Université de Montréal

Étude de transformations grammaticales pour l'entraînement de grammaires probabilistes hors-contexte

par
Ngoc Tran Nguyen

Département d'informatique et de recherche opérationnelle
Falculté des Arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de M.Sc.
en informatique

Décembre 2002

Université de Montréal
Falculté des études supérieures

Étude de transformations grammaticales pour l'entraînement de grammaires probabilistes hors-contexte

présenté par
Ngoc Tran Nguyen

a été évalué par un jury composé des personnes suivantes

Balázs Kégl
Président

Philippe Langlais
Directeur de recherche

Victor Ostromoukhov
Rapporteur

Mémoire accepté le 5 février 2003

Résumé

Un effort important a été produit ces dernières années pour proposer des modèles probabilistes capables d'analyser grammaticalement des textes. Ces modèles qui sont appris automatiquement à partir de corpus analysés grammaticalement par des linguistes évoluent et deviennent conceptuellement très compliqués. Dans ce mémoire, nous allons étudier des transformations simples qui sont applicables sur les corpus d'entraînement. Nous avons trouvé que certaines de ces transformations permettent à des modèles très simples d'obtenir des performances comparables à des modèles beaucoup plus compliqués proposés dans la littérature.

Mots-clé : traitement des langues naturelles, grammaire probabiliste hors-contexte, transformation grammaticale, Penn Treebank, Parseval.

Abstract

An important effort to propose probabilistic models that are capable of analyzing grammatically texts has been witnessed in recent years. During this period, these models were automatically trained from corpora analyzed grammatically by linguists, and thus became tremendously conceptual complicated. We expect to study in this report simple transformations applicable on the training materials. Some of these transformations will prove the ability to allow very simple models to obtain performances comparable to much more complicated one.

Keywords : natural language processing, probabilistic context free grammar, grammatical transformation, Penn Treebank, Parseval.

Table de matières

Liste des sigles et abréviations

CNF	Chomsky normal form
CFG	Context free grammar
CYK	Algorithme Cock-Younger-Kasami
PCFG	Probabilistic context free grammar
POS	Part-of-speech
PTB	Corpus Penn Treebank
WSJ	Wall Street Journal
P(X|Y)	Probabilité de X étant donné Y
PARSEVAL	Métriques pour évaluer l'analyseur syntaxique
STL	Standard Template Library
n/a	not available

Liste des tables

Table de figures

Remerciements

Je voudrais remercier le Professeur *Philippe Langlais*, mon directeur de recherche, qui m'a beaucoup aidé dans les études des grammaires probabilistes et notamment m'a aidé à résoudre les problèmes rencontrés. Sans son encouragement, ce mémoire ne serait pas ce qu'il est. Je tiens également à remercier tous les membres du laboratoire de Recherche Appliquée en Linguistique Informatique pour leur assistance aussi bien matérielle que morale et qui m'ont permis de faire ce mémoire dans de bonnes conditions.

Chapitre 1. Introduction

L'informatique est depuis plusieurs années entrée dans notre quotidien. Il est donc impératif de doter l'ordinateur de compétences linguistiques afin de le rendre apte à communiquer avec un humain, ce qui est déjà possible dans des situations très précises. Des majordomes téléphoniques permettent par exemple déjà, grâce à la reconnaissance vocale d'acheminer un appel automatiquement à son destinataire via un dialogue minimaliste. Des serveurs vocaux interactifs permettent également de faire des tâches spécifiques (par exemple la réservation de billets d'avions, l'interrogation de bases de spectacles, etc.) ne nécessitant que peu de connaissances linguistiques mais plutôt une bonne modélisation acoustique. Dans le domaine de l'écrit, certains systèmes permettent d'aider un utilisateur à trouver des éléments de réponse dans des bases de documents. Ce type de système est amené à se développer avec la taille grandissante des informations que l'on trouve sur le web ou détenues par des sociétés. L'information est devenue une ressource qu'on commercialise, et tout système facilitant son traitement est pertinent dans ce contexte. Pour améliorer la fonctionnalité des systèmes existants (de l'oral ou de l'écrit), il est impératif d'étendre leur capacité de compréhension, ce qui requiert le traitement adéquat de l'analyse syntaxique et l'analyse sémantique.

On s'intéresse dans cette étude à l'analyse syntaxique parce qu'elle facilite le traitement du sens d'une phrase. L'analyse syntaxique est le domaine de la linguistique qui s'occupe de l'étude des règles de bonne formation des phrases. Notons que le concept de phrase implique déjà un niveau d'abstraction assez élevé. Prenons les exemples suivants:

1. tables fauteuils murs planchers lits
2. Cette petite fille est assez rapide.

La plupart des gens auraient tendance à considérer le deuxième exemple comme une phrase, mais pas le premier. C'est que la notion de phrase implique un niveau minimal de structure. Les éléments sont reliés de façon régulière. Cette régularité se manifeste à deux niveaux: la forme et le sens.

Du point de vue formel, on constate que, contrairement à l'exemple *(1)* l'exemple *(2)* comprend un ordre (on dit *cette petite fille* mais pas *petite cette fille*), et une série de dépendances (le choix d'un nom féminin implique le choix d'un adjectif féminin, qui se termine par *-e*, et le choix d'un nom singulier implique le choix d'un verbe singulier). Du point de vue sémantique, on remarque que la phrase se caractérise par le fait de porter un contenu qui représente en quelque sorte la composition de ses composantes. Ainsi, *cette petite fille* permet d'indiquer l'existence d'une fille en particulier (*cette*), le fait que cette fille n'est pas vieille et le fait qu'il s'agit d'une fille. Ces deux niveaux de structure supposent une connaissance de la langue. Si on ne parle pas français, on ne peut pas savoir que *(2)* est une phrase mais que *(1)* n'en est pas une. Les suites de mots que nous entendons et que nous lisons tous les jours sont classées comme des phrases ou non selon ces connaissances.

On peut représenter l'ordre et les dépendances d'une phrase au moyen d'un arbre. Cet arbre comprend des branches reliant des nœuds. Chaque nœud sauf le premier est relié à un nœud supérieur (son père), et chaque nœud sauf les derniers est relié à des nœuds inférieurs (ses fils). On peut voir dans l'arbre le produit de l'application en série d'une liste de règles, chacune produisant une relation entre une mère et sa ou ses filles, comme l'illustre l'exemple suivant :

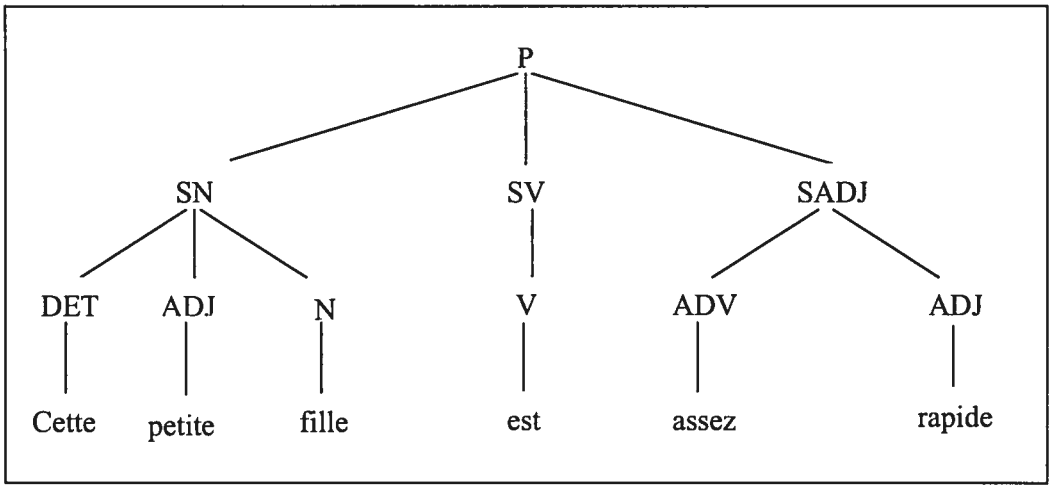


FIG. 1.1 Analyse syntaxique de la phrase : "cette petite fille est assez rapide"

On peut représenter un tel arbre à l'aide d'un ensemble de règles de réécriture dans lesquelles une seule unité se réécrit comme une suite d'unités. On exprime la réécriture au moyen d'une flèche. Ainsi, la règle qui réécrit le *P* (Phrase*)* comme *SN* (syntagme nominal) plus *SV* (syntagme verbal) et *SADJ* (syntagme adjectival) s'écrit:

$$P \rightarrow SN\ SV\ SADJ$$

On voit que cette règle de réécriture définit non seulement la présence des éléments mais aussi leur ordre. La grammaire d'une langue est constituée d'un ensemble de règles comme celle-ci (grammaires hors-contexte ou CFG dans la suite pour *Context Free Grammar*). L'analyse syntaxique joue un rôle important dans la détermination du sens d'une phrase. Une bonne analyse permet de bien représenter la sémantique, une facette importante du traitement des langues naturelles.

Pendant longtemps, la modélisation syntaxique d'une langue naturelle était essentiellement affaire de linguistes. Les modèles résultants sont généralement incomplets et ambigus, et leur maintenance est une tache ardue qui se heurte à deux problèmes suivants [Rajman, 1996] :

- la couverture linguistique: un modèle est, soit trop permissifs (existence des phrases acceptées par le modèle et pourtant linguistiquement invalides), soit trop restrictifs (existence de phrases rejetées par le modèle bien linguistiquement valides),
- l'ambiguïté d'analyse: la complexification des modèles syntaxiques liée à la prise en compte de phénomènes linguistiques de plus en plus fins se traduit, à couverture linguistique égale, par une augmentation, souvent dramatique, du nombre de structures syntaxiques potentielles associées aux phrases. Il devient alors de plus en plus difficile de sélectionner, parmi les structures produites, celles qui sont les bonnes.

Pour tenter de résoudre ces problèmes, on peut envisager une sophistication accrue des modèles utilisés, par exemple par un enrichissement des formalismes permettant une meilleure représentation de la complexité des langages étudiés. Cependant, la mise en œuvre des modèles plus complexes rend la tâche de modélisation encore plus ardue et augmente de ce fait, de façon considérable, les délais de mise au point. Par ailleurs, la complexité des modèles rend également difficile la mise en œuvre de techniques automatiques ou semi-automatiques d'aide à la conception ou de validation. La modélisation syntaxique fine est un but louable

mais ardu, aussi de nombreux chercheurs s'intéressent à la dérivation automatique d'un modèle syntaxique à partir de grammaires hors-contexte probabilisées (PCFG dans la suite pour *Probabilistic Context Free Grammar*). Cette approche permet d'associer à chaque arbre d'analyse syntaxique (comme celui de la figure *FIG 1.1)* un score qui représente l'exactitude de cet arbre. Ce score est calculé à partir du score associé à chaque règle de grammaire. Cette approche est devenue d'autant plus séduisante qu'il est maintenant possible d'apprendre les règles et les scores à partir d'un corpus annoté (un ensemble d'arbres comme celui de la *FIG 1.1)*. L'idée de cette approche est de partir de modèles syntaxiques simples mais trop permissifs et/ou trop ambigus, et d'utiliser les probabilités pour adapter progressivement le langage reconnu à la réalité observée.

- soit par réduction de la couverture du modèle (si, pour certaines séquences, la probabilité de leur production par le modèle devient nulle ou, éventuellement, inférieure à un seuil fixé),
- soit, et c'est probablement le cas le plus fréquent, par réduction de l'ambiguïté du modèle, par la mise en œuvre d'un mécanisme de décision permettant de filtrer, sur la base des probabilités conditionnelles correspondantes, les structures syntaxiques associées à une phrase donnée.

De tels modèles devraient être capable d'isoler un ensemble de règles minimaliste qui rend compte le mieux possible des phénomènes linguistiques rencontrés dans la langue. En pratique, les modèles proposés (voir le chapitre 4) font tous face au même problème qui est celui de la sous-représentation des données (beaucoup de règles non rencontrées ou vues une seule fois, trop de règles de nature trop spécifique, etc). Ces modèles diffèrent essentiellement dans la manière avec laquelle ils capturent les informations linguistiques et comment les estimées des paramètres sous-jacents sont lissés. Notre réponse au problème consiste à effectuer des opérations de transformation simples sur le corpus PTB avant entraînement de la grammaire. Nous étudions plusieurs transformations et leur impact sur la qualité des modèles produits. Nous décrivons en particulier une combinaison de ces transformations qui permet d'obtenir un modèle simple (donc lissé) avec des performances comparables à celles mesurées sur avec les meilleurs modèles probabilistes du moment. L'organisation de ce mémoire est comme suit:

- Chapitre 2 : Dans ce chapitre, nous présentons les notions de CFG, PCFG, ainsi que deux algorithmes génériques d'analyse syntaxique. Nous présentons

également le protocole d'évaluation que nous avons utilisé pour mesurer la qualité de nos modèles.

- Chapitre 3 : Nous présentons le corpus Penn Treebank que nous utilisons pour entraîner et tester nos modèles.
- Chapitre 4 : Nous proposons une brève revue des principaux modèles PCFG décrits dans la littérature et reportons les expériences que nous avons réalisées sur le Penn Treebank. À travers ces résultats, nous montrons que notre approche est simple et efficace comparée avec celle des autres chercheurs.
- Chapitre 5 : Nous dressons les conclusions de ce travail et discutons des pistes de recherche.

Chapitre 2. Modèle statistique PCFG

Depuis les travaux de Charniak [Charniak, 1996], les Grammaires Probabilistes Hors Contexte (Probabilistic Context Free Grammar ou PCFG) connaissent un regain d'intérêt. Nous décrivons formellement cette notion puis décrivons les problèmes qu'il faut résoudre pour les utiliser dans des applications à savoir : l'apprentissage des paramètres d'une PCFG et l'analyse d'une phrase à l'aide d'une PCFG. Nous décrivons en particulier deux algorithmes d'analyse génériques Cock-Younger-Kasami [Younger D, 1967] et CHART (*Chart parsing*) [Charniak, 1993]. Nous comparons les deux algorithmes et précisons les raisons pour lesquelles nous avons utilisé dans notre étude CYK. Enfin, nous présentons les mesures PARSEVAL pour évaluer la performance de différents analyseurs syntaxiques. Ce chapitre est organisé comme suit:

- Section 2.1 : nous présentons formellement les grammaires hors-contexte (Context Free Grammar ou CFG), et discutons du problème de l'ambiguïté avec les CFG.
- Section 2.2 : nous définissons les grammaires probabilistes hors-contexte et les problèmes associés.
- Section 2.3 : nous décrivons le problème de l'apprentissage de PCFG à partir d'un corpus annoté (décrit au chapitre 3); nous introduisons le lecture à l'estimation par maximum de vraisemblance et précisons comment nous associons dans cette étude une probabilité à une règle.
- Section 2.4 : nous discutons deux algorithmes génériques d'analyse syntaxique (CYK et CHART). Nous proposons une extension de l'algorithme CYK qui permet de gérer des grammaires qui ne sont pas exprimées en forme normale de Chomsky.

- Section 2.5 : nous décrivons les mesures PARSEVAL utilisées dans la littérature pour évaluer la performance des analyseurs syntaxiques.

2.1 Grammaire hors contexte (CFG)

Dans l'ensemble de ce chapitre, nous utilisons la notation de la *Table 2.1*. D'après la classification de Chomsky, il y a 4 types de grammaires selon la forme des règles de production:

- type 0: $\alpha \rightarrow \beta \quad \alpha, \beta \in (\mathrm{N} \cup \mathrm{T})^*$
- type 1: $\alpha \rightarrow \beta$ où $|\alpha| \leq |\beta| \, \alpha, \beta \in (\mathrm{N} \cup \mathrm{T})^*$ *Contexte-Sensitive*
- type2: $A \rightarrow \alpha$ où $A \in T$, $\alpha \in (\mathrm{N} \cup \mathrm{T})^*$ *Grammaire Hors Contexte*
- type3: $A \rightarrow aB$ ou $A \rightarrow a \; \mathrm{A} \in \mathrm{N}, \mathrm{a} \in \mathrm{T}$ *Langage régulière*

Nous nous intéressons ici aux grammaires de type 2 pour plusieurs raisons :

- elles sont bien maîtrisées et on connaît des algorithmes d'analyse relativement efficaces,
- elles permettent de rendre compte d'un grand nombre de phénomène linguistique (voir [Manning et Schutze, 1999] pour de plus amples discussions sur l'adéquation des grammaires hors-contexte pour modéliser la langue naturelle).

Une CFG se définie donc par un 4-tuplet <N,T,R,S> où

N est l'ensemble fini des symboles non terminaux
T est l'ensemble fini des symboles terminaux
R est l'ensemble fini de règles r_i de la forme: $A \rightarrow \alpha$ avec $\mathrm{A} \in \mathrm{N}$ et $\alpha \in (\mathrm{N} \cup \mathrm{T})^*$
S est l'axiome (le symbole non terminal initial)
$w_1^n = w_1 w_2 w_3 ..... w_n$ fait référence à l'entrée (la chaîne à analyser)

Table 2.1 Définition de grammaire hors-contexte

On considère ces grammaires comme les grammaires hors-contexte car l'application d'une règle se fait de manière inconditionnelle: le contexte du non terminal substitué n'est pas considéré. Il existe par opposition des grammaires de type *1* (*contexte-sensitive*) mais les propriétés de ces grammaires sont moins étudiées et les algorithmes d'analyse moins efficaces.

On dit que N domine la chaîne $w_i.....w_j$, ce que l'on note $N \Rightarrow^* w_i.....w_j$, si on peut dériver depuis N la chaîne w_i^j en un nombre de dérivation fini (une dérivation consiste à l'application d'une règle de la grammaire).

Une CFG non ambiguë est une CFG qui ne produit qu'un seul arbre d'analyse pour une phrase du langage. À l'inverse, une CFG est ambiguë s'il existe au moins une phrase du langage décrit possédant deux arbres différents par cette grammaire. Voici un exemple

Soit la grammaire G où S est l'axiome		
S → NP VP	P → with	NP → ears
PP → P NP	V → saw	NP → saw
VP → V NP	NP → NP PP	NP → stars
VP → VP PP	NP → astronomers	NP → telescopes

Table 2.2 Exemple de grammaire hors-contexte

Prenons cette grammaire et la phrase à analyser: w_i^5 = *astronomers saw stars with ears*. Cette phrase contient deux arbres d'analyse possibles (voir les figures 2.1 et 2.2) avec G dont les interprétations sont les suivantes:

t_1 = les *astronomers* ont vu des étoiles qui avaient des oreilles

t_2 = les *astronomers* ont vu des étoiles en utilisant des oreilles

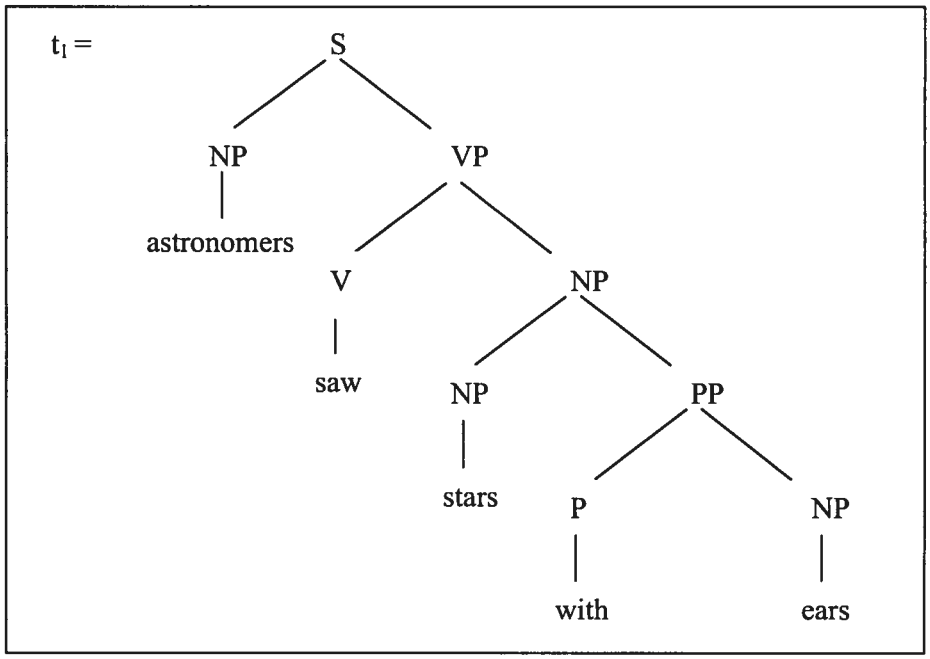


FIG. 2.1 Première possibilité d'arbre analysé par la grammaire G

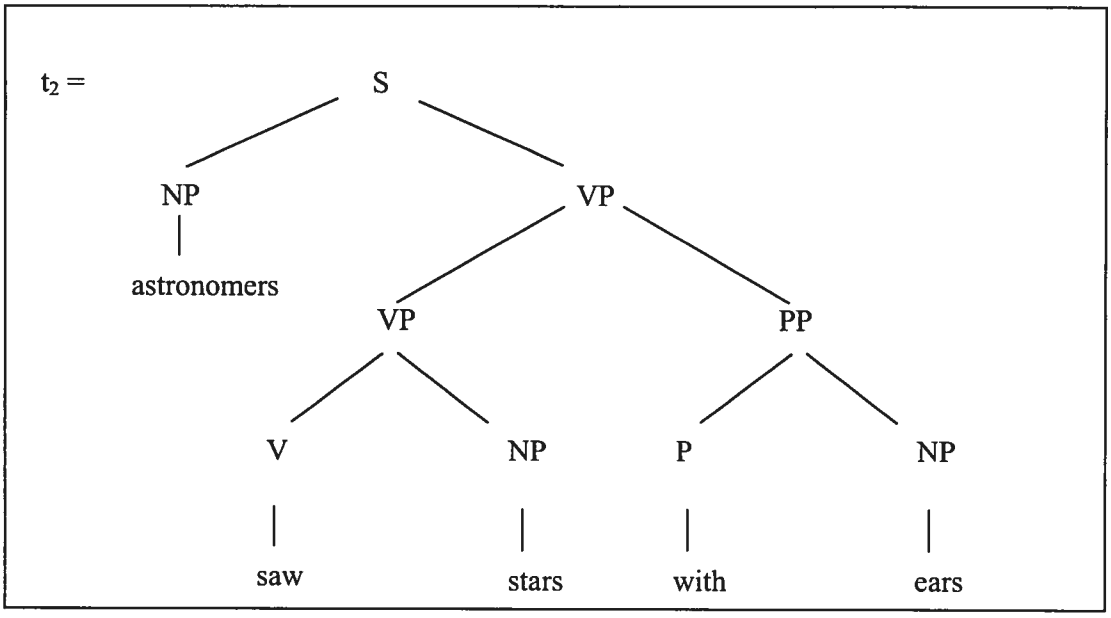


FIG. 2.2 Deuxième possibilité d'arbre analysé par la grammaire G

Dans l'approche traditionnelle, choisir la bonne analyse parmi l'ensemble des analyses proposées est une affaire d'expertise linguistique. Les PCFG offrent un mécanisme formel autorisant la sélection de la bonne analyse parmi l'ensemble des analyses possibles d'une phrase. Certains correcteurs orthographiques qui utilisent de telles grammaires se contentent souvent de trouver une analyse (parmi celles possibles) pour valider une phrase. Nous décrivons donc dans la section suivante ces grammaires probabilisées et illustrons ce mécanisme de sélection (et ses faiblesses).

2.2 Grammaires probabilistes hors-contexte (PCFG)

Une grammaire probabiliste hors-contexte est une extension naturelle d'une CFG et se définie par un 5-tuplet < N, T, R, S, P> où:

N	est l'ensemble fini des symboles non terminaux
T	est l'ensemble fini des symboles terminaux
R	est l'ensemble fini des règles r_i de la forme: $A \rightarrow \alpha \, A \in N, \alpha \in (N \cup T)^*$
S	est l'axiome de départ
P	est l'ensemble des probabilités p_i associées aux règles r_i telles que $\sum_{\alpha} p(A \rightarrow \alpha) = 1, \forall A \in N \, et \, \alpha \in (N \cup T)^*$

Table 2.3 Définition d'une grammaire probabiliste hors-contexte (PCFG)

La probabilité d'un arbre syntaxique est obtenue en multipliant la probabilité de chaque règle utilisée pour construire l'arbre. Reprenons notre exemple de grammaire (*table 2.2*), avec maintenant une probabilité associée à chaque règle (nous discutons plus tard des façons d'obtenir ces probabilités).

Soit la grammaire G où S est l'axiom		
1.S → NP VP [1.0]	5.NP → NP PP [0.4]	9.NP → ears [0.18]
2.PP → P NP [1.0]	6. V → saw [1.0]	10.NP → saw [0.04]
3.VP → V NP [0.7]	7.P → with [1.0]	11.NP → stars [0.18]
4.VP → VP PP [0.3]	8.NP → astronomers [0.1]	12.NP → telescopes [0.1]

Table 2.4 Exemple de grammaire probabiliste

Avec la grammaire *G,* nous avons toujours deux arbres d'analyse de la phrase '*astronomers saw stars with ears'* qui sont rappelés dans les figures 2.3 et 2.4 (les nombres cerclés indiquent la règle utilisée), ce qui permet de sélectionner l'arbre le plus probable (étant donnée G) t_1. La différence entre ces deux arbres est l'attachement prépositionnel *PP,* il est attaché au *NP* pour t_1 et au *VP* pour t_2.

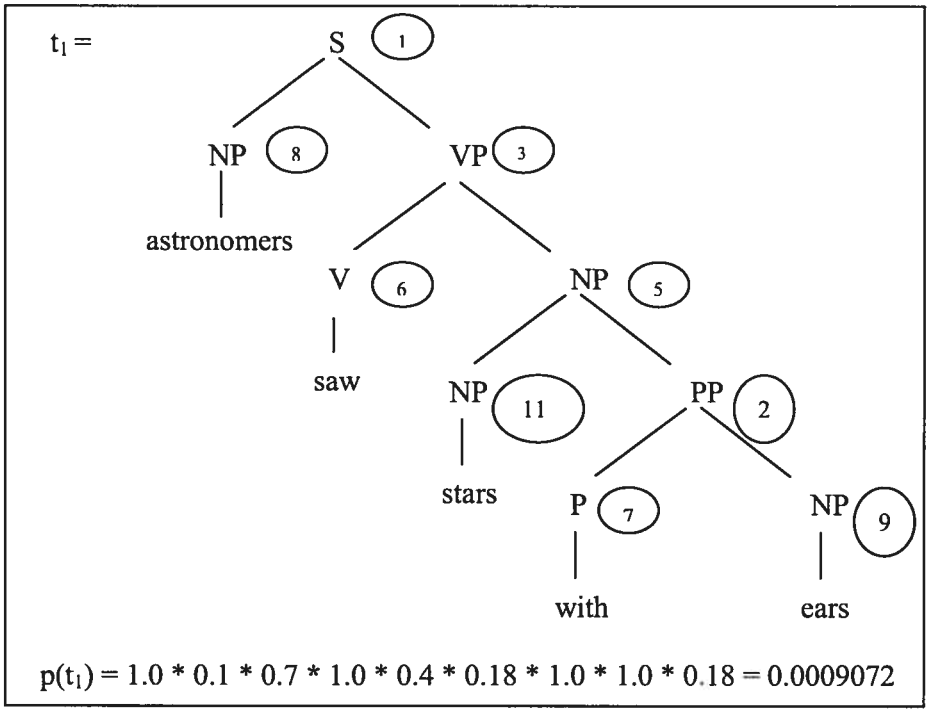


FIG. 2.3 L'un des deux arbres possibles pour la phrase exemple et sa probabilité associée

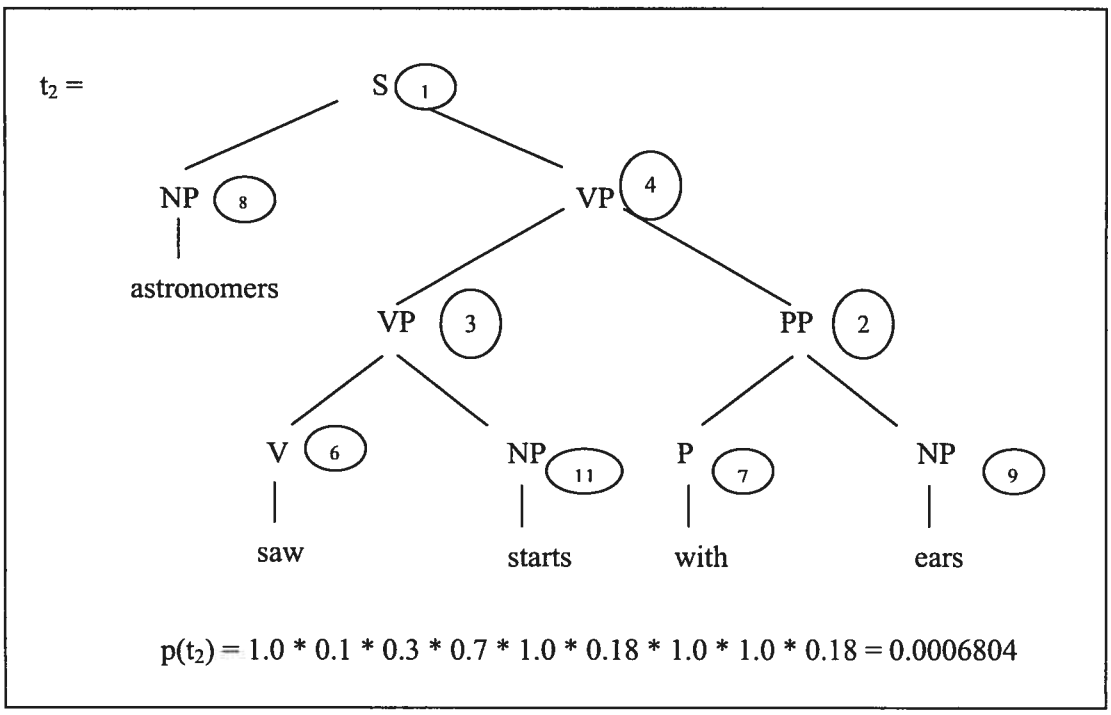


FIG. 2.4 L'autre arbre d'analyse possible
pour la phrase exemple et sa probabilité associée

Bien qu'une PCFG nous permet d'établir un ordre sur les arbres d'analyse syntaxique possibles, cet ordre n'est pas toujours pertinent. Considérons par exemple ces deux phrases mettant en relief une ambiguïté classique, l'ambiguïté de rattachement prépositionnel :

Alice bought a plant with Mary

Alice bought a plant with yellow leaves

Une PCFG entraînée sur un corpus de grand taille a toutes les chances de "préférer" l'attachement de la phrase prépositionnelle *PP* (*'with yellow leaves'*) plus souvent sur le group nominal *NP* (*'a plant'*) que sur le group verbal *VP* (*'bought'*) (par exemple dans notre expérience sur le corpus Penn Treebank, la règle $NP \rightarrow NP\ PP$ a une probabilité de *0.111* tandis que la règle $VP \rightarrow VP\ PP$ a une probabilité de 0.00002). Cette préférence favorise une analyse incorrecte de la première phrase. Charniak (1993) a cependant montré que l'usage d'une PCFG jumelé à des mécanismes de désambiguïsation spécifique permet de palier ce problème.

En sus du mécanisme de désambiguïsation (même imparfait) que nous venons de décrire, les PCFG possèdent sur les CFG un avantage lorsque l'on tente d'apprendre des règles. Comme le mentionné Charniak (1993), les CFG ne

peuvent être apprises qu'en présence d'exemples positifs et négatifs, alors que dans le cas de PCFG, seuls les exemples positifs sont nécessaires. Un corpus de phrases d'une langue donnée est donc en théorie suffisant pour l'apprentissage des PCFG.

Enfin, une PCFG, à contrario des CFG, peut être utilisée comme un modèle de langue qui consiste à estimer la probabilité d'une phrase dans une langue donnée. Un tel modèle de n-gramme prend contexte lexical en considération (n – 1 mots précédants) tandis qu'un PCFG ne l'utilise pas. Par contre, un modèle de langue PCFG peut considérer les dépendances à distance (les relations syntaxiques). En pratique, Charniak (1993) a montré qu'une PCFG donne un modèle de langue moins bon qu'un modèle n-gramme où n est supérieur à 1 (bigramme ou plus), il est cependant possible de combiner avec succès les deux modèles afin de capturer les relations locales entre les mots ainsi que de tenir compte des dépendances à distance [Benedi et al, 2000].

Dans le modèle PCFG, il existe deux grands problèmes:

- apprendre la grammaire, ce qui implique d'apprendre à la fois les règles et leurs probabilités,
- analyser une phrase étant donnée une grammaire afin d'obtenir l'arbre le plus probable.

Nous abordons ces deux problèmes de manière générale dans les sections suivantes.

2.3 L'apprentissage PCFG

Il y a deux approches reliées à l'apprentissage d'une PCFG : l'apprentissage des règles elles-mêmes et l'association d'une probabilité à chaque règle. Le choix de l'approche dépend entre autres des corpus disponibles (voir à ce sujet le chapitre 3). Si on a un corpus étiqueté (chaque mot de la phrase est étiqueté par une étiquette syntaxique), ou simplement des corpus textuels (non étiqueté), on doit effectuer l'apprentissage des règles elle mêmes. On a besoin pour cela de techniques d'estimation spécifiques, à base de réestimations itératives convergeant vers un maximum de vraisemblance (l'algorithme inside-outside [Baker ,1979] [Charniak, 1993]. Malheureusement, les performances de cet algorithme ne sont limitées [Rajman, 1996] [Miles et al, 1997]. En fait, il est plus simple et plus efficace de dériver les règles d'un corpus arboré comme celui que nous utilisons

dans cette étude. Avec un tel corpus, seul un estimateur de la probabilité à associer à chaque règle est nécessaire.

Nous introduisons ici quelques définitions et notations qui nous permettrons de décrire l'estimateur que nous utilisons dans nos expériences. Tout d'abord, chaque règle d'une grammaire hors-contexte est définie dans un format $LHS \rightarrow RHS$ (*LHS* représente la partie à gauche, *RHS* représente la partie à droite). On appelle: *règle lexicale* toute règle de la forme $A \rightarrow a$ où $A \in N$ et $a \in T$, *règle unitaire* toute règle de la forme $A \rightarrow B$ où $A, B \in N$, *règle interne* toute règle de la forme $A \rightarrow N_1 N_2 \alpha$ où $N_1, N_2 \in N, \alpha \in N^*$. Un exemple de chaque règle est donné dans la figure suivante:

```
Règle lexicale : NN→week

Règle unitaire : NP→NNS

Règle interne  : NP→DT JJS NN
```

Le but de l'apprentissage d'une *PCFG* est ici de définir la probabilité conditionnelle $P(LHS \rightarrow RHS \mid LHS)$ associée à chaque règle. Cette approche est principalement basée sur la théorie de probabilité que nous abordons de manière rapidement.

2.3.1 La théorie de probabilité

Nous supposons quelques définitions standard de la théorie de probabilité [Collins, 1999]. Si l'ensemble ς est un espace d'événements discrets et P est une distribution de probabilité sur cet espace, alors:

$$0 \leq P(A) \leq 1 \quad pour \quad tous \quad A \in \varsigma \quad et \quad \sum_{A \in \varsigma} P(A) = 1$$

Dans la plupart des exemples, P est une fonction d'un vecteur de paramètres Θ. Nous écrivons $P(A \mid \Theta)$ comme la probabilité de A étant donné un vecteur de paramètres Θ. L'espace des paramètres Ω est alors l'espace $\{\Theta \mid P(A \mid \Theta)$ est une mesure de probabilité sur $A\}$.

Prenons un exemple classique, celui du jet d'une pièce de monnaie qui peut apparaître comme pile (P) ou la face (F), où la probabilité de pile est p. Dans ce cas:

- L'espace d'événement ς est l'ensemble {P,F}
- le vecteur de paramètres Θ est constitué d'un seul élément p
- La probabilité $P(A|\Theta)$ est définie comme p si A = P, $1-p$ si A = F
- L'espace de paramètre Ω est un ensemble [0,1] (p doit prendre les valeurs entre 0 et 1 pour que $P(A|\Theta)$ soit une mesure de probabilité)

Estimation par maximum de vraisemblance

Supposons maintenant qu'on dispose d'une séquence de n événements indépendants mais tirés de la même distribution ς $X = \{X_1, X_2, X_3, \ldots\ldots, X_n\}$. Par exemple on jette une pièce de monnaie 5 fois et on observe $X = <P, F, F, P, F>$. Avec une telle observation, on peut calculer $\hat{\Theta}$, l'estimation de Θ.

Une méthode très générale est l'estimation par maximum de vraisemblance qui consiste à maximiser la probabilité de l'observation. Supposons que les événements sont indépendants l'un de l'autre, la fonction de probabilité L de l'observation est définie par :

$$L(X|\Theta) = \prod_{i=1..n} P(X_i | \Theta)$$

L'estimation par maximum de vraisemblance $\hat{\Theta}$ est la valeur de Θ (dans l'espace de paramètre Ω) qui maximise cette fonction de vraisemblance

$$\hat{\Theta}_{ML} = \arg\max_{\Theta\in\Omega} L(X|\Theta)$$

Dans l'exemple des pièces de monnaie, la vraisemblance de l'échantillon X = <P, F, F, P, F> est:

$$L(<P,F,F,P,F>|\Theta) = p(1-p)(1-p)p(1-p) = p^2(1-p)^3$$

Et l'estimation de p qui maximise cette vraisemblance est

$$\hat{p}_{ML} = \arg\max_{p\in[0,1]} p^2(1-p)^3 = \frac{2}{5}$$

On peut pour s'en persuader dériver cette équation ou simplement considérer l'inégalité de Cauchy : $p^2(1-p)^3 \le \frac{3^3}{2^3} p^2(\frac{2}{3} - \frac{2}{3}p)^3 \le \frac{3^3}{2^3}(\frac{2}{5})^5$, où $p^2(1-p)^3$

est maximal si et seulement si $p = \frac{2}{5}$. Intuitivement, cette estimation consiste à compter le nombre d'observations positives (ici la sortie d'un pile) et à normaliser ce compte par le nombre total de tirages. C'est ce qu'on appelle également l'estimée par la fréquence relative (la fréquence d'un événement est le nombre de fois où cet événement se produit)

$$\text{fréquence relative} = \frac{\text{fréquence d'occurence}}{\text{nombre total d'expériences}}$$

Revenons à notre problème d'estimation de la probabilité des règles d'une CFG. Si S est une phrase à analyser et T est un arbre d'analyse dérivé en utilisant n règles de la forme $LHS \rightarrow RHS$ et si nous supposons que ces dérivations sont indépendantes les unes des autres, alors la probabilité d'un arbre T étant donnée la phrase S est:

$$P(T \mid S) = \prod_{i=1..n} P(LHS_i \rightarrow RHS_i \mid LHS_i)$$

Les paramètres de chaque règle sont estimés en utilisant l'estimation de maximum de vraisemblance (la fréquence relative), c'est à dire

$$P(LHS_i \rightarrow RHS_i \mid LHS_i) = \frac{Count(LHS_i, RHS_i)}{Count(LHS_i)} \qquad (2.1)$$

Où $Count(LHS_i, RHS_i)$ est le nombre de fois où $< LHS_i, RHS_i >$ apparaît dans les données d'apprentissage et $Count(LHS_i) = \sum_{RHS_i \in Y} Count(LHS_i, RHS_i)$ est le nombre de fois où $< LHS_i >$ est vu dans les données d'apprentissage. La formule 2.1 est celle que nous utilisons dans notre travail. Par exemple, pour notre corpus d'entraînement, la règle $NP \rightarrow NP\ PP$ apparaît *34965* fois et le symbole non terminal NP apparaît *313160* fois. La probabilité associée à cette règle est donc *34965/313160.*

2.4 Analyseur syntaxique statistique

En général, le modèle d'analyse syntaxique définit la probabilité conditionnelle *P(T|S)*, pour chaque arbre d'analyse syntaxique T étant donné une phrase S. L'analyseur syntaxique est un algorithme qui va chercher l'arbre T qui maximise cette quantité.

$$\hat{T} = \arg\max_T P(T \mid S) = \arg\max_T \frac{P(T,S)}{P(S)} = \arg\max_T \max P(T,S)$$

Il existe plusieurs algorithmes génériques qui permettent d'effectuer cette maximisation. Nous décrivons l'algorithme CYK et l'algorithme d'analyse tabulaire (Chart parsing) qui sont très couramment utilisés.

2.4.1 Analyseur syntaxique CYK (Cock-Younger-Kasami)

L'idée principale de CYK [Younger D, 1967] est de considérer toutes les dominances d'un non terminal jusqu'à obtenir un non terminal qui domine la phrase entière. C'est à dire qu'il existe une dérivation de la chaîne à analyser à l'axiome de la grammaire. Les règles lexicales sont premièrement utilisées pour chercher les dominances maximales de longueur 1. L'algorithme examine ensuite toutes les parties droites de règles pour découvrir l'ensemble des dominances maximales de longueur quelconque m. Pour chaque règle de la forme $A \rightarrow A_1 A_2 .. A_{k-1} A_k$, il cherche un ensemble de span $\{m_1 m_2 .... m_{k-1} m_k\}$ qui satisfait deux conditions suivantes :

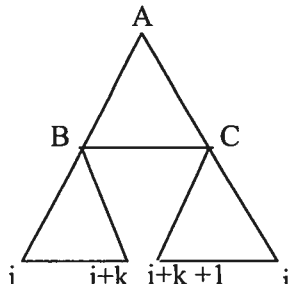
- $\sum_{i=1..k} m_k = m$ où A_i domine $m_i \forall i = 1..k$ **(I)**
- $score(A) = p(A \rightarrow A_1 A_2 .. A_{k-1} A_k) * score(A_1) .... score(A_k)$ où $score(A_i)$ est la probabilité maximale de A_i dominant m_i $\forall i = 1..k$ **(II)**

Tout d'abord, on considère la première version de cet algorithme qui présuppose que toutes les règles sont au format CNF (Chomsky Normal Form). En format *CNF*, chaque règle prend un des deux formes suivantes:

$A \rightarrow BC$ où *A, B, C* sont des non terminaux

$A \rightarrow a$ où *A* est non terminal , *a* est terminal

L'algorithme est basé sur une structure de donnée que nous représentons formellement par *best[i,A,j]* et *back [i,A,j]=<r,k>*. *Best[i,A,j]* représente le score maximal que peut obtenir *A* lorsqu'il domine w_i^j, *back [i,A,j]=<r,k>* est un doublet qui mémorise l'information nécessaire pour retrouver cette dominance maximale. En pratique, le numéro *r* de la règle $A \rightarrow BC$ employée pour dériver *A*



ainsi que le point de coupure *k* (voir la figure ci-dessous) suffisent à retrouver l'arbre le plus probable (s'il existe) à partir de *back [1,S,n]* (où n est la longueur de la phrase à analyser).

Cet algorithme cherche l'arbre le plus probable en temps $O(n^3 |N|^3)$ (où n est la longueur de la phrase à analyser et N est le nombre de symboles non terminaux)

```
best[1..n, 1..N, 1..n] := 0
 back[1..n,1..N, 1..n] := NULL
    for all k ∈ [1..n] do
        for all rule A → w_k ( soit r l'indice de cette règle) do
            best[k,A,k] = p(A → w_k)
            back[k, A, k] = <r,0>

      for all l ∈ [1,n] do
        for all s ∈ [1, n - l + 1] do
            for all k ∈ [ s, s+ l-1) do
              for all r ∈ [ 1, |R| ] do
                score = p( A → BC ) * best[s, B, k] * best [k +1, C, s + l –1]
                if ( score > best[s, A, s+l –1] ) then (garder l'arbre le plus
                    best[s, A, s+ l –1] =score                probable)
                    back[s, A, s+l-1] = <r,k>

best [1, S, n] est la probabilité de la meilleure analyse
```

Table 2.5 Algorithme CYK pour grammaire CNF

Exemple : On illustre l'algorithme avec la grammaire de la table 2.4 (qui contient 12 règles) et la phrase à analyser 'astronomers saw stars with ears'. Dans cette illustration, la table 2.6 représente la structure best (la valeur de la ligne i et la colonne j permet de trouver le symbole non terminal dominant de i à j avec sa probabilité), la table 2.7 représente la structure back (la valeur de la ligne i et la colonne j permet de trouver le symbole non terminal dominant de i à j avec la règle utilisée et sa coupure).

	1	2	3	4	5
1	NP(0.1)				S(0.0009072)
2		NP(0.04) V(1.0)	VP(0.126)		VP(0.006804) VP(0.009072)
3			NP(0.18)		NP(0.01296)
4				P(1.0)	PP(0.18)
5					NP(0.18)

Table 2.6 Structure best

	1	2	3	4	5
1	NP<8,0>				S<1,1>
2		NP<10,0> V<6,0>	VP<3,2>		VP<4,3> VP<3,2>
3			NP<11,0>		NP<5,3>
4				P<7,0>	PP<2,4>
5					NP<9,0>

Table 2.7 Structure back

Tout d'abord, les structures de *best* et *back* sont remplies à l'aide des règles lexicales telles que avec la règle numéro *8* (*best[1,NP,1] = 0.1* et *back[1,NP,1] = <8,0>)*; ou encore la règle *10 (best[2,NP,2] = 0.04, back[2,NP,2] =<10,0>)* etc. Les règles unitaires et internes sont ensuite utilisées pour remplir les structures *best* et *back.* Par exemple, on peut appliquer la règle 3 (*VP→ V NP)* car *V* domine le span *(2,2)* (grâce à la règle *6*) et *NP* domine le span *(3,3) (*grâce à la règle *11).* Donc par combinaison, (*k=2*) on obtient *best[2,VP,3]=0.126* qui est la multiplication de *0.7* (la probabilité de la règle *3*), de *1.0* (le score associé à *best[2,V,2])* et de *0.18* (*best[3,NP,3]*). De la même manière, on peut appliquer la règle *5* (*NP→NP PP*) car *NP* domine le span *(3,3)* (grâce à la règle *11*) et *PP* domine le span (*4,5*) (en raison de la règle *2*) avec la combinaison *k=3*. La case *(2,5)* dans ces tables illustre l'opération de maximisation faite tout au long du processus. Supposons que la règle *4* (*VP→ VP PP*) ait été appliquée pour le span (*2,5*), donnant ainsi un score de *0.006804*. Si maintenant la règle *3* (*VP →V NP*) est appliquée pour le même span, alors son score est de *0.009072,* le score est meilleur que le précédant, c'est donc l'application de la règle 3 qui est conservée. Finalement, l'algorithme s'arrête par l'application de la règle 1 (*S → NP VP*) pour le span *(1,5) (k = 1, score = 0.0009072)*. L'arbre d'analyse de la *figure 2.3* est obtenu en utilisant la structure *back (table 2.7).*

L'algorithme que nous venons de décrire fonctionne pour des grammaires au format CNF. Ce format n'est pas adapté aux besoins classiques de l'analyse grammaticale (multiplication des règles, structure moins apparante, désavantage lorsqu'on utilise l'évaluation PARSEVAL, etc). Par cette raison, nous proposons une extension de cette algorithme à toute grammaire de type 2. Les règles peuvent avoir maintenant plus de deux non terminaux en partie droite (par exemple *NP→DT JJS NN)* et la structure *back[i,N,j] = <r,k>* n'est plus suffisante pour

garder les informations qui permettent de savoir quelle partie de la règle va dominer quelle portion de la phrase. On modifie donc la structure *back* comme suit: $back[i,A,j] = <r,\{V\}>$ où $r=A \rightarrow A_1A_2...A_k$ et $V = \{v_1.....v_{k-1}\}$, $back[i,A,j]$ signifie qu'en appliquant la règle *r*, *A* domine de i à j, A_m domine de v_{m-1} à v_m où $1 \leq m \leq k$, $v_0 = i$, $v_k = j$. L'algorithme devient alors :

```
best[1..n, 1..N, 1..n] := 0
back[1..n, 1..N, 1..n] := {}
    for all k ∈ [1..n] do
        for all rule A → w_k ( soit r est l'indice de cette règle) do
            best[k, A ,k] = p(A → w_k)
            back[k, A ,k] = {r, {k}}
        for all l ∈ [1,n] do
            for all s ∈ [1, n - l + 1] do
                for all r ∈ [ 1, |R| ] do
                    split(r, 0, s, s, s + l, {}, 0);
```

Table 2.8 Algorithme CYK pour non CNF

Split est une fonction qui cherche les possibilités de dominance du non terminal A de *s* à *s + l* inclusivement en satisfaisant les deux conditions (I) et (II). Split est décrit d'une façon récursive comme suit :

```
split(rule, n, mstart, start, stop, cuts, score)
if ( n = rhs(rule)) then
    if (start > stop) then
        if best[mstart, lhs(rule), stop] < score* p(rule) then
            best[mstart, lhs(rule), stop] = score * p(rule)
            back[mstart, lhs(rule), stop] = {rule, cuts}
        else
            for all j ∈ [ start, stop – rhs(rule) + n + 1] do
                s = best[start, rhs(rule,n), j]
                if (s > 0) then
                    cuts = cuts + {j}
                    if (score >0) then split(rule, n+ 1, mstart, j+1, stop, cuts, score* s)
                    else split(rule, n+ 1, mstart, j+1, stop, cuts, s)
                    cuts = cuts - {j}
où
rhs(rule) retourne le nombre de non terminaux de la partie droite de la règle.
rhs(rule, n) retourne le non terminal à la n-ième position de la partie droite de la
règle.
lhs(rule) retourne le non terminal de la partie gauche de la règle.
p(rule) est la probabilité de la règle
best [1, S, n] est la probabilité de la meilleure analyse.
```

Table 2.9 Description de la fonction split de l'algorithme CYK étendu

Nous illustrons cette extension sur un exemple: soit la règle r=S →NP VP PP. Pour chercher la possibilité de dominance S de 1 à 5, on appelle split(r,0,1,1,5,{},0). La fonction split va chercher toute les possibilités suivantes (i-j indique une dominance du non terminal associé de i à j)

NP	VP	PP
1_1	2_2	3_5
1_1	**2_3**	**4_5**
1_1	2_4	5_5
1_2	3_3	4_5
1_2	3_4	5_5
1_3	4_4	5_5

Avec les structures qu'on a (les structures best et back des tables 2.6 et 2.7), Split trouve que NP domine de 1 à 1, VP domine de 2 à 3, PP domine de 4 à 5.

Il reste cependant un problème de cet algorithme avec les règles unitaires. Par exemple, quand on considère la règle A→B à partir de 3 à 5. À ce moment là, on n'a pas B dominant [3,5], c'est pourquoi on ne peut pas ajouter A dominant [3,5]. On ne peut donc pas trouver tous les symboles non terminaux qui peuvent dominer [3,5]; l'arbre d'analyse syntaxique qu'on trouve à la fin de cet algorithme n'est donc pas nécessairement l'arbre le plus probable. Pour résoudre ce problème, on peut organiser l'ensemble de règles en deux sous ensembles, le sous ensemble de règles internes [R_i] et le sous ensemble de règles unitaires [R_u]. Chaque fois qu'on peut appliquer une règle unitaire, on reconsidère toutes les règles unitaires. L'algorithme CYK pour non CNF est finalement modifié comme suit :

```
best[1..n, 1..N, 1..n] := 0
back[1..n, 1..N, 1..n] := {}
    for all k ∈ [1..n] do
        for all rule A → w_k ( soit r est l'indice de cette règle) do
            best[k, A ,k] = p(A → w_k)
            back[k, A ,k] = {r, {k}}
        for all l ∈ [1,n] do
            for all s ∈ [1, n - l + 1] do
                for all r ∈ [ 1, |R_i| ] do
                    split(r, 0, s, s, s + l, {}, 0);
            for all s ∈ [1, n - l + 1] do
                for all r ∈ [ 1, |R_u| ] do
                    split(r, 0, s, s, s + l, {}, 0)
                    s'il y a un changement, repéter pour r ∈ [ 1, |R_u| ];
```

Table 2.10 Algorithme modifié pour non CNF

2.4.2 Analyseur syntaxique CHART

L'analyse syntaxique CHART [Charniak, 93] a trois structures de données principales: un diagramme, une liste de clés, et un ensemble d'arrêtes. Un diagramme est un ensemble d'entrées de diagramme; chaque entrée comprend le nom du symbole non terminal qui couvre le span considéré. Une liste de clés est une pile d'entrée de diagramme qui est en attente pour entrer dans le diagramme. Il y a seulement deux fonctions pour la pile : empiler et dépiler. Un ensemble d'arrêtes représente les règles qui sont en train d'être traitées. La première version de CHART que nous donnons recherche tous les arbres d'analyse syntaxique possibles. L'algorithme est décrit comme suit :

Pour remplir le diagramme

Répéter jusqu'à ce que la liste de clés soit vide

1. *Enlever un élément de la liste de clé (dépiler)*
2. *Si l'élément existe dans le diagramme, ajouter la liste des arrêtes de cet élément à la liste déjà existante dans le diagramme et aller à l'étape 1, sinon aller à l'étape 3*
3. *Ajouter un élément de la liste dans le diagramme*
4. *Pour chaque règle qui commence par cet élément, ajouter une arrête pour cette règle au diagramme*
5. *Pour chaque règle qui a besoin de cet élément, ajouter une arrête étendue*
6. *Si l'arrête est finie, ajouter cet élément à la liste de clés avec le type approprié, le point de départ, la longueur et la liste des arrêtes (empiler)*

Pour étendre une arrête e avec un élément c:

1. *Créer une nouvelle arrête e'*
2. *Mettre start(e') à start(e)*
3. *Mettre end(e') à end(e) + c*
4. *Mettre rule(e') à rule(e) avec o déplacé après c*

Table 2.11 Algorithme CHART

Nous illustrons cette algorithme en reprenant notre grammaire exemple de la table 2.4 et la phrase à analyser 'astronomers saw stars with ears'

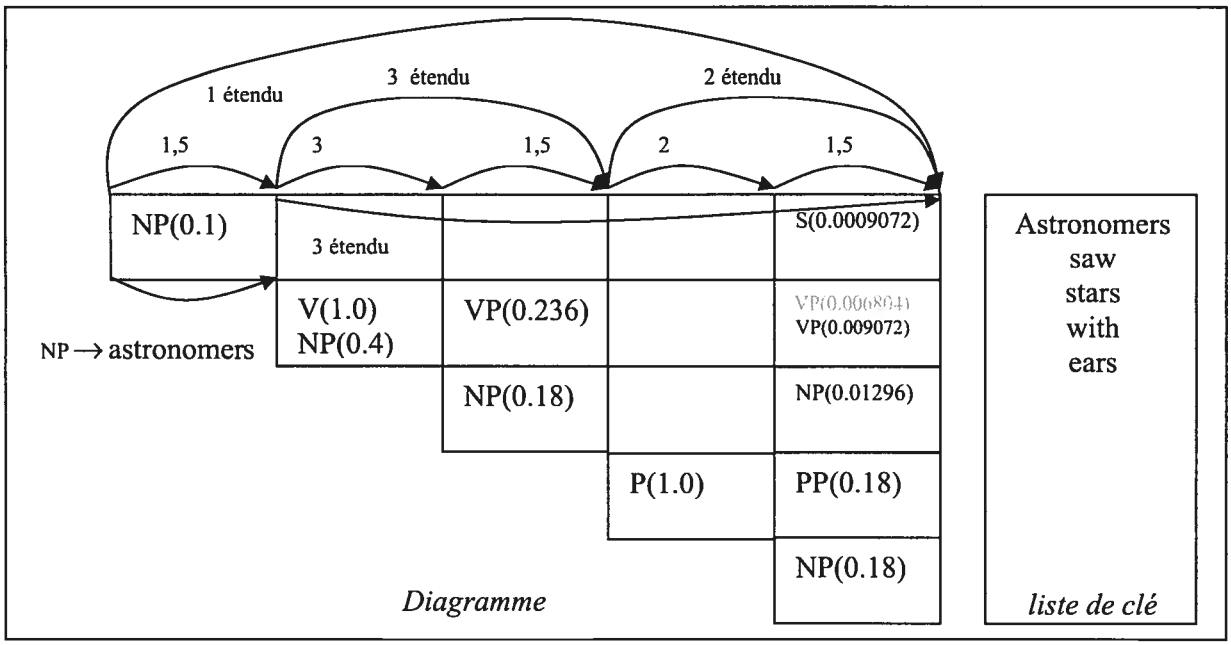


FIG. 2.5 Exemple d'analyse syntaxique selon l'algorithme CHART

L'algorithme commence avec la liste de clés contenant, en séquence, l'ensemble des mots à analyser (voir la figure 2.5). À l'étape *1*, *'astronomer'* est dépilé. Aux étapes *2,3, NP* est ajouté dans le diagramme avec la probabilité *0.1*. À l'étape *4*, on ajoute les arrêtes pour les règles *1* et *5* car leur partie gauche commence par *NP*. Pour la deuxième itération, *V et NP* sont ajoutés, l'arrête pour la règle *3* est ajoutée. Pour la troisième itération, après avoir ajouté *NP* et les arrêtes pour les règles *1 et 5*, on doit ajouter une arrête étendue pour la règle 3 à l'étape *5*. La règle *3* est donc complétement appliquée et VP est ajouté dans la liste de clés à l'étape 6. Finalement, S est ajouté dans le diagramme et la liste de clés est vide. L'algorithme s'arrête, les arbres d'analyse comme t_1 (voir la figure 2.3) et t_2 (voir la figure 2.4) sont dérivés en utilisant les informations des arrêtes.

Avec cet algorithme, on peut obtenir tous les arbres d'analyse syntaxique possibles étant donné un ensemble de règles et une phrase à analyser. Cependant, cette version de l'algorithme boucle en présence de règles comme: $A \rightarrow B$ et $B \rightarrow A$, parce que l'ajout de B dans CHART va produire l'ajout de A et vice versa.

Dans nos expériences, nous cherchons seulement l'arbre le plus probable. C'est la raison pour laquelle, nous étendons l'algorithme décrit par Charniak comme suit:

2. *si l'élément existe dans le diagramme :*

   - *Si la probabilité de cet élément est inférieuere ou égale à celle de l'élément considéré, aller à l'étape 1.*
   - *Si la probabilité de cet élément est supérieure à celle de l'élément considéré, aller à l'étape 3.*

Avec cette modification, on a toujours un seul symbole non terminal dans le diagramme qui domine un ensemble de mots donnés. L'arbre d'analyse t_2 (voir la figure 2.4) est éliminé. Cette modification permet d'éviter le bouclage à l'infini. Par exemple, on a B dominant [3,5] avec la probabilité 0.005 et on a deux règles $A \rightarrow B$ [0.002] et $B \rightarrow A$ [0.004], tout d'abord, on arrive à ajouter A [3,5] dans le diagramme avec la probabilité [10^{-5}] , cela entraîne la considération de la règle $B \rightarrow A$ [0.004]. B[3,5] obtient la probabilité [4.10^{-8}] tandis que la probabilité de B dans le diagramme est 0.005. On évite ainsi de boucler.

Cet algorithme s'arrête si et seulement si la liste de clés est vide, mais la longueur de cette liste est changée dynamiquement (l'ajout à l'étape *6*). La complexité de cet algorithme dépend énormément de l'implémentation et de la structure de donnée qu'on utilise (dans la partie Annexe de [Collins, 2000], on peut trouver son calcul avec son implémentation). Avec notre implémentation, on essaie de réduire le plus possible le temps d'exécution à l'étape 4,5 c'est à dire, on utilise les structures de données supplémentaires pour permettre de considérer le plus vite toutes les règles qui commencent par l'élément qu'on vient d'ajouter au diagramme et celles qui peuvent être étendues par cet élément.

Les deux algorithmes que nous avons décrit (CYK et CHART) ont été implémentés en C++ avec une utilisation massive des collections de la librairie STL (*Standard Template Library*). Nous avons observé que l'algorithme CYK est meilleur que celui de CHART par rapport au temps d'exécution et à l'espace mémoire utilisé, ce qui nous a amené à ne considérer que CYK dans la suite de nos expériences.

2.5 ParseEval

Nous discutons dans cette dernière section du problème de l'évaluation d'un analyseur syntaxique. Le groupe *Grammar Evaluation Interest* a proposé une technique d'évaluation qui est actuellement la plus largement utilisée et qu'on désigne par PARSEVAL ([Harison at al, 1991] ou [Black et al. 91]). Cette

méthode compare la structure syntaxique produite par l'analyseur syntaxique avec celle d'un corpus de référence annoté syntaxiquement (Treebank). L'idée est de calculer le nombre de parenthèses se croisant par phrase. Concrètement, PARSEVAL comprend les trois mesures de base: précision (*precision*), rappel (*recall*) et croisement (*crossing-brakets*), qui sont décrites dans la table *2.12* où un constituant est un non terminal qui domine de telle position à telle position inclusivement dans la phrase (par exemple, dans la figure 2.6, le constituant S(0,12) signifie que le non terminal S domine de 0 à 12).

Rappel = $\frac{\text{nombre de constituants corrects de l'arbre produit par le parseur}}{\text{nombre de constituants de l'arbre annoté dans Treebank}}$
Précision = $\frac{\text{nombre de constituants corrects de l'abre produit par le parseur}}{\text{nombre de constituants de l'arbre produit par le parseur}}$
Croisement = nombre de constituants qui violent la frontière de constituants de l'arbre annoté dans Treebank (un exemple détaillé est décrit ci-dessous)

Table 2.12 Métriques PARSEVAL

Une phrase est correctement analysée si le taux de croisement est nul, la précision et le rappel sont à 1 ou 100%. Pour illustrer ces métriques, nous considérons l'exemple suivant :

Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group

```
L'arbre de référence dans PTB
( (S
    (NP (NNP Mr.) (NNP Vinken) )
    (VP (VBZ is)
      (NP
        (NP (NN chairman) )
        (PP (IN of)
          (NP
            (NP (NNP Elsevier) (NNP N.V.) )
            (, ,)
            (NP (DT the) (NNP Dutch) (VBG publishing) (NN group) )))))
    (. .) ))

Constituants:
S(0,12), NP(0,1), NNP(0,0), NNP(1,1), VP(2,11), VBZ(2,2), NP(3,11),
NP(3,3), NN(3,3), PP(4,11), IN(4,4), NP(5,11), NP(5,6), NNP(5,5),
NNP(6,6), ,(7,7), NP(8,11), DT(8,8), NNP(9,9), VBG(10,10), NN(11,11),
.(12,12)

L'arbre proposé par l'analyseur syntaxique testé
(S
  (NP
   (NNP Mr.)
   (NNP Vinken))
  (VP
   (VBZ is)
   (NP
    (NP
     (NP
      (NN chairman))
     (PP
      (IN of)
      (NP
       (NNP Elsevier)
       (NNP N.V.))))
    (, ,)
    (NP
     (DT the)
     (NNP Dutch)
     (VBG publishing)
     (NN group))))
  (. .))

Constituants:
S(0,12), NP(0,1), NNP(0,0), NNP(1,1), VP(2,11), VBZ(2,2), NP(3,11),
NP(3,6), NP(3,3), NN(3,3), PP(4,6), IN(4,4), NP(5,6), NNP(5,5),
NNP(6,6), ,(7,7), NP(8,11), DT(8,8), NNP(9,9), VBG(10,10), NN(11,11),
.(12,12).
```

FIG. 2.6 Measures PARSEVAL

Dans cet exemple, les positions sont calculées à partir de 0 et les constituants reportés en gras apparaissent en même temps dans l'arbre de référence et l'arbre proposé de l'analyseur syntaxique testé. On a donc:

Précision = 20/22 = 0.909091; Rappel =20/22 = 0.909091; Croisement = 1 (NP(3,6) croise NP(5,11)).

Il est important de considérer ces trois mesures en même temps car il est facile d'augmenter arbitrairement la précision ou le rappel au détriment des autres mesures. Ceci s'observe sur l'exemple suivant où la précision est parfaite (100%), le croisement est nul alors que le rappel est faible (33%):

Treebank : *[[They [came]][yesterday]]*

Sortie de l'analyseur: *[They came yesterday]*

PARSEVAL est une technique simple qui permet d'évaluer différents analyseurs indépendamment du système mais il a certains inconvénients. En particulier, Lin (1995) observe qu'une seule erreur dans l'analyse syntaxique peut être comptée plusieurs fois ce qui provoque une mauvaise évaluation des performances de l'analyseur syntaxique testé.

Exemple : considérons les trois phrases suivantes

(a) [I [saw [[a man] [with [[a dog] and [a cat]]]] [in [the park]]]]

(b)[I [saw [[a man] [with [[a dog] and [[a cat [in [the park]]]]]]]]

(c) [I [saw [a man] with [a dog] and [a cat] [in [the park]]]]

(a) est l'analyse correcte de la phrase, *(b)* et *(c)* sont respectivement les analyses de deux analyseurs différents. Dans l'analyse *(b)*, il y a seulement une erreur à cause de l'attachement de *PP* (*in the park*) à *'a cat'* tandis que l'analyse (*c)* a plusieurs erreurs. *(b)* est donc plus satisfaisante que *(c)*. Cependant, avec PARSEVAL, *(b)* est pire que *(c): (b)* donne une précision de *63.6%,* un rappel de *60%* et un taux de croisement de *3,* alors que *(c)* donne une précision de *100%, un rappel* de *70%,* un taux de croisement de *0)*.

Malgré cette faiblesse, PARSEVAL est utilisé par plusieurs chercheurs pour vérifier la qualité de leurs modèles. Nous l'utilisons également pour nos expériences.

Chapitre 3. Corpus

Un corpus est une collection de données langagières qui sont sélectionnées et organisées selon l'objectif (étude grammaticale, test d'un système de reconnaissance de la parole, etc.). Dans le cadre du traitement automatique des langues naturelles, les corpus sont souvent utilisés pour entraîner et tester des modèles. Nous distinguons dans la suite deux types de corpus : les corpus étiquetés et les corpus annotés. Par corpus étiqueté, nous entendons un corpus de textes dont les mots sont associés à leur étiquette syntaxique (*part-of-speech ou POS).* De tels corpus sont par exemple souvent utilisés par entraîner des *taggeurs,* qui sont des programmes que certaines analyseurs syntaxiques utilisent (voir chapitre 4). Par corpus annoté, nous désignons les corpus où les phrases ont été arborées (la structure de chaque phrase du corpus est explicitée). L'organisation de ce chapitre est comme suit:

- Section 3.1 : nous parlons des corpus utiles : les corpus étiquetés Brown et BNC, les corpus annotés Lancaster/IBM Treebank, Susanne et Corfrans.
- Section 3.2 : nous décrivons le corpus annoté Penn Treebank et la présentation des portions *Wall Street Journal* de ce corpus que nous utilisons comme corpus d'entraînement et de test.

3.1 Corpus utiles

Dans cette section, nous considérons quelques corpus qui peuvent servir à la mise au point d'analyseurs syntaxiques.

Brown (*Standard Corpus of American English*) [Brown corpus] est un corpus étiqueté d'un million de mots en anglais américain extraits de textes américains publiés en 1961. La compilation de ce corpus a commencé en 1979 à l'université Brown aux États-Unis par W.N. Francis et H. Kucera. Le corpus intègre 15 genres de textes différents (reportage, presse, textes littéraires, textes scientifiques et techniques, textes de fiction, etc.) et utilise 87 étiquettes syntaxiques (syntagmes) simples.

BNC (*British National Corpus*) [BNC corpus] est un corpus étiqueté de 100 millions de mots anglais; commencé en 1991 et terminé en 1994 à partir de textes de plusieurs sources: journaux régionaux et nationaux, textes de fiction, les livres académiques etc.

Lancaster/IBM Treebank [Lancaster corpus] est un corpus arboré d'un million de mots provenant de l'agence *Associated Press*, d'un million de mots extraits des textes du parlement canadien, de 200,000 mots de *American Printing House for the Blind,* et de 800,000 mots de manuels IBM.

Susanne (***Surface and Underlying Structural ANalyses of Natural English***) [Sampson, 1995] est un corpus annoté de 130,000 mots réalisé par Geoffrey Sampson et constitué de 64 extraits de 500 textes tirés du Brown. Chaque phrase est assortie d'un arbre syntaxique fournissant des étiquettes catégorielles et fonctionnelles.

Corfrans (***CORpus de référence annoté pour la syntaxe en FRANçaiS***) l'objectif du projet Corfrans est de produire un corpus de qualité de taille moyenne (200,000 du journal Le Monde, 200,000 mots du Hansard, 100,000 du journal L'hebdo), représentatif des principales difficultés grammaticales rencontrées en français norme. Le projet est coordonné par l'équipe Talana (Université Denis Diderot Paris 7), et mené conjointement par le Talana, le LATL (Université de Genève) et le laboratoire RALI (Université de Montréal).

À ces corpus anglophones et francophones, s'ajoutent des corpus annotés portant sur d'autres langues comme l'allemand (NEGRA corpus de l'université de

Saarlandes), l'espagnol (corpus de l'université de Autonoma de Madrid), l'italien (Turin University Treebank) et le japonais [Japonais corpus].

3.2 Penn Treebank

Dans cette section, nous décrivons le corpus Penn Treebank (PTB). Ensuite, nous précisons la portion du *Wall Street Journal* (WSJ) que nous utilisons dans notre étude.

3.2.1 Introduction

Le PTB a été constitué entre 1989 et 1995 par l'université de Pennsylvania. C'est un corpus annoté d'environ 4.5 millions de mots d'anglais-américain. Les textes de ce corpus sont composés de plusieurs sources différentes : articles du *Wall Street Journal* (WSJ), résumés de textes du Département à l'énergie, bulletins du Département d'agriculture, une bibliothèque de textes américains, le Brown corpus réétiqueté, des phrases des manuels d'IBM, des phrases du corpus d'ATIS (corpus de parole spontanée transcrite), des phrases prononcées à la radio, des messages MUC-3 (informations sur les activités terroristes en Amérique du Sud), etc. Ce corpus est disponible pour les membres du "Linguistic Data Consortium" en ligne et par CD-Rom. Le PTB est en fait un corpus dont l'annotation a été produite de façon semi-automatisée. L'étiquetage et le parenthésage ont été sont automatiquement produits puis corrigés par des annotateurs humains. L'étiquetage du PTB est basé sur celui du Brown Corpus (87 étiquettes simples) mais avec un jeu d'étiquettes réduit (décrit en annexe) [Satorini, 1990] [Marcus et al, 1993]. Le parenthésage syntaxique (syntactic bracketing) proposé dans la première version (terminée en 1992) était simple, n'indiquait pas les dépendances et les structures non-contigues [Santorini et al, 1991] et manquait de rigueur et de consistance. Une nouvelle version de l'annotation du PTB a été produite comme avec mission de maintenir la consistance avec la première version, tout en proposant l'annotation d'une structure prédicat-argument (une sorte de représentation sémantique [Marcus et al., 1995]) que certains analyseurs utilisent. Nous illustrons cette annotation sur la phrase exemple: " *The market crumbled* "

```
( (S
    (NP-SBJ
       (DT The)
       (NN market) )
    (VP
       (VBD crumbled) )
    (. .) ))
```

Table 3.1 Phrase exemple de PTB

Dans cet exemple, tout d'abord *'The'* est étiqueté par *DT (determiner), 'market'* par *NN (noun), 'crumbled'* par *VBD (past tense) et '.'* par *.(sentence-final punctuation). DT* et *NN* sont ensuite annotés par *NP-SBJ* (un nom qui joue le rôle du sujet), *VBD* par *VP (verbal phrase). NP-SBJ, VP* et . sont enfin annotés par *S(simple declarative clause)* (voir [Marcus et al., 1995] pour de plus amples explications sur cette nouvelle annotation).

3.2.2 Corpus d'entraînement et de test

À l'instar d'autres études [Magerman, 1995], [Charniak, 1997], [Collins, 1999], [Goodman, 1997], nous avons décidé de restreindre le PTB aux seuls textes du WSJ. La portion WSJ comprend 24 sections, nous utilisons les sections 2-21 pour l'entraînement et la section 23 pour le test. Le corpus d'entraînement comprend (*39832* phrases, *39548* mots différents), la distribution de la longueur de ces phrases est comme suit:

Longueur	Nombre de phrase	Pourcentage
]0,12]	6046	15.17%
]12, 20]	10621	26.67%
]20, 40]	20098	50.45%
]40,100]	3064	7.7%
]100,141]	3	0.01%

Le corpus de test comprend (*2416* phrases, *8423* mots), la distribution de la longueur de ces phrases est comme suit:

Longueur	Nombre de phrase	Pourcentage
]0,12]	402	16.64%
]12, 20]	632	26.16%
]20, 40]	1211	50.13%
]40,100]	171	7.07%

Ce découpage est celui que respectent la plupart des chercheurs travaillant à l'élaboration de grammaires probabilistes.

Chapitre 4. Expériences

Dans ce chapitre, nous présentons les différentes expériences que nous avons effectuées en utilisant le modèle PCFG avec le corpus PTB. Tout d'abord, nous proposons un survol de la littérature consacrée à l'induction de grammaires probabilistes à partir du PTB. Nous décrivons les principaux modèles proposés et leurs résultats. Nous présentons ensuite nos expériences sur les transformations faites au PTB et les résultats des modèles associés. Nous identifions la transformation la plus pertinente et proposons une solution simple mais efficace au traitement des mots inconnus et au filtrage des règles de grammaire en vue de simplifier le modèle. En combinant ces techniques, nous proposons une suite des transformations des arbres du PTB qui permet d'obtenir des résultats comparables aux meilleurs modèles décrits dans la littérature. L'organisation de ce chapitre est comme suit :

- Section 4.1: nous mentions brièvement les modèles dominants la littérature consacrée [Magerman, 1995], [Charniak, 1996], [Charniak, 1997], [Collins, 1996], [Collins, 1997] , [Collins, 1999], [Goodman, 1997].
- Section 4.2: nous décrivons les transformations que nous testons, présentons leurs résultats et identifions la transformation qui offre les meilleures performances.
- Section 4.3: le traitement des mots inconnus et le filtrage des règles de grammaire sont décrits.
- Section 4.5 : nous décrivons finalement la transformation retenue en combinant la meilleure transformation de la section 4.2 et la meilleure méthode de filtrage de la section 4.3.

4.1 Modèles précédants

Dans cette section, on passe en revue les modèles statistiques les plus populaires de la littérature consacrée.

4.1.1 [Magerman, 1995]

Il a décrit l'analyseur syntaxique SPATTER (Statistical PATTErn Recognizer) qui est totalement différent des analyseurs syntaxiques basés sur les règles de grammaire. Cet analyseur statistique est basé sur la technique d'apprentissage d'arbres de décision qui construit l'arbre d'analyse syntaxique complet pour chaque phrase. Cette étude reporte un taux de précision supérieur à ceux reporté par d'autres auteurs à cette époque.

4.1.2 [Charniak, 1996]

C'est le premier à avoir publié les résultats de l'entraînement par fréquence relative (voir l'équation 2.1) de règles dérivées du PTB. Son travail a permis de montrer que cette méthode simple d'extraction des règles était une alternative bien plus intéressante à celle de découvrir automatiquement les règles à partir d'un corpus textuel (ou étiqueté) [Rajman, 1996]. Les modèles qui suivent ont été fortement influencés par cette étude et tentent d'introduire l'information lexicale dans le processus pour palier aux problèmes du modèle PCFG de base que nous avons mentionné dans le chapitre 2.

4.1.3 [Collins, 1996]

En constatant le rôle important de l'information lexicale dans l'analyse syntaxique, Collins a essayé de prendre en compte les informations lexicales en modélisant les relations mot de tête – modificateur (*head-modifier*) entre les paires de mots. L'analyseur syntaxique recherche l'arbre le plus probable en introduisant la décomposition suivante :

$$T_{meilleur} = \arg\max P(T \mid S) = \arg\max P(B, D \mid S) = \arg\max P(B \mid S) * P(D \mid B, S)$$

Où B est la phrase réduite et D est l'ensemble des dépendances entre les paires de mot de B. Une étape préliminaire à l'analyse syntaxique consiste à taggeur la phrase *S à* analyser; l'auteur utilise pour cela le taggeur de l'anglais distribué gratuitement sur Internet et décrit dans [Ratnaparkhi, 1996]. La phrase réduite est alors obtenue en supprimant les ponctuations et en réduisant tous les

syntagmes en *baseNPs (non recursive NP)*. L'analyseur va chercher l'arbre qui donne *P(D|B,S)* le plus probable. La figure 4.1 donne un exemple d'analyse et illustre les étapes intermédiaires qui sont mises en œuvre.

John Smith , the president of IBM announced his resignation yesterday.

taggeur: John/NNP Smith/NNP, the/DT president/NN of/IN IBM/NNP, announced/VBD his/PRP$ resignation/NN yesterday/NN.

*baseNPs : [John **Smith**] [the **president**] of [**IBM**] announced [his **resignation**] [**yesterday**]*

*(B)= <Smith,**NNP**> <president, **NN**> <of, **IN**><IBM, **NNP**><announced, **VBD**> <resignation, **NN**> <yesterday, **NN**>*

L'analyse syntaxique de la phrase et l'ensemble de dépendances approprié D

(D)={<Smith →announced, Smith→ president, of →president, of →IBM, resignation→announced, yesterday→announced}

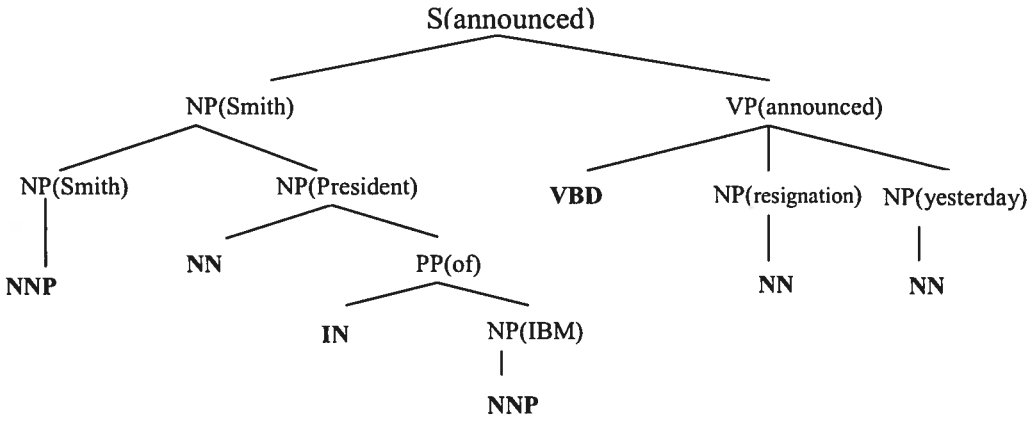


FIG. 4.1 Exemple d'analyse syntaxique de [Collins, 1996]

Ce modèle assez simple (entraînement des paramètres à partir d'un corpus de 40,000 phrases en moins de 15 minutes) montrait des performances comparables aux meilleurs analyseurs statistiques du moment.

4.1.4 [Charniak, 1997]

L'idée principale est de calculer la probabilité *p(h|hp,t,p)*, c'est-à-dire la probabilité du mot de tête *h* du non terminal *t* étant donné que *hp* est le mot de tête du non terminal *p* qui est parent de *t* (*p* →....*t*...).

Exemple : p(profits|rose,np,s) est la probabilité de np dont le mot de tête est 'profits' étant donné que 'rose' est le mot de tête de s.

Ensuite, l'analyseur syntaxique fonctionne comme suit : à partir de la phrase, on cherche l'ensemble de l'arbre d'analyse syntaxique en utilisant un modèle PCFG [Charniak, 1996] et l'algorithme CHART avec le filtrage proposé dans [Caraballo et Charniak, 1996]. Puis, en utilisant la probabilité *p(h|hp,t,p)*, l'analyseur choisit le meilleur arbre d'analyse. Les résultats obtenus (*PTB version II*) sont meilleurs que [Magerman, 1995] et [Collins, 1996]. Le taggeur est inclu dans l'analyse syntaxique.

4.1.5 [Collins, 1997]

Dans cet article, il propose un nouveau modèle statistique génératif de grammaire lexicalisée hors contexte. Un modèle génératif considère que maximaliser *P(T , S)* est équivalent à maximaliser *P(T | S)*, c'est à dire

$$T_{meilleur} = \arg\max P(T \mid S) = \arg\max_{T} \frac{P(T,S)}{P(S)} = \arg\max_{T} P(T,S)$$

Il propose trois nouveaux modèles; modèle *1* est une extension de [Collins, 1996] et utilise une PCFG lexicalisée (en associant un mot de tête à chaque syntagme); modèle *2* étend l'analyseur syntaxique du modèle *1* pour prendre en compte la distinction de *complement/adjunct* en ajoutant les probabilités sur les frames de subcatégorisation pour les mots de tête; modèle *3*, dans le but de permettre à l'analyse syntaxique d'extraire la structure *predicate-argument* (chapitre 3) propose un traitement probabiliste pour le problème du *wh-movement* (chapitre 3) qui est un facteur important dans l'extraction de structure prédicat-argument.

Pour tous les trois modèles, le taggeur est compris dans le modèle. Pour les mots inconnus, le taggeur de Ratnaparkhi (1996) est utilisé pour déterminer leur étiquette syntaxique. Avec le modèle *3*, les résultats obtenus sont meilleurs que ceux de [Collins, 1996], l'amélioration reste cependant minime pour la complexité introduite (environ 2% de gain).

4.1.6 [Goodman, 1997]

Comme dans d'autres articles [Collins, 1996, 1997], [Charniak, 1996], *Goodman* tente de prendre en compte l'information du contexte pour améliorer la

performance de l'analyseur syntaxique. Dans cet article, l'auteur propose une nouvelle forme de grammaire appelée grammaires de traits probabilistes (*Probabilistic Feature Grammars* ou PFG). Les PFG ajoutent des traits au non terminal de chaque nœud comme le mot de tête ou le nombre de constituants grammaticaux. Les PFG sont entraînées sur le PTB et les PFG peuvent être analysées grâce à la programmation dynamique. Par exemple, l'analyse syntaxique de la phrase *'The man dies'*, avec la grammaire (*S* → *NP VP ; NP* → *the man ; VP*→ *dies*) donne un arbre d'analyse suivant (*S (NP the man) (VP dies)*). À partir de cet arbre, on ne peut que dire que la phrase comprend seulement *NP* (*noun phrase*) et *VP* (*verb phrase*), mais on ne peut pas capturer certaines restrictions telles que : *NP* et *VP* doivent être accordés en genre et en nombre. La nature du verbe rend certains mots du group sujet plus probables que d'autres (*'man'* est plus probable que *'spaghetti'*). On peut capturer ces informations en utilisant les PFG avec des particularités telles que : la catégorie, le nombre (singulier ou pluriel) et le mot de tête. La règle des PFG est décrite comme suit:

(S, singular, die) →(NP, singular, man) (VP, singular, dies)

Avec le taggeur inclu dans le modèle, les résultats se situent entre ceux de [Collins, 1996 1997] et [Magerman, 1995]; mais en utilisant le taggeur de Ratnaparkhi (1996) comme le taggeur, ce modèle donne des résultats meilleurs que ceux de [Collins, 1996].

4.1.7 [Collins, 1999]

Cet article propose une extension aux modèles proposés précédemment par l'auteur en ajoutant les traitements spéciaux pour les *baseNPs*, le traitement de la coordination, et le traitement de la ponctuation. Les résultats obtenus par ce nouveau modèle étaient les meilleurs résultats du moment.

4.1.8 Résultats

À part le modèle proposé par Magerman (1995) qui n'utilise pas les PCFG, tous les modèles passés en revue utilisent les modifications des PCFG dans le but d'ajouter statistiquement des informations que les PCFG ne capturent pas. [Collins, 1996, 1997, 1999] utilisent [Ratnaparkhi, 1996] pour étiqueter les mots inconnus; les analyseurs syntaxiques [Magerman, 1995], [Charniak, 1996, 1997], [Goodman, 1997] comprennent quant à eux leur propre taggeur. Tous les modèles que nous venons de présenter utilisent les corpus d'entraînement et de test que

nous avons décrit au chapitre 3. PARSEVAL est utilisé pour évaluer la performance des modèles, les résultats de ces modèles sont présentés ci-dessous:

Modèle	pour la longueur <=40			pour la longueur <=100		
	Rappel	Précision	Croisement	Rappel	Précision	Croisement
[Charniak, 1996]	80.40	78.80	n/a	n/a	n/a	n/a
[Magerman, 1995]	84.60	84.90	1.26	84.00	84.30	1.46
[Collins, 1996]	85.80	86.30	1.14	85.30	85.70	1.32
[Charniak, 1997]	87.50	87.40	1.00	n/a	n/a	n/a
[Collins, 1997]	88.10	88.60	0.91	87.50	88.10	1.07
[Goodman, 1997]	84.80	85.30	1.21	n/a	n/a	n/a
[Collins, 1999]	**88.60**	**88.70**	**0.90**	**88.00**	**88.30**	**1.05**

Table 4.1 Résultats des modèles décrits

C'est le modèle de Collins (1999) qui obtient les meilleurs résultats. C'est également le modèle qui fait une plus large part aux informations linguistiques.

Dans la section suivante, on va parler d'une suite d'expériences qu'on a effectué en utilisant les transformations simples appliquées sur le PTB avant d'utiliser le modèle PCFG de base.

4.2 Transformations

Dans cette section, nous considérons différentes transformations de PTB avant d'utiliser le modèle PCFG. Dans PTB, il y a des informations associées à chaque symbole non terminal dédiées à l'extraction de la structure prédicat-argument (chapitre 3), il faut supprimer ces informations indésirables. C'est pourquoi, dans la première transformation, nous traitons seulement sur les symboles non terminaux avant d'extraire des règles de grammaire. Le meilleur traitement est considéré comme le modèle PCFG de base. Les transformations qui suivent essaient de prendre en considération les informations lexicales et contextuelles manquant dans le modèle PCFG de base. Afin de comparer les résultats avec les modèles précédants, nous utilisons les corpus d'entraînement et de test (décrit dans le chapitre 3) dans toutes ces expériences. Le diagramme ci-dessous illustre toutes nos transformations. L'étape de retransformation sert à éliminer les informations associées aux syntagmes (si elles existent) avant d'utiliser les mesures PARSEVAL.

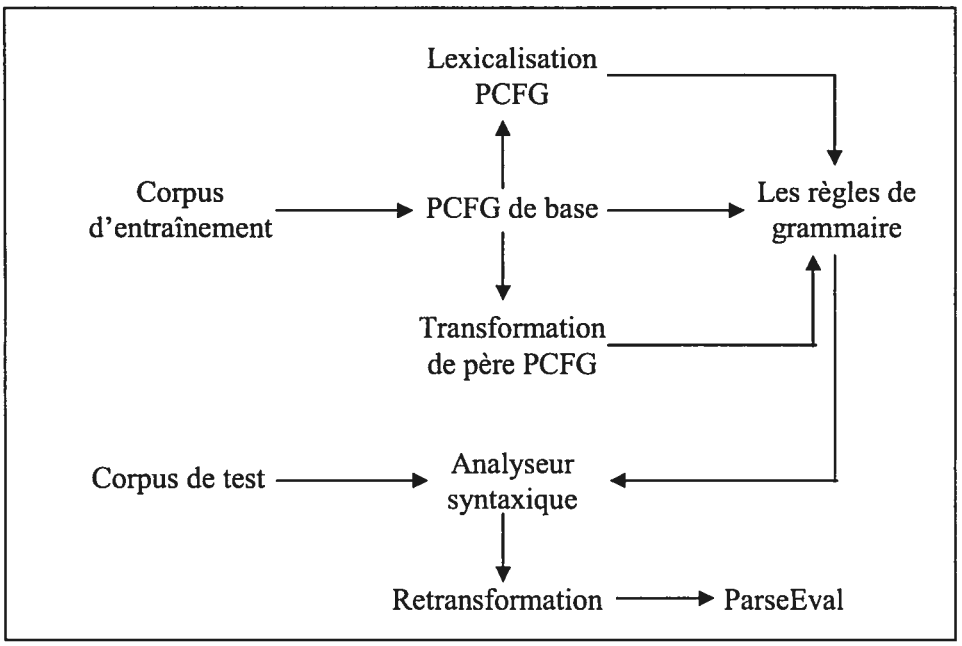


FIG. 4.2 Diagramme d'application des transformations

Pour présenter clairement les résultats de différentes transformations, nous utilisons les notations suivantes :

N1 : Nombre de phrases que l'analyseur syntaxique arrive à reconnaître avec le même symbole non terminal à la racine de celui de l'arbre de référence.

Il arrive effectivement que l'arbre le plus probable trouvé par l'analyseur ne soit pas un arbre dominé par le même symbole non terminal que celui de la référence.

N2 : Nombre de phrases que l'analyseur syntaxique arrive à reconnaître avec un symbole à la racine différent de celui de la référence.

Exemple: dans la phrase testée, on a le symbole S comme racine mais l'analyseur syntaxique donne le symbole NP qui domine toute la phrase.

N3 : Nombre de phrase où l'analyseur syntaxique n'arrive pas à trouver un symbole non terminal qui domine toute la phrase (analyse partielle).

N4 : Nombre de phrases que l'analyseur syntaxique n'arrive pas à reconnaître à cause de mots qui n'existent pas dans le corpus d'entraînement (les mots inconnus).

Dans tous les résultats présentés, pour calculer *Rappel, Précision et Croisement*, on prend en compte seulement *N1* et *N2,* c'est-à-dire les phrases pour lesquelles au moins un arbre complet a été trouvé. Bien que ces résultats ne soient

pas comparables aux autres études, ils servent le besoin de comparaison entre les différentes transformations que nous effectuons.

4.2.1 Modèle PCFG de base

Pour la première version du PTB que [Charniak 1996] a utilisé, il n'y a pas de symboles non terminaux tels que UCP, LST, QP, CONJP, NX, PRT, PRN, RRC, INTJ, PRT|ADVP, FRAG. La table ci-dessous contient les 72 symboles non terminaux dans la portion du WSJ du PTB version II.

UCP VP CONJP LST SQ QP WHPP PP SBARQ WHADVP ADVP WHNP NX PRT PRN NP RRC INTJ SINV PRT|ADVP FRAG SBAR WHADJP ADJP X S NAC RBS RBR WDT VBZ WP VBP VBN VBG VBD VB UH TO RP RB SYM PRP$ PRP NN JJS JJR NNS NNP MD -LRB- LS POS JJ IN FW '' EX NNPS WRB -RRB- DT CD CC WP$ PDT `` : . , $ #

Table 4.2 Les 72 symboles non terminaux du PTB version II

Pour sélectionner un modèle PCFG de base efficace, on a testé trois méthodes différentes d'extraction des règles à partir d'un arbre du PTB. Tout d'abord, la première méthode élimine toutes les informations indésirables associées aux symboles non terminaux et nous supprimons également tous les symboles inutiles comme l'index *'-1'*, le symbole –NONE-, le terminal *-1 (par exemple *NP-SBJ-1* devient *NP-SBJ*). Ensuite, la deuxième méthode supprime également les étiquettes fonctionnelles (par exemple *NP-SBJ* devient *NP*); la troisième garde enfin seulement les règles grammaires contenant les symboles non terminaux utilisés par [Charniak, 1996] (par exemple *QP*). Nous illustrons ces trois méthodes sur l'exemple de la table 4.3.

```
( (S
    (NP-SBJ-1
      (NP (NNS Assets) )
      (PP (IN of)
        (NP (DT the) (CD 400) (JJ taxable) (NNS funds))))
    (VP (VBD grew)
      (PP-EXT (IN by)
        (NP
          (QP ($ $) (CD 1.5) (CD billion) )
          (-NONE- *-1) ))
      (PP-TMP (IN during)
        (NP (DT the) (JJS latest) (NN week) ))
      (, ,)
      (PP-DIR (TO to)
        (NP
          (QP ($ $) (CD 352.7) (CD billion) )
          (-NONE- *U*) )))
    (. .) ))
```

Table 4.3 Phrase exemple pour nos trois méthodes d'extraction des règles

Méthode 1:

On garde tous les non terminaux et on enlève tous les indices qui y sont attachés. Nous obtenons donc pour notre exemple les règles suivantes:

Règles internes	Règles unitaires	Règles lexicales
NP → DT JJS NN	NP → NNS	NN → week
NP → DT CD JJ NNS	NP → QP	IN → during
NP-SBJ → NP PP		IN → by
PP → IN NP		IN → of
PP-EXT → IN NP		CD → 352.7
PP-TMP → IN NP		CD → billion
PP-DIR → TO NP		CD → 1.5
QP → $ CD CD		CD → 400
S → NP-SBJ VP .		JJS → latest
VP → VBD PP-EXT PP-TMP , PP-DIR		JJ → taxable
		TO → to
		DT → the
		NNS → funds
		NNS → Assets
		VBD → grew
		. → .
		, → ,
		$ → $

Table 4.4 Ensemble des règles pour la méthode 1

Méthode 2:

On supprime toutes les étiquettes fonctionnelles comme – TMP, -SBJ, -DIR etc (voir la table 5.3 pour l'ensemble des étiquettes fonctionnelles) à partir de l'ensemble de règles de la méthode 1. Dans cet exemple, les règles unitaires et les règles lexicales sont les mêmes que dans la méthode 1 mais nous obtenons les règles internes suivantes (les règles en gras sont changées):

NP → DT JJS NN
NP → DT CD JJ NNS
NP → NP PP (au lieu de la règle NP-SBJ→NP PP)
PP → IN NP (au lieu de la règle PP-EXT→IN NP
et PP-TMP→IN NP)
PP → TO NP (au lieu de la règle PP-DIR→TO NP)
QP → $ CD CD
S→NP VP . (au lieu de la règle S → NP-SBJ VP.)
VP→VBD PP PP , PP (au lieu de la règle
VP → VBD PP-EXT PP-TMP , PP-DIR)

Table 4.5 Règles internes pour la méthode 2

Méthode 3 :

À partir de l'ensemble de règles obtenu par la méthode 2, nous n'utilisons que le sous-ensemble de non terminaux utilisé par Charniak (1996). Avec

l'exemple précédant, les règles lexicales sont les mêmes que dans la méthode 2 mais les règles internes et les règles unitaires sont différentes.

Règles internes	Règles unitaires
NP → DT JJS NN	NP → NNS
NP → DT CD JJ NNS	NP → QP est éliminé
NP → NP PP	
PP → IN NP	
PP → TO NP	
QP → $ CD CD est éliminé	
S → NP VP .	
VP → VBD PP PP , PP	

Table 4.6 Ensemble des règles pour la méthode 3

Pour calculer les probabilités des règles, on utilise la formule 2.1 décrite dans le chapitre *2.*

Exemple: la probabilité de la règle CD → 400 est 1/4=0.25 pour notre exemple.

Résultats

Dans ce modèle, on a énuméré trois méthodes différentes pour extraire les règles de grammaire probabilistes. Voici les résultats

Méthode	Règles internes	Règles unitaires	Règles lexicales
(1)	25140	530	51659
(2)	**14736**	**221**	**51659**
(3)	11866	148	51659

Table 4.7 Nombre des règles du modèle PCFG de base

Méthode	N1	N2	N3	N4	Rappel	Précision	Croisement
pour la longueur <=12							
(1)	328	0	0	74	87.48	88.61	0.0640
(2)	**328**	**0**	**0**	**74**	**89.19**	**90.62**	**0.0731**
(3)	321	7	0	74	88.69	90.55	0.0823
pour la longueur <=20							
(1)	763	0	0	271	84.72	86.34	0.2726
(2)	**763**	**0**	**0**	**271**	**86.81**	**88.87**	**0.3866**
(3)	754	9	0	271	86.20	88.56	0.3805
pour la longueur <=40							
(1)	1442	0	0	803	82.20	83.94	0.8044
(2)	**1442**	**0**	**0**	**803**	**84.92**	**87.28**	**0.9334**
(3)	1429	13	0	803	84.14	86.75	0.9632

Table 4.8 Résultats du modèle PCFG de base

À travers les résultats présentés au-dessus, on peut constater que la méthode *(2)* est la meilleure parmi les autres. Le nombre de règles de la méthode *(1)* est

environ deux fois plus que celui de la méthode *(2) (25670 vs 14957),* mais ses résultats sont moins bons que *(2)* car les étiquettes fonctionnelles *(-TMP, -ADV, -EXT* etc*)* sont ajoutées afin de favoriser l'extraction du prédicat-argument de PTB (version *II)*. Cet ajout distingue les syntagmes syntaxiques de même fonction dans la phrase en plusieurs différents syntagmes. Par exemple dans la phrase (voir la t*able 4.3)*, on a deux règles *PP-EXT→IN* et *PP-TMP→IN* selon *(1)* au lieu d'une seule règle dans la méthode *(2) PP→IN*. Cela augmente le nombre de règles et affecte la distribution de probabilité; la performance de l'analyseur syntaxique est donc affectée. La meilleure performance de *(2)* comparée avec *(3)* prouve que l'annotation du PTB (version *II*) est meilleure que celle de la version I (les évaluations théoriques sur les représentations différentes d'arbre sont présentées dans [Johnson, 1998]). Dans *(2),* le nombre de phrases sans analyse à cause des mots inconnus est de *803* phrases. Ce nombre est élevé (56% des phrases de test), mais la moitié des mots inconnus sont constitués de données chiffrées et de mots composés. Nous proposons un traitement de ce problème dans la section 4.4.

La méthode donnant les meilleurs résultats a été utilisée comme modèle PCFG de référence de cette étude. Il s'agit du modèle PCFG obtenu avec la méthode 2.

4.2.2 Lexicalisation de PCFG

Comme nous l'avons vu, une des faiblesses du modèle PCFG est de sous-exploiter les informations lexicales. En fait, la sortie d'un analyseur syntaxique PCFG est influencée seulement par la séquence de tags (*POS*). Dans le but de prendre en considération les informations lexicales comme dans [Collins, 1996, 1997, 1999], ou [Charniak, 1997], nous avons étudié différentes façons de lexicaliser le PTB. Par lexicaliser nous entendons: associer un mot de tête à chaque non terminal. Dans le cas des règles lexicales (*NP→ cat*), le mot *cat* est le mot de tête du non terminal *(NP)*. Dans le cas d'une règle unitaire *(A→ B),* le mot de tête associé au non terminal de gauche *(A)* est le mot de tête associé au non terminal de droite *(B)*. Pour les règles internes, on distingue deux situations. Les règles dont la partie gauche est un *NP,* auquel cas les règles suivantes sont appliquées :

- si le dernier non terminal de la partie droite de la règle est *POS (Possessive ending par exemple 's)*, retourner le mot de tête de *POS*.
- sinon chercher de la droite vers la gauche dans la partie droite de la règle le premier non terminal qui appartient à l'ensemble *NN, NNP, NNPS, NNS, POS* ou *JJR*. C'est lui qui donnera la tête à associer au non terminal, de la partie droite.
- sinon chercher de gauche à droite dans la partie droite de la règle le premier non terminal qui est *NP*. C'est lui qui transmet alors la tête;

Les règles dont la partie gauche est un non terminal autre que *NP*, auquel cas les règles de la *table 4.9* sont utilisées. Dans cette table, le symbole de la colonne *'Parent non terminal'* est le symbole non terminal de la partie gauche de la règle considérée, le mot de tête de ce symbole est le mot de tête du premier symbole non terminal dans la colonne *'Priority List'* quand on le cherche dans la direction indiquée dans la colonne *'Direction'* (*'Left'* signifie qu'on cherche de la gauche vers la droite, *'Right'* signifie qu'on cherche de la droite vers la gauche).

Prenons un exemple d'arbre entièrement lexicalisé pour illustrer le fonctionnement de ces règles (voir la figure 4.3). Dans cet arbre, chaque symbole non terminal est associé à son mot de tête. Nous cherchons à identifier le mot de tête à associer au VP. La règle concernée est $VP \rightarrow VBD\ NP$ et c'est donc la ligne 18 de la table 4.9 qui nous indique la marche à suivre. On cherche de la gauche vers la droite (car la colonne *'Direction'* est *'Left'*) le premier non terminal parmi ceux indiqués dans la colonne *'Priority List'* (*TO, VBD, VBN, MD, VBZ, VBG, VBP, VP, ADJP, NN, NNS, NP)*. Dans ce cas, il s'agit de *VBD*. C'est donc le mot de tête associé à ce non terminal qui sera la tête du non terminal *VP*. Il s'agit ici du mot *'bought'*.

	Parent non terminal	Direction	Priority List
1	ADJP	Left	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
2	ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
3	CONJP	Right	CC RB IN
4	FRAG	Right	
5	INTJ	Left	
6	LST	Right	LS :
7	NAC	Left	NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
8	PP	Right	IN TO VBG VBN RP FW
9	PRN	Left	
10	PRT	Right	RP
11	QP	Left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
12	S	Left	TO IN VP S SBAR ADJP UCP NP
13	RRC	Right	VP NP ADVP ADJP UCP NP
14	SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
15	SINV	Left	VBZ VBD VBP VB MD VP S SINV ADJP NP
16	SQ	Left	VBZ VBD VBP VB MD VP SQ
17	UCP	Right	
18	VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP
19	WHADJP	Left	CC WRB JJ ADJP
20	WHADVP	Right	CC WRB
21	WHNP	Left	WDT WP WP$ WHADJP WHPP WHNP
22	WHPP	Right	IN TO FW

Table 4.9 Règles de lexicalisation

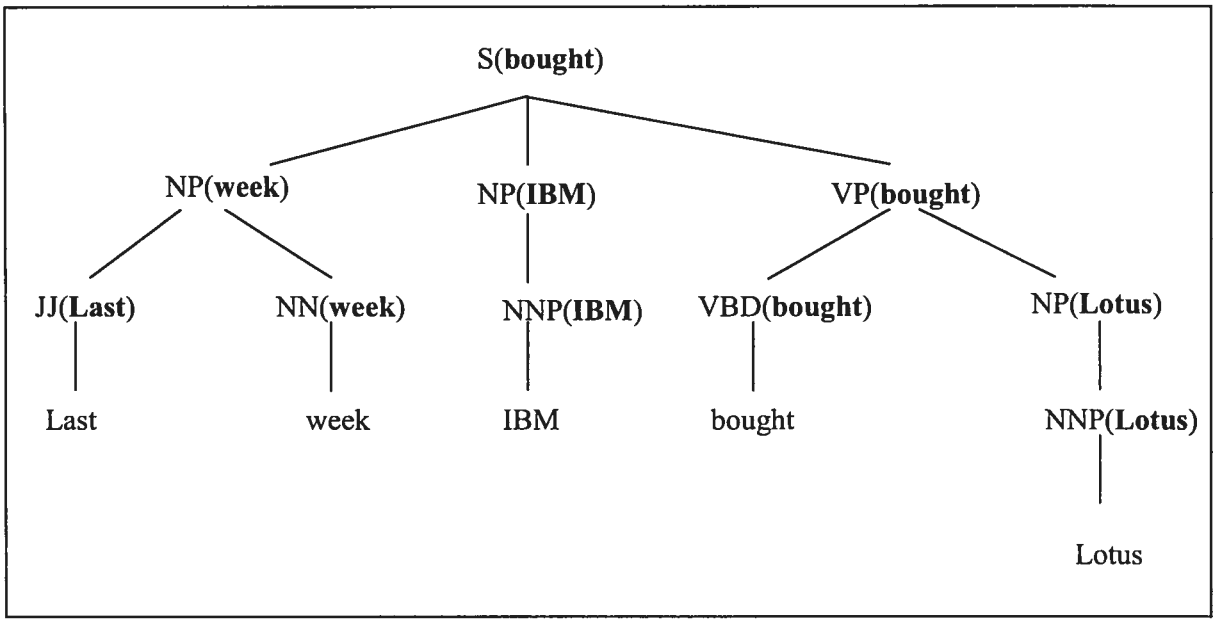


FIG. 4.3 Arbre totalement lexicalisé

À partir de cet arbre, on peut extraire l'ensemble de règles lexicalisées suivantes.

```
S(bought)   →  NP(week) NP(IBM) VP(bought)
NP(week)    →  JJ(Last) NP(week)
NP(IBM)     →  NNP(IBM)
VP(bought)  →  VBD(bought) NP(Lotus)
NP(Lotus)   →  NNP(Lotus)
JJ(Last)    →  Last
NN(week)    →  week
NNP(IBM)    →  IBM
VBD(bought) →  bought
NNP(Lotus)  →  Lotus
```

Table 4.10 Ensemble des règles lexicalisées obtenues à partir de l'arbre lexicalisé de la figure 4.2

Il est également possible de lexicaliser partiellement un arbre. Nous reportons en *FIG. 4.3* l'arbre lexicalisé uniquement pour les symboles *S, NP et VP.*

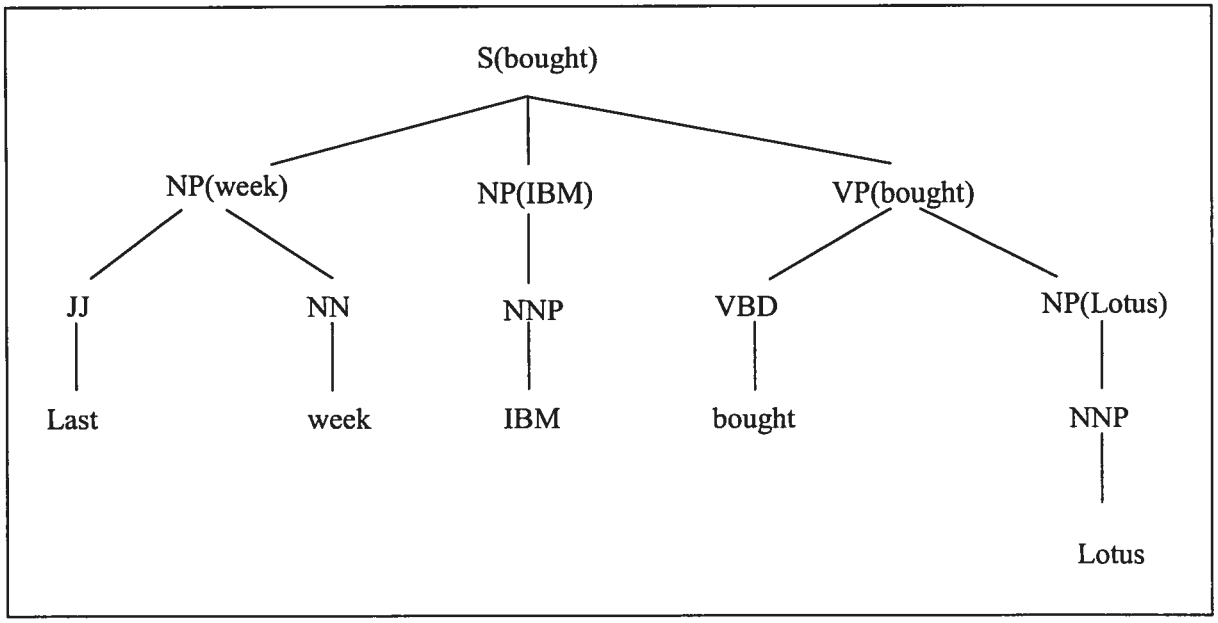


FIG. 4.4 Arbre partiellement lexicalisé pour les S, NP, VP

À partir de cet arbre, on obtient l'ensemble de règles suivantes:

```
S(bought)    →  NP(week) NP(IBM) P(bought)
NP(week)     →  JJ NP(week)
NP(IBM)      →  NNP
VP(bought)   →  VBD NP(Lotus)
NP(Lotus)    →  NNP
JJ           →  Last
NN           →  week
NNP          →  IBM
VBD          →  bought
NNP          →  Lotus
```

Table 4.11 Ensemble des règles partiellement lexicalisées de notre exemple

Pour calculer les probabilités des règles lexicalisées, on utilise la formule 2.1 décrite dans le chapitre *2.*

Exemple: pour calculer la probabilité de la règle

`S(bought)  →  NP(week) NP(IBM) VP(bought)`

$$P(\texttt{NP(week) NP(IBM) VP(bought)} \mid \texttt{S(bought)}) = \frac{Count(S(bought) \rightarrow NP(week)NP(IBM)VP(bought))}{Count(S(bought))}$$

Il est donc bien évident que lexicaliser toutes nos règles a une influence sur le modèle PCFG résultant. En particulier, nous augmentons de manière importante le nombre de règles et augmentons donc les chances de ne voir ces règles qu'un nombre limité de fois, rendant donc notre estimée de probabilité bruitée.

Résultats

Dans cette expérience, tout d'abord, nous appliquons le modèle PCFG de base sur le corpus du PTB, puis nous lexicalisons les arbres dans PTB comme décrit ci-dessus avant l'extraction des règles de grammaires. Pour la première expérience, nous appliquons la lexicalisation entière (pour tous les symboles non terminaux); nous comparons cette approche à la lexicalisation partielle avec VP, NP, PP respectivement.

Les résultats avec la lexicalisation entière

Nous présentons les résultats en prenant comme référence le modèle PCFG de base que nous avons décrit. Le nombre de règles obtenu en lexicalisant complètement les arbres très important (voir table 4.12). Le nombre de règles a une influence directe sur les temps de calcul, aussi ne donnons les résultats que pour les phrases de notre corpus de test d'un longueur inférieure ou égale à 20 mots.

Règles internes	Règles unitaires	Règles lexicales
330077	18935	51659
14736	221	51659

Table 4.12 Nombre des règles du modèle PCFG entièrement lexicalisé

Longueur	N1	N2	N3	N4	Rappel	Précision	Croisement
<=12	38	1	289	74	95.38	95.03	0.0000
PCFG	328	0	0	74	88.19	90.62	0.0731
<=20	54	1	708	271	95.17	95.16	0.0740
PCFG	763	0	0	271	86.81	88.87	0.3866

Table 4.13 Résultats du modèle PCFG entièrement lexicalisé

Bien que les résultats de *Rappel* et *Précision* soient élevés et le *Croisement* est bas (calculés pour *N1* et *N2* seulement), cette transformation a le nombre de phrase non analysées (*N3)* le plus haut (*708* phrases). Nous sommes en présence d'un problème de sur-entraînement en présence d'un corpus sur-spécialisé (sparseness problem).

En effet, la lexicalisation entière rend les règles très spécifiques. Lorsque l'analyse syntaxique propose un arbre à une phrase de test, cet arbre est presque exactement celui de la référence (dans *39* phrases reconnues *(N1 + N2)* pour la longueur <=12, il y a *25* arbres d'analyse syntaxique dont la précision et le rappel sont *1* et le croisement est *0*, pour la longueur <=*20*, on en a *30* parmi *55*). La lexicalisation entièrement donc clairement pas une solution intéressante.

Les résultats avec la lexicalisation partielle

Dans cette expérience, après avoir lexicalisé entièrement les arbres du PTB, on garde seulement les mots de tête associés avec les non terminaux *VP, NP, PP*. Les résultats qu'on obtient sont présentés ci-dessous:

VP

Règles internes	Règles unitaires	Règles lexicales
68340	5204	51659
14736	221	51659

Table 4.14 Nombre des règles du modèle PCFG lexicalisé avec VP

Longueur	N1	N2	N3	N4	Rappel	Précision	Croisement
<=12	328	0	0	74	90.23	91.07	0.0579
PCFG	328	0	0	74	88.19	90.62	0.0731
<=20	763	0	0	271	88.35	89.83	0.2922
PCFG	763	0	0	271	86.81	88.87	0.3866
<=40	1442	0	0	803	86.24	88.10	0.8037
PCFG	1442	0	0	803	84.92	87.28	0.9334

Table 4.15 Résultats du modèle PCFG lexicalisé avec VP

Comparé avec les résultats du modèle PCFG de base, on peut trouver que la lexicalisation PCFG avec VP augmente un peu la performance de l'analyseur syntaxique (*1.55% Rappel*, 0.94% *Précision*) mais le nombre de règles est presque cinq fois supérieur. Ce résultat est assez intuitif. La nature du verbe conditionne en pratique les règles à appliquer ce que notre modèle capture. Nous illustrons ceci sur la phrase exemple : *'These stocks eventually reopened'*

(S (NP (DT These) (NNS stocks) (ADVP (RB eventually) (VP (VBN reopened) (. .)) Rappel :90.9% Précision:90.9%	(S (NP (DT These) (NNS stocks) (ADVP (RB eventually) (VP (VBD reopened) (. .)) Rappel :100% Précision :100%
PCFG de base	lexicalisation PCFG avec VP

Table 4.16 Arbres les plus probables proposés par le modèle PCFG de base et le modèle partiellement lexicalisé sur VP

Avec le modèle PCFG de base, l'arbre trouvé par l'analyseur syntaxique est faux à cause de la distribution de probabilité des règles : *VP→VBN [1704/14602], VBN→reopened [6/20024], VP→VBD [2655/14602], VBD→reopened [2/29889]* (*VBD* signifie '*past tense*', *VBN* signifie '*past*

participle'). Avec la lexicalisation sur VP, ces règles VP(verbe)→VBD, VP(verbe)→VBD reçoivent une probabilité conditionnelle au verbe, l'analyse bénéficie de cette information additionnelle (voir table 4.16). En fait, la règle *VP(verbe)→VBD* reçoit une probabilité de *1877/5995* alors que la règle *VP(verbe)→VBN* reçoit une probabilité de *4/5995*. Ce qui veut dire que le modèle préférera la forme '*past tense*' à la forme '*verb phrase*' en présence du seule verbe '*reopened*'.

En comparant la lexicalisation entière PCFG pour les phrases dont la longueur est inférieure à 20 (<=20), il y a *704* phrase sans analyse (*N3*) parmi *1024* phrases. On voit donc qu'avec cette méthode, le nombre de phrase sans analyse (*N4)* est la seule cause d'erreur (*N3 = 0*). Nous préférons donc la lexicalisation partielle sur VP à la lexicalisation complète.

PP

Règles internes	Règles unitaires	Règles lexicales
23051	308	51659
14736	221	51659

Table 4.17 Nombre des règles du modèle PCFG lexicalisé avec PP

Longueur	N1	N2	N3	N4	Rappel	Précision	Croisement
<=12	324	4	0	74	89.30	90.39	0.0701
PCFG	328	0	0	74	88.19	90.62	0.0731
<=20	763	0	0	271	87.71	88.81	0.3420
PCFG	763	0	0	271	86.81	88.87	0.3866
<=40	1442	0	0	803	85.19	87.16	0.8932
PCFG	1442	0	0	803	84.92	87.28	0.9334

Table 4.18 Résultats du modèle PCFG lexicalisé avec PP

Avec la lexicalisation partielle sur PP, le nombre de règles augment de 56% mais la performance de l'analyse syntaxique est presque la même que le modèle PCFG de base. En effet, *81%* des règles impliquant le non terminal *NP* sont des règles *PP→IN NP (préposition IN (with, in, as, for after etc) suivie d'un groupe nominal NP),* le mot de tête du groupe prépositionnel *PP* est donné par *IN*. Ce modèle ne capture donc pas vraiment d'information supplémentaire dans ce cas. Ceci explique que les deux modèles donnent des performances comparables

NP

Règles internes	Règles unitaires	Règles lexicales
160268	9939	51659
14736	221	51659

Table 4.19 Nombre des règles du modèle PCFG lexicalisé avec NP

Longueur	N1	N2	N3	N4	Rappel	Précision	Croisement
<=12	328	0	0	74	88.78	90.02	0.0518
PCFG	328	0	0	74	88.19	90.62	0.0731
<=20	763	0	0	271	86.06	88.12	0.3145
PCFG	763	0	0	271	86.81	88.87	0.3866
<=40	1442	0	0	803	83.89	86.25	1.0215
PCFG	1442	0	0	803	84.92	87.28	0.9334

Table 4.20 Résultats du modèle PCFG lexicalisé avec NP

Avec lexicalisation partielle sur NP, le nombre de règles augmente de presque 12 fois par rapport au modèle PCFG de référence. La performance de l'analyse résultant est cependant moins bonne. Nous expliquons cela par l'exemple : on a la phrase à tester '*Third-quarter profits from gasoline were weaker.*'

(S **(NP** **(NP** **(JJ Third-quarter)** **(NNS profits))** **(PP** **(IN from)** **(NP** **(NN gasoline))))** (VP (VBD were) (ADJP (JJR weaker))) (. .)) Rappel : 100% Précision :100%	(S **(NP** **(JJ Third-quarter)** **(NNS profits))** **(NP** **(IN from)** **(NN gasoline))** (VP (VBD were) (ADJP (JJR weaker))) (. .)) Rappel : 78.57% Précision :91.66%
PCFG de base	Lexicalisation PCFG avec NP

Table 4.21 Comparaison des deux analyses proposées par le modèle PCFG de base et le modèle partiellement lexicalisé sur NP

Avec le modèle PCFG de base, l'analyseur syntaxique arrive à donner un bon arbre mais avec le modèle lexicalisation partielle sur NP, l'arbre trouvé n'est pas le bon. Dans le modèle de base, la règle $NP \rightarrow IN\ NN$ a une basse probabilité de *70/313160.* La lexicalisation partielle a pour effet d'augmenter la probabilité de cette règle (le mot de tête de *NP* est celui de *NN*) par conséquent, les structures qui apparaissent très rarement (comme $S \rightarrow NP\ NP\ VP\ .$ *(132/95404)* comparé à $S \rightarrow NP\ VP\ .$*(17703/95404))* ont la chance d'obtenir une probabilité élevée.

À partir des résultats de lexicalisation partielle sur *VP, NP et PP,* on peut observer donc que la lexicalisation partielle PCFG sur *VP* donne les résultats les plus intéressants (une légère augmentation de *1.55%* en *Précision* et de *0.94%* en *Rappel* et ainsi qu'une réduction de *4.30%* en *Croisement* comparé avec le modèle PCFG de base). Ce modèle entraîne cependant une augmentation conséquente du

nombre de règles (*68340 vs 14736)* et ralenti donc de manière notable les temps d'analyse. C'est donc une question de compromis qualité/temps de réponse.

Le but de la lexicalisation d'un modèle PCFG est de prendre en considération les informations lexicales afin d'augmenter la performance de l'analyse syntaxique. L'approche que nous venons de présenter est une approche simple qui permet l'ajout d'informations lexicales aux PCFG. Avec cette approche, on peut trouver qu'on pourrait augmenter un peu la performance de l'analyse syntaxique au détriment de la rapidité de l'analyseur résultant.

4.2.3 P-transformation de PCFG

Nous avons testé plusieurs transformations contextuelles, mais seule la transformation que nous décrivons maintenant s'est avérée intéressante. Nous appelons cette transformation la P-transformation. Elle consiste à ajouter à chaque symbole non terminal X le père de X, cette information distingue différents contextes dans lesquels le symbole non terminal X apparaît (dans la section du résultat, nous en parlons d'une façon plus détaillée), nous examinons l'exemple suivant pour justifier ce type de transformation.

```
Arbre d'origine
( (S
    (NP (NNP Mr.) (NNP Vinken) )
    (VP (VBZ is)
      (NP
        (NP (NN chairman) )
        (PP (IN of)
          (NP
            (NP (NNP Elsevier) (NNP N.V.) )
            (, ,)
            (NP (DT the) (NNP Dutch) (VBG publishing) (NN group) )))))
    (. .) ))

Après la P-transformation
( (S
  (NP__S (NNP__NP  Mr.)(NNP__NP  Vinken))
  (VP__S (VBZ__VP  is)
       (NP__VP
         (NP__NP (NN__NP  chairman))
         (PP__NP (IN__PP  of)
           (NP__PP
             (NP__NP (NNP__NP  Elsevier)(NNP__NP  N.V.))
             (,__NP  ,)
             (NP__NP (DT__NP  the)(NNP__NP  Dutch)(VBG__NP  publishing)
                     (NN__NP  group))))))
  (.__S  .)))
```

FIG. 4.5 Arbre avant et après la P-transformation

Après avoir transformé les arbres du PTB, on obtient les règles de grammaire comme d'habitude (la formule 2.1). Cette transformation est pour la première fois présentée dans [Johnson, 1998], dans son article, en supposant que le corpus comprend seulement deux structures d'arbre importantes, il a prouvé théoriquement que cette transformation augmente la performance du modèle PCFG. Il a entraîné un modèle sur les sections 2-21 et a testé ses performances sur la section 22; il a obtenu aussi une précision de 80% et un rappel : 79.20% (une augmentation d'environ 8% pour la précision et le rappel).

Résultats

Les résultats obtenus par cette P-transformation sont de loin supérieurs à ceux obtenus avec un modèle PCFG de base.

Règles internes	Règles unitaires	Règles lexicales
21940	**745**	**59936**
14736	221	51659

Table 4.22 Nombre des règles de transformation de père PCFG

Longueur	N1	N2	N3	N4	Rappel	Précision	Croisement
<=12	**328**	**0**	**0**	**74**	**93.12**	**93.04**	**0.0457**
PCFG	328	0	0	74	88.19	90.62	0.0731
<=20	**763**	**0**	**0**	**271**	**91.64**	**91.97**	**0.1926**
PCFG	763	0	0	271	86.81	88.87	0.3866
<=40	**1441**	**0**	**1**	**803**	**90.29**	**90.69**	**0.5988**
PCFG	1442	0	0	803	84.92	87.28	0.9334

Table 4.23 Résultats de transformation de père PCFG

Avec cette transformation, on observe qu'il y a une augmentation importante en *Rappel* (*6.32%)* et en *Précision (3.90%)* ainsi qu'une réduction notable du taux de croisement (*35.84%)*. Il est intéressant de noter que les erreurs d'analyse sont maintenant essentiellement dûes aux mots inconnus, ce qui n'était pas le cas des grammaires obtenues par lexicalisation.

Cette transformation montre également une augmentation raisonnable du nombre de règles (*21940 vs 14736*). Les temps d'analyse sont donc raisonnablement plus longs. Nous illustrons maintenant la façon dont cette transformation capture l'information pertinente.

Exemple : dans PCFG de base, on a deux structures de l'arbre suivantes

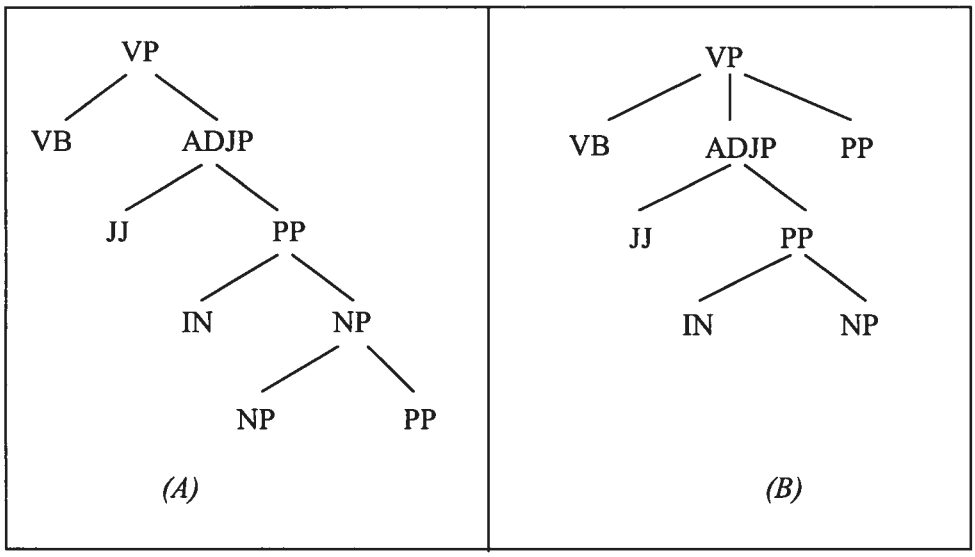


FIG. 4.6 Deux arbres possibles pour une séquence d'étiquettes VB JJ IN NP PP

Prenons par exemple une phrase constituée de la séquence d'étiquette *VP JJ IN NP PP* et considérons les deux arbres représentés en *FIG. 4.6.* Supposons de plus que *VP →VB ADJP (a), VP →VB ADJP PP (b), NP→NP PP (c).* L'analyseur donne l'arbre d'analyse *(A)* si et seulement si $a*c > b$ *ou* $b/a < c$. avec le modèle PCFG de base, on a *a=733/146022, b=117/146022, c=34965/313160 = 0.111* et *b/a = 0.159* et donc l'arbre *(B)* sera toujours sélectionné, bien que dans un grand corpus l'arbre *(A)* est plus fréquent que l'arbre *(B).* Dans la PCFG de base, la probabilité *c* ne dépend que de la fréquence de la règle *NP→NP PP* (et ce quelque soit le contexte d'application de cette règle). Ici nous mesurons *c= 34965/313160.* Avec la P-transformation, *c* dépend seulement de la règle NP_PP→ *NP_NP PP_NP, c=12616/88395=0.14.* Cette prise en compte d'un contexte plus spécifique est illustrer sur cet exemple où la phrase à analyser est "`They will be available in minimum denominations of $ 10,000.`", avec les deux arbres de la figure 4.6 ou la position en gras contient la séquence d'étiquettes que nous venons d'étudier. Avec la P-transformation, *b/a = 0.125 et c= 0.14,* l'arbre syntaxique préféré est alors l'arbre *(B)* qui rattache le groupe prépositionnel au group nominal '`minimum denomination`' alors que le modèle PCFG de base préfère l'analyse *(A)* où le groupe prépositionnel est rattaché au verbe '`will`'.

(S (NP (PRP They)) (VP (MD will) **(VP** **(VB be)** **(ADJP** **(JJ available)** **(PP** **(IN in)** **(NP** **(JJ minimum)** **(NNS enominations))))** **(PP** (IN of) (NP ($ $) (CD 10,000))))) (. .)) Rappel :85.71 % Précision:90%	(S (NP (PRP They)) (VP (MD will) **(VP** **(VB be)** **(ADJP** **(JJ available)** **(PP** **(IN in)** **(NP** **(NP** **(JJ minimum)** **(NNS denominations))** **(PP** (IN of) (NP ($ $) (CD 10,000))))))) (. .)) Rappel :100% Précision :100%
PCFG de base	P-transformation

Table 4.24 Comparaison des deux analyses proposés par le modèle PCFG de base et la P-tranformation

4.3 Traitements des mots inconnus et filtrages

Nous avons dans la section précédante observé que de nombreuses phrases restaient sans analyse. Dans le cas de la P-transformation, ces erreurs sont essentiellement dûes aux mots inconnus, c'est-à-dire, les mots qui n'ont pas été vus lors de l'entraînement. Nous proposons un traitement simple des mots inconnus qui est une alternative à l'utilisation d'un étiqueter (taggeur) externe, comme cela est proposé dans plusieurs études (Collins, Charniak, Goodman). Nous nous intéressons également dans cette section à vérifier dans quelle mesure le retrait de règles dont la fréquence est faible a un impact significatif sur les résultats.

4.3.1 Méthodologie

Nous avons analysé les mots inconnus d'un corpus de texte et avons constaté que de nombreux mots inconnus des modèles pourraient être classés en *macro-classes* :

- les données chiffrées qui représentent 10% des mots inconnus de notre corpus. Nous leur associons à la tokenization le token *N*

  Exemple : 0.0005, 1,000,000

- les mots composés représentent 20% des mots inconnus, nous remplaçons chaque mot d'un mot composé par le token *A*

  Exemple : anti-war-related est remplacé par *A-A-A*

  area-code est remplacé par A-A

- les mots contiennent les caractères non alphabétiques (par exemple . ' - etc), nous remplaçons chaque chiffre par le token *N,* chaque mot par *A*

  '50 est remplacé par *'N*

  .what est remplacé par *.A*

  1962-85 est remplacé par *N-N*

  81-year-old est remplacé par *N-A-A*

Ces *macro_classes* sont attribuées au moment de la tokenization, les mots normaux (exemple : *chaiman, publishing etc)* ne sont quant à eux pas modifiés.

Méthode 1

Pour tout symbole non terminal *N* en partie gauche d'une règle lexicale, nous ajoutons une règle lexicale $N \rightarrow UNK$ avec la fréquence 1. L'entraînement de la grammaire tient compte de l'ajout de ces règles. Ce traitement simple donne à chaque non terminal une probabilité différente de générer le mot inconnu *UNK.* Moins le non terminal est présent dans le corpus d'entraînement, plus la probabilité qu'il produise le mot *UNK* est grande. Ainsi dans notre corpus, la règle $NN \rightarrow UNK$ obtient la plus faible probabilité (1/132936) alors que la règle $WP\$ \rightarrow UNK$ obtient la plus élevée *(1/169).*

Méthode 2

Dans la méthode précédante, la probabilité des règles $N \rightarrow UNK$ dépendait de la fréquence de *N.* Ici nous associons une probabilité fixée à toutes les règles $N \rightarrow$ *UNK (0.00001).* Les distributions de probabilité sont bien entendu renormalisées pour tenir compte de cet ajout. L'idée sous-jacente à cette méthode est de laisser les règles internes décider de l'étiquette d'un mot inconnu.

Méthode 3

Nous utilisons la méthode 1, mais supprimons de plus toutes les règles internes de fréquence 1. *8609* règles sont ainsi retirées dans le cas d'une PCFG de base.

Méthode 4

Nous combinons ici la méthode *2* et le filtrage des règles internes de fréquence 1.

4.3.2 Résultats

Voici les résultats de toutes les méthodes que nous venons de décrire.

Méthode	Règles internes	Règles unitaires	Règles lexicales
(1)(2)	**14736**	**221**	**38010**
(3)(4)	6189	159	38010

Table 4.25 Nombre des règles du traitement des mots inconnus

Méthode	N1	N2	N3	Rappel(R)	Précision(P)	Croisement
	pour la longueur <=20					
(1)	**1034**	**0**	**0**	**86.17**	**88.31**	**0.308**
(2)	**1034**	**0**	**0**	**83.54**	**83.52**	**0.436**
(3)	1034	0	0	88.30	88.15	0.324
(4)	1034	0	0	83.09	83.26	0.532
	pour la longueur <=40					
(1)	**2245**	**0**	**0**	**83.37**	**85.74**	**0.756**
(2)	**2245**	**0**	**0**	**80.43**	**80.34**	**1.275**
(3)	2242	0	3	83.46	85.52	1.055
(4)	2242	0	3	79.75	79.97	1.586

Table 4.26 Résultats du traitement des mots inconnus

On voit clairement que la technique de traitement des mots inconnus de la *Méthode 1* est plus favorable que celle de la *Méthode 2*. De manière intéressante, l'élimination des règles de fréquence *1* (*8609* règles) n'affecte pas beaucoup la performance de l'analyseur syntaxique mais accélère les temps d'analyse *R :83.37%* et *P* :85.74% (*méthode 1) vs R:83.46% et P :85.52% (méthode 3), R :80.43 et P :80.34 (méthode 2) vs R :79.75 et P :79.97 (méthode 4)* (bien qu'il y ait *3* phrases sans analyse, le taux d'erreur reste très bas *3/2245*), c'est-à-dire les règles qui apparaissent rarement ne contribuent pas aux décisions de désambiguïsation dans le processus de l'analyse syntaxique. À travers ces remarques, nous retenons la méthode *3* pour le traitement des mots inconnus et le raffinement d'un modèle PCFG.

4.4 Transformation Retenue

Dans toutes les transformations que nous avons effectuées sur le corpus PTB, la P-transformation, une transformation simple, est la plus efficace. Le traitement des mots inconnus et le filtrage proposé dans la méthode *3* de la section 4.4.1 est

également une solution simple mais efficace au traitement des mots inconnus. Nous présentons maintenant les résultats obtenus en combinant ces deux transformations et comparons les performances du modèle résultant à ceux obtenus par le meilleur des modèles décrits en section 4.1.

Pour mémoire, le diagramme suivant résume notre méthodologie :

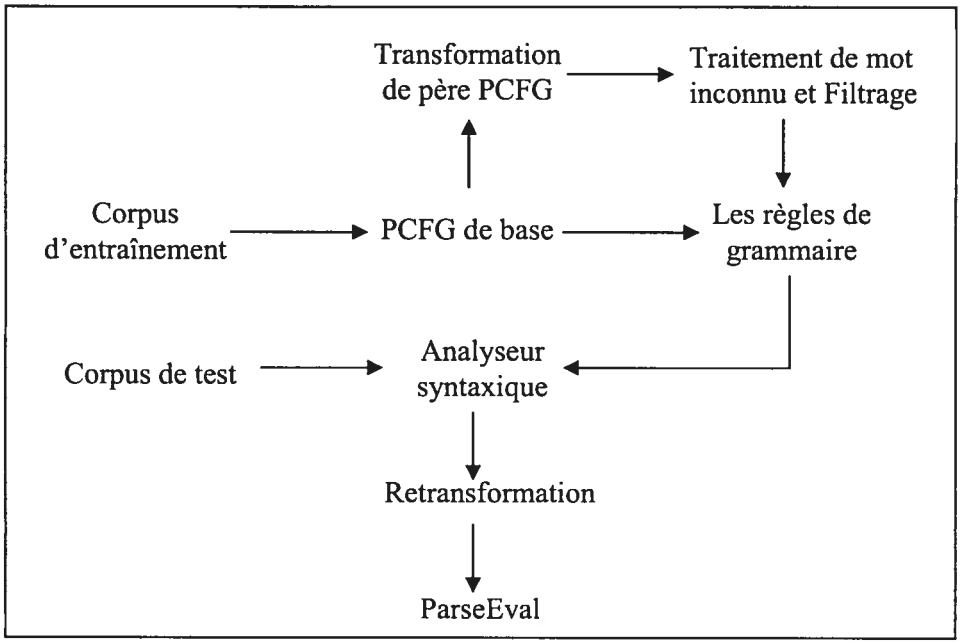


FIG. 4.7 Diagramme de transformation proposée

Nous donnons la *Précision,* le *Rappel* et le *Croisement* sur N1, N2 et N3 (la *Précision* et le *Rappel* sont assignés à *0*, le *Croisement* est nul). Il s'agit d'une comparaison directe au modèle de Collins (1999).

Règles internes	Règle unitaire	Règles lexicales
9068	**504**	**45139**

Table 4.27 Nombre de règles

Longueur	N1	N2	N3	Rappel	Précision	Croisement
<=12	402	4	0	90.91	90.55	0.0562
<=20	1027	6	1	89.37	89.43	0.2456
<=40	2230	6	9	87.27	87.38	0.8393
[Collins,1999]	n/a	n/a	n/a	88.60	88.70	0.90
<=100	2401	6	9	86.99	87.06	1.0381
[Collins,1999]	n/a	n/a	n/a	88.00	88.30	1.05

Table 4.28 Résultats de transformation proposés

Nous pouvons le constater, notre modèle se compare au modèle Collins (1999). Les performances que nous obtenons sont légèrement inférieures, mais notre modèle est de loin beaucoup plus simple. Rappelons que le modèle de

Collins (1999) utilise un modèle complexe traitant des dépendances entre les mots, nécessitant l'identification de groupes nominaux de base (*baseNP*), le traitement spécifique de la ponctuation et le taggeur de Ratnaparkhi (1996) pour gérer les mots inconnus. Notre modèle quant à lui contient en tout et pour tout "que" 9572 règles de grammaire probabilisées et le taggeur est une résultante de l'analyseur. Notons enfin que la performance de notre modèle sont supérieures à celle des autres modèles que nous avons décrits.

Chapitre 5. Conclusion

Les applications nécessitant le traitement des langues naturelles ont souvent besoin d'une étape d'analyse syntaxique partielle ou pas (exemple : question-réponse, traduction automatique, etc). Nous avons montré dans ce mémoire que les modèles PCFG pouvaient servir cette cause. L'entraînement de tels modèles peut être relativement simple et les performances obtenues suffisantes.

Nous avons décrit, étendu et implémenté deux algorithmes génériques d'analyse syntaxique qui prennent en entré un modèle PCFG. Nous avons observé un meilleur comportement de l'algorithme CYK quant aux temps de traitement et à l'occupation mémoire. Nous avons ensuite proposé et étudié un ensemble de transformations simples qui peuvent être appliquées à un corpus arboré dans le but de capturer de l'information linguistique que modèle de base ignore. Nous montrons qu'une combinaison de plusieurs de ces transformations permet d'obtenir un modèle d'une performance comparable aux meilleurs modèles décrits dans la littérature.

En particulier, nous montrons que la P-transformation (prise en compte de l'information du père d'un non terminal par l'application d'une règle) et le traitement même simple des mots inconnus sont deux étapes garante d'un modèle de qualité.

Ces résultats sont très encourageants ne sont pas sans problème. Tout d'abord, il convient de mentioner que dans le cadre d'une application réelle, les temps de calculs de notre implémentation de CYK ne sont pas optimaux (pour une grammaire d'environ 10000 règles, analyser une phrase de 12 mots prend en moyen 5 seconds, le temps d'analyse d'une phrase de 20 est de 35 seconds, analyser de phrase de 40 mots prend 41 minutes). Nous avons tenté en vain d'appliquer des techniques classiques de filtrage comme la recherche en faisceau (beam search) pour réduire les temps de calcul sans détériorer les performances. La situation n'est cependant pas sans espoir car il est possible de réduire encore plus le nombre de règles de la grammaire. Considérons par exemple les règles suviantes : (1) NP $\rightarrow$NP PP [34965/313168], (2)

NP →NP PP PP [1860/313168], (3) NP →NP PP PP PP [98/313168], (4) NP →NP PP PP PP PP [1/313168]. En appliquant plusieurs fois la règle (1), on obtient les règles (2), (3) et (4), avec une probabilité plus haute; les trois dernières règles sont donc inutiles.

Au delà des problèmes de temps, il faut dans des situations réelles d'application se doter d'un mécanisme d'analyse partielle lorsque la phrase ne peut être analysée complétement.

Nous n'avons pas abordé dans ce mémoire d'autres estimateurs que celui de la fréquence relative. D'autre modèles qui prendraient explicitement de l'information linguistique (par opposition à notre capture passive de l'information) devraient s'avérer plus performants. Nous pensons en particulier à des modèles capables de regrouper ensembles des règles qui se ressemblent d'un point de vue fonctionnel; ce qui permettrait également d'améliorer leur estimation (plus de représentants canoniques et de limiter le nombre de règles du modèle).

Enfin, mentionnons qu'outre le modèle PCFG qui présente l'avantage d'être simple et d'être efficace. Il existe d'autres approches à l'analyse syntaxique statistique [Rajman, 1996] comme le modèle des grammaires à substitution d'arbres (Tree Substitution Grammars ou TSG) et le modèle des grammaires d'arbres adjoints lexicalisées (Lexicalized Tree-Adjoining Grammars ou LTAG). Le modèle des grammaires TSG probabilistes [Resnik, 1992] nous permettra de développer les notions d'analyse syntaxique par échantillonnage (Monte Carlo Parsing) et d'analyse syntaxique guidée par les données (Data Oriented Parsing ou DOP). Les grammaires LTAG probabilistes [Schabes, 1992] constitueront quant à elle une bonne introduction à la probabilisation de modèles syntaxiques dépassant le cadre des simples grammaires non contextuelles.

Annexe

Les syntagmes du PTB

CC	Coordinating conjunction
CD	Cardinak number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition/subord, conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predterminer
POS	Possessive ending
PRP	Personal pronoun
PP$	Possessive pronoun
RB	Adverb, comparative
RBR	Adverb, superative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol (mathematical or scientific)
TO	*to*
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund / present partciple
VBN	Verb, past participle
VBP	Verb, non-3rd ps. sing. present
VBZ	Verb, 3rd ps. sing. present
WDT	wh-determiner
WP	wh-pronoun
WP$	Possesive wh-pronoun
WRB	wh-adverb
#	Pound sign
$	Dollar sign
.	Sentence-final punctuation
,	Comma
:	Colon, semi-colon
(	left bracket character
)	Right bracket character
''	Straight double quote
‘	Left open single quote
“	Left open double quote
’	Right close single quote
”	Right close double quote

Table 5.1 Ensemble des étiquettes (POS)

Tags	Marks
ADJP	Adjective phrase
ADVP	adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Clause introduced by subordinating conjunction
SBARQ	Direct question introduced by wh-word or wh-phrase
SINV	Declarative sentence with subject-aux inversion
SQ	Subconstituent of SBARQ excluding wh-wordor wh-phrase
VP	verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-preposition phrase
X	Constituent of unknown or uncertain category
Null elements	
*	« «
0	Zero variant of that in subordinate clauses
T	Trace
NIL	Marks position where preposition is interpreted

Table 5.2 Ensemble des syntagmes syntaxtiques

Tags	Marks
Text Categories	
-HLN	headline and datelines
-LST	list markers
-TTL	titles
Grammatical Functions	
-CLF	true clefs
-NOM	non NPs that function as NPs
-ADV	clausal and NP adverbials
-LGS	logical subjects in passive
-PRD	non VP predicates
-SBJ	surface subject
-TPC	topicalized and fronted constituents
-CLR	closely related
Semantic Roles	
-VOC	vocatives
-DIR	direction & trajectory
-LOC	location
-MNR	manner
-PRP	purpose and reason
-TMP	temporal phrase

Table 5.3 Ensemble des étiquettes fonctionnelles

Bibliographie

[Younger D, 1967] Younger D: Recognition of Context Free Language in time n^3, Inform. Control, vol . 10-2, 189-208.

[Baker, 1979] J.K. Baker: Trainable grammars for speech recognition. In D. Klatt and J. Wolf, editors, Speech Communication Papers for the 97th Meeting of the Acoustical Society of America, pages 547-550, 1979.

[Bahl et al, 1983] Bahl L.R, Jelinek F., and Mercer R. L: Maximum Likelihood Approach to Continuous Speech Recognition in IEEE Transaction on Pattern Analysis and Machine Intelligence, PAMI-5(2): 179-190, 1983.

[Satorini 1990] Part-of-Speech Tagging Guidelines for the PTB Project.

[Black et al., 1991] E.Black, S. Abnery, D.Flickinger. A procedure for quantitatively comparing the syntactic coverage of English grammars. DARPA Speech and Natural Language Workshop, pages 306- 311.

[Santorini et al, 1991] Satorini, Beatrice and Marcinkiewiez : Bracketting Guidelines fot the PTB Project , Department of Computer and Information Service, University of Pennsylvania.

[Harison at al, 1991] P Harrison, Abney S., Black E., Flickinger D., Gdaniec C., Grishman R., Hindle D., Infria B., Marcus M., Santorini B. & Strzalkowski : Evaluation syntax performance of parser/grammars of English. In proceedings of the Workshop on Evaluating Natural Language Processing System, ACL.

[Schabes, 1992] Schabes Y : Stochastic lexicalized tree-adjoining grammars, in Proceedings COLING 92, Nantes, France.

[Resnik, 1992] Resnik P : Probabilistic tree-adjoining grammar as a framework for statistical natural language processing, in Proceedings COLING 92, Nantes, France.

[Marcus et al, 1993] Mitchell Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz : Building a large annotated corpus of English : The PTB Computational Linguistics 19 (1993), 313-330.

[Brill, 1993] Eric Brill Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach in proceedings of the 31st Annual Meeting of the Association for Computational Linguistic, 259-265.

[Karlsson et al, 1995] Karlsson, Voutilainen, Heikkilä, Antilla.. Constraint Grammar: A language-independent system for parsing unrestricted text, Mouton de Gruyter, Berlin, 1995.

[Sampson G, 1995] English for the computer: The SUSANNE corpus and analytic scheme, Clarendon Press, Oxford.

[Marcus et al., 1995] Mitchell Marcus, Grace Kim, Mary Ann, Marcinkiewicz, Robert MacIntyre, Anne Bies, Mark Ferguson, Karent Katz, Britta Schasberger : The Penn Treebank : annotating predicate argument structure.

[Ramsaw et al, 1995]. L. A. Ramshaw and M. P. Marcus. 1995. Text chunking using transformation-based in proceedings of the Third Workshop on Very Large Corpora .

[Lin 1995] Dekang Lin : A Dependency based Method for Evaluating Broad Coverage Parser in proceeding of IJCAI-95.

[Lauer, 1995] Mark Lauer: Corpus statistics meet the noun compound: come empirical results, in proceedings of the 33rd Annual Meetings of the Association for Computational Linguistics, 47 –55.

[Magerman 1995] D. Magerman: Statistical Decision-Tree Models for Parsing. In Proceeding of the 33rd Annual Meeting of the Association for Computational Linguistics, pages 276-283.

[Karlsson et al, 1995] Karlsson, Voutilainen, Heikkilä, Antilla.. Constraint Grammar: A language-independent system for parsing unrestricted text, Mouton de Gruyter, Berlin, 1995.

[Caraballo et Charniak, 1996] Sharon A. Caraballo and Eugene Charniak : Figures of Merit for Best-First Probabilistic Chart Parsing in proceedings of the Conference in Empirical Methods in Natural Language Processing, 127-132.

[D.Manning et Carpenter, 1996] Cristopher D.Manning et Bob Carpenter: Probabilistic Parsing Using Left Corner Language Models.

[Ratnaparkhi, 1996] Adwait Ratnaparkhi : A Maximum Entropy Model for Part-of-Speech Tagging In Proceedings of the Empirical Methods in Natural Language Processing Conference, May 17-18, 1996. University of Pennsylvania.

[Rajman, 1996] Approche probabiliste de l'analyse syntaxique, T.A.L., 36, n 1-2, p. 1-43.

[Collins 1996] Michael Collins : A new Statistical parser based on bigram lexical dependencies In proceedings of the 34th Annual Meeting of the Association for Computational Linguistic (ACL), 1996.

[Charniak 1996] Eugene Charniak : Tree-bank grammars, in Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park 1031-1036.

[Charniak et al, 1997] Eugene Charniak, Glenn Carroll, John Adcock, Anthony Cassandra, Yoshihiko Gotoh, Jeremy Katz, Mitchael Littman, et John McCann: Taggeurs for Parsers.

[Charniak, 1997] Eugene Charniak : Statistical parsing with a context-free grammar and word statistics, in proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI Press/MIT Press, Menlo Park (1997).

[Miles et al, 1997] Miles Osborne and Ted Briscoe: Learning Stochastic Categorial Grammars, CoNLL97 Computational Natural Language Learning.

[Collins 1997] Michael Collins : Three Generative, Lexicalised Models for Statistical Parsing , Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European. Chapter of the Association for Computational Linguistics.

[Goodman, 1997] Joshua Goodman : Probabilistic Feature Grammars. In Proceeding of the fourth International Workshop on Parsing Technologies.

[Johnson, 1998] Mark Johnson, Brown University : PCFG models of linguistic tree representations.

[Carroll et al, 1998] John Carroll, Ted Briscoe, Antonio Sanfilippo : Parser Evaluation: a Survey and a New Proposal.

[Brill et al, 1998] Eric Brill et Jun Wu : Classifier Combination for Improved Lexical Disambiguation in Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics.

[Carroll et al., 1999] John Carroll, Guido Minnen, Ted Briscoe : Corpus Annotation for parser evaluation in Liguistically Interpreted Corpora (LINC –99) workshop at the 9th Conference of European Chapter of the Association for Computational Linguistics (EACL-99) Bergen, Norway, June 1999.

[Collins, 1999] Michael Collins : Head-Driven Statistical Models for Natural Language Parsing, University of Pennsylvania, Ph.D. Dissertation.

[Ratnaparkhi, 1999] Adwait Ratnaparkhi : Learning to parse natural language with maximum entropy models. Machine Learning 34 1/2/3, 151-176.

[Chiang ,2000] David Chaing : Statistical parsing with an automatically-extracted tree adjoining grammar In Proceedings of ACL 2000, Hong Kong, October 2000, pages 456-463.

[Charniak, 2001] Eugene Charniak : Immediate-Head Parsing for Language Models, meeting of the Association for Computational Linguistics.

[Collins, 2000] Michael Collins : Discriminative reranking for natural language parsing. In proceedings of the International Conference on Machine Learning (ICML 2000).

[Benedi et al, 2000] Benedi J.M, Sanchez J.A.: Combination of N-Grams and Stochastic Context-Free Grammars for Language Modeling, 2000.

[Lancaster corpus] University Centre for computer corpus REsearch on Language of Lancaster University: Corpus Annotation
http://ww.comp. lancs.ac. uk/ computing/ research/ ucrel /annotation.html# treebank

[BNC corpus] http://www.hcu.ox.ac.uk/BNC/

[Brown corpus] Department of Linguistics Brown University http://khnt.hit.uib.no/icame/manuals/brown/INDEX.HTM#bc9

[Japonais corpus] http://www-nagao.kuee.kyoto-u.ac.jp/index-e.html
[Charniak, 93] : Eugene Charniak, Statistical Language Learning, MIT Press.

[Manning et Schutze, 1999] : Christopher D.Manning et Hinrich Schutze, Foundation of Statistical Natural Language Processing.