natural Language Processing. *Artificial Intelligence*, 69(1-2):123–164.

[Vilain and Day, 1996] Vilain, M. B. and Day, D. (1996). Finite-state phrase parsing by rule sequences. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 274–279, Copenhagen, Denmark.

[Wiren, 1987] Wiren, M. (1987). A comparison of rule invocation strategies in context-free chart parsing. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark.