UNIVERSITÉ DE LORRAINE | UFR MATHÉMATIQUES ET INFORMATIQUE

Universite de Lorraine

Master Sciences Cognitives

Option : Traitement Automatique des Langues

Solution Ranking for a Symbolic Parser

Master Thesis, Academic year 2012-2013

Author:
Tatiana Ekeinhor

Supervisor:
Dr. Bruno Guillaume

Sémagramme Team

Loria
Laboratoire lorrain de recherche en informatique et ses applications

June, 2013

Acknowledgements

I would like to express my gratitude to my supervisor Bruno, for his guidance and availability during all my work.

I am also grateful to all those who helped me, with their advice and support during my school year and even more my thesis.

Finally I wish to thank my parents, Nyedji and Faadji, for their support and encouragement throughout my study.

Abstract

Solution Ranking for a Symbolic Parser

by Tatiana Ekeinhor

Ranking is one of the methods used to improve data-driven parsers. The task of the ranker is to define a function that will assign a score to each parse candidate tree obtained in the parsing process.

We want to apply such models to resolve ambiguity problems in LEOPAR, a grammar-driven parser. LEOPAR produces a high number of parse solutions especially when the size of the input sentence grows. We want to choose the most suitable solutions among the parse candidates proposed by LEOPAR. We have to notice that a scoring system exists already in LEOPAR. It is based on some handcrafted rules.

We present in this document a ranking solution for LEOPAR based on statistical techniques. The candidate parses provided by LEOPAR are used as input for our ranker. We test our approach on the Sequoia TreeBank and obtain an improvement of the system compared to the handcrafted rules.

Contents

Abbreviations

NLP: **N**atural **L**anguage **P**rocessing

LEOPAR: e**LE**ctr**O**static **PAR**ser

MIRA: **M**argin **I**nfused **R**elaxed **A**lgorithm

ACG: **A**bstract **C**ategorial **G**rammar

CG: **C**ategorial **G**rammar

LFG: **L**exical **F**unctional **G**rammar

HPSG: **H**ead-Driven **P**hrase **S**tructure **G**rammar

TAG: **T**ree **A**djoining **G**rammar

IG: **I**nteraction **G**rammar

FRIG: **F**rench **I**nteraction **G**rammar

SVM: **S**upport **V**ector **M**achine

PCFG: **P**robabilistic **C**ontext **F**ree **G**rammar

LAS: **L**abelement **A**ttachement **S**core

PA: **P**assive **A**ggressive

POS: **P**art **O**f **S**peech

1 Introduction

A natural language parser is a program that builds the grammatical structure of natural language input strings. It decomposes sentences from natural language into syntactic parts and defines the relationship between these parts. Parsing is a traditional problem in the Natural Language Processing field and this task reveals to be hard. Indeed, natural language is complex by its *ambiguity* (human language allows more than one syntactical decomposition), *variability* (rules do not cover every detail in natural language) and *openness* (human language is continuously changing) (Barbero and Lombardo, 1995).

The core of our work deals especially with the *ambiguity* problem. All our examples will be in French as the tools we will use are developed for French language. Let us consider an example to get an idea of the problem:

(1.1) *La belle ferme la porte.*
THE BEAUTY CLOSE THE DOOR.
The beauty closes the door.

(1.2) *La belle ferme la porte.*
THE PRETTY FARM IT CARRY
The pretty farm carries it.

(1.3) *La belle ferme la porte.*
THE BEAUTY FIRM IT CARRY
The firm beauty carries it.

The interpretation 1.1 is the most obvious one. The 1.2 comes from the fact that *belle* can be an adjective and *ferme* a noun. We then obtain *the pretty farm carries it.* The last one has the most tricky interpretation with *ferme* seen like a modifier of *belle* and *porte* being a noun. The resulting meaning can be seen in 1.3.

Handling this ambiguity problem consists in choosing the best parse from candidate analyses provided by a parser. Each analysis is built according to different techniques. We distinguish grammar-driven techniques and data-driven techniques for parsing.

Grammar-driven parsers use linguistic resources (grammars) in order to process parsing when data-driven parsers use machine learning methods based on corpora resource to do so. For data-driven parsers, there is a natural way to choose among the candidate solutions because the parsing provides score or probability to each parse according to decisions made during the process.

In our work, we will look for a way to assign a score to the candidate analyses of a grammar-driven parser, LEOPAR (Guillaume and Perrier, 2009).

2 From Parsing to Ranking

2.1 Parsing

Parsing refers to the process of analysing the syntactic structure of a given input. It consists in decomposing a complex structure into its components. It was first applied to programming languages for their compilation and interpretation. Then it was used on natural language. As it is the case for programming languages, parsing helps the interpretation.

"It is generally accepted that finding the structure of a sentence can be useful in determining its meaning." (Charniak, 1997).

Parsing can be seen as a twofold task. The first one is to determine all the possible syntactic structures of sentences and the second is to select the most suitable ones. Depending on the way descriptions are made, we distinguish two major representations of the syntactic structure of a sentence: the *constituent structure*, also called *phrase-structure* (or constituency tree) and the *dependency structure* (or dependency tree). In *phrase-structure* representation, sentences are represented by phrases (non-terminal nodes), structural categories (non-terminal labels) and, sometimes, functional categories (grammatical functions). Constituency rules describe how to combine phrases (like noun phrases, verbal phrases...), in order to obtain a sentence. Figure 2.1 presents a phrase-structure representation of the sentence 2.1:

(2.1) *Jean regarde la fille.*
JOHN LOOK AT THE GIRL.
John is looking at the girl.

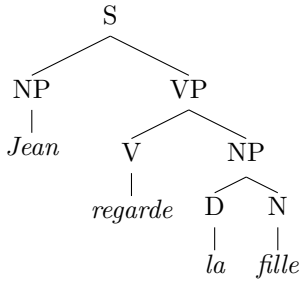


FIGURE 2.1: Example of constituency tree

In *dependency* view, as in figure 2.2, the main verb is the structural center of the representation. All other syntactic units depend, either directly or indirectly, on this verb.

Even if both approaches (phrase-structure vs dependency structure) are often presented in opposition, we have to notice that they are complementary.

The dependency theory (Tesnière and Fourquet, 1959) is based on the idea that the syntactic structure of a sentence consists of *binary asymmetrical relations* that express functional categories

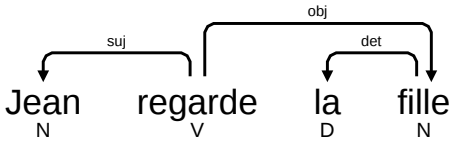


FIGURE 2.2: Example of dependency tree

(arc labels) between the words of the sentence. This kind of relation links (directed arcs) a governor (or head) to a dependent (or modifier). In dependency parsing, a dependency tree is the result of the computation, where nodes are the word of the input sentence, and the arcs are the binary relations from head to dependent. On representations, arrows go from governor to dependent. In our example of figure 2.2, the wordform *regarde* governs *fille* with the relation *obj*. Dependency representation is also found more adequate for languages which allow greater freedom in word order.

Some of the criteria for establishing a dependency relation between a head H and a dependent D, in a construction C, are of type (Zwicky, 1985):

- H determines the syntactic category of C
- H is mandatory, D may be optional
- The form of D depend on H
- ...

Natural language parsers are programs that analyse the structure of a sentence. These programs are built in different ways. According to the source of the knowledge involved for the analysis of a sentence there are two main types of parsers: *grammar-driven* and *data-driven*. Grammar-oriented and data-oriented systems are not mutually exclusive. There exist grammar-driven parsers that include statistical disambiguation components (like in the XTAG Project (XTAG Research Group, 2001)). LEOPAR parser (Guillaume et al., 2008) is a grammar-driven parser. In this work, we aim to incorporate statistical information in order to resolve ambiguity problems and rank the output obtained after the parsing phase of LEOPAR.

2.2 Grammar-driven Parsing

A grammar-driven parser is a system where grammars are employed to deduce the parses of an input string of a natural language.

How is a sentence structured in natural language? It is in response to this question that grammars have been developed. Grammars are sets of rules that govern how syntactical units (clauses, phrases, ...) are structurally constructed. They also help determining whether a sentence belongs to a language or not. The goal of a grammar is to model a language and to bring out linguistic knowledge involved in that language.

A formalism is a precise specification of a language. A grammar formalism is a linguistic description device. Many grammar formalisms exist like Lexical Functional Grammar (LFG) (Kaplan and Bresnan, 1982), Head-Driven Phrase Grammar (HPSG) (Pollard and Sag, 1994), Tree Adjoining Grammar (TAG) (Joshi and Schabes, 1997) just to mention a few.

In the grammar-driven approach, parsing can be considered as the process of solving a set of constraint problems where the grammatical rules define those constraints. More concretely, a sentence is seen as a sequence of words. By analysing them, the parser assigns lexical functions and defines head-dependent relations. We can cite LEOPAR parser and Link Parser (Sleator and Temperley, 1993) as grammar-driven parsers. One problem of this approach is that the used grammar may not be exhaustive. It is an approximation of the language that it aims to process. Robustness is defined as the capacity of a parser to analyze any input sentence, and the lack of robustness is a weak point of the grammar-driven parser systems. The last but not least need of this method that makes it hard to implement is that, it requires considerable linguistic and computational expertise.

2.2.1 Interaction Grammars

In the following, we will introduce how the parsing system we want to improve works. LEOPAR is a grammar-based parser that uses a grammar formalism called Interaction Grammar (IG). It is a lexicalized phrase-structure grammar as opposed to dependency structure grammar.

"*Interaction grammars combine two key ideas: the grammar is viewed as a constraint system, which is expressed through the notion of tree description, and the resource sensitivity of natural languages is used as a syntactic composition principle by means of a system of polarities*" (Guillaume and Perrier, 2009).

IG mixes properties of different grammar formalisms. Therefore this formalism uses the tree description notion of grammars like Tree Adjoining Grammar (TAG). It is inspired by the Categorial Grammars (CGs). Indeed CGs are based on the principle of composition and are organized according to the view that syntactic constituents should generally be combined as functions or according to a function-argument relationship. In IG it is expressed through a *polarity* system where linguistic resources are considered as consuming resources. The polarity expresses the saturity of syntactic tree, in other words their capacity to be combined with other elementary trees.

A tree description is a set of nodes and hierarchical relations between them. Nodes represent phrases and the hierarchical relations represent how phrases are embedded one inside another. The syntactic properties of the phrases are expressed through features structures. A tree description is then a specification of a set of syntactic trees and each tree is a model of the tree description.

"[...] A positive node represents an available grammatical constituent whereas a negative node represents an expected grammatical constituent". (Guillaume and Perrier, 2009)

IG also uses the features structure views of Unification Grammars like Lexical Functional Grammar (LFG) and Head-driven Phrase Structure Grammar (HPSG). IG specifies tree categories and operations like superposition and composition. The tree description is a model of specification that associates a model to a word. It can be viewed as a set of partially specified tree which are superposed under polarity controls. Figure 2.3 shows an example of tree description.

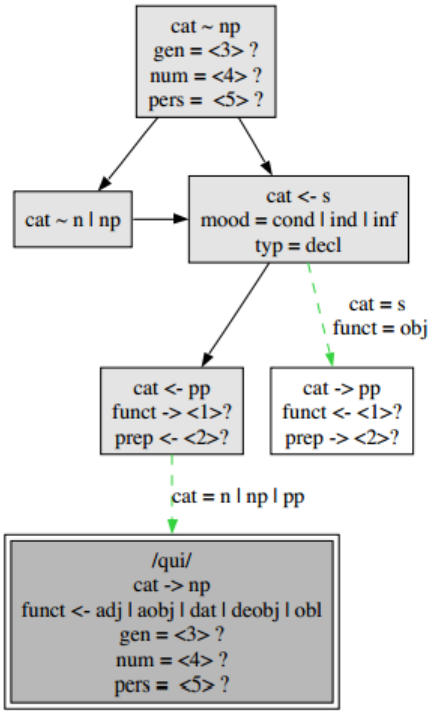


FIGURE 2.3: Tree description associated with the French relative pronoun *qui* (who) used inside a prepositional complement

A polarized tree description is a set of constraints both on the tree structure and on the feature structures of syntactic trees. There is a distinction between Polarize Tree Description (PTD) and Elementary Polarized Tree Description (EPTD). A PTD is a structure used during the parsing process. An EPTD is an elementary tree defined in the grammar. FRIG (FRench IG) is a set of linguistic resources that embeds a grammar, *frigram*, and a lexicon, *frilex* for the French language. There are two levels in the grammar formalism: the *source grammar* and the *object grammar*. The source grammar represents linguistic generalisations and is written by a human.

The object grammar is used by a Natural Language Processing (NLP) system and results from the compilation of the first one. The tool used to develop the grammar is called XMG (Duchier et al., 2005). XMG provides a high level language for writing source grammars and a compiler which transforms the grammars into operational object grammars. Frigram is a relatively large coverage grammar that contains about 4,000 EPTDs. Frilex is a lexicon of 528,949 French word entries. [1] In figure 2.4, we can see an example of tree associated with the sentence *Jean la voit*.

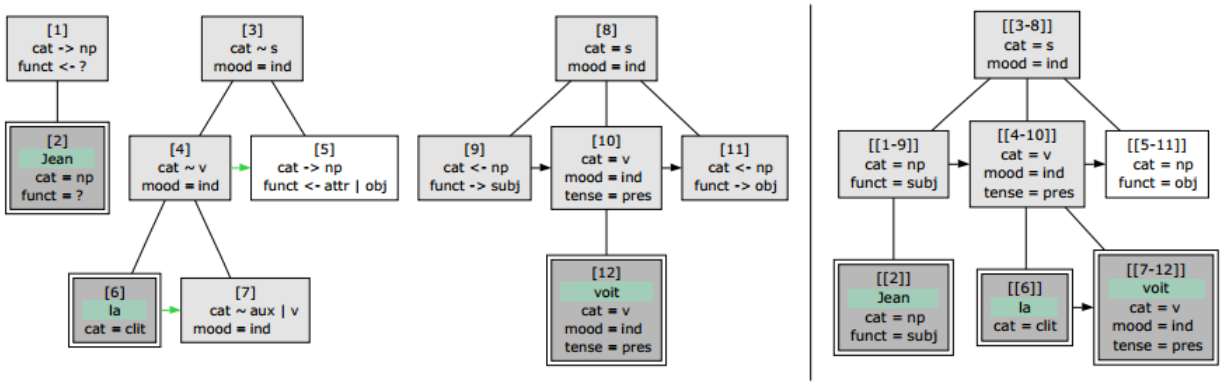


FIGURE 2.4: PTD associated with the sentence *Jean la voit* and its minimal saturated model

2.2.2 LEOPAR

In order to make experiment, a parser based on IG was developed and is named LEOPAR[2]. LEOPAR (e**LE**ctr**O**static **PAR**ser) is a syntatic parser based on Interaction Grammar formalism. In order to parse a sentence, each word of the sentence is associated with an elementary tree (EPTD) contained in the lexicon. This action is denoted by *lexical selection*. The number of possible lexical selections may present an exponential complexity in the length of the sentence. Specifics methods were designed to remove most of the wrong lexical selection at a minimal cost. *"The lexical disambiguation module checks the global neutrality of every lexical selection for each polarized feature: a set of trees bearing negative and positive polarities can only be reduced to a neutral tree if the sum of the negative polarities for each feature equals the sum of its positive polarities."* (Guillaume et al., 2008)

The resulting lexical selections are presented in the form of an automaton, and are sent to LEOPAR. LEOPAR processes the composition of two syntactic tree descriptions. This operation is seen as an electrostatic process during which an elementary operation consists of the identification of two nodes from the anchored grammar.

"[...] Negative nodes tend to merge with positive nodes of the same type and this mechanism of neutralization between opposite polarities drives the composition of syntactic trees to produce trees

[1] For one word we can have many entries. For instance for a verb, one entry defining it as a transitive verb and another as an intransitive one.

[2] LEOPAR is freely available under the CeCILL license at http://www.loria.fr/equipes/calligramme/

in which all polarities have been saturated". This operation is called neutralization of polarities" . (Guillaume and Perrier, 2009)

The parsing succeeds if the resulting tree has only neutral polarities. LEOPAR is built to produce phrase-structure trees but It contains a constituency to dependency converter, so It can output dependency structures. In appendix A.1 an example of final tree and its transformation into dependency structure is provided.

LEOPAR has a command line and a graphical user interface. The parsing process can be visualised at each step via LEOPAR in order to check the behaviour of the grammar and the lexicon. LEOPAR allows the user to process the parsing manually and the user can choose himself the lexical selection of the sentence and neutralize the nodes for the selection.

In the two steps of lexical disambiguation and parsing, all possible solutions are kept without discarding even the less probable ones. This leads to a large amount of possible syntactic structures. We will use dependency structures in our work since the final output of LEOPAR is a dependency tree. Now the goal is to use statistical methods in order to discriminate the solutions of the parsing.

2.2.2.1 LEOPAR Scoring Method

There is no natural way to choose among parses of a grammar-driven system when it provided several parses in the theory. However by including some statistical information or inference rules, a score or weight can be assigned to parses. LEOPAR uses handcrafted rules to compute score of parses. The weight of a dependency tree as computed by LEOPAR parser is a sum of values of the two operations. The first operation follows rules below:

- the starting value is set to 1,000,
- $-50\times$ number_of_node (of a sentence),
- each dependency tree will receive a weight value.

The value obtained after this computation is then add to the result value of a second operation. This second operation attributes a value according to dependency relation with following configuration, $\delta = dep\_position - gov\_position$:

- $-|\delta|$: allow to penalize long dependencies,
- a value given by the label, the part of speech (POS) of governor that follows rules listed in the table 2.1.

Type of dependency	Value
passive	50
aff	40
agt	40
obj	if $gov = v \mid aux \rightarrow 40$ else 30
suj	if $gov = v \mid aux \rightarrow 40$ else 30
iobj	30
agt tense	30 30
mod	if $gov = v$ and $dep = adj \rightarrow -30 - 10 * delta$ else if $gov = v \mid aux$ and $\delta > 0 \rightarrow 20$ else 10
_	10

TABLE 2.1: Table with the rules used according to dependency type

2.3 Data-driven Parsing

The second framework is the statistical data-driven parsing. We can cite (Charniak, 2000), (Klein and Manning, 2003) or (Nivre et al., 2006), just to mention a few of the works in this domain. Here the parsing process is based on machine learning and the knowledge of the language is gained from hand-parsed corpora. The parser is trained on corpora and tries to produce the most accurate analysis. A formal model defines possible analysis. A sample data is used to train the model and an inference is made based on previous assumption for parsing.

In data-driven approaches, another split can be made according to the model used. On one hand we have the graph-based model. Here, dependency trees are seen like graphs. After decomposing a graph into arcs, scores are attributed to arcs. Parsing is processed by looking for the highest scoring graphs. On the second hand we have transition-based models, where dependency trees are seen like transition states. Parsing is performed by constructing the optimal transition sequence using the transition history.

As we said, data-driven methods are machine learning approaches to natural language parsing. In others words models are built using machine learning on existing data. Before introducing what machine learning is and how parsing is modelled in data-driven systems, we will shortly remind what probability theory is.

2.3.1 Probability Theory

A probability theory can be understood as a mathematical model for the intuitive notion of uncertainty. To represent this notion or uncertainty we need a *probability space* defined regarding

an *experiment*.

Definition 2.1. A *probability space* is a triple $(\Omega, \mathcal{F}, P)$ consisting of:

- A sample space Ω which is a set of outcomes (Ω is non-empty).
- $\mathcal{F} \subset \mathcal{P}(\Omega)$ a set of subsets of Ω called events.
- P a probability measure which is a function from $\mathcal{F}$ to $[0, 1]$.

Definition 2.2. A set S is said to be *countably infinite*, if there is a one-to-one correspondence between $\mathbb{N}$ and S.

For all the following definitions Ω is finite or has a *contably infinite* number of outcomes. This is the *discrete probability theory*. In the generalized theory called *continuous probability theory*, $\Omega = \mathbb{R}^k$, $k \in \mathbb{N}$.

Definition 2.3. An event $A \in \mathcal{F}$ is a set of elements of Ω. $P(A)$ represents how likely is the outcome to be member of A.

Definition 2.4. We denote by N the number of elements in Ω. An experiment is a countable set of outcomes, together with a function $p : \Omega \to [0, 1]$ with the property:

$$p(\omega) = \frac{1}{N}, \omega \in \Omega$$

$$\sum_{\omega \in \Omega} p(\omega) = 1$$

The function p is called a *probability mass function*.

Definition 2.5. For all $A \subseteq \Omega$ we define the *probability measure* P of an event A as:

$$P(A) = \sum_{\omega \in A} p(\omega)$$

P has to obey three rules:

- $P(A) \geq 0$
- $P(\Omega) = 1$
- For two disjoint events A and B, $P(A \cup B) = P(A) + P(B)$.

(2.2) We consider an experiment which consists in rolling a fair dice. There are six possible outcomes: Obtain 1 (o_1), Obtain 2 (o_2), Obtain 3 (o_3), Obtain 4 (o_4), Obtain 5 (o_5), Obtain 6 (o_6).

$\Omega = \{o_1, o_2, o_3, o_4, o_5, o_6\}$, $|\Omega| = 6$

$p(o_k) = \frac{1}{6}$, $k = 1, ...6$

There are 2^6 possible events. For instance A_1 = (Obtaining an odd number), A_2 = (Obtaining a prime and even number), A_3 = (Obtaining an odd or even number), A_4 = (Obtaining an odd and even number), A_5 = (Obtaining an even number), A_6 = (Obtain 2, 3, 4 or 5).

The associated probabilities are displayed in table 2.2.

Event	Probability
A_1: odd	$\frac{3}{6}$
A_2: prime and even	$\frac{1}{6}$
A_3: odd or even	$\frac{6}{6}$
A_4: odd and even	$\frac{0}{6}$
A_5: even	$\frac{3}{6}$
A_6: $\{o_2, o_3, o_4, o_5\}$	$\frac{4}{6}$

TABLE 2.2: Table of events and associated probabilities

Definition 2.6. Suppose A and B are events in a sample space and suppose that $P(B) > 0$. The *conditional probability* of A given B is defined as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$P(A \cap B)$ is also denoted by $P(A, B)$.

(2.3) Let us consider our dice example. Suppose that we are looking for the probability that the number 6 comes up, knowing that the outcome is greater than 4.

$P(o_6) = \frac{1}{6}$, $P(\{5, 6\}) = \frac{2}{6}$, $P(\{o_6\} \cap \{5, 6\}) = \frac{1}{6}$

Then $P(\{o_6\}|\{5, 6\}) = \frac{1}{2}$

Given A and B, we can introduce two main rules of probability:

- **product rule**: $P(A, B) = P(B|A)P(A)$.
- **sum rule**: $P(A) = \sum_B P(A, B)$.

x	X
o_1	-1
o_2	1
o_3	-1
o_4	1
o_5	-1
o_6	1

TABLE 2.3: A random variable with it values

Theorem 2.7. *Bayes' Rules*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In the Bayesian theory, we have such interpretation:

- $P(A)$ is called the *prior*, it is the initial degree of belief in A.
- $P(A|B)$ is the *posterior*, it is the degree of belief having accounted for B.
- The quotient $\frac{P(B|A)}{P(B)}$ represents the support B provides for A.

Theorem 2.8. *Let $A_1, ..., A_n$ be partitions of Ω such that $P(A_k) > 0$ and B be any event such that $P(B) > 0$. The sum rule gives:*

$$P(B) = \sum_{j=1}^{n} P(B|A_j)P(A_j)$$

Then we obtain the generalization of the Bayes rules for all k:

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{j=1}^{n} P(B|A_j)P(A_j)}$$

(2.4) Suppose that we roll a dice. Each time an odd number comes up we lose one euro, and when an even number comes up we win one euro. We write $\omega = (\omega_1, ..., \omega_n)$, where $\omega_i = 1$ if the i'th roll is an even number and $\omega_i = -1$ otherwise. The winning after n rolls is:

$$S(\omega) = \sum_{i=1}^{n} \omega_i$$

The winning S is a mapping from $\Omega \to \mathbb{R}$. Such mappings are called *random variables*

Definition 2.9. A *random variable* is a mapping from $\Omega \to \mathbb{R}$.

We will write $\{X : x\}$ for $\{\omega : X(\omega)\}$ and $P(X = x)$ for $P(\{\omega : X(\omega) = x)\}$

Definition 2.10. We describe as the *probability mass function* of a random variable X the function $p_X : \mathbb{R} \to [0, 1]$ given by:

$$p_X(x) = P(X = x)$$

Definition 2.11. The *distribution function* of a random variable X is the function $F_X : \mathbb{R} \to [0, 1]$ given by:

$$F_X(x) = P(X \geq x)$$

There are many different distributions functions like the normal distribution, the Poisson distribution, the binomial distribution and so on.

Definition 2.12. We consider two random variables $X = (x_1, ...x_n)$, $Y = (y_1, ...y_m)$.

The *joint* probability of X and Y is denoted by: $P(X = x_i, Y = y_j)$ and represents the probability that X takes the value x_i and Y the values y_j.

The *conditional* probability of X and Y is denoted by: $P(X = x_i|Y = y_j)$ and represents the probability the X takes the value x_i knowing that Y took the value y_j.

2.3.2 Machine learning

"Machine Learning is a field of study that gives machine the ability to learn without being explicitly programmed". Arthur Samuel

"A machine is said to learn from experience E with respect to a task T and a performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". Tom Mitchell

Consider that we want to predict whether an incoming mail is a spam or not. The goal is to build a machine that will take mails as input and will predict *spam* or *not spam* as the output. To do so, a large set of mails marked as *spam* or *not spam* is used to set the parameters of an adaptive model. A function is obtained, which takes an input and returns the corresponding output. This is determined during a phase called *training phase* or *learning phase*, based on training data. After the training phase, the system has to identify the output of new input data, this is the *test phase*.

According to the type of data and the problem to solve, we distinguish between:

- **Supervised Learning**: In this case, the training data include input examples along with their corresponding output values. This kind of data is called labelled data. Cases like predicting the output of input instances, where these outputs are discrete class values, are called a *classification task* (e.g. Text classification, Part of Speech Tagging...). When the output consists of one or more continuous variables then it is called a *regression task*.

- **Unsupervised Learning**: Here, the training data consist of example input without any corresponding output value. We speak about unlabelled data. The goal of this method is to find similarity in data and create groups or classes according to that similarity. This is called *clustering* (e.g. Image recognition, Voice separation...).

- **Semi-Supervised Learning**: In this case, the training data combines both labelled and unlabelled examples (e.g. some statistical parsers...). It mixes the advantage of both supervised and unsupervised learning. Unlabelled data are used because they are less expensive to build when labelled data bring information used to improve the accuracy of the algorithm.

- **Reinforcement Learning**: *"The problem is to find the suitable action in a given situation in order to maximize a reward. There is no example of optimal output, the system has to discover it by mean of trial and error. A general feature of reinforcement learning is the trade-off between exploration, in which the system tries out new kinds of actions to see how effective they are, and exploitation, in which the system makes use of actions that are known to yield a high reward. Too strong a focus on either exploration or exploitation will yield poor results."* (Bishop et al., 2006) (e.g. multi-agent system, flocking simulation...).

In our work, we are interested in supervised methods.

2.3.2.1 Supervised Methods

In this section, we will talk about the classification problem in supervised machine learning. The reason is that we are so far, only interested in categorical output (parse selection) and not in numerical one.

Definition 2.13. We consider a set V with the addition $(+)$ and multiplication $(\cdot)$ operations. V is a vector space over a Field F if and only if:

- $(V, +)$ is a commutative group which means that:
  - $\forall\ (\vec{u}, \vec{v}) \in V \times V,\ \vec{u} + \vec{v} = \vec{v} + \vec{u}$,
  - $\forall\ (\vec{u}, \vec{v}, \vec{w}) \in V \times V \times V,\ \vec{u} + (\vec{v} + \vec{w}) = (\vec{u} + \vec{v}) + \vec{w}$,
  - V contains the *additive identity* element denoted by $\vec{0}_V$, such that $\forall\ \vec{u} \in V,\ \vec{u} + \vec{0}_V = \vec{0}_V + \vec{u} = \vec{u}$,
  - $\forall\ \vec{u} \in V$, the equation $x + \vec{u} = \vec{0}_V$ and $\vec{u} + x = \vec{0}_V$ have a solution in V called additive inverse and denoted by $-\vec{u}$.

- $(\cdot)$ verifies the following properties:
  - $\forall\ (\alpha, \beta) \in F \times F$, and $\vec{u} \in V,\ \alpha \cdot (\beta \cdot \vec{u}) = (\alpha \cdot \beta) \cdot \vec{u}$
  - $\forall\ \vec{u} \in V,\ 1_F \cdot \vec{u} = \vec{u}$

– $\forall\ (\alpha, \beta) \in F \times F$, and $\vec{u} \in V$, $(\alpha + \beta) \cdot \vec{u} = \alpha \cdot \vec{u} + \beta \cdot \vec{u}$,

– $\forall\ \alpha \in F$, $\forall (\vec{u}, \vec{v}) \in V \times V$, $\alpha \cdot (\vec{u} + \vec{v}) = \alpha \cdot \vec{u} + \alpha \cdot \vec{v}$

An element of V is called vector.

The aim of the classification is to take an input vector $\vec{x}$ and to assign it one of a K-discrete classes C_k where $k = 1, ...K$. For K = 2 we speak about binary classification. For instance, deciding whether a mail is a spam or not, whether a transaction is fraudulent or not, whether a tumor is benign or malignant.

There are many classes of models for solving classification tasks. The goal of algorithms used in each model is to define a function that will assign to each input vector a value y. We distinguish:

- Model with a discriminant function : $y(x) = w^T x + w_0$, where w^T is the column vector associated to the row vector w

  w is called weight vector and w_0 is called bias. An input vector x is assigned to class $\mathcal{C}_1$ if $y(x) \geq 0$, and to class $\mathcal{C}_2$ otherwise. The decision boundary correspond to $y(x) = 0$, which corresponds to a $(D-1)$-dimensional hyperplane within the D-dimensional input space. The perceptron algorithm (Rosenblatt, 1958) belongs to this class of model.

- Another approach is to model the *conditional probability distribution* $p(C_k|x)$ in an inference stage, and subsequently use this distribution to make optimal decisions. Two techniques exist to determine $p(C_k|x)$:

  – One technique is to model it directly, for example by representing them as parametric models and then optimizing the parameters using a training set. Those are referred to, as the probabilistic discriminative models.

  – Alternatively, a generative approach (generative because it looks for $p(x, C_k)$, the joint probability of x and C_k) can be adopted in which $p(x|C_k)$ is modelled, together with the prior probabilities $p(C_k)$ for the classes, and then the required posterior probabilities using Bayes' theorem will be :

  $$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}$$

  This is the probabilistic generative model. Iterative Reweighted Least Squares (IRLS) is an algorithm of this class of models. It is used to find maximum likelihood [3] solution of the parameters of the model.

[3]The likelihood of a set of parameter values, θ, given outcomes x, is equal to the probability of those observed outcomes given those parameter values, that is $\mathcal{L}(\theta|x) = P(x|\theta)$

2.3.3 Supervised data-driven Parsing: Process

As we said before, ambiguity is one of the central problems in Natural Language Parsing. Indeed, one sentence (even a short and simple one) can have a large number of parses. Statistical parsing approaches try to resolve the ambiguity problem by assigning a probability to each parse tree, thereby ranking trees in matter of relevancy.

In many statistical models, the probability for each candidate tree is calculated as a product of terms, each term corresponding to some substructure within the tree. In order to generate the most likely derivation of a sentence a probabilistic model is required. Most employed model theories are generative. A generative model is a model that uses joint probabilities (as opposed to discriminative model where conditional probabilities are used). We will expose a specific kind of generative models, the *history-based model* as described in (Collins and Koo, 2005). In history-based models, a parse tree is represented as a sequence of decisions, the decisions being made in some derivations of the tree. Each decision has an associated probability, and the product of these probabilities defines a probability distribution over possible derivations.

We can describe parsing as a machine learning task to induce a function f from $\mathbf{X}$ to $\mathbf{Y}$, given a training data.

$\mathbf{X}$ is a set of sentences, $\mathbf{Y}$ a set of parse trees. $Y(X)$ is defined as the candidate parses for a sentence X, $Y(X) \subset \mathbf{Y}$.

We are looking for a model that assigns a probability $P(X, Y)$ to every pair sentence-parse (X, Y). Now, the question is how to define $P(X, Y)$.

In history-based models, a mapping is defined between each (X, Y) and a decision sequence $< d_1, ..., d_n >$. A decision sequence can be the sequence of rules or parse moves that build (X, Y) in some order. Given all that, we can compute the probability of a tree as follows:

$$P(X, Y) = \prod_{i=1}^{n} P(d_i | \Phi(d_1, ..., d_{i-1}))$$

$(d_1, ..., d_{i-1})$ is the history of the i'th decision. Φ is a function that groups decisions into equivalent classes.

All that can be summarized within three essential components:

- A formal model
- A sample of data set of pairs $(X, Y_\star)$, where $Y_\star$ is the correct tree of X
- An inducing inference scheme that defines actual analysis of a sentence

An example of this model is the *Probabilistic Context Free Grammars* (PCFG). The inference scheme represents the decisions sequence. For a PCFG, it is the top-down, left most derivation rule of a tree.

Ranking the candidate parses provided by a first parsing process is one solution to improve statistical parsers. It is a solution that uses statistical information downstream of the parsing process. We have to notice that there are other models like in (Nivre and McDonald, 2008), where different models are combined or like the Supertagger (Bangalore and Joshi, 1999), where statistical computations are operated upstream of the process of parsing. We have made the choice to operate downstream the parsing process. We will describe now how the ranking works.

2.3.4 Ranking

Ranking is an ordering relation. Dependency tree ranking is the task of selecting the most plausible dependency tree for a given sentence. In figure 2.5, we can see an example of ambiguity in the syntactic decomposition of the sentence 2.5 :

(2.5) *Jean regarde la fille sur le balcon.*
JOHN LOOK AT THE GIRL ON THE BALCONY.
John is looking at the girl on the balcony.

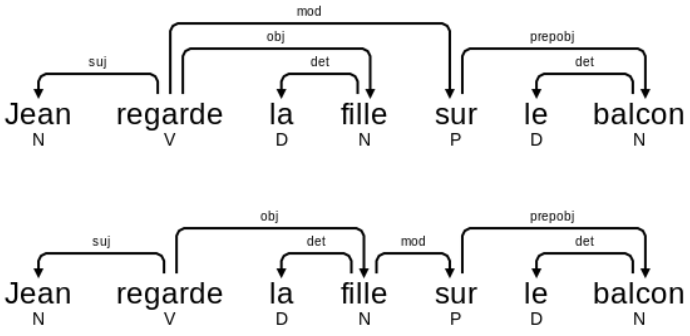


FIGURE 2.5: Example of parse tree provided by LEOPAR

We can see this problem as a machine learning task. There are many applications, the design of search engines, information extraction platforms, movie recommendation systems... This is the same problem when we want to order parse tree results of syntactical derivation process. There are two general settings for this problem: the *score-based* and the *preference-based* settings. By using information in data, a learning algorithm will define a ranking prediction function. This predictor is a real-value function called score function: the scores assigned to the input by this function determine their ranking. To do so, many techniques exist: in the score-based system there are: Margin-based algorithm, Support Vector Machine based ranking algorithm or boosting algorithm for ranking...

The preferred based settings consist of two stages. In the first stage, a sample of labelled pairs S is used to learn a preference function h that assigns a higher value to a pair (u, v) when u is preferred to v or is to be ranked higher than v, and smaller values in the opposite case. This preference function can be obtained as the output of a standard classification algorithm trained on S. Then, the preference function h is used to determine a ranking on input.

We expect that the methods succeed in making generalizations about features seen in a given corpus and achieve a good performance in order to make accurate predictions. In what follows we will present the score-based method we have chosen.

3 Ranking Model

Reranking is one of the issues that statistical parsing has to handle in order to improve their systems performance. We call it reranking because parsing already provides a ranking for trees with the probabilities computed during the process. In our work, we will refer to this solution as *ranking* because we are applying this method to the outputs of a grammar-driven parse which are not naturally ranked.

To do so, we will use a model that will allow us to discriminate a good analysis from a worse one. This model differs from the generative one involved in the parsing task. We will focus on finding the features that promote or penalize a dependency tree. In this context we will consider on one hand corpora where the gold standard is unique for every sentence. On the other hand, dependency structures are dependency trees [1].

3.1 Problem Definition

In what follows we will describe the notation and definitions we will use in the rest of the document:

Let $\mathcal{X}$ be the set of natural language sentences and $\mathcal{Y}$ the set of dependency structures.

Parser: We call $parse$ a function which takes a sentence as input and returns a set of dependency structures. For a given $x \in \mathcal{X}$, $parse(x) \subset \mathcal{Y}$. Note that $|parse(x)|$ varies according to x. So that, some sentences have zero, one or several parses. We are interested in the case where $parse(x) \neq \emptyset$

Annotated Corpus: An Annotated Corpus $\mathcal{C}$ is defined as a set of couples $(x, y^\star) \subset (\mathcal{X} \times \mathcal{Y})$, where x is a sentence and $y^\star$ the dependency structure considered as the correct parse for the sentence x. $y^\star$ is called the gold standard. $|\mathcal{C}|$ is the number of sentences in the corpus C.

The Labeled Attachment Score (LAS) of a dependency structure y according to the gold standard $y^\star$ measures the quality of a description tree with respect to the gold standard. It is denoted by:

$$LAS_{y^\star}(y) = \frac{\text{number of relations in } y \text{ equals to } y^\star}{\text{number of relations in } y^\star}$$

Feature: A $feature$ is a function that maps a dependency structure into numerical values. f is thus an $\mathbb{R}$-valued function of the set of dependency structures.

$f : \mathcal{Y} \to \mathbb{R}$

A $feature\ vector$ $\vec{f}$ is a m-dimensional vector of numerical features. $\vec{f} = (f_1, ..., f_m)$. $\vec{f}(y) = (f_1(y), ..., f_m(y)) \in \mathbb{R}^m$, $f_j(y)$ represents the value of the j'th feature on y.

[1] dependency structures are such that y and $y_\star$ have the same number of relations

Weight Vector: Given a feature vector $\vec{f}$, a weight vector $\vec{\theta}$ is a m-dimensional vector of $\mathbb{R}^m$, $\vec{\theta} = (\theta_1, ...\theta_m)$. Each θ_j represents the weight of a f_j .

Score: The score of a parse tree y is the dot product between the vector $\vec{\theta}$ and the feature function $f(y)$ defined as:

$$score_\theta(y) = \sum_{j=1}^{m} \theta_j \cdot f_j(y)$$

Knowing that the parser produces a set of candidates we will use a discriminative model to attribute a score to each of them, based on specific features.

3.2 Features For Ranking

The features describe a parse in matters of syntactic phenomenon. The performance of a ranker depends on the relevancy of the feature patterns extracted from the inputs. In our case, the inputs are different parses of a sentence.

When selecting a feature, a trade-off must be made between a feature as general as possible, so that it can handle a great number of parses and as specific as possible, in order to be able to distinguish a good parse from a bad one.

A parse is mapped to a feature vector:

$$f(y) = (f_1(y), ..., f_m(y))$$

Every f_j is associated with a configuration or schema, and the value of $f_j(y)$ corresponds to the number of times this configuration appears in y.

For instance $f_{(regarde,V,G,R,suj)}$ counts the number of times that the verb (V) *regarde* governs (G) the relation *suj* with an arrow going to the right (R).

We present below some feature schemata as it was proposed in (Le Roux et al., 2011).

Unigram: The instance of this class describes one dependency relation in a sentence. A dependency relation l, between two positions i and j governed by w_i is denoted by $w_i \xrightarrow{l} w_j$.

We create two sextuplets, one for the governor w_i, the other for the dependent w_j with this schema [word, lemma, part of speech (pos), status of dependency (governor or head), direction of the dependency (left (L) or right (R)), type of dependency]. Here is an example for the sentence in 3.1:

(3.1) *Jean* *regarde* *la* *fille.*
JOHN LOOK AT THE GIRL.
John is looking at the girl.

For the dependency *obj* between *regarde* and *fille* figure 3.1 we have these patterns :

`[regarde, regarder, V, G, R, obj]` and `[fille, fille, N, D, R, obj]`

(3.2) *Jean regarde la mer.*
JOHN LOOK AT THE SEA.
John is looking at the sea.

In order to have features that can be as general as possible we can "underspecify" the previous patterns. To get an idea we consider sentences 3.1 and 3.2.

The dependency *obj* between *regarde* and *fille* will give this configuration for one underspecification:

`[_, regarder, V, G, R, obj]` and `[_, fille, N, D, R, obj]`.

With two underspecifications we obtain:

`[_, _, V, G, R, obj]` and `[_, _, N, D, R, obj]`.

Besides, the dependency *obj* between *regarde* and *mer* will give with one underspecification:

`[_, regarder, V, G, R, obj]` and `[_, mer, N, D, R, obj]`.

With two underspecifications we obtain the same patterns as for sentence 3.1:

`[_, _, V, G, R, obj]` and `[_, _, N, D, R, obj]`.

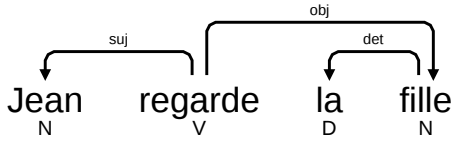


FIGURE 3.1: Example of dependency tree

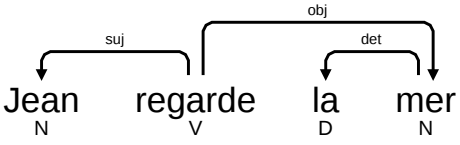


FIGURE 3.2: Example of dependency tree

The underspecification technique can be applied to all the types of schemata. We will avoid mentioning it for the rest of the schemata.

Bigram: The instances of this schema indicate the two co-occuring members of a dependency relation l.

Given the dependency relation $w_i \xrightarrow{l} w_j$, we create one tuple with this configuration:

[word w_i, lemma w_i, part of speech w_i, word w_j, lemma w_j, part of speech w_j , distance between position i and j, direction of the dependency, type of dependency].

(3.3) For the dependency *obj* between *regarde* and *fille*, we have these patterns: `[regarde, regarder, V, fille, fille, N, 2, R, obj]`

Syntactic Context: For the instances of this schema, we consider two dependencies $w_i \xrightarrow{l} w_j \xrightarrow{m} w_k$. One tuple is created with the configuration: [word w_i, lemma w_i, part of speech w_i, word w_j, lemma w_j, part of speech w_j, word w_k, lemma w_k, part of speech w_k, distance between position i and j, distance between position i and k, direction of the dependency l, type of dependency l, direction of the dependency m, type of dependency m].

(3.4) We consider the dependency *obj* between *regarde* and *fille*, and the dependency *det* between *fille* and *la*. We obtain :

`[regarde, regarder, V, fille, fille, N, la, le , D, 2, 1, R, obj, L, det]`

3.3 A Discriminative model

We will use a score-based model for the ranking. A score is attributed to each parse. We remind that the score is the dot product between the vector θ and the feature function $f(y)$:

$$score_\theta(y) = \sum_{j=1}^{m} \theta_j \cdot f_j(y)$$

The subset of k best-ranked dependency structure is denoted by $\hat{\mathcal{Y}}_k$ as below:

$$\hat{\mathcal{Y}}_k = \underset{y \in parse(x)}{\text{argmax}} \; score(y)$$

The vector $\vec{\theta}$ is estimated from the training data. In order to achieve this, machine learning multi-class classification algorithms (Collins and Koo, 2005) or (Le Roux et al., 2011) are used for ranking a parser output. The section below will explain how we estimate the parameters of our model.

3.3.1 MIRA: Theory

(Crammer and Singer, 2003) presented a learning algorithm for online large margin classification called MIRA (Margin Infused Relaxed Algorithm). In its binary form MIRA is closed to the perceptron algorithm (Rosenblatt, 1958). We will use its adaptation to the multi-class classification task.

MIRA is an online algorithm. This means that it operates incrementally and adjusts the models parameter progressively. For each sentence it makes a prediction. This prediction is compared to the correct parse (the gold standard) and an update is made if the ranker is wrong. This problem could then be expressed as follows:

$$\begin{aligned} & \min \|\theta\| \\ \text{subject to: } & score(y_t^\star) - score(y_t) \geq L(y_t^\star, y_t) \\ & \forall y_t \in parse(x_t) \end{aligned} \tag{3.5}$$

$L(y_t, y_t^\star)$ is the loss of a parse y_t relative to the correct tree $y_t^\star$. It is the distance between y_t and the correct one $y_t^\star$. The higher this value, the closer is the parse tree to the reference. In our specific case, we use the Labeled Attachment Score of the parse tree to define the measure L.

$$L(y_t, y_t^\star) = 1 - LAS_{y^\star}(y_t)$$

The largest loss a dependency tree can have is 1 and the minimal is 0. The minimal value expresses that the parse is identical to the gold standard, the maximal value the opposite.

We refer to $score(y_t^\star) - score(y_t)$ as the margin attained on round t. The goal of the algorithm is to achieve a margin at least as large as the loss of the incorrect tree. The more errors a tree has, the farther away its score will be from the score of the correct tree.

The task of MIRA is to incrementally learn the weight vector $\vec{\theta}$, according to the constraints specified above. On each update, it keeps the smallest norm possible of the changed θ subject to margin specification. This is formalized this way:

$$\begin{aligned} & \min \frac{1}{2}\|\theta^{(t+1)} - \theta^{(t)}\| \\ \text{subject to } & score(y_t^\star) - score(y_t) \geq L(y_t^\star, y_t) \\ & \forall y_t \in parse(x_t) \end{aligned} \tag{3.6}$$

Assuming that the parsing may fail to propose a parse that is identical to the gold standard, we define the oracle o_t as the parse y_t which is the closest to the reference. In other words the oracle

is the parse with the minimal loss L. We redefine equation 3.6 as follows:

$$\begin{aligned} \min \frac{1}{2}\|\theta^{(t+1)} - \theta^{(t)}\| \\ \text{subject to } score(o_t) - score(y_t) \geq L(y_t^\star, y_t) \\ \forall y_t \in parse(x_t) \end{aligned} \tag{3.7}$$

Instead of trying to satisfy the margin constraint for all the $parse(x_t)$ we make the assumption that only the margin which matters is the one that involves the incorrect tree with the highest score. We do so because it may be difficult or impossible to satisfy all the margin constraints knowing that for some sentences the number of parses is quite large.

Then equation (3.7) becomes equation (3.8). The benefit of this last transformation is that the simplification leads to a closed form solution. We finally have the optimization problem expressed as:

$$\begin{aligned} \min \frac{1}{2}\|\theta^{(t+1)} - \theta^{(t)}\| \\ \text{subject to } score(o_t) - score(y_{(t)}) \geq L(y_t^\star, y_t) \\ \forall y_t \in \mathcal{Y}_k \end{aligned} \tag{3.8}$$

The algorithm update the weight $\vec{\theta}$ based on the optimization formula above. Going through the dataset once for updating $\vec{\theta}$ is called a turn or iteration. It is advised to perform several turns in order to improve the accuracy of the system.

Resolution

The update makes sure that the prediction of the current instance does not change too radically the prediction made for the previous instances. We consider the case where $|\mathcal{Y}_k| = 1$. The equation 3.8 is resolved in (Crammer and Singer, 2003) by constructing the Lagrangian of equation 3.8 and taking partial derivative. This yield to equation 3.9, using the Hildreth algorithm (Censor and Zenios, 1997) that how to determine α_t when there is one constraint in the option problem. The solution of the optimization is then:

$$\begin{aligned} \theta^{(t+1)} = \theta^{(t)} + \alpha_t(f(o_t) - f(\hat{y_t})) \\ \text{with } \alpha_t = \frac{\Lambda}{\|f(o_t) - f(\hat{y_t})\|^2} \\ \Lambda = \max(0, L(y_t^\star, y_t) - (score(o_t) - score(\hat{y_t}))) \end{aligned} \tag{3.9}$$

The algorithm is described below:

Algorithm 3.1 MIRA Algorithm

Require: θ
  for s=1,2,...,S **do**
    for each sentence **do**
      Provide the parses
      Extract the feature $f(y)$ from parses
      Compute the $score_\theta$
      Predict $\hat{y}$ the best ranked parse using $score_\theta$
      Let the oracle o be the parse with $\min L$
      Define $\Lambda \leftarrow \max(0, L - (score(o) - score(\hat{y}))$
      if $o \neq \hat{y}$ **then**
        Define $\alpha \leftarrow \dfrac{\Lambda}{\|f(o) - f(\hat{y})\|^2}$
        Update $\theta^\star \leftarrow \theta + \alpha(f(o) - f(\hat{y}))$
      end if
    end for
  end for

4 Experiments

4.1 Data

The Sequoia Treebank (Candito et al., 2012) is a freely available annotated French corpus. It is developed in the Alpage team (Université Paris 7) by Marie Candito and Djamé Seddah. The corpus is freely available under the free license LGPL-LR[1]. The Sequoia Treebank contains 3,240 sentences, from the French Europarl, the regional newspaper L'Est Républicain, the French Wikipedia and documents from the European Medicines Agency.

Each sentence is annotated for part-of-speech and phrase-structure following the French Treebank guidelines (Abeillé and Barrier, 2004). The constituency trees were then automatically converted to dependency trees.

The corpus is provided in various forms:

- $*+fct.mrg\_strict$ files are files with one sentence per line, with phrase-structure in bracketed format, in which the phrase-structures are strictly compliant to the French Treebank annotation scheme.
- $*.xml$ files are the XML FrenchTreebank version for the $*+fct.mrg\_strict$ files
- $*+fct.mrg$ files are the same sentences, still in bracketed format, but with phrase-structures compliant to the ftb-uc Treebank instantiation.
- $*conll$ files contain the dependency trees, obtained through automatic conversion, in CoNLL 2006 tabulated format.

For our experiments we use the *.conll* files of the version 5.1 of Sequoia TreeBank. The CoNLL format is a data format for representing dependency tree, which is illustrated by figure 4.1. It was introduced during the CoNLL (Conference on Computational Natural Language Learning), in order to set up a common way to represent dependency structures.

4.2 Preprocessing

Among the 3,204 sentences of the Sequoia Treebank, we choose 717 sentences that correspond to well formed and short sentences which length is greater than 2 and lower than 14. From the 717 sentences, we retain only sentences for which LEOPAR proposed more than one parse, meaning for 436 sentences. The different parses for a sentence are stored in a *.conll* file like in figure 4.2. The weight value is the one computed by LEOPAR during the parsing process.

[1] Lesser General Public License For Linguistic Resources

1	Ils	cln	CL	CLS	s=suj	4	suj	4	suj			
2	ont	avoir	V	V	m=ind|n=p|p=3|t=pst	4	aux_tps	4	aux_tps			
3	été	être	V	VPP	m=part|t=past	4	aux_pass	4	aux_pass			
4	interpellés		interpeller		V	VPP	g=m|m=part|n=p|t=past	0	root	0	root	
5	sur	sur	P	P	_	4	mod	4	mod			
6	le	le	D	DET	g=m|n=s|s=def	7	det	7	det			
7	champ	champagne	N	NC	g=m|n=s|s=c	5	obj	5	obj			
8	.	.	PONCT	PONCT	s=s	4	ponct	4	ponct			

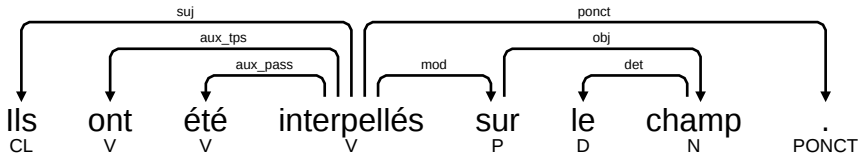


FIGURE 4.1: Example of a *.conll* file, with its associated dependency tree

4.3 Experimental Setup

We remind that our goal is to build a specific type of classifier: a ranker, on the output of the parsing of LEOPAR parser. We are using the MIRA algorithm and we want to determine the parameter θ of the system.

$$score_\theta(y) = \sum_{j=1}^{m} \theta_j \cdot f_j(y)$$

4.3.1 Features

We used the unigram feature schema and a modification of this features to create another class of features by including the distance of the governor-dependent relations. The different configurations are presented below:

- A set of features of the unigram category: `[word, lemma, pos, dependency status, dependency direction, dependency type]`.
- A set of features of the unigram category with the underspecification of the word: `[_, lemma, pos, dependency status, dependency direction, dependency type]`.
- A set of features of the unigram category with underspecification of the word and the lemma: `[_, _, pos, dependency status, dependency direction, dependency type]`.

```
% weight=823
1 ils       il          CL      _   pro_type=clit                                  4   suj       _   _
2 ont       avoir       V       _   mood=ind|num=pl|pers=3|tense=pres             4   aux_tps   _
3 été       être        V       _   gen=m|mood=pastp|num=sg                       4   aux_pass  _
4 interpellés           interpeller V   _   gen=m|mood=pastp|num=pl                0   root
5 sur       sur         P       _           4   p_obj     _   _
6 le        le          D       _   gen=m|num=sg                                 7   det       _   _
7 champ     champ       N       _   gen=m|noun_type=count#anim#abstr#mass|num=sg  5
8 .         .           PONCT   _           4   ponct     _   _

% weight=820
1 ils       il          CL      _   pro_type=clit                                  3   suj       _   _
2 ont       avoir       V       _   mood=ind|num=pl|pers=3|tense=pres             3   aux_tps   _
3 été       être        V       _   gen=m|mood=pastp|num=sg                       0   root      _   _
4 interpellés           interpeller V   _   gen=m|mood=pastp|num=pl                3   ats
5 sur       sur         P       _           4   p_obj     _   _
6 le        le          D       _   gen=m|num=sg                                 7   det       _   _
7 champ     champ       N       _   gen=m|noun_type=count#anim#abstr#mass|num=sg  5
8 .         .           PONCT   _           3   ponct     _   _
```

FIGURE 4.2: Example of the *.conll* file representing different parses provided by LEOPAR

.

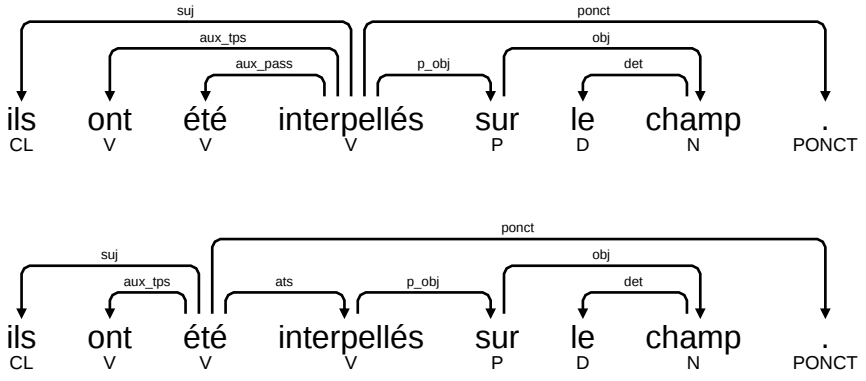


FIGURE 4.3: Trees associated with the file in figure 4.2

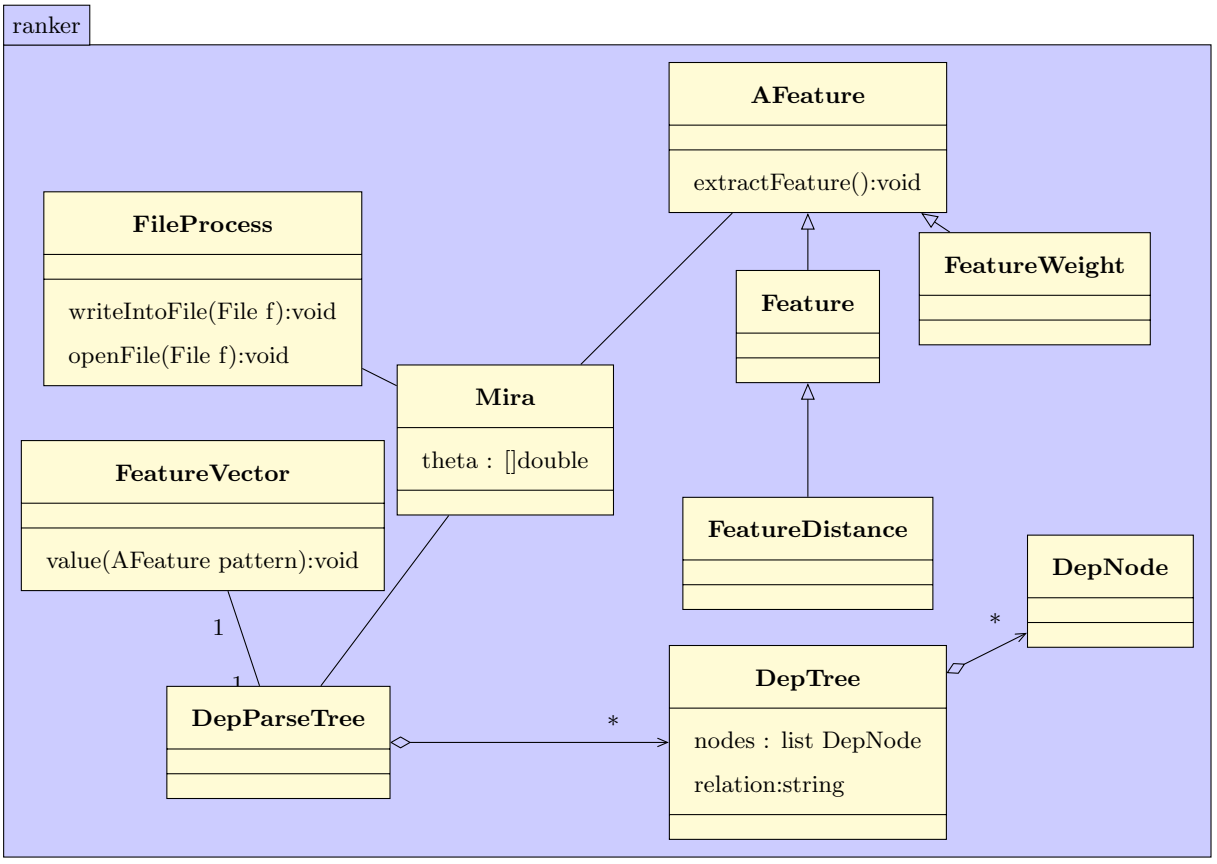
.

- A combination of the these first three features.
- A set of features of the unigram category with dependency length that we will denoted by *Unigram DepLength*: `[_, lemma, pos, dependency status, dependency direction, dependency type, dependency length]`.

- A set of features of the Unigram DepLength with the underspecification of the word: `[_, lemma, pos, dependency status, dependency direction, dependency type, dependency length]]`.
- A set of features of the unigram DepLength with the underspecification of the word and the lemma: `[_, lemma, pos, dependency status, dependency direction, dependency type, dependency length]]`.
- A set of features of the three features above.

4.3.2 Implementation

We have implemented the MIRA algorithm in JAVA. A overview of the UML class diagram of our architecture is presented below:



DepNode-DepTree-DepParseTree:

We represent our corpus by the *DepParseTree* structure. A *DepParseTree* stands for different sentences associated with their parses. A parse is a *DepTree* that is defined as a collection of nodes *DepNode* together with the list of parents and children.

AFeature-Feature-FeatureDistance-FeatureWeight:

The *AFeature* class models the notion of features that *Feature*, *FeatureDistance* and *FeatureWeight* extend. *Feature* represents the schema that we have called Unigram. It contains

methods that give the underspecification of Unigram configuration. *FeatureDistance* represents the Unigram DepLength and inherits the *Feature* Class. The weight computed by LEOPAR is also used as a feature that we model with the class *FeatureWeight*.

FeatureVector:

A *FeatureVector* is the object representation of a vector of features.

FileProcess:

This class contains several tools for the file computation task.

Mira:

MIRA is the implementation of the MIRA Algorithm.

4.3.3 Experiment parameters

The experiment was configured to use one oracle and a number of hypotheses that can be chosen for each instantiation of the experiment. We remind that, for one sentence we can have a large number of parses.

Sentences number	271	130	38	42	14	28	13	17	11	15	2	8	5	15	6	3	2	3	2
Parses number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Sentences number	5	7	1	2	6	4	6	1	10	2	7	1	2	2	3	2	2	1	1	1
Parses number	20	21	22	23	24	25	26	27	28	30	32	33	34	35	36	40	42	43	44	48

Sentences number	1	2	2	1	1	1	1	1	1	1	3	1	2	1	1	1	1	3
Parses number	51	54	56	58	60	62	63	68	73	98	102	103	105	159	225	247	396	406

TABLE 4.1: Number of parses according to sentences

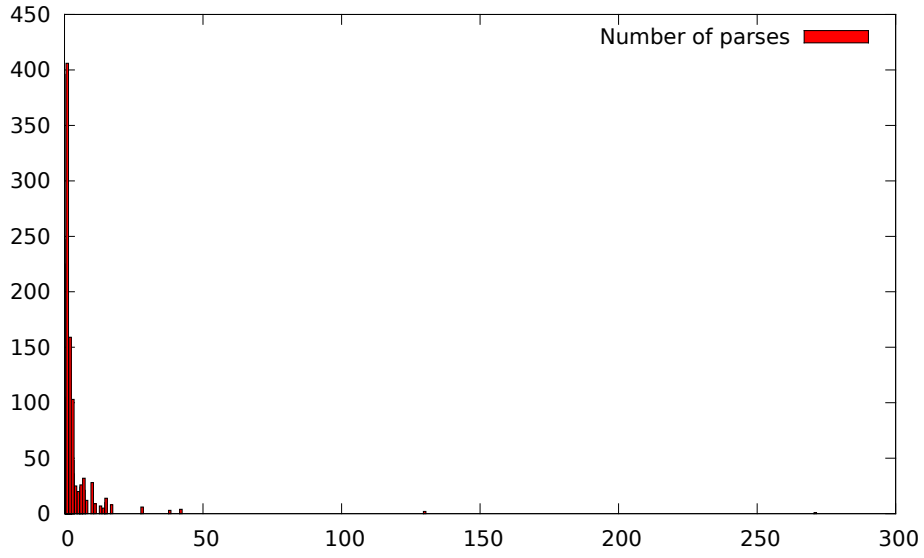


FIGURE 4.4: Histogram of number of parses evolution

On one hand, we conducted an experiment on all the parses. On the second hand, we reduced the number of parses to the 5-best list of sentences according to the scoring metric of LEOPAR. Some simple sentences received fewer than 5 parses. We partitioned and validated our dataset using 10-fold cross-validation rules. We used a null start weight $\vec{\theta} = \vec{0}$ with an average of 4,000 features. We call *turn* the fact that we iterate over the whole dataset one time, choosing randomly the order of the data examples. For our experiment we made 50 turns. At each turn, the weight is computed recursively as it has been said earlier, and we keep the weight of the last data example of the current turn as the starting weight for the next turn.

4.3.4 Results

The Labeled Attachment Score is an evaluation metric that computes the proportion of "scoring" tokens that are assigned both to the correct head and the correct dependency relation label. The LAS of the corpus is the mean of the LAS of the best ranked parse for each sentence weighted by the length of the sentence. This is the metric we will use for evaluating our work.

In order to compare the prediction performance of our system we consider two reference values:

- Oracle: The best prediction that the ranker can process, the one closer to the gold standard.
- AntiOracle: The worst prediction that the ranker can made, the further to the gold standard.

Oracle and Antioracle are the references for the comparison of the performance of the systems below:

- LEOPAR Prediction: The prediction made by LEOPAR according to its handcrafted rules.
- Ranker: The prediction of our ranker.

The graphics of figure 4.5 and figure 4.6 show the evolution of the LAS through 50 turns. We notice that the performance of the ranker depends on the used feature schemata. With the *Unigram DepLength* schema with the 5-best list, we obtain the best results, whereas with the *Unigram DepLength* with two underspecifications we obtain not really good results. This suggests that the second feature schema, as shown in figure 4.6, is too general to discriminate good parses from the worse ones.

Our ranker is able to select better parses than the evaluator of LEOPAR depending on the chosen feature schemata. The values of the LAS for the 50'th θ for two schemata is presented in table 4.2.

In order to evaluate the convergence of θ, we computed $\|\theta_{i+1} - \theta_i\|^2$. As we can see in figure 4.7, no update of θ has been performed from the 33^{th} turns until the end. On the other side, in figure 4.8, θ is still changing. This suggests that the stopping criterion is not suitable enough.

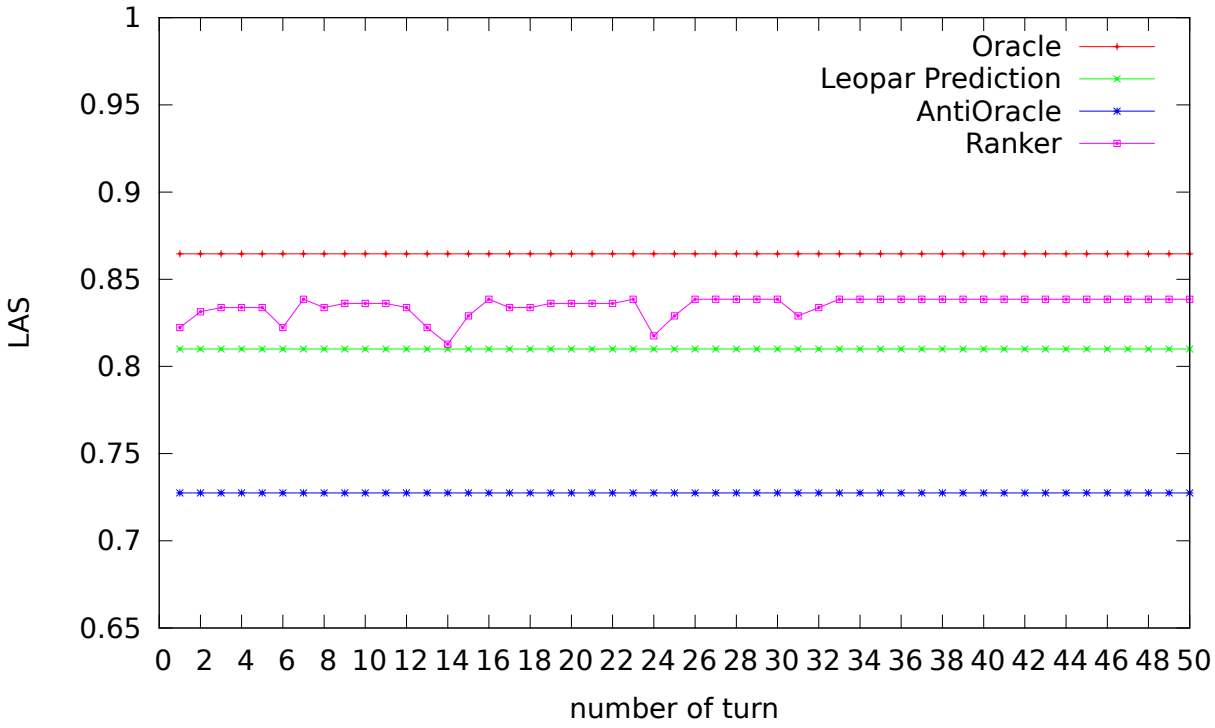


FIGURE 4.5: LAS: Unigram DepLength 5-best

System	Unigram DepLength 5-best	Unigram DepLength 5-best 2 underpsecfication
Oracle	0.864	0.864
LEOPAR	0.809	0.809
Ranker	0.838	0.798

TABLE 4.2: LAS for the different Features used

(Hasler et al., 2011) proposed two stopping criteria: *"stop running the algorithm when no update has been formed during a full epoch"* or *"stop when during three consecutive epochs the sums of all updates in each dimension has not changed by more than a predefined value"*.

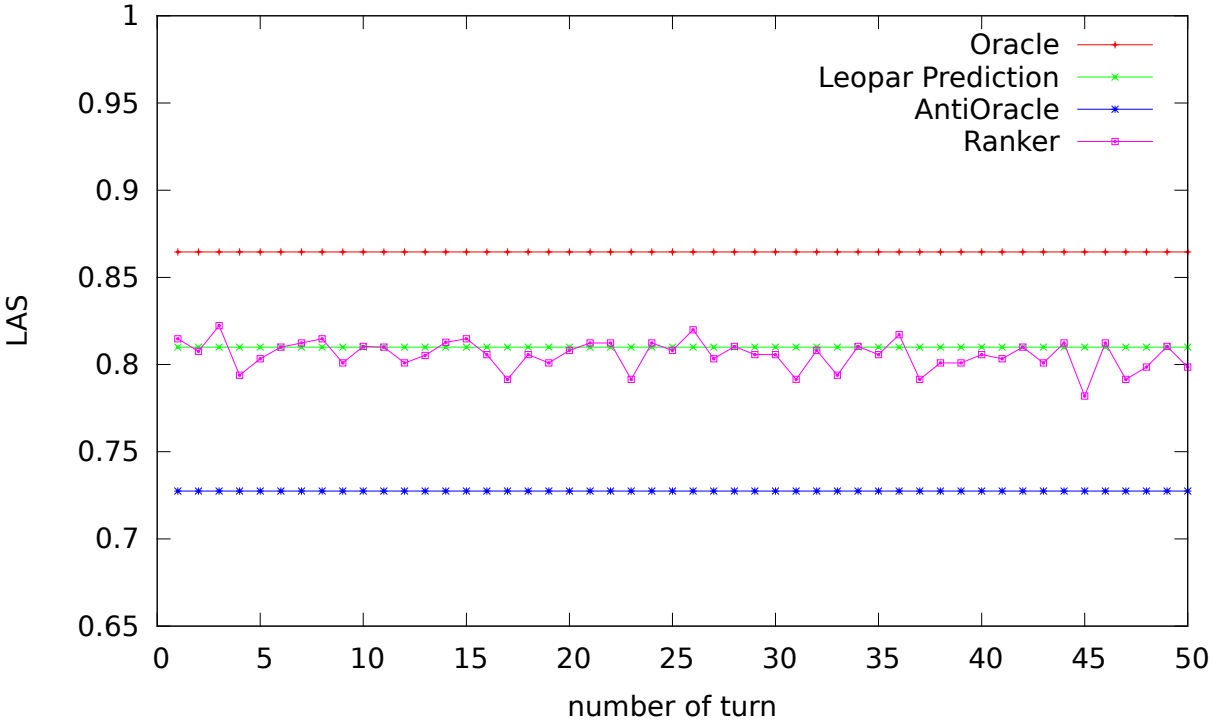


FIGURE 4.6: LAS: Unigram DepLength with two underspecifications

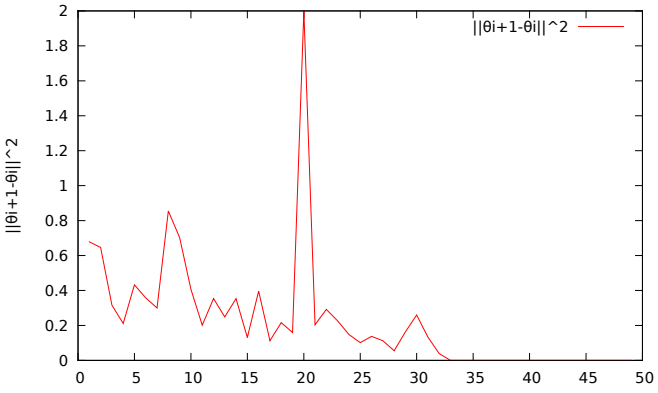


FIGURE 4.7: Norm: Unigram DepLength 5-best

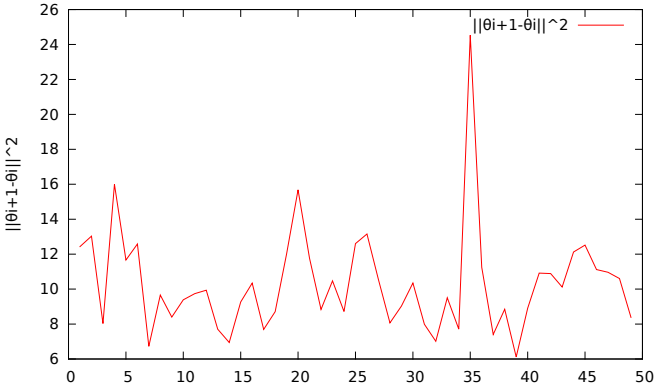


FIGURE 4.8: Norm: Unigram DepLength with two underspecifications

5 Conclusions

In this thesis, we focused on machine learning techniques to improve parsers, downstream the parsing process. Indeed, our goal was to incorporate statistical methods on a grammar-driven parser, LEOPAR, in order to rank the different parses that it provides. LEOPAR parser integrated already a scoring method based on handcrafted rules. We investigated multi-class classification methods and chose to apply the online margin algorithm called MIRA to solve our problem. We built a classifier used for the specific task of ranking. We evaluated our ranker on the Sequoia TreeBank.

We noticed that the performance of our ranker is largely related to different parameters like the size of our dataset, the used feature schemata, the stopping criteria... Even though the data set used was small, we achieve slightly better results compared to LEOPAR scoring method. Further works could be to investigate:

- Increasing the size and improving the quality of the used treebanks. Statistical method inferences depend on the treebank used to build their models. The quality of treebanks is crucial is the efficiency of such models.

- Using other feature schemata. As we have seen, our ranker performance depends as well, on the used feature schemata quality. However, we have to notice that there is no automatic way to know if the feature schema has a significant effect on the performance of the system. As has said Charniak in (Charniak and Johnson, 2005), *"[...] developing feature schemata is much more of an art than a science"*.

- We have to remind that there are other statistical models for ranking in the literature such as the maximum entropy model (Charniak and Johnson, 2005), the Markov Randoms Fields, Boosting Approaches (Collins and Koo, 2005) or generative models like the one developed in (Sangati et al., 2009).

In our work, we made cooperate a statistical technique and a handcrafted-grammar parser in order to improve the performance of the parser for the parse selection task. Maybe, the key idea will be to use a hybrid parsing system that will use both two approaches directly during the parsing process. Hybrid parsing systems use probabilistic models during the parsing process, before tree combination, i.e. upstream the parsing process. Recent works, (XTAG Research Group, 2001), (Kanayama et al., 2000), (Toutanova et al., 2002), suggest that hybrid systems can contribute to the improvement of the parsing systems accuracy.

A Appendix

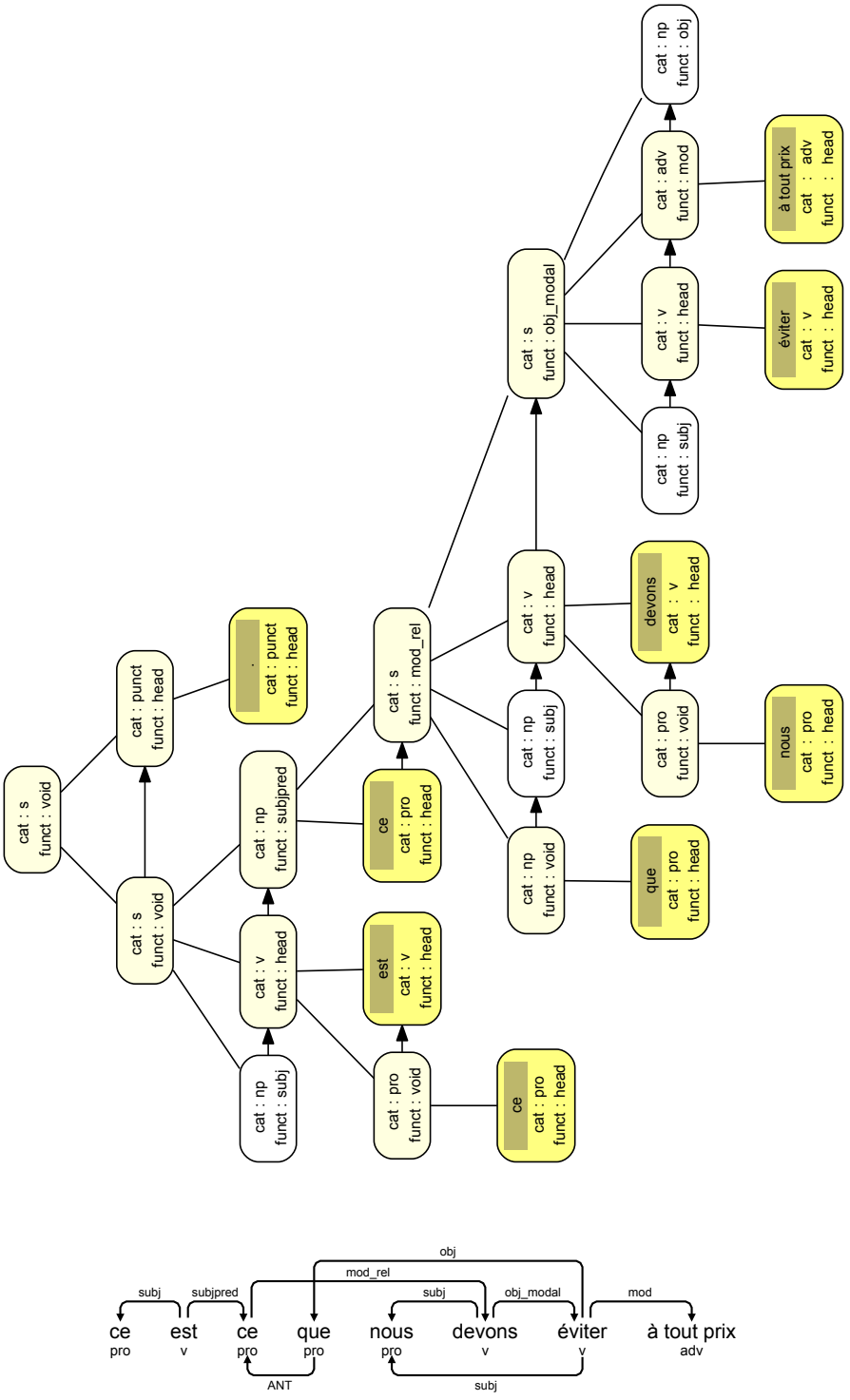


FIGURE A.1: Example of ambiguity in trees

List of Figures

Bibliography

Anne Abeillé and Nicolas Barrier. Enriching a french treebank. In *Proc. of LREC'04*. Citeseer, 2004.

Srinivas Bangalore and Aravind K Joshi. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265, 1999.

Cristina Barbero and Vincenzo Lombardo. Dependency graphs in natural language processing. In *Topics in Artificial Intelligence*, pages 115–126. Springer, 1995. URL http://link.springer.com/content/pdf/10.1007/3-540-60437-5_11.

Christopher M Bishop et al. *Pattern Recognition and Machine Learning*, volume 4. springer New York, 2006.

Marie Candito, Djamé Seddah, et al. Le corpus Sequoia: annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In *TALN 2012-19e conférence sur le Traitement Automatique des Langues Naturelles*, 2012. URL http://aclweb.org/anthology-new/F/F12/F12-2024.pdf.

Yair Censor and Stavros Andrea Zenios. *Parallel optimization: Theory, algorithms, and applications*. Oxford University Press on Demand, 1997.

Eugene Charniak. Statistical techniques for natural language parsing. *AI magazine*, 18(4):33, 1997. URL url={http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1320}.

Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics, 2000.

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on the ACL*. Association for Computational Linguistics, 2005. URL http://acl.ldc.upenn.edu/P/P05/P05-1022.pdf.

Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational linguistics*, 31(1):25–70, 2005.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003. URL http://dl.acm.org/citation.cfm?id=944936.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*,

7:551–585, 2006. URL http://eprints.pascal-network.org/archive/00002147/01/CrammerDeKeShSi06.pdf.

Denys Duchier, Joseph Le Roux, and Yannick Parmentier. The metagrammar compiler: An nlp application with a multi-paradigm architecture. In *Multiparadigm Programming in Mozart/Oz*, pages 175–187. Springer, 2005.

Bruno Guillaume and Guy Perrier. Interaction grammars. *Research on Language and Computation*, 7(2-4):171–208, 2009. doi: 10.1007/s11168-010-9066-x. URL http://hal.inria.fr/inria-00568888.

Bruno Guillaume, Joseph Le Roux, Jonathan Marchand, Guy Perrier, Karën Fort, and Jennifer Planul. A toolchain for grammarians. In *Coling 2008*, pages 9–12, Manchester, Royaume-Uni, 2008. URL http://hal.inria.fr/inria-00336333.

Eva Hasler, Barry Haddow, and Philipp Koehn. Margin infused relaxed algorithm for Moses. *The Prague Bulletin of Mathematical Linguistics*, 96(1):69–78, 2011. URL http://ufal.mff.cuni.cz/pbml/96/art-hasler-haddow-koehn.pdf.

Aravind K Joshi and Yves Schabes. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer, 1997.

Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, and Jun'ichi Tsujii. A hybrid japanese parser with hand-crafted grammar and statistics. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 411–417. Association for Computational Linguistics, 2000.

Ronald M Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. *Formal Issues in Lexical-Functional Grammar*, pages 29–130, 1982.

Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003. URL http://nlp.stanford.edu/~manning/papers/unlexicalized-parsing.pdf.

Joseph Le Roux, Benoît Favre, Seyed Abolghasem Mirroshandel, Alexis Nasr, et al. Modèles génératif et discriminant en analyse syntaxique: expériences sur le corpus arboré de Paris 7. *Actes de TALN 2011*, 1:371–383, 2011. URL http://hal.archives-ouvertes.fr/hal-00702424/.

Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. *Proceedings of ACL-08: HLT*, pages 950–958, 2008.

Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, volume 6, pages 2216–2219, 2006.

Carl Pollard and Ivan A Sag. *Head-driven phrase structure grammar*. University of Chicago Press, 1994.

Frank Rosenblatt. The perceptron. *Psych. Rev*, 65(6):386–408, 1958.

Federico Sangati, Federico Zuidema, and Rens Bod. A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conferenceee on Parsing Technologies*, pages 238–241. Association for Computational Linguistics, 2009. URL http://dl.acm.org/citation.cfm?id=1697285.

Daniel DK Sleator and Davy Temperley. Parsing english with a link grammar. *The Third International Workshop on Parsing Technologies*, pages –, 1993.

Lucien Tesnière and Jean Fourquet. *Eléments de syntaxe structurale*, volume 1965. Klincksieck Paris, 1959.

Kristina Toutanova, Christopher Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. Parse disambiguation for a rich hpsg grammar. In *First Workshop on Treebanks and Linguistic Theories (TLT2002), 253-263*. Stanford InfoLab, 2002.

XTAG Research Group. A lexicalized tree adjoining grammar for english. Technical report, Technical Report IRCS-01-03, IRCS, University of Pennsylvania, 2001.

Arnold M Zwicky. Heads. *Journal of linguistics*, 21(1):1–29, 1985.