MODÈLES SYNTAXIQUES PROBABILISTES NON-GÉNÉRATIFS

THÈSE N^{O} 2705 (2002)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Antoine ROZENKNOP

ingénieur diplômé de l'ENST, Paris, France
et de nationalité française

acceptée sur proposition du jury:

Dr M. Rajman, directeur de thèse
Prof. H. Bourlard, rapporteur
Dr J.-C. Chappelier, rapporteur
Prof. G. Coray, rapporteur
Dr A. Drygajlo, rapporteur
Prof. M. El-Bèze, rapporteur,
Dr F. Yvon, rapporteur

Lausanne, EPFL
2003

Version abrégée

Modèles syntaxiques probabilistes non génératifs

Ce travail traite de modèles utilisés, ou utilisables en traitement automatique du langage naturel, lorsque l'on cherche une interprétation syntaxique d'un énoncé. Cette interprétation peut être utilisée comme une information supplémentaire pour des traitements ultérieurs de l'énoncé, visant par exemple à le représenter sémantiquement. Elle peut aussi servir plus simplement comme un filtre pour sélectionner des énoncés appartenant à un langage spécifique, comme en reconnaissance de la parole.

L'interprétation syntaxique d'un énoncé en langage naturel étant le plus souvent ambiguë, la probabilisation de l'espace des arbres syntaxiques peut permettre de faciliter la tâche d'analyse : lorsque plusieurs analyses sont possibles, on peut alors extraire l'interprétation la plus probable, ou encore classer les interprétations selon leurs probabilités. On s'intéresse ici aux versions probabilistes des grammaires hors-contexte (PCFG) et des grammaires à substitution d'arbre (PTSG).

La probabilisation des grammaires est réalisée à partir d'ensembles d'arbres syntaxiques d'apprentissage, représentant plus ou moins fidèlement le langage modélisé. Dans un premier temps, en nous appuyant sur des exemples jouets, nous exhibons un certain nombre de faiblesses des méthodes usuelles d'apprentissage, dues à l'absence de critère d'apprentissage explicite (modèle STSG DOP), ou à l'emploi de critères qui considèrent ces grammaires comme génératives (passage de la grammaire à un énoncé) plutôt que dédiées à l'analyse (passage d'un énoncé à son interprétation).

Dans un deuxième temps, nous proposons de nouvelles méthodes d'apprentissage, développées à partir des critères classiques de maximum d'entropie, et de maximum de vraisemblance. Ces critères sont décrits de façon à correspondre à une tâche d'analyse syntaxique plutôt que de génération d'un langage. Des algorithmes d'apprentissage spécifiques sont nécessaires pour leur mise en application, mais l'analyse syntaxique subséquente peut utiliser des algorithmes classiques.

Enfin, nous nous intéressons également aux problèmes de complexité de l'analyse syntaxique, particulièrement saillants avec les PTSG. Nous décrivons des classes de PTSG permettant l'analyse d'un énoncé avec une complexité polynomiale en sa longueur. Nous élaborons enfin des méthodes pour obtenir de telles PTSG à partir d'un corpus arboré, et les utilisons de façon à pouvoir tester les critères d'apprentissage en terme de résultats sur des données « réelles ».

English summary

Nongenerative Probabilistic Syntactic Models.

This work deals with models used, or usable in the domain of Automatic Natural Language Processing, when one seeks a syntactic interpretation of a statement. This interpretation can be used as additional information for subsequent treatments, that can aim for instance at producing a semantic representation of the statement. It can also be used as a filter to select utterances belonging to a specific language, among several hypotheses, as done in Automatic Speech Recognition.

As the syntactic interpretation of a statement is generally ambiguous with natural languages, the probabilisation of the space of syntactic trees can help in the analysis task : when several analyses are competing, one can then extract the most probable interpretation, or classify interpretations according to their probabilities. We are interested here in the probabilistic versions of Context-Free Grammars (PCFGs) and Substitution Tree Grammar (PTSGs).

Syntactic treebanks, which as much as possible account for the language we wish to model, serve as the basis for defining the probabilistic parameters of such grammars. First, we exhibit in this thesis some drawbacks of the usual learning paradigms, due to the use of arbitrary heuristics (STSG DOP model), or to the use of learning criteria that consider these grammars as generative ones (creation of sentences from the grammar) rather than dedicated to analysis (creation of analyses from the sentence).

In a second time, we propose new methods for training grammars, based on the traditional Maximum Entropy and Maximum Likelihood criteria. These criteria are instanciated so that they correspond to a syntactic analysis task rather than a language generation task. Specific training algorithms are necessary for their implementation, but traditional algorithms can cope with those models for the task of syntactic analysis.

Lastly, we invest the problem of time complexity of syntactic analysis, which is a real issue for the effective use of PTSGs. We describe classes of PTSGs that allow the analysis of a sentence in polynomial complexity. We finally describe a method that enable the extraction of such a PTSG from the set of subtrees of a treebank. The PTSG produced by this method allows us to test our non-generative learning criterium on "realistic" data, and to give a statistical comparison between this criterium and the usual heuristic criterium in term of analysis performance.

Table des matières

Chapitre 1

Introduction

1.1 Linguistique informatique

La *linguistique informatique* est une discipline issue des développements de l'informatique dans le domaine des sciences du langage. Dans un sens large, elle couvre toutes les applications informatiques ayant un rapport avec le langage naturel, c'est-à-dire le langage tel qu'il est parlé et compris par les êtres humains, par opposition aux langages « artificiels » développés en mathématique, en informatique (langages de programmation)... Quarante ans d'efforts de recherche ont réussi à montrer que si la production et la compréhension d'énoncés en langues naturelles sont des tâches élémentaires pour les humains, les machines ont en revanche un mal extrême à traiter ces problèmes. Malgré tout, avec les développements technologiques considérables dont ont profité les ordinateurs ces dernières années, des applications de la linguistique informatique ont commencé à pouvoir être réalisées dans divers domaines. Non seulement les progrès techniques ont permis ces réalisations, mais ils *suscitent* eux-mêmes une demande croissante de telles applications, donc un effort de recherche croissant, du fait de l'envahissement de nos vies quotidiennes par les ordinateurs et par des quantités d'informations gigantesques qu'ils permettent de mettre à disposition.

1.1.1 Applications

Quelques exemples d'applications peuvent mettre en lumière l'importance de la linguistique informatique pour l'informatique en général, et pour les disciplines du langage en particulier :

- la *traduction automatique* occupe une place privilégiée dans le domaine de la linguistique informatique, du fait qu'elle en a constitué le point de départ, dès le début des années 1950. Elle fait intervenir toutes les composantes de la linguistique informatique, de l'analyse à la génération, certains projets y faisant également intervenir la reconnaissance

et la synthèse de la parole (Verbmobil, [Niemann 97]). Les besoins en traduction sont également en constante augmentation, ce qui lui garantit un marché substantiel. Les systèmes commercialisés (SYSTRAN) reposent sur des analyses linguistiques encore sommaires, et délivrent des qualités de traduction généralement médiocres, mais pouvant être utiles lorsqu'il s'agit de traiter de gros volumes textuels ou d'avoir un aperçu vague d'un texte ;

- des *correcteurs orthographiques* apparaissent fréquemment dans les applications de traitement de texte. Si les premiers systèmes se bornaient à comparer l'orthographe des mots avec ceux d'une liste préétablie, des systèmes plus sophistiqués sont apparus, qui effectuent une analyse grammaticale et arrivent à repérer des fautes d'accord, ce qui nécessite de connaître les relations grammaticales intervenant entre les mots ;

- la *recherche d'information* consiste en la conception de programmes capables de rechercher dans de très grandes ensembles de données (internet, par exemple) des documents exprimés en langue naturelle pour en extraire les informations désirées. Les programmes disponibles sont essentiellement basés sur la recherche de mots-clefs, ou sur des statistiques sur le vocabulaire employé dans les documents. Des outils de recherche plus performants apparaissent cependant, qui prennent en compte les variations morphologiques, la synonymie, les relations syntaxiques [Besançon 02].

- la *reconnaissance automatique de la parole* peut consister en un système de capture de mots-clefs, permettant par exemple de comprendre des ordres simples, ou en un système de « dictée vocale », qui serait une sorte de machine à écrire à laquelle on dicte un texte au lieu de le taper. Dans ce cas, la difficulté première à laquelle doit faire face cette application est le découpage d'un flot de parole en une séquence de mots. Étant données les nombreuses incertitudes liées à la reconnaissance phonétique de sons proches (« plante/pleinte ») ainsi que les cas d'homophonie (« sang/sans/cent », « on naît/on est »), une analyse linguistique s'avère d'un grand secours pour aider à déterminer les choix les plus plausibles sur la base de contraintes linguistiques.

- la *synthèse de la parole* est l'opération inverse, selon laquelle une machine lit un texte à haute voix, par l'intermédiaire d'un système de synthèse, qui traduit une suite de phonèmes en un flot de parole. Ces systèmes, bien que faisant des progrès réguliers, n'atteignent pas la performance humaine, en particulier pour les aspects prosodiques (intonation), le traitement des homographes hétérophones (mots s'écrivant de la même façon et se prononçant différemment : « les poules

du couvent couvent ») ou le traitement des liaisons entre les mots (qui n'est pas systématique). Une analyse d'ordre syntaxique peut souvent donner des indications sur la façon de prononcer un énoncé ;

Très différentes les unes des autres à maints égards, toutes les applications décrites ont pourtant une exigence commune : la nécessité d'une analyse linguistique précise pour obtenir des résultats de bonne qualité. Cette analyse relève de niveaux différents, qui correspondent aux composantes traditionnelles d'une grammaire : la *morphologie*, la *syntaxe*, la *sémantique* et la *pragmatique*, auxquelles il faut ajouter la *phonétique* et la *phonologie* dans le cas des applications en parole. La composante syntaxique est souvent considérée comme centrale dans la grammaire, et c'est celle qui nous préoccupe dans cette thèse. Depuis [Chomsky 57], qui a jeté les bases de théories syntaxiques suffisamment explicites pour que l'on puisse tenter de les implémenter, de nombreux formalismes grammaticaux ont vu le jour. Chomsky jugeait les grammaires régulières et non contextuelles insuffisantes pour la description du langage naturel, et a proposé des « modèles transformationnels ». Plusieurs formalismes proposés plus tard sont basés sur un mécanisme d'unification, comme les GPSG (grammaires syntagmatiques généralisées), HPSG (grammaires de tête), LFG (grammaires lexicales fonctionnelles)... (voir [Miller 90] pour une description de ces différents formalismes).

Les modèles considérés dans cette thèse sont pourtant basés sur les grammaires hors-contextes (CFG : context-free grammar), ainsi que sur les grammaires à substitution d'arbres (TSG : Tree Substitution Grammar) qui sont construites à partir des CFG. Malgré leur inaptitude à *décrire* efficacement le langage naturel, soulignée par Chomsky, ces grammaires peuvent servir de *briques de base* pour construire des interprétations syntaxiques d'un énoncé, ou pour la représentation syntaxique. Ils se prêtent de plus particulièrement bien à des algorithmes d'analyse efficaces, ainsi qu'à leur dérivation sous forme de modèles probabilistes (du moins en comparaison avec les grammaires d'unification).

1.1.2 Modèles probabilistes

Comme nous l'avons souligné, les langages naturels sont caractérisés par leur *variabilité*, et leur *ambiguïté*, qui apparaissent à tous les niveaux linguistiques :

- la variabilité se traduit dans les mécanismes de production d'un énoncé par le fait qu'une même idée peut s'exprimer de différentes manières. Par exemple, au niveau syntaxique, « Réponds-moi ! »/« Peux-tu me répondre, s'il-te-plaît ? »/« Merci de me répondre ! » ont le même sens, sans avoir la même forme ; au niveau lexical, cette variabilité

est principalement due à l'existence de synonymes (« matou »/« minet »/« chat »).

– l'ambiguïté est une variabilité dans le sens inverse, en analyse : un même énoncé peut avoir plusieurs sens. Nous avons déjà évoqué l'ambiguïté phonologique (« sang »/« sans ») et lexicale (« couvent »). Un exemple bien connu d'ambiguïté syntaxique est : « la petite brise la glace », dans lequel « brise » et « glace » peuvent jouer tous deux le rôle du verbe ou du nom.

Pour résoudre les problèmes d'ambiguïté en analyse, on peut parfois s'appuyer sur des niveaux linguistiques supérieurs : dans « les poules du couvent couvent », la syntaxe interdit par exemple d'avoir un verbe après un « du », et l'on en déduit que le premier « couvent » est un nom, et le deuxième est un verbe de façon à ce que la phrase ait un sens. Les humains sont étonnamment doués pour de telles analyses. Il est rare qu'ils n'arrivent pas, en utilisant leurs connaissances linguistiques et pragmatiques, à désambiguïser un énoncé[1]. Un modèle informatique idéal d'analyse ferait sans doute appel à tous ces niveaux, sur le modèle humain, mais le fait est que les algorithmes utilisés par l'homme pour cette tâche demeurent un grand mystère, et l'on ne sait si la puissance actuelle des ordinateurs peut ou non permettre d'utiliser toutes ces connaissances pour désambiguïser aussi bien qu'un esprit humain.

Plutôt que de chercher à combattre la variabilité du langage naturel par tous les moyens, on peut alors en *rendre compte* en décrivant plus ou moins le langage comme un phénomène aléatoire. Pour la production automatique d'énoncés, cet aspect aléatoire apparaît comme une façon de rompre la monotonie des phrases produites. En analyse, il rend compte du fait que le système ne possède pas les connaissances nécessaires pour désambiguïser un énoncé, dont l'analyse lui paraît de fait aléatoire.

Cependant, le langage n'apparaît pas comme *totalement* aléatoire, même pour un système n'ayant des connaissances que sur un niveau linguistique particulier : sans être capable d'expliquer pourquoi, il peut quand même observer que certaines formes sont plus fréquentes que d'autres. Par exemple, un système de synthèse vocale qui voudrait imiter un enfant peut *observer* que « Maman » est un mot fréquent dans son domaine, sans pour autant savoir ni où ni quand utiliser ce mot s'il veut se faire passer pour un enfant.

En bref, le phénomène aléatoire, qui décrit le langage du point de vue du système automatique, peut être probabilisé de manière à « coller » au mieux

[1] la preuve en est que lorsque cela arrive, on appelle cela un quiproquo, l'effet de surprise provoque en général le rire.

à un domaine particulier.

La description d'un langage comme un phénomène probabiliste peut être utilisée dans différentes perspectives :

- classification de documents : de nombreux systèmes de classification automatique de documents reposent sur de simples statistiques sur l'occurrence de mots [Besançon 02] ; les paramètres des modèles de *grammaires probabilistes* peuvent servir à caractériser le style d'un document, voire à deviner son auteur ;

- reconnaissance de la parole : il semble que ce domaine soit celui qui profite le plus des modèles probabilistes ; il a très tôt utilisé des modèles de bigrammes, qui représentent les statistiques sur l'adjacence de deux mots dans le langage. De nombreuses recherches visent à enrichir les statistiques prises en compte, en considérant des séquences de plus de deux mots ou en ajoutant des informations d'ordre syntaxique à ces modèles, ou encore en utilisant des modèles de grammaires probabilistes tels que les SCFG. La probabilisation du langage induite par ces modèles permet à un système de reconnaissance de choisir parmi différentes interprétations celle qui est la plus *plausible* [Boite 00, Chappelier 99c].

- interprétation et analyse syntaxique : les modèles syntaxiques probabilistes affectent des probabilités aux structures syntaxiques possibles dans l'analyse d'une phrase ; celles-ci peuvent être utilisées pour lever l'ambiguïté, en décidant par exemple de choisir l'analyse la plus probable comme solution.

Les grammaires PCFG et PTSG, étudiées dans cette thèse, sont des versions probabilistes des grammaires CFG et TSG.

1.1.3 Approches ascendante et descendante des modèles syntaxiques

Dans cette thèse, nous faisons la distinction entre deux approches du traitement du langage, en particulier pour les modèles syntaxiques :

- l'approche *descendante* considère un modèle syntaxique comme un moyen de produire des énoncés ; on qualifie alors ce modèle de *modèle génératif* ;
- l'approche *ascendante* considère un modèle syntaxique comme un moyen de faire l'*analyse* syntaxique d'un énoncé, c'est-à-dire de construire des structures syntaxiques *au-dessus* d'un énoncé, qui lui sont compatibles au sens de la grammaire.

Lorsque les modèles sont probabilisés, cette distinction se répercute sur les probabilités qui nous intéressent :

- dans une approche descendante, les paramètres du modèle servent à calculer la probabilité de génération d'un *énoncé par un processus stochastique de génération*;
- dans une approche ascendante, on cherche à calculer la probabilité d'une *analyse syntaxique à partir d'un énoncé.*

Remarquons que ces termes sont aussi utilisés pour décrire des classes d'algorithmes d'analyse syntaxique : les algorithmes ascendants partent de la phrase pour créer des structures syntaxiques, alors que les algorithmes descendants commencent par créer des structures syntaxiques, avant de vérifier si elles peuvent ou non correspondre à la phrase. Dans la suite, la distinction *ascendante/descendante* sera réservée à la description des algorithmes d'analyse. On distinguera les types de modèles probabilistes sous-jacents par les termes *génératif*[2] et *non-génératif*[3].

L'approche probabiliste générative est largement plus utilisée que l'approche non-générative. Cela est peut-être dû à l'origine Chomskienne générative des formalismes informatiques, ou au succès rencontré par les modèles probabilistes en reconnaissance de la parole, où l'on cherche à probabiliser les énoncés, de façon à pouvoir choisir le plus plausible en cas d'ambiguïté. Toujours est-il que cette approche est très utilisée y compris pour les systèmes d'analyse !

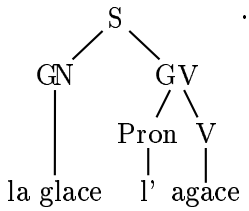
Cette distinction étant à la base de ce travail de thèse, illustrons-la par un petit exemple; on considère une grammaire hors-contexte contenant les règles suivantes :

$S \rightarrow GN\ GV$	$V \rightarrow$ brise
$GV \rightarrow V\ GN$	$V \rightarrow$ glace
$GV \rightarrow Pron\ V$	$V \rightarrow$ agace
$GN \rightarrow$ la petite	$Pron \rightarrow$ la
$GN \rightarrow$ la glace	$Pron \rightarrow$ l'
$GN \rightarrow$ la petite brise	

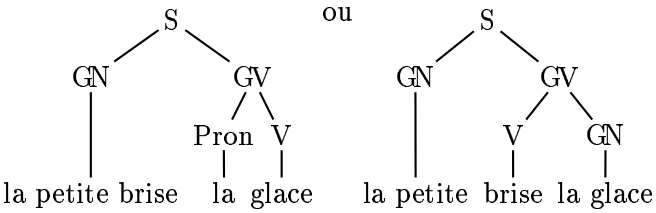
On peut considérer cette grammaire comme un ensemble de *contraintes* sur les structures syntaxiques du langage, la première règle indiquant par exemple qu'une phrase (« S ») peut être constituée d'un groupe nominal (« GN ») suivi d'un groupe verbal (« GV »). On peut utiliser cette grammaire

[2] modèle génératif : probabilisation descendante des arbres syntaxiques;
[3] modèle non-génératif : probabilisation ascendantes des arbres syntaxiques.

pour faire de l'analyse syntaxique de phrases simples, comme « la glace l'agace », dont l'analyse peut se représenter par un arbre :



Mais cette grammaire est ambiguë ; la phrase « la petite brise la glace » a plusieurs analyses possibles :



Cette grammaire peut également être considérée comme une grammaire générative, qui produit des phrases par un processus stochastique ; celui-ci construit des arbres en partant du symbole initial « S », par l'application successive de règles tirées au sort. Dans cette optique, on peut orienter le tirage en associant des *probabilités de tirage* à chacune des règles. On dispose d'un ensemble de phrases, qui représentent un domaine d'application. On constate que dans le domaine qu'on essaie de modéliser les structures $GV \rightarrow V\,GN$ sont plus fréquentes que les structures $GV \rightarrow Pron\ V$. On va donc orienter le tirage au sort des règles en leur affectant des probabilités, en s'assurant que la probabilité de la première règle est supérieure à celle de la seconde. La grammaire hors-contexte stochastique (SCFG) qu'on obtient peut alors être utilisée en reconnaissance de la parole, à la fois pour restreindre le nombre d'interprétations possibles d'un énoncé (i.e. affaiblir jusqu'à écarter l'hypothèse d'interprétations sans structure possible, comme « l'agace la glace ») et pour classer les hypothèses selon leurs probabilités d'être émises par le processus : à score acoustique équivalent, on placera l'hypothèse « la petite brise la glace » en meilleure position que « la petite brise l'agace », du fait qu'elle peut être produite par la règle $GV \rightarrow V\,GN$, alors que l'autre ne peut s'analyser qu'avec la règle $GV \rightarrow Pron\ V$.

En outre, la grammaire obtenue, considérée comme un modèle génératif, peut être utilisée dans un processus d'analyse, car le processus ainsi obtenu ne probabilise pas seulement les phrases d'un langage, mais également les arbres syntaxiques créés par l'application successive des règles hors-contextes. Avec cette probabilisation, on ne peut certes pas décider de façon sûre quelle

interprétation donner à la phrase « la petite brise la glace », mais néanmoins, on peut dire que la seconde interprétation est la plus probable, compte tenu du domaine considéré lors de la probabilisation des règles.

En résumé, le modèle syntaxique, ambigu à la base, a été paramétré comme un modèle génératif, puis utilisé comme un modèle analytique. Plusieurs questions se posent, dont les deux suivantes sont les plus immédiates :

- Comment choisir les paramètres du modèle pour le faire correspondre au mieux à un domaine, représenté par un corpus ?
- Le modèle obtenu est-il adapté à une tâche de désambiguïsation syntaxique ?

1.1.4 Apprentissage des paramètres

Les probabilités des modèles génératifs sont en général déterminées à partir d'un corpus d'apprentissage, qui représente le domaine de l'application considérée. Ce corpus sera, dans notre cas, constitué d'un ensemble d'arbres syntaxiques produits par des experts humains, que l'on considère comme des exemples de ce que le modèle doit produire. Les probabilités des règles sont alors déterminées de façon à ce que le processus produise les arbres *les plus proches* de ceux du corpus. Pour les calculer automatiquement, on a besoin de *critères mathématiques* définissant cette proximité, ainsi que de *méthodes algorithmiques* pour maximiser ces critères.

Différentes approches peuvent être utilisées pour le choix des paramètres du modèle.

- l'approche *heuristique* consiste déterminer le modèle et ses paramètres selon des règles définies à l'avance ; c'est l'approche utilisée dans le modèle DOP STSG sur lequel nous reviendrons au chapitre 4 ;

- l'approche *empirique* pose la forme du modèle, et détermine la valeur de ses paramètres en testant manuellement son comportement pour plusieurs valeurs ; c'est par exemple une méthode pour fixer les paramètres α, β utilisés dans le mélange de modèles acoustique et linguistique en reconnaissance de la parole [Boite 00] : on teste les performances du modèle pour plusieurs valeurs de α et β, et l'on choisit les valeurs qui produisent les meilleures performances ;

- l'approche par le critère de *maximum de vraisemblance* : la forme du modèle[4] est fixée, la valeur des paramètres est calculée de façon à maxi-

[4]On entend par *forme du modèle* la famille paramétrique de fonctions à laquelle ap-

miser un *critère* mathématique, la vraisemblance du corpus, qui est liée à la probabilité affectée au corpus par le modèle ; cette méthode se rapproche de la méthode empirique, à la différence près que l'utilisation de ce critère mathématique permet de déterminer automatiquement les paramètres par des algorithmes qui convergent vers la solution, au lieu d'explorer l'espace des solutions pour en extraire la meilleure.

– l'approche par le critère de *maximum d'entropie* ; ici, la forme du modèle n'est pas fixée, mais on fixe les caractéristiques que l'on observe dans le corpus. Par exemple, on choisit d'observer la fréquence des règles hors-contextes apparaissant dans le corpus. Le modèle obtenu par cette approche est celui qui va produire des analyses qui auront les mêmes caractéristiques en moyenne que celles qu'on a observées dans le corpus. Lorsque plusieurs modèles peuvent respecter cette contrainte, on choisit celui d'entropie maximale, i.e. celui qui *rend le modèle le plus ambigu possible*, c'est-à-dire le plus indéterminé : on cherche à obtenir un modèle avec aussi peu d'*a prioris* que possible sur les caractéristiques que l'on *n'a pas choisi d'observer* dans le corpus.

Le plus utilisé des critères est celui du maximum de vraisemblance, qui maximise simplement la probabilité de produire le corpus par une série de tirages. Dans le cas des SCFG, les probabilités des règles que l'on obtient sont proportionnelles à leur fréquence d'apparition dans le corpus ce qui rend l'algorithme d'apprentissage trivial. Cet apprentissage est *supervisé*, dans le sens où l'on observe directement dans le corpus les étapes du processus génératif. Lorsque le corpus n'est constitué que de phrases, qui sont les productions finales du processus, on peut effectuer un apprentissage *non supervisé*, avec ce même critère de maximum de vraisemblance, qui repose alors sur la probabilité des phrases plutôt que celle des arbres. Pour trouver les paramètres qui le maximisent, on doit alors utiliser des algorithmes plus complexes, dont le plus connu est l'algorithme EM[5] [Dempster 77].

Lorsque l'on dispose d'un corpus d'arbres, on peut non seulement l'utiliser pour faire l'apprentissage des paramètres des règles d'une grammaire hors-contexte, mais également l'utiliser directement pour l'analyse syntaxique. C'est ce que fait le modèle DOP[6], basé sur l'observation que *les adultes tendent à traiter les énoncés sur la base de leur expérience linguistique passée*

partient la distribution de probabilités du modèle. Par exemple, dans une SCFG (voir à la section 2.2), la probabilité d'un arbre est de la *forme* $p(t) = \prod_{r \in t} p_r^{n(r,t)}$, où t est un arbre, r une règle de grammaire hors-contexte, $n(r,t)$ le nombre d'occurrences de r dans t, et p_r est un paramètre du modèle associé à la règle r. L'apprentissage du modèle consiste alors uniquement à déterminer les paramètres p_r ; la forme du modèle est prédéfinie.

[5]pour Expectation Maximization.

[6]DOP = Data Oriented Parsing = Analyse orientée par les données.

[Scha 90]. DOP fait l'analyse d'une nouvelle phrase en commençant par se « remémorer » tous les sous-arbres apparaissant dans le corpus. Puis ces sous-arbres sont assemblés, de façon similaire aux règles d'une SCFG, pour créer les analyses de la phrase. Le processus d'assemblage des sous-arbres est gouverné par leurs fréquences relatives dans le corpus d'apprentissage, considérées comme des probabilités, et la théorie des probabilités est mise en œuvre pour le calcul de la probabilité des analyses ainsi créées, afin de sélectionner celle de plus forte probabilité. Bien que ce modèle soit à la base attaché à un problème d'analyse, ses implémentations actuelles se conforment aux modèles de grammaires probabilistes génératives, en conditionnant la probabilité affectée aux sous-arbres par la catégorie de leur racine : elles supposent l'existence d'un processus stochastique sous-jacent, qui crée des arbres en partant d'un nœud initial « S », auquel on substitue un sous-arbre de racine « S », aux feuilles duquel on substitue à nouveau des sous-arbres de racine correspondante, et ainsi de suite jusqu'à l'obtention d'un arbre complet. Il s'agit d'un modèle à substitution d'arbres stochastique, ou STSG[7]. Dans une STSG, les arbres qui viennent se substituer aux feuilles constituent en quelque sorte les règles de la grammaire ; on les appelle *arbres élémentaires* [Bod 93].

On peut noter qu'une STSG dont les arbres élémentaires sont uniquement des arbres de profondeur 1 peut s'identifier à une SCFG. Dans ce cas, la probabilité de substitution des arbres, posée comme leur fréquence relative dans le corpus, est exactement celle qu'on obtient en maximisant le critère de vraisemblance du corpus. Cependant, dans le cas où l'on considère des arbres de profondeur supérieure comme faisant partie des arbres élémentaires, le choix de leur fréquence relative comme probabilité de substitution ne repose sur aucun critère mathématique et relève plutôt d'une approche heuristique.

1.1.5 Utilisation en analyse

La réponse à la deuxième question évoquée plus haut (à savoir : les modèles obtenus sont-ils adaptés à une tâche d'analyse) est mitigée. Les probabilités des modèles génératifs permettent indéniablement d'améliorer l'analyse syntaxique, dans le sens où la désambiguïsation qu'ils apportent est statistiquement meilleure qu'un tirage au sort parmi les différentes hypothèses. En revanche, il apparaît que ces paramètres ne sont pas *les plus adaptés* à la tâche d'analyse. Dans ce travail, on verra sur quelques exemples particuliers très simples que ces modèles peuvent même se comporter de façon contraire à ce que l'on voudrait : des analyses apparaissant plus souvent que d'autres dans le corpus d'apprentissage peuvent se voir affectées d'une probabilité

[7]STSG, pour Stochastic Tree Substitution Grammar.

plus faible[8].

L'hypothèse défendue ici est que ce type de « biais » vient de ce que les paramètres sont déterminés selon une approche générative (*descendante*) de production à partir des catégories syntaxiques supérieures, alors que l'analyse est par définition non-générative (*ascendante*). Pour vérifier cette hypothèse, et pour tenter d'améliorer les résultats des modèles syntaxiques probabilistes en analyse, cette thèse s'attache donc à définir des méthodes de probabilisation non génératives, directement liées à la tâche d'analyse.

1.2 Survol des modèles étudiés

Cette thèse traite de différents modèles de grammaires probabilistes, dont les caractéristiques peuvent être résumées par la figure 1.1.

On distingue :
- les CFG : grammaires hors-contextes (Context-Free Grammars) ;
- les TSG : grammaires à substitution d'arbres (Tree Substitution Grammars) ;

Ces grammaires se déclinent sous différentes formes de grammaires probabilistes (PCFG et PTSG) : la forme *stochastique* considère ces grammaires comme des *processus stochastiques* descendants, ou génératifs (SCFG, STSG), alors que la forme *gibbsienne* les considère plutôt comme des modèles d'analyse (GCFG, GTSG).

Les grammaires considérées diffèrent par :

- la *profondeur maximale de leurs règles* :
  une règle de réécriture d'une grammaire hors-contexte peut en effet être considérée comme un arbre de profondeur 1, le symbole de tête, ou « membre de gauche », correspondant à la racine de l'arbre, et ses symboles de queue, ou « membre de droite » aux feuilles. Vues sous cet angle, les PCFG apparaissent comme des cas particuliers de PTSG.

- la *nature des paramètres associés à leurs règles* :

  - Ces paramètres sont des probabilités dans les grammaires stochastiques. Celles-ci modélisent des processus aléatoires, dont l'opération élémentaire est d'appliquer à un symbole X une règle parmi

[8] ces « biais » sont constatés p.68 pour les modèles SCFG, et p.79 pour les grammaires DOP STSG.

		PCFG			PTSG	
		SCFG	TCFG	GCFG	STSG	GTSG
profondeur des règles		$= 1$			≥ 1	
paramètres λ_r associés auxrègles r	forme standard	probabilité $p(r) = e^{\lambda_r}$	potentiel λ_r	potentiel λ_r	probabilité $p(r) = e^{\lambda_r}$	potentiel λ_r
	domaine de λ_r	$\mathbb{R}^-$	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}^-$	$\mathbb{R}$
	contrainte	$\sum_{r \in \mathcal{R}_X} e^{\lambda_r} = 1$	$\emptyset$	$\emptyset$	$\sum_{r \in \mathcal{R}_X} e^{\lambda_r} = 1$	$\emptyset$
Critère d'apprentissage		ML $\prod p(t\|rac(t))$	$\left\{ \begin{array}{r} \text{ML} \\ \prod p(t\|rac(t)) \end{array} \right\}$ + intuitif	MCL $\prod p(t\|w(t))$	*intuitif*	MCL $\prod p(t\|w(t))$
Complexité de l'analyse		polynomiale			NP-difficile ou polynomiale	

FIG. 1.1 – Caractéristiques des grammaires étudiées ($rac(t)$ et $w(t)$ sont resp. la racine et les feuilles de l'arbre t).

l'ensemble $\mathcal{R}_X$ de ses règles ayant X pour tête. Les paramètres sont alors des répartitions de probabilités sur les ensembles $\mathcal{R}_X$ pour chaque symbole non-terminal X. Ils doivent donc être compris entre 0 et 1, et respecter les contraintes stochastiques habituelles : $\sum_{r \in \mathcal{R}_X} p(r) = 1$.
  - Dans les autres grammaires, les paramètres sont des *potentiels*, pouvant prendre une valeur réelle quelconque.

- le *critère* utilisé pour effectuer le calcul des paramètres à partir d'un corpus d'exemples :

  - le maximum de vraisemblance (ML : Maximum Likelihood) est un critère d'apprentissage courant pour des modèles stochastiques. Il est fondé sur la probabilité $P(Corpus|\lambda)$, probabilité affectée par le modèle de paramètres λ au corpus d'exemples. Dans le cas d'un modèle stochastique, c'est la probabilité pour que le processus associé *produise* le corpus.
  - le maximum de vraisemblance conditionnelle (MCL : Maximum Conditionnal Likelihood) est le critère développé dans cette thèse pour les modèles GCFG et GTSG. Il est fondé sur la probabilité $P(Corpus|\lambda, phrases)$, probabilité affectée par le modèle aux arbres du corpus, *connaissant leurs feuilles*, les feuilles des arbres correspondant aux phrases du corpus.
  - Dans les deux cas ML et MCL, l'apprentissage consiste dans le choix des paramètres qui vont maximiser le critère.

- la complexité de l'analyse syntaxique.
  On entend par analyse syntaxique le problème MPP[9] d'extraction de **l'arbre** d'analyse le plus probable parmi les arbres pouvant correspondre à une phrase donnée. La complexité du problème MPP, pour les PTSG, dépend des arbres élémentaires qui les constituent. L'une des difficultés majeures associées à ce type de grammaires est que dans le cas général, ce problème est NP-difficile : on ne connaît pas d'algorithme qui puisse le résoudre en un temps polynomial en fonction des tailles de la phrase analysée et de la grammaire (voir [Sima'an 96a] et Annexe A).

Un modèle intermédiaire : la Grammaire Hors Contexte Thermodynamique (TCFG)

La figure 1.1 fait référence à un modèle de grammaire hors-contexte *thermodynamique* (TCFG) [Rozenknop 99], auquel il ne sera plus fait mention par la suite, mais qui a mené au développement des modèles « gibbsiens »

[9]MPP : Most Probable Parse = Analyse la plus probable

présentés aux chapitres 5 et 3. Il apparaît comme un modèle hybride entre les STSG dont il partage le critère d'apprentissage ML, et les GTSG, dont il partage la forme des paramètres.

L'objectif à l'origine de ce modèle était de pouvoir définir des paramètres d'une CFG compatibles avec les probabilités SCFG de génération des arbres d'un corpus, mais qui ne soient pas normalisés comme des probabilités, de façon à éviter que les « scores » des gros arbres soient ridiculement faibles par rapport à ceux des petits (ce qui pose problème lorsque l'on utilise ces modèles en reconnaissance de la parole). Le modèle TCFG répond à cette exigence, car les probabilités de génération n'y apparaissent que lorsque l'on *contraint* l'ordre dans lequel les règles s'appliquent (i.e. de la racine vers les feuilles). Pour l'apprentissage des potentiels, on peut considérer que le corpus a été produit par un processus stochastique, dont on calcule les probabilités à l'aide du critère ML. On en déduit alors les potentiels λ par la formule $p(X \rightarrow \alpha) = C_X e^{\frac{\lambda}{T}}$, où T et C_X sont des constantes. En analyse, le « score » des arbres est ensuite pris comme son potentiel, qui est la somme des potentiels de ses règles.

Le modèle TCFG a été effectivement implémenté, en utilisant quelques méthodes heuristiques pour fixer les valeurs des constantes. Les modèles obtenus se sont révélés équivalents aux modèles SCFG du point de vue de leurs performances en analyse syntaxique. Les TCFG pourraient s'avérer utiles en reconnaissance de la parole. On peut d'ailleurs remarquer que les constantes déterminées empiriquement joueraient exactement le rôle des constantes α, β utilisées lors du mélange des modèles de langage aux modèles acoustiques[10] ; elles sont juste plus nombreuses, et leur calcul empirique nécessite plus d'efforts.

Ce modèle permet de construire une vision théorique à peu près cohérente de l'utilisation de processus génératif en analyse, ou en reconnaissance de la parole. Il est similaire dans sa forme aux GCFG, auxquelles est dédiée la section 3.2, et les algorithmes d'apprentissage d'une partie de ses paramètres sont ceux des SCFG, bien connus. Son intérêt se restreint à ses objectifs initiaux : l'utilisation d'une grammaire hors-contexte en reconnaissance de la parole. Comme il n'apporte pas de nouveauté ni d'amélioration dans les systèmes purement d'analyse syntaxique, et qu'il reste à spécifier des méthodes pour fixer judicieusement certains de ses paramètres (C_X), ce modèle ne sera pas exposé plus en détail dans ce document : il apparaît ici uniquement comme un modèle intermédiaire, qui a inspiré le travail sur les GCFG.

[10] on a déjà fait allusion à ces constantes p.18

1.3 Objectifs

Les buts de ce travail sont les suivants :

- corriger les biais observés en analyse syntaxique dus aux approches génératives de l'apprentissage des modèles de langage stochastiques,
- rendre *possible* l'utilisation des grammaires à substitution d'arbres en analyse syntaxique, i.e. rendre possible l'extraction de PTSG pour lesquelles l'analyse syntaxique peut se faire en temps polynomial par rapport à la taille des données.

1.4 Structure du document

Le chapitre 2 donne un éventail non-exhaustif des différentes grammaires probabilistes étudiées dans la littérature. On y introduit en particulier les deux grammaires dont les extensions font l'objet de cette thèse : les grammaires hors-contextes probabilistes (SCFG) et les grammaires à substitution d'arbres probabilistes (STSG).

Le chapitre 3 donne une version probabiliste non générative des grammaires hors-contextes, les GCFG. On y détaille les notions de Maximum de vraisemblance et de Maximum d'entropie, utilisées comme critères d'apprentissage de leurs paramètres, ainsi que les algorithmes permettant de réaliser cet apprentissage à partir d'un corpus d'analyses. Le modèle obtenu supprime certains biais observés avec une SCFG, ce qui y est illustré sur un exemple jouet, tiré d'un article de M. Johnson [Johnson 98]. Du point de vue statistique, les résultats en analyse sont légèrement meilleurs que ceux obtenus avec une SCFG.

Le chapitre 4 détaille le modèle STSG DOP, et son comportement parfois indésirable en analyse, signalé par K. Sima'an [Sima'an 96b]. Dans le cas des STSG, ce comportement est dû à l'heuristique utilisée pour le calcul de leurs paramètres. K. Sima'an [Sima'an 99] propose une autre heuristique qui corrige en partie les défauts qu'il signale. Son analyse a été augmentée par la proposition d'une troisième méthode, à base de maximum de vraisemblance. Nous évoquons, dans une deuxième section, divers travaux tentant de répondre au problème de la complexité de l'analyse syntaxique soulevé par les STSG.

Nous donnons dans le chapitre 5 notre version probabiliste non générative des grammaires à substitution d'arbre, les GTSG. Il faut noter que le passage d'une STSG à une GTSG est substantiellement plus difficile qu'avec celui me-

nant des SCFG aux GCFG, du fait que le corpus d'apprentissage, constitué d'arbres d'analyses, ne donne pas explicitement la liste des arbres élémentaires utilisés dans les substitutions : les paramètres que l'on cherche à définir ne sont pas attachés à des éléments directement observables dans le corpus. L'apprentissage ne peut être totalement supervisé, et l'on est confronté au problème de l'évaluation de parties cachées du modèle.

Le chapitre 6 s'attache au problème de la complexité de l'analyse dans le cadre des modèles à substitution d'arbre. Nous y caractérisons un type de grammaires, dites grammaires à substitution d'arbres polynomiales, qui permettent d'effectuer des analyses en un temps polynomial en fonction de la taille de la phrase analysée. La solution trouvée passe par la sélection des arbres élémentaires constituant la grammaire. Nous définissons également une méthode pratique de sélection de ces arbres élémentaires à partir d'un corpus.

Les conclusions de ce travail et les perspectives qu'il ouvre sont données au chapitre 7.

Nous donnons également en annexes quelques compléments : l'annexe A est la démonstration de K. Sima'an de la NP-difficulté du problème de l'analyse syntaxique avec une PTSG. L'annexe B, librement inspirée de [Goodman 98], décrit l'algorithme Inside-Outside, essentiel dans l'apprentissage des paramètres de nos modèles non-génératifs.

Chapitre 2

Grammaires probabilistes

2.1 Grammaires hors-contextes

Avant de s'intéresser aux formalismes des grammaires probabilistes, nous donnons une rapide introduction à la théorie des grammaires hors-contextes (CFG : Context-Free Grammar). « Le concept d'un langage hors-contexte a été introduit par Chomsky en 1959 [Chomsky 59] dans le but de trouver un modèle mathématique raisonnable pour des langages naturels, tels que l'anglais, le français, etc. » [Ginsburg 66].

Formellement, une CFG est un quadruplet (V_N, V_T, P, S), où V_N et V_T sont les symboles de la grammaire, S est le symbole *initial*, ou *axiome*, et P est un ensemble de règles de productions (ou règles de réécritures). Les symboles de V_N sont appelés *non-terminaux*, et peuvent être récrits par l'application des règles de P. Les symboles de V_T sont appelés *terminaux* et ne peuvent être récrits. Les règles de P sont de la forme $A \rightarrow \alpha$, où A est un non-terminal et $\alpha \in (V_N \cup V_T)^*$ est une chaîne de symboles grammaticaux. A est appelé *partie gauche* ou *tête* de la règle (LHS : left-hand side), et α *partie droite* ou *queue* (RHS : right-hand side).

Une chaîne $\alpha A \gamma$ peut être récrite par $\alpha\beta\gamma$, ce qu'on note $\alpha A\gamma \Rightarrow \alpha\beta\gamma$, si la règle $A \rightarrow \beta$ existe. Une *dérivation* est une séquence de telles réécritures. $\Rightarrow^*$ désigne la fermeture réflexive et transitive de la relation de dérivation. Une *leftmost derivation* (dérivation par la gauche) est une dérivation dans laquelle chaque étape récrit le symbole non-terminal le plus à gauche de la chaîne.

Le *langage* $L(\mathcal{G})$ d'une grammaire $\mathcal{G}$ est l'ensemble des chaînes de terminaux pouvant être dérivées du symbole initial S :

$$L(\mathcal{G}) = \{x \int V_T^* | S \Rightarrow^* x\}$$

Donnons un exemple de grammaire hors-contextes.

S est le symbole initial de la grammaire hors-contexte suivante :

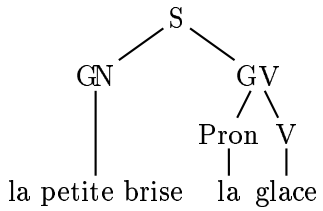


FIG. 2.1 – Exemple d'arbre de dérivation dans une CFG

$S \rightarrow GN\ GV$	$V \rightarrow$ brise
$GV \rightarrow V\ GN$	$V \rightarrow$ glace
$GV \rightarrow Pron\ V$	$V \rightarrow$ agace
$GN \rightarrow$ la petite	$Pron \rightarrow$ la
$GN \rightarrow$ la glace	$Pron \rightarrow$ l'
$GN \rightarrow$ la petite brise	

Les non-terminaux de la grammaire sont S, GN, GV, V, $Pron$. Les terminaux sont « brise », « glace », « agace », « la petite », « la », « la glace », « l' » et « la petite brise ». Une dérivation à gauche dans cette grammaire peut être :

$$\begin{aligned} S &\Rightarrow GN\ GV \\ &\Rightarrow \text{la petite brise} GV \\ &\Rightarrow \text{la petite brise} Pron\ V \\ &\Rightarrow \text{la petite brise la } V \\ &\Rightarrow \text{la petite brise la glace} \end{aligned}$$

L'arbre de dérivation correspondant est donné par la figure 2.1. De cette grammaire peuvent être dérivées d'autres chaînes, comme « la petite brise l'agace », par exemple.

2.2 Grammaires hors-contextes probabilistes

Les grammaires hors-contextes probabilistes (PCFG : Probabilistic Context-Free Grammars, ou SCFG : Stochastic Context-Free Grammar) sont les plus standards et les mieux comprises des grammaires probabilistes. En général, les PCFG sont définies comme des grammaires hors-contextes étendues par une fonction qui assigne une probabilité à chaque règle. Formellement, une *grammaire hors-contexte probabiliste* est un *quintuplet* (V_N, V_T, S, P, π), où (V_N, V_T, S, P) est une grammaire hors-contexte, et $\pi : P \rightarrow [0, 1]$ est une fonction assignant une probabilité à chaque règle de telle façon que pour chaque symbole non-terminal $A \in V_N$:

$$\sum_{A \rightarrow \alpha \in P} \pi(A \rightarrow \alpha) = 1$$

La probabilité d'un arbre de dérivation de la grammaire est définie comme le produit des probabilités des sous-arbres apparaissant immédiatement sous son nœud supérieur et de la probabilité de la règle de réécriture appliquée au non-terminal du nœud supérieur. Formellement, étant donné un arbre de dérivation $\sigma = \sigma(\sigma_1 \dots \sigma_n)$, $\sigma \in P$, et σ_i des arbres de dérivations, la probabilité de la dérivation est donnée par :

$$\begin{aligned} p(\sigma()) &= \pi(\sigma) \\ p(\sigma(\sigma_1 \dots \sigma_n)) = \pi(sigma) \prod_{i=1}^{n} p(\sigma_i) \end{aligned}$$

Si plus d'une dérivation produisent la même chaîne, la somme des probabilités de toutes les dérivations possibles est prise comme la probabilité de la chaîne :

$$p(w) = \sum_{\sigma \in Arbres(\mathcal{G})/feuilles(\sigma)=w} p(\sigma)$$

où $Arbres(\mathcal{G})$ est l'ensemble des arbres que la grammaire $\mathcal{G}$ peut produire. De façon similaire, on peut associer à une chaîne non pas dans la somme des probabilités de ses dérivations, mais la probabilité maximale parmi les probabilités de ses dérivations.

Une des propriétés des PCFG est que la probabilité d'une phrase décroît avec sa longueur. Cela correspond à l'intuition que les longues phrases ont une plus petite probabilité d'occurrence. Comme on le verra, cette propriété peut être indésirable dans certaines applications en langage naturel. Pour une discussion détaillée de l'aspect naturel des PCFG, on peut se référer à [Suppes 72]. Les aspects théoriques des PCFG sont détaillés dans [Grenander 67, Booth 73, Wetherell 80, ter Doest 94].

2.2.1 Apprentissage supervisé

Les PCFG peuvent facilement être inférées à partir d'un corpus arborisé. Étant donné un ensemble $E = \{e_1, e_2, \dots, e_N\}$ d'arbres (le corpus arborisé, ou *treebank*), la probabilité d'une règle $A_i \rightarrow \alpha_j$, où $A_i \in V_N$ et $\alpha_j \in (V_N \cup V_T)^*$, est définie par :

$$\pi(A_i \rightarrow \alpha_j) = \frac{n(A_i \rightarrow \alpha_j)}{n(A_i)}$$

où $n(A_i)$ représente le nombre d'occurrences du symbole A_i dans la treebank, et $n(A_i \rightarrow \alpha_j)$ représente le nombre d'occurrences de la règle $A_i \rightarrow \alpha_j$.

Une PCFG dont les probabilités des règles sont estimées par cette formule induit la distribution de probabilité sur les arbres qui possède le maximum de vraisemblance de générer les données de la treebank ; la preuve en est donnée dans [Fu 75]. Abney [Abney 97a] utilise le terme de « Expected Rule Frequency » (ERF) pour cette estimation.

2.2.2 Apprentissage non-supervisé

On peut inférer les paramètres d'une PCFG par le biais de l'algorithme *Inside-Outside* [Lari 90]. L'algorithme Inside-Outside est une généralisation de l'algorithme de Baum-Welch utilisé pour l'apprentissage des paramètres des Modèles de Markov Cachées (HMM : Hidden Markov Models) [Baum 72]. Charniak [Charniak 93] expose comment l'algorithme Inside-Outside peut être dérivé de l'algorithme de Baum-Welch.

L'idée de l'algorithme Inside-Outside est d'utiliser les probabilités courantes des règles de la grammaire pour estimer à partir d'un échantillon incomplet les probabilités des dérivations, puis de mettre à jour les probabilités des règles, avec une fréquence déterminée. Chaque itération de l'algorithme commence par calculer les probabilités *inside* et *outside* de toutes les phrases de l'échantillon. Ces probabilités sont en fait des fonctions ayant comme arguments une phrase w de l'échantillon, des indices indiquant la partie de la phrase considérée, et un non-terminal A_k. Avec ces arguments la probabilité *inside* $I_w(i, j, A_k)$ est la probabilité que A_k se récrive en $w_{i+1} \dots w_j$. La probabilité *outside* est la probabilité que la forme $w_1 \dots w_i A_k w_{j+1} \dots w_{|w|}$ puisse être dérivée du symbole initial.

Les probabilités des règles convergent vers des valeurs telles que la PCFG présente une probabilité localement maximale d'engendrer l'échantillon.

L'algorithme de ré-estimation peut être utilisé pour affiner les probabilités des règles d'une PCFG existante, ou pour calculer ces probabilités à partir de zéro. La première utilisation est dite *incrémentale*. Dans le second cas, la grammaire initiale est composée de toutes les règles CNF[1] possibles étant donnés les ensembles des non-terminaux et des terminaux. Le processus doit alors être initialisé avec des probabilités non-nulles appropriées (tirées aléatoirement ou équiprobables).

2.2.3 Discussion

Il est actuellement reconnu que les PCFG manquent de pouvoir de représentation dans un sens probabiliste, et qu'elles ne sont pas suffisamment flexibles pour incorporer des informations probabilistes autres que les probabilités des règles.

Sensitivité au contexte Pour la désambiguïsation de phénomènes tels que l'attachement des groupes prépositionnels et les dépendances à longue distance, la portée des probabilités des règles est trop petite. Dans les PCFG, la probabilité d'une règle exprime la probabilité qu'elle soit utilisée pour récrire son symbole de gauche ; ceci est l'hypothèse hors-contexte. Mais si l'on veut modéliser statistiquement un phénomène tel que l'attachement

[1] Une règle CNF (Chomsky Normal Form) est une règle ayant pour partie droite exactement deux non-terminaux, ou exactement un terminal.

d'un groupe prépositionnel, on a besoin de modéliser des caractéristiques des arbres syntaxiques allant au-delà de ces règles hors-contextes. Un grand nombre de travaux sur les modèles de langages probabilistes se concentrent sur des statistiques plus riches, dépendantes du contexte. Une amélioration peut être de conditionner les probabilités des règles par des caractéristiques des arbres syntaxiques externes à ces règles. La section 2.3 relate certaines de ces approches. Une autre solution fondamentale est d'abandonner les grammaires hors-contextes, et d'utiliser des formalismes grammaticaux différents. Les sections 2.5 à 2.7 donne une vue des efforts faits dans ce sens.

Information lexicale Les PCFG ne sont pas flexibles quant à l'incorporation d'informations statistiques autres que les fréquences de règles. Par exemple, les statistiques de co-occurrence (n-grammes) de mots ou de catégories syntaxiques contiennent des informations importantes sur les dépendances lexicales. L'incorporation dans une PCFG de telles informations obscurcit la vue qu'on peut avoir de ce qui est réellement modélisé par le modèle statistique obtenu : « A difficulty with the hybrid approaches, however, is that they have unclear what the statistical model of the language is. In probabilistic context-free grammar and in surface-string analysis of lexical co-occurrence, there is a precise definition of the *event space* – what events go into making up a sentence. (...) The absence of such a characterization in the hybrid approaches makes it more difficult to identify what assumptions are being made, and gives such work a decidedly empirical flavor. » [Resnik 92]. Les formalismes de grammaires d'arbres traitées à la section 2.5 fournissent un cadre pour l'incorporation de statistiques lexicales. Les méthodes à base de maximum d'entropie, traitées à la section 3.1 permettent l'incorporation de sources d'informations arbitraires dans les modèles statistiques.

Nature générative Une hypothèse implicite, dans la définition des PCFG, est que l'application d'une règle change la probabilité de l'arbre de dérivation final. La distribution de probabilité sur le langage d'une grammaire est définie d'une manière générative, une phrase étant produite à partir d'un symbole initial. Si elle est effectivement justifiée lorsque que l'on utilise ce modèle pour produire des phrases d'un langage, cette propriété apparaît en revanche comme une limitation lorsque l'application visée est différente : pourquoi se limiter à des grammaires de type génératif pour faire de l'analyse syntaxique, par exemple ? Les modèles alternatifs présentés dans cette thèse (GCFG et GTSG) sont des exemples de modèles non-génératifs qui apparaissent plus adaptés pour une telle tâche.

2.3 Extension des grammaires probabilistes hors-contextes

Comme on l'a vu dans les sections précédentes, les PCFG ne sont pas assez riches pour capturer des phénomènes qui sont hors de portée des règles de grammaire hors-contextes. Dans cette section, nous donnons quelques approches qui tentent de répondre à ce problème en conditionnant les probabilités des règles sur des informations qui leur sont externes.

2.3.1 Grammaires stochastiques faiblement restreintes

Dans [op den Akker 94], les auteurs proposent un léger enrichissement du formalisme des PCFG dans le but d'obtenir quelque sensitivité au contexte pour les probabilités des règles. Leur idée est d'assigner aux règles des probabilités dépendant de la position où le non-terminal devant être récrit est introduit. Leur argumentation repose sur le fait que la connaissance de la règle qui a introduit le non-terminal améliore la sensitivité au contexte du modèle probabiliste.

Formellement, une grammaire stochastique faiblement restreinte (appelée WRSG pour « Weakly Restricted Stochastic Grammar ») est un quintuplet (V_N, V_T, S, P, π) où (V_N, V_T, S, P) est une grammaire hors-contexte et π une fonction assignant à chaque occurrence d'un non-terminal A_{ij} (qui dénote la $j^{\text{ème}}$ occurrence de A_i en partie droite de règle) une fonction π_{ij} qui assigne à chaque règle de tête A_i une probabilité. La probabilité qu'une règle récrive un non-terminal dépend donc de la règle qui a introduit ce non-terminal.

Une WRSG peut être transformée en une PCFG stochastiquement fortement équivalente. Cela implique que l'augmentation du pouvoir descriptif probabiliste vienne de la taille de la grammaire, et non d'un formalisme plus puissant. Les auteurs donnent également une adaptation de l'algorithme Inside-Outside pour réaliser l'apprentissage des probabilités d'une WRSG directement à partir d'une treebank, ou à partir de phrases du langage.

2.3.2 History-based grammars

Dans [Black 92] est introduit un modèle sophistiqué de grammaire probabiliste, appelé *history-based grammar*[2] (HBG). L'idée est ici que les probabilités sont conditionnées par l'historique des règles de grammaire déjà appliquées, dans une dérivation par la gauche. HBG est un cadre pour des modèles de grammaires probabilistes riches. Les probabilités des règles sont conditionnées par la classe d'équivalence de la séquences des règles précédemment appliquées. Formellement, la probabilité d'une dérivation $\sigma = \sigma_1 \dots \sigma_n, \sigma \in P$, est définie par

[2] Grammaire basée sur l'historique

$$p(\sigma) = \prod_{i=1}^{m} p(\sigma|[\sigma_1 \dots \sigma_{i-1}])$$

où m est le nombre d'étapes de dérivation, et $[\sigma_1 \dots \sigma_{i-1}]$ est la classe d'équivalence des règles de grammaires appliquées précédemment.

La grammaire utilisée pour l'expérimentation avec le modèle HBG est une grammaire écrite manuellement, utilisant l'unification. Les terminaux et non-terminaux sont des paires « type-valeur » similaires à celles du modèle de Goodman ([Goodman 97] (voir section 2.6.2). La différence est que certaines valeurs sont compilées dans la grammaire hors-contexte. La PCFG ainsi construite est alors entraînée sur la base « Lancaster Treebank » par un algorithme Inside-Outside. La grammaire ainsi obtenue est utilisée pour l'évaluation subséquente du modèle HBG, et pour « bootstrapper » l'induction de la HBG.

Pour obtenir la HBG, la PCFG est mise en action sur la Lancaster Treebank. Chaque arbre le plus probable est considéré comme un évènement s'il correspond à l'analyse de la treebank. Chaque nœud de ces arbres est annoté par des caractéristiques syntaxiques (Syn) et sémantiques (Sem), par ses têtes primaires et secondaires, et par la règle qui lui est appliquée lors de sa réécriture. Le modèle proposé associe à chaque nœud de chaque arbre d'analyse la probabilité conditionnelle suivante :

$$p(Syn, Sem, R, H_1, H_2|Syn_p, Sem_p, R_p, I_{pc}, H_{1p}, H_{2p})$$

Les indices p indiquent les caractéristiques du nœud parent, en I_{pc} représente l'index du nœud dans la partie droite de la règle qui est appliquée à son nœud parent. Cette règle est notée R_p. Un arbre de décision statistique (voir [Magerman 94] pour une explication) est alors utilisé pour approximer cette probabilité par la probabilité qu'une règle soit utilisée dans la réécriture d'un nœud dans l'arbre syntaxique :

$$p(R|Syn, Sem, Syn_p, Sem_p, R_p, I_{pc}, H_{1p}, H_{2p})$$

Les performances de la HBG montrent une nette amélioration par rapport aux performances de la PCFG construite lors de la phase de création de l'espace des évènements d'arbres d'analyse.

E. Charniak a donné une méthode pour le calcul des probabilités des règles dans une grammaire HBG, qui repose sur les modèles à maximum d'entropie [Charniak 00] : la probabilité d'appliquer une règle de récriture r conditionnellement à l'historique H est alors de la forme :

$$p(r|H) = \frac{1}{Z(H)} e^{\sum_i \lambda_i f_i(r,H)}$$

où les λ_i sont des poids positifs ou négatifs qui indiquent l'importance relative d'une observation f_i. f_i est une fonction d'observation de (r, H). Par exemple, un f_i possible pourrait valoir 1 si H contient le symbole de tête de r, et 0 sinon. Ce modèle s'apparente aux modèles gibbsiens présentés dans cette thèse, à la différence qu'il reste un modèle génératif : les probabilités $p(r, H)$ sont ensuite combinées par un opérateur multiplicatif pour calculer la probabilité *de produire* un arbre.

2.4 Probabilisation des actions d'un analyseur syntaxique

Une autre approche pour enrichir une grammaire statistique consiste en l'extension de l'analyseur. Chaque algorithme d'analyse exécute un ensemble limité d'actions pour analyser une phrase. Typiquement, les analyseurs LR[3] exécutent les actions « shift » et « reduce »[4], les analyseurs « left-corner » exécutent « shift », « hypothesise » et « complete »[5]. Chacune de ces actions reçoit une probabilité, et la probabilité d'une dérivation peut alors être calculée comme le produit des probabilités des actions qui ont mené à sa construction. Les probabilités des actions peuvent être calculées à partir d'une SCFG initiale, ou induites d'une base d'arbres syntaxiques. Dans cette section sont présentés deux modèles d'analyseurs probabilistes : un analyseur LR compilé à partir d'une SCFG, et un analyseur « left-corner » qui estime les probabilités à partir d'une base d'arbres.

2.4.1 Analyse probabiliste GLR

Wright et Wrigley [Wright 91] étendent l'analyse LR généralisée en compilant les probabilités à l'intérieur des tables d'analyse. Ils décrivent comment les probabilités des SCFG sont transformées en probabilités d'un analyseur LR. Des algorithmes sont exhibés pour des analyseurs SLR, LALR et LR canonique. La transformation repose sur l'idée qu'une transition d'état LR correspond aux actions « shift » et « réduire ».

De plus, les auteurs donnent une interprétation bayésienne des probabilités des actions LR, proposent une solution pour le problème des mots inconnus, et montrent qu'un modèle probabiliste LR a de bien meilleures performances que des modèles de Markov.

[3] Left-Right = Gauche-Droite
[4] « placer » et « réduire »
[5] « placer »,« hypothétiser » et « compléter »

2.4.2 Analyse probabiliste « left-corner »

Manning et Carpenter [Manning 97] présentent un analyseur probabiliste « left-corner » à base de piles[6], dont les actions sont probabilisées. Un tel analyseur utilise une pile pour garder une trace des constituants qu'il a trouvés et de ceux qu'il recherche.

Les actions typiques d'un analyseur « left-corner » sont : *shift*, *attach*, et *lc-project*. Une action *shift* déplace le point derrière une catégorie lexicale, une action *attach* le déplace derrière un constituant reconnu, et une action *lc-project* prédit de nouveaux constituants en utilisant la relation « left-corner » (coin gauche). Pour une explication de l'analyse « left-corner » à base de piles, on peut se référer à [op den Akker 88, Nederhof 94].

La probabilité d'une dérivation « left-corner » est définie comme le produit des probabilités des actions exécutées durant l'analyse. Si les probabilités de chaque action sont conditionnées par l'historique des actions, nous avons :

$$P(t) = \prod_i P(C_i|C_1, \ldots, C_{i-1})$$

Les auteurs reconnaissent que de telles probabilités conditionnelles ne peuvent pas être facilement estimées en pratique. Ils proposent alors deux modèles pour classifier les historiques des actions de l'analyseur. On peut noter que le même genre de problème apparaît avec les grammaires « History-based » (voir section 2.3.2), où la classification est définie en fonction de l'historique des règles appliquées. Dans le modèle le plus simple, l'historique des actions est classifié en fonction de la catégorie « left-corner » courante et de la catégorie du but. Le modèle probabiliste est légèrement plus riche d'après les auteurs que le formalisme SCFG. Des expériences sur la Penn Treebank II exhibent de meilleures performances de ce modèle en comparaison avec une SCFG standard.

Dans le modèle le plus complexe, la classification des historiques intègrent une information supplémentaire, qui est la taille de la pile. Mais aucune expérience n'est relatée sur ce modèle.

2.5 Formalismes d'arbres

Dans cette section sont présentées deux formalismes grammaticaux basés sur des arbres syntaxiques : les grammaires stochastiques à adjonction d'arbres lexicalisés (SLTAG[7]) et les grammaires stochastiques à substitution d'arbres (STSG[8]).

[6] left-corner stack parser

[7] SLTAG : Stochastic Lexicalized Tree Adjunction Grammar

[8] STSG : Stochastic Tree Substitution Grammar.

2.5.1 Grammaire à adjonction d'arbres

Les grammaires stochastiques à adjonction d'arbres lexicalisés, apparaissant dans [Schabes 92], sont des extensions stochastiques des grammaires à adjonction d'arbres lexicalisés (LTAG[9]) [Joshi 92]. On peut se référer à [Joshi 87] pour une introduction aux grammaires à adjonction d'arbres.

Schabes [Schabes 92] donne une définition des SLTAG en définissant d'abord les grammaires stochastiques indexées linéaires (SLIG[10]), et en montrant comment leurs langages stochastiques sont liés. Nous présentons ici une définition plus intuitive des SLTAG, données dans [Resnik 92]. Une LTAG est une paire (I, A), où I est un ensemble d'arbres initiaux, et A un ensemble d'arbres auxiliaires. Chaque arbre auxiliaire possède une feuille dont le symbole correspond à celui du nœud racine. Il est appelé nœud *de pied*[11], et le chemin allant de la racine au pied est appelé *spine*.

Adjonction et substitution sont les deux opérations permettant de construire des arbres plus complexes à partir de I et A. Chaque arbre doit contenir une feuille lexicalisée.

Substitution Chaque arbre α de $I \cup A$ possède un sous-ensemble (potentiellement vide) de feuilles marquées comme pouvant être substituées par un arbre initial ; ce sous-ensemble est noté $s(\alpha)$;

$S(\alpha, \alpha', \eta)$ représente la substitution du nœud η dans l'arbre α par l'arbre α'. S désigne l'ensemble de toutes les substitutions.

Adjonction L'opération d'adjonction remplace un nœud γ d'un arbre par un arbre auxiliaire ; la racine et le pied de cet arbre auxiliaire doivent évidemment porter le même symbole que le nœud remplacé : ce nœud ne peut pas être un nœud substituable. L'ensemble des nœuds d'adjonction d'un arbre α est noté $a(\alpha)$;

$A(\alpha, \beta, \eta)$ représente l'adjonction d'un arbre auxiliaire β dans l'arbre α au nœud η ; $A(\alpha, none, \eta)$ représente un non-évènement : aucun adjonction n'a été effectuée au nœud η ; on note A l'ensemble de toutes les adjonctions.

Les LTAG sont étendues par des statistiques en introduisant les fonctions de probabilité suivantes :

- $P_I : I \rightarrow [0, 1]$ assigne une probabilité à chaque arbre initial, et vérifie :

$$\sum_{\alpha \in I} P_I(\alpha) = 1$$

- $P_S : S \rightarrow [0, 1]$ assigne une probabilité à chaque substitution et vérifie :

$$\forall \alpha \in I \cup A : \forall \eta \in s(\alpha) : \sigma_{\alpha' \in I} P_S(\alpha, \alpha', \eta) = 1$$

[9] LTAG : Lexicalized Tree Adjunction Grammar.
[10] SLIG : Stochastic Linear Indexed Grammar.
[11] *foot node.*

– $P_A : A \rightarrow [0,1]$ assigne une probabilité à chaque adjonction et vérifie :

$$\forall \alpha \in I \cup A : \forall \eta \in s(\alpha) : \sigma_{\beta \in A \cup \{none\}} P_A(\alpha, \beta, \eta) = 1$$

La probabilité d'une dérivation est le produit des probabilités des opérations de substitution et d'adjonction, et de l'arbre initial. Si la dérivation est donnée par $\tau = \sigma_1, \ldots, \sigma_i \in S \cup A$ et que l'arbre initial est α_0, on a alors :

$$p(\tau) = P_I(\alpha_0) \prod_i P(\sigma_i)$$

Schabes [Schabes 92] donne un algorithme pour calculer la probabilité d'une phrase (de complexités $O(n^6)$ en temps, et $O(n^4)$ en taille), dérive un algorithme Inside-Outside pour les SLTAG et décrit des expériences dans lesquelles des SLTAG sont construites automatiquement à partir d'un corpus en utilisant l'algorithme Inside-Outside. Un des avantages des SLTAG sur les SCFG est que l'algorithme d'apprentissage converge plus rapidement. L'auteur soutient que cela est dû au manque de représentation des influences du lexique sur les distributions dans les SCFG, alors que les SLTAG intègrent des statistiques à la fois sur les distributions lexicales et et les distributions hiérarchiques.

Une restriction intéressante du formalisme SLTAG est donné par les grammaires à insertion d'arbres lexicalisés (SLTIG[12]) [Schabes 93]. Les SLTIG sont basées sur les LTIG [Schabes 94] et les LTIG sont obtenues par restriction des LTAG, en interdisant certains arbres auxiliaires. Cette restriction est réalisée si :

- il n'y a pas d'arbre auxiliaire ayant des branches non-vides à la fois à gauche et à droite du spine ;
- un arbre auxiliaire gauche (droite) ne peuvent être adjoints à un nœud situé sur le spine d'un arbre élémentaire droite (gauche) ;
- il n'y a pas d'adjonction autorisée à droite (gauche) du spine d'un arbre auxiliaire gauche (droite).

Les LTIG lexicalisent les CFG sans changer l'ensemble des arbres qui peuvent être dérivés. De plus, Schabes et Waters [Schabes 94] donnent une construction pour transformer une CFG quelconque en une LTIG.

L'analyse avec une SLTIG peut se faire en temps $O(n^3)$, et l'apprentissage Inside-Outside d'une SLTIG en temps $O(n^4)$. Les auteurs indiquent que l'on peut construire un algorithme Inside-Outside de complexité $O(n^3)$ en temps.

2.5.2 Data Oriented Parsing

Avec le Data Oriented Parsing[13] (DOP) [Bod 97, Bod 98] sont définies les grammaires stochastiques à substitution d'arbres (STSG[14]), qui généralisent

[12] SLTIG : Stochastic Lexicalised Tree Insertion Grammar.
[13] analyse syntaxique basée sur les données
[14] STSG : Stochastic Tree Substitution Grammar.

les SCFG. Une STSG est une PCFG dont l'ensemble des règles est remplacé par un ensemble d'arbres. Formellement, une STSG est un quintuplet (V_N, V_T, S, T, π), où

- V_N est un ensemble fini de non-terminaux ;
- V_T est un ensemble fini de terminaux ;
- $S \in V_N$ est le symbole initial (ou axiome) ;
- T est un ensemble fini d'arbres élémentaires ; un arbre élémentaire est un arbre dont le nœud racine et les nœuds et intérieurs sont des non-terminaux, et dont les feuilles sont des symboles terminaux ou non-terminaux ; $root(t)$ désigne le non-terminal racine de l'arbre t ;
- π est une fonction qui assigne une probabilité à chaque arbre élémentaire, de façon à ce que pour chaque non-terminal A :

$$\sum_{t|root(t)=A} \pi(t) = 1$$

La notion de dérivation est basée sur l'opérateur binaire de substitution $\cdot$. Si t_1 et t_2 sont des arbres, alors $t_1 \cdot t_2$ est l'arbre qui résulte de la substitution de t_2 à la feuille non-terminale[15] la plus à gauche de t_1. Une « dérivation par la gauche » est définie comme une séquence d'arbres élémentaires $(t_1, \ldots, t_n)$ telle que $t_1, \ldots, t_n \in R$, la racine de t_1 porte le label S, et les feuilles de $t_1 \cdot \cdots \cdot t_n$ sont des terminaux. La probabilité d'une dérivation est le produit des probabilités des arbres élémentaires qui la constituent. Un arbre syntaxique peut évidemment posséder de multiples dérivations (par la gauche) ; la probabilité d'un arbre est définie comme la somme des probabilités de ses différentes dérivations. Le langage d'une STSG est défini comme l'ensemble des chaînes pour lesquelles un arbre d'analyse existe ; la probabilité d'une chaîne est la somme des probabilités de ses différents arbres d'analyse, et vaut également la somme des probabilités de ses différentes dérivations.

Les STSG sont faiblement équivalentes aux SCFG, dans le sens où l'ensemble des langages définis par des STSG est le même que celui des langages définis par des SCFG [Bod 98] (p.29). Bod montre que les STSG sont plus puissantes que les SCFG, non seulement parce qu'il est des STSG pour lesquelles n'existe aucune SCFG fortement équivalente, mais aussi *stochastiquement*, lorsque une SCFG et une STSG sont fortement équivalentes [Bod 98] (p.33).

Une STSG peut facilement être définie à partir d'un corpus d'arbres en extrayant tous les sous-arbres qu'il contient, en les comptant et en normalisant par le nombre total de sous-arbres ayant la même racine. Les STSG deviennent évidemment très grandes, comparées à des SCFG extraites du même corpus d'arbres. Pour 'contenir' la taille d'une STSG, on peut restreindre le type d'arbres élémentaires à extraire. Bod applique des restrictions

[15] Une feuille non-terminale est une feuille portant un symbole non-terminal, appartenant à V_N.

sur la profondeur des arbres élémentaires. La grammaire reste cependant très grande, et l'analyse une opération coûteuse. Alors que, souvent, la taille de la grammaire n'est pas prise en compte dans le calcul de la complexité temporelle de l'analyse, du fait qu'elle est constante, cette constante peut ici devenir tellement grande qu'elle domine le facteur relatif à la longueur de la phrase.

Bod définit l'analyse syntaxique comme une procédure d'échantillonnage. Si durant l'analyse, montante ou descendante, plusieurs sous-arbres peuvent se substituer au même nœud, alors l'analyseur sélectionne un sous-arbre par échantillonnage, le substitue, et continue.

Goodman [Goodman 96] a montré que pour chaque STSG, on peut construire une SCFG fortement équivalente, dans le sens où pour chaque dérivation dans la STSG, il existe une dérivation dans la SCFG qui a la même probabilité, et vice-versa. Il présente alors un algorithme d'analyse de complexité temporelle $O(n^3)$. L'analyse ne se fait pas alors avec le critère de la dérivation, ni de l'arbre le plus probable, mais selon le critère du *Maximum de Constituants*[16] : on peut avec cette grammaire extraire l'analyse dont l'*espérance du nombre de constituants corrects* est maximale.

Collins [Collins 02] a donné une méthode des plus intéressantes pour estimer les paramètres d'un modèle DOP, qui repose sur l'algorithme du « *Voted Perceptron* ». Cette méthode se rapproche de très près du modèle « GTSG » présenté au chapitre 5, en ceci qu'il produit un modèle discriminant : il optimise les paramètres du modèle de façon à favoriser les bonnes solutions en analyse syntaxique *par rapport aux mauvaises*. Il permet de plus d'utiliser la base d'apprentissage elle-même comme modèle, sans avoir à passer par une représentation intermédiaire (la liste de tous les sous-arbres de la base d'apprentissage) qui peut être trop coûteuse en place mémoire. À l'aide des paramètres ainsi extraits, on peut alors faire une analyse syntaxique en calculant un score pour chaque arbre compatible avec la phrase à analyser. Comme le nombre de ces arbres peut être très grand, cela nécessite un préfiltrage, en choisissant de ne regarder par exemple que les N meilleures solutions proposées une SCFG.

Si l'on compare les formalismes STSG et SLTAG, on constate que le formalisme SLTAG est plus puissant du fait de l'opération d'adjonction, les SLTAG pouvant, en théorie, capturer autant de dépendances stochastiques que les STSG [Bod 98] (pp.36-37). Dans les deux formalismes, les statistiques de co-occurrences ou autres statistiques lexicales peuvent facilement être représentées. La principale différence tient aux philosophies qui les ont motivés. Dans la recherche sur DOP, le formalisme STSG est appliqué pour utiliser un corpus de données annotées comme un modèle de performance du langage humain, alors que le formalisme TAG est linguistiquement motivé pour le développement « manuel » de grammaires par des experts.

[16] Maximum Constituant Parse.

2.6 Grammaires d'unification

Étendre une grammaire d'unification par des statistiques n'est pas simple. En principe, on ne peut attacher des probabilités aux règles d'une grammaire d'unification de la même façon qu'on le fait avec les SCFG. Du fait que l'unification ne peut pas toujours se faire, de telles probabilités induites, et les distributions qu'elles impliqueraient, seraient invalides. Cependant, on trouve dans la littérature des expériences fructueuses, qui appliquent de telles probabilités aux règles.

2.6.1 Analyse probabiliste LR avec grammaire d'unification

Dans [Carroll 92] et [Briscoe 93] est décrite une approche probabiliste de l'analyse syntaxique par des grammaires d'unification. Briscoe et Carroll ont utilisé la grammaire ANLT[17] à large couverture, contenant environ 800 règles, et un lexique d'environ 64000 entrées construit à partir du LDOCE[18]

Les caractéristiques les plus importantes de la grammaire sont compilées dans son « ossature » hors contexte, à partir de laquelle un analyseur LR non-déterministe est construit. Les probabilités sont assignées aux transitions dans la table d'action, par un processus d'apprentissage supervisé, basé sur le calcul de la fréquence avec laquelle ces transitions sont utilisées dans un corpus d'« historiques d'analyses » construit par un humain à l'aide d'une version interactive de l'analyseur [Carroll 92].

Un analyseur LR muni de probabilités sur les transitions entre états peut distinguer différents arbres de dérivation créés à partir des mêmes règles hors-contextes. Cela n'est pas le cas pour une SCFG, dans laquelle la probabilité d'un arbre de dérivation est indépendante de l'ordre dans lequel les règles ont été appliquées.

2.6.2 Grammaires de traits probabilistes

Dans les grammaires de traits probabilistes (les PFG[19], introduites par Goodman [Goodman 97], les terminaux et les non-terminaux sont des vecteurs de traits. Typiquement, une règle binaire PFG ressemble à :

$$(a_1 \dots a_g) \rightarrow (b_1 \dots b_g)(c_1 \dots c_g) \tag{2.1}$$

Les traits sont instanciés un par un, et chaque instanciation est considérée comme un évènement. Deux types d'évènements sont distingués : les *évènements binaires* et les *évènements initiaux*. Un évènement binaire est

[17] Alvey Natural Language Tools.
[18] Longman Dictionary of Contemporary English.
[19] PFG : Probabilistic Feature Grammars.

l'application d'une règle telle que (2.1). Un évènement initial est l'introduction de la racine d'un arbre. Les deux types sont modélisés de façon probabiliste par des « *EventProb* »s. Un EventProb est un triplet $e = (K, N, F)$, où :

- K est un ensemble de traits conditionnants (les traits connus) ;
- $N = N_1, N_2, \ldots, N_n$ est une liste ordonnée de traits conditionnants (les nouveaux traits) ;
- $F = f_1, f_2, \ldots, f_n$ est une liste ordonnée de fonctions ;
  la valeur de $f_i(n_i, k_1, \ldots, k_k, n_1, \ldots, n_{i-1})$ est la probabilité conditionnelle $P(N_i = n_i | K_1 = k_1, \ldots, K_k = k_k, N_1 = n_1, N_{i-1} = n_{i-1})$, c'est-à-dire la probabilité pour que le trait N_i reçoive la valeur n_i, étant donnés les traits connus $K_1, \ldots, K_k$ et les nouveaux traits d'index inférieurs $N_1, \ldots, N_{i-1}$.

Avec la règle donnée par (2.1), un évènement binaire est représenté par :

$$e_b = (\{a_1 \ldots a_g\}, (b_1 \ldots b_g, c_1 \ldots c_g), F_b)$$

ce qui signifie que le trait-fils $b_1 \ldots b_g$ et $c_1 \ldots c_g$ sont conditionnés par les traits-enfants antérieurs et par tous les traits-parents $a_1 \ldots a_g$. Un évènement initial est exprimé par :

$$e_s = (\{\}, (a_1 \ldots a_g), F_s)$$

ce qui signifie que les traits-parents se conditionnent réciproquement.

Comme le nombre de probabilités conditionnelles devant être estimées devient facilement gigantesque, le problème de la disparité des données se pose sérieusement. Goodman utilise du *lissage* pour répondre à ce problème. Le lissage assure qu'un évènement apparaissant avec une fréquence nulle dans un espace d'évènements reçoivent une probabilité non nulle dans le modèle.

Le papier expose l'algorithme Inside-Outside pour les PFG, et décrit des expériences sur la Penn Treebank II. Des performances comparables à celles l'état de l'art y sont rapportées.

2.6.3 Grammaires stochastiques attribut-valeur

Une approche très avancée des grammaire stochastique s'inspire de la théorie des champs aléatoires (*random field theory*), appliqués dans [Mark 91] au mélange de modèles de grammaires hors-contextes et de bigrammes lexicaux, et dans [Della Pietra 97] à l'orthographe. La théorie des champs aléatoires est bien connue en analyse bayésienne d'images [Winkler 95]. Il est expliqué dans [Abney 97b] comment l'idée générale de l'induction des champs aléatoires peut être appliquée à l'estimation des paramètres d'une grammaire stochastique attribut-valeur (SAVG[20]).

[20]SAVG : Stochastic Attibute-Value Grammar.

Une grammaire attribut-valeur (AVG), selon Abney, est une grammaire hors-contexte possédant des labels d'attributs et des équations de chemin. Les labels d'attributs identifient de manière unique les symboles de la partie droite des règles. Les équations de passage spécifient les chemins dans l'arbre de dérivation qui mènent au même nœud-enfant. Les AVG peuvent se représenter par des structures de traits sur un ensemble de types de catégories syntaxiques, et de labels de traits sous forme d'entiers qui représentent l'ordre des nœuds-enfants.

Abney explique que l'estimation par maximum de vraisemblance des SCFG ne peut pas être appliquée aux grammaires attribut-valeur, aux grammaires d'unification, aux HPSG ni aux autres grammaires basées sur l'unification, car l'application des règles ne vérifie pas les critères d'indépendance nécessaires. Il propose l'application des modèles à maximum d'entropie (MaxEnt) avec des propriétés de treillis. Ces propriétés détectent des sous-structures des structures de trait, et de nouvelles propriétés peuvent être construites à partir des structures existantes par combinaison des structures détectées.

2.7 Extensions de Logique à base de Contraintes

Les grammaires hors-contextes peuvent aisément être spécifiées dans un langage de programmation logique tel que Prolog. De plus, l'analyse syntaxique au sens d'une grammaire est équivalente à la requête d'un but correspondant au nœud supérieur d'un arbre de dérivation. Les grammaires de clauses définies (DCG[21]) sont basées sur certaines définitions Prolog qui facilitent encore la définition de grammaires [Pereira 80]. La spécification de grammaires attribut-valeur et de grammaires d'unification est également immédiate. Dans cette section, nous donnons des extensions probabilistes de formalismes grammaticaux basés sur un langage contraint sous-jacent.

2.7.1 CUF probabiliste

Eisele [Eisele 94] définit une extension probabiliste du *Comprehensive Unification Formalism*[22] (CUF) [Dörre 93, Dörre 94]. CUF est un langage déclaratif pour décrire un langage naturel.

Eisele indique que les définitions CUF peuvent s'exprimer de façon relationnelle comme des clauses de la forme :

$$r \leftarrow q_1 \wedge \cdots \wedge q_n \wedge \phi \tag{2.2}$$

où $r, q_1, \ldots, q_n$ sont des prédicats avec différentes variables en arguments, et ϕ une contrainte, dans le langage de contrainte sous-jacent (par exemple

[21] DCG : Definite Clause Grammars.
[22] formalisme d'unification élargi

Prolog), liée à ces variables. Il propose d'associer à ces clauses des probabilités de façon à ce que les probabilités de toutes les clauses d'un même prédicat somment à 1. La probabilité d'une preuve, la contrepartie intuitive d'une dérivation, est alors le produit des probabilités des clauses qui la constituent. Du fait de la possibilité d'échecs, les probabilités de toutes les preuves d'un but particulier ne somment pas, en général, à 1.

Pour entraîner une spécification P-CUF sur un corpus de requêtes, on applique l'algorithme de Baum-Welch. Il requière des probabilités initiales associées aux clauses de la spécification, puis ré-estime à chaque itération les probabilités des clauses en leur affectant leur fréquence *estimée* normalisée sur le corpus. Cet algorithme est similaire à l'algorithme Inside-Outside.

Riezler [Riezler 96] observe que cette approche n'est correcte que pour le cas hors-contexte. Dans les cas où l'hypothèse d'indépendance des clauses de la spécification CUF est cassée (le ϕ de la forme (2.2) est non vide), l'estimation des fréquences des clauses résulte en un modèle probabiliste de CUF inconsistant.

2.7.2 Logique à base de contraintes pondérées

Dans [Riezler 96] sont également données la syntaxe et la sémantique d'une extension pondérée de la logique à base de contraintes. L'article définit la logique à base de contraintes, et donne formellement pour cette logique des spécifications de clauses définies. L'extension quantitative des spécifications de clauses définies est ensuite donnée, comme une annotation des clauses définies.

$$r \leftarrow_f q_1 \wedge \cdots \wedge q_n \wedge \phi$$

où $f \in [0,1]$. Le poids d'une preuve est défini comme le minimum des poids des clauses utilisées dans la preuve. Si plusieurs preuves existent, le poids maximum est choisi. Ainsi, les arbres usuels **et/ou** pour la représentation de toutes les preuves possibles deviennent des arbres **min/max**.

L'application que Riezler a en tête est la spécification de grammaires à base de clauses définies pondérées. Dans cette perspective, l'interprétation **min/max** des preuves est fortement liée à l'interprétation **min/max** des arbres de dérivation des grammaires floues [Lee 69].

Deux interprétations possibles de l'extension pondérée sont présentées. L'une des interprétations possibles est que le poids d'un arbre de preuve donne le « degré de grammaticalité » ; cette interprétation correspond clairement à la notion de « degré d'appartenance » en algèbre floue des ensembles [Zadeh 65]. Et l'autre interprétation, plus courante, est de considérer les poids comme des valeurs de préférence. Cependant, le papier n'apporte pas d'information sur le problème de l'estimation des paramètres. L'auteur, conscient de ce problème, écrit : « Regarding the interest in computational

linguistics problems such as ambiguity resolution, however, a necessary prerequisite for a more sophisticated semantics for probabilistically interpreted quantitative CLP is the development of a probabilistic model for CLP which allows for correct parameter estimation from empirical data. » (p.364). Le travail de Riezler présenté dans la prochaine section répare ce défaut en définissant un modèle probabiliste basé sur des modèles à maximum d'entropie.

2.7.3 Logique probabiliste à base de contraintes

Dans [Riezler 98b] et [Riezler 98a], une extension probabiliste de la programmation logique à base de contraintes est présentée, qui s'appuie sur la méthode MaxEnt (Maximum d'entropie). L'idée est d'induire une distribution de probabilités qui donne la probabilité d'une preuve conditionnée par une requête. Riezler introduit un apprentissage non supervisé de programmes logiques à base de contraintes probabilistes (CLP[23]), ainsi qu'une version révisée de l'Improved Iterative Scaling pour des données incomplètes ; cette version est nommée Iterative Maximization (IM). Les données sont incomplètes, car si plus d'une preuve est possible pour une requête, leurs fréquences respectives sont inconnues. L'algorithme IM tient compte de cette incomplétude.

L'espace des évènements pour IM est défini comme suit. Y désigne l'ensemble des requêtes observées, et X l'ensemble des preuves. Soit $c(y)$ le nombre de fois où y est observé, et $proofs(y)$ l'ensemble des preuves pour y compte tenu du programme P. Chaque $x \in X$ est une preuve pour une requête $y \in Y$. Le nombre d'occurrences d'une preuve x de y est inconnu.

Deux distributions de probabilités sont alors définies, l'une sur X et l'autre sur Y, notées p_X et p_Y respectivement. p_X est une distribution MaxEnt, et p_Y dépend de p_X comme suit :

$$p_Y(y) = \sum_{x \in proofs(y)} p_X(x)$$

Étant donné le programme P, l'algorithme IM détermine les paramètres $\alpha \in \mathbb{R}^k$ de p_X tels que la vraisemblance d'observer Y est maximisée :

$$L(\alpha) = \prod_{y \in Y} p_Y(y)^{c(y)}$$

L'algorithme IM est donné par l'itération :

$$\alpha^{(n+1)} = M(\alpha^{(n)}$$

[23]CLP : Constraint Logic Programming

où

$$\begin{aligned} M(\alpha) &= \hat{\gamma} + \alpha \\ \hat{(}\gamma) &= \operatorname*{Argmax}_{\gamma \in \mathbb{R}^k} A(\gamma, \alpha) \\ A(\gamma, \alpha) &= \sum_{y \in Y} (1 + p(x|y)[\gamma \cdot f] - p_X[\sum_i \bar{f}_i e^{\gamma_i f_\#}]) \\ p(x|y) &= \frac{p_\lambda(x)}{\sum_{x \in proofs(y)} p_X(x)} \end{aligned}$$

La fonction auxiliaire A est une limite inférieure du gain de vraisemblance[24] si les paramètres α sont ajustés par γ. Cette ré-estimation est très similaire à celle de l'algorithme IIS. Dans les deux algorithmes, une fonction auxiliaire, servant de limite inférieure à la différence de vraisemblance, est maximisée.

[24] c'est-à-dire de $L(\gamma + \alpha) - L(\alpha)$.

Chapitre 3

GCFG : Modèles de grammaires hors-contextes à Maximum d'Entropie

Ce chapitre expose le premier modèle de grammaire non-générative défini dans cette thèse (qui est aussi le plus « simple »), nommé GCFG, pour Gibbsian Context-Free Grammar, ou Grammaire hors-contexte de Gibbs [Rozenknop 02b]. Le qualificatif « Gibbsian » est essentiellement dû au fait qu'il s'agit d'un modèle à maximum d'entropie qui s'inspire des champs de Gibbs [Winkler 95]. C'est un modèle probabiliste, très semblable par sa forme à une grammaire hors-contexte stochastique, hormis ses paramètres qui ne sont pas des probabilités mais des « potentiels » pouvant prendre n'importe quelle valeur réelle. La principale nouveauté de ce modèle n'est cependant pas la nature de ses paramètres, mais la manière dont on les calcule à partir d'un corpus.

Avant d'exposer le modèle GCFG, ce chapitre propose une introduction générale aux modèles à maximum d'entropie, et établit les correspondances entre les principes de maximum d'entropie et de maximum de vraisemblance, utiles dans notre recherche de modèles de grammaires non-génératives. L'exposé de ces principes est basé sur [Berger 96].

3.1 Modélisation à maximum d'entropie

On considère un processus aléatoire qui produit en sortie une valeur y, appartenant à un ensemble fini $\mathcal{Y}$ donné. En produisant y, le processus peut être influencé par quelque information contextuelle x, membre d'un ensemble fini $\mathcal{X}$. Dans le cas classique d'une SCFG, le processus produit une règle hors-contexte (par exemple $NP \rightarrow Det\ N$) à partir d'un non-terminal de

tête[1] (NP dans l'exemple) ; $\mathcal{X}$ est alors l'ensemble des non-terminaux de la grammaire, et $\mathcal{Y}$ l'ensemble de ses règles. x est restreint à un non-terminal, qui est aussi la tête de la règle hors-contexte. On peut imaginer étendre le contexte x, par exemple en adjoignant au non-terminal, tête de règle, ses non-terminaux frères dans un arbre syntaxique.

Le problème est de construire un modèle stochastique représentant le comportement du processus aléatoire. Un tel modèle est une méthode d'estimation des probabilités conditionnelles pour que le processus produise y connaissant le contexte x. On notera $p_\Theta(y|x)$ la probabilité que le modèle attribue à y dans le contexte x, Θ représentant l'ensemble des paramètres du modèle. On notera $\mathcal{P}$ l'ensemble des distributions de probabilités conditionnelles. Ainsi, p_Θ est un élément de $\mathcal{P}$.

3.1.1 Base d'apprentissage

Pour étudier le processus aléatoire défini à la section précédente, on observe son comportement, en collectant N échantillons (x_1, y_1), $(x_2, y_2), \ldots,$ (x_N, y_N) supposés indépendants. Dans l'exemple de la SCFG, ces échantillons sont simplement des non-terminaux x associés à des règles de grammaire y de tête x. On peut imaginer que ces échantillons aient été produits par un expert humain à qui on aurait donné un ensemble de phrases et qui en aurait fait l'analyse syntaxique.

On peut résumer la base d'apprentissage par sa distribution de probabilité empirique $\tilde{p}$, définie par :

$$\tilde{p}(x, y) = \frac{1}{N} * \text{nombre d'occurrences de } (x, y) \text{ dans l'échantillon.}$$

3.1.2 Statistique, Indicateurs et Contraintes

Le but est de construire un modèle statistique du processus qui a produit la base d'apprentissage, « résumée » par $\tilde{p}(x, y)$. La construction de ce modèle repose sur un ensemble de statistiques observées dans la base d'apprentissage. Dans le présent exemple, on observe uniquement les statistiques sur la réécriture d'un non-terminal membre de gauche de règle en tous les symboles de membre de droite de règle.

Pour exprimer le fait qu'un non-terminal **NP** se récrit **Det N**, on introduit une fonction indicatrice :

$$f_0(x, y) = \begin{cases} 1 & \text{si } y = \textbf{NP} \rightarrow \textbf{Det N} \text{ et } x = \textbf{NP}, \\ 0 & \text{sinon.} \end{cases}$$

[1] Dénomination :
tête = membre de gauche d'une règle hors-contexte,
queue = membre de droite d'une règle hors-contexte.

On pourrait observer d'autres indicateurs, moins collés au modèle de grammaire hors-contexte, comme par exemple :

$$f_1(x,y) = \begin{cases} 1 & \text{si le premier non-terminal de queue de } y \text{ est } \mathbf{Det} \text{ et } x = \mathbf{NP}, \\ 0 & \text{sinon.} \end{cases}$$

ou encore :

$$f_2(x,y) = f_2(y) = \begin{cases} 1 & \text{si le second symbole de queue de } y \text{ est } \mathbf{N}, \\ & \text{et que le précédent est } \mathbf{Det}, \\ 0 & \text{sinon.} \end{cases}$$

Cette fonction indicatrice peut prendre d'autres valeurs que 0 ou 1. Par exemple, ce pourrait être :

$$f_3(x,y) = \begin{cases} \text{nombre d'occurrences de } \mathbf{Det} \text{ en queue de } y & \text{si } x = \mathbf{NP}, \\ 0 & \text{sinon.} \end{cases}$$

On s'intéresse à l'espérance de f selon la distribution empirique $\tilde{p}(x,y)$. On note cette valeur :

$$E_{\tilde{p}}(f) = \sum_{x,y} \tilde{p}(x,y) f(x,y)$$

On peut exprimer toute statistique des échantillons comme l'espérance d'une fonction indicatrice appropriée f. On appelle une telle fonction un **indicateur**.

Lorsqu'on juge qu'une statistique est importante[2], on peut l'inclure au modèle en imposant à celui-ci de s'accorder avec elle. Ceci est fait en contraignant l'espérance affectée par le modèle à l'indicateur correspondant f. L'espérance de f dans le modèle $p(x,y)$ est :

$$E_p(f) = \sum_{x,y} \tilde{p}(x) p(y|x) f(x,y)$$

où $\tilde{p}(x)$ est la distribution empirique de x dans la base d'apprentissage. On contraint cette espérance à être la même que l'espérance de f dans la base d'apprentissage. C'est-à-dire que l'on requiert l'égalité :

$$E_p(f) = E_{\tilde{p}}(f) \tag{3.1}$$

En combinant les équations précédentes, on obtient :

$$\sum_{x,y} \tilde{p}(x) p(y|x) f(x,y) = \sum_{x,y} \tilde{p}(x,y) f(x,y) \tag{3.2}$$

[2]La *sélection* des indicateurs considérés n'entre pas dans le cadre de cette thèse. Elle peut être faite manuellement par des experts, ou induite automatiquement. Lire [Della Pietra 97] à ce sujet.

L'équation (3.2) est une **contrainte** linéaire sur les fonctions $p(y|x)$, les termes $\tilde{p}(x,y)$, $\tilde{p}(x)$ étant des constantes observées sur le corpus, et f une fonction indicatrice fixée. En se restreignant aux modèles $p(x,y)$ pour lesquels (3.2) est valide, on élimine les modèles qui ne sont pas en accord avec la base d'apprentissage sur le nombre de fois où le processus stochastique devrait produire l'indicateur f.

En résumé, on a un moyen de représenter des phénomènes statistiques inhérents à un échantillon de données ($E_{\tilde{p}}(f)$), ainsi qu'un moyen d'exiger que le processus recrée ces phénomènes ($E_p(f) = E_{\tilde{p}}(f)$).

3.1.3 Principe du Maximum d'Entropie

Supposons que l'on ait n fonctions indicatrices f_i, qui déterminent les statistiques que l'on pense importantes dans la modélisation du processus. On voudrait que le modèle soit en accord avec ces statistiques. C'est-à-dire que l'on voudrait que p soit dans le sous-ensemble $\mathcal{P}_\mathcal{C}$ de $\mathcal{P}$ défini par :

$$\mathcal{P}_\mathcal{C} = \{p \in \mathcal{P} \text{ t.q. } E_p(f_i) = E_{\tilde{p}}(f_i) \text{ pour } i \in \{1,2,...,n\}\} \qquad (3.3)$$

La figure 3.1 donne une interprétation géométrique de ce schéma, et illustre le fait que les contraintes peuvent être inconsistantes dans le cas général, aucune distribution de probabilité ne pouvant alors les satisfaire toutes. Dans le cas présent, cependant, les contraintes linéaires sont extraites du corpus d'apprentissage, et ne peuvent pas, par construction, être inconsistantes. De plus, les contraintes linéaires que l'on va considérer ne sont pas forcées de déterminer $p \in \mathcal{P}$ de façon unique, (figure 3.1 (c)). Au contraire, l'ensemble $\mathcal{P}_\mathcal{C} = \mathcal{P}_{\mathcal{C}1} \cup \mathcal{P}_{\mathcal{C}2} \cup \ldots \cup \mathcal{P}_{\mathcal{C}n}$ des modèles possibles peut être infini.

Parmi les modèles $p \in \mathcal{P}_\mathcal{C}$, le principe de maximum d'entropie consiste en le choix de la distribution la plus uniforme [Berger 96], ou encore la plus ambigue : celle qui introduit le moins d'*a prioris* sur les évènements qui ne font pas partie des fonctions indicatrices choisies. Reste à savoir ce que l'on entend par « uniforme ». Une mesure mathématique de l'uniformité d'une distribution de probabilité conditionnelle $p(y|x)$ est fournie par l'**entropie conditionnelle** :

$$H_p(Y|X) = -\sum_{x,y} \tilde{p}(x)p(y|x)\ln p(y|x)$$

Pour souligner que l'entropie dépend de la distribution de probabilité, on utilisera par abus de notation $H(p)$ à la place de $H_p(Y|X)$. Il faut remarquer que cette mesure dépend aussi de la distribution empirique $\tilde{p}(x)$, x n'étant pas probabilisé par la distribution $p(y|x)$. En réalité, pour coller à la définition classique de l'entropie conditionnelle, on peut dire que $H_p(Y|X)$ est ici l'entropie conditionnelle de la distribution pseudo-empirique de (x,y) : $p'(x,y) = \tilde{p}(x)p(y|x)$.

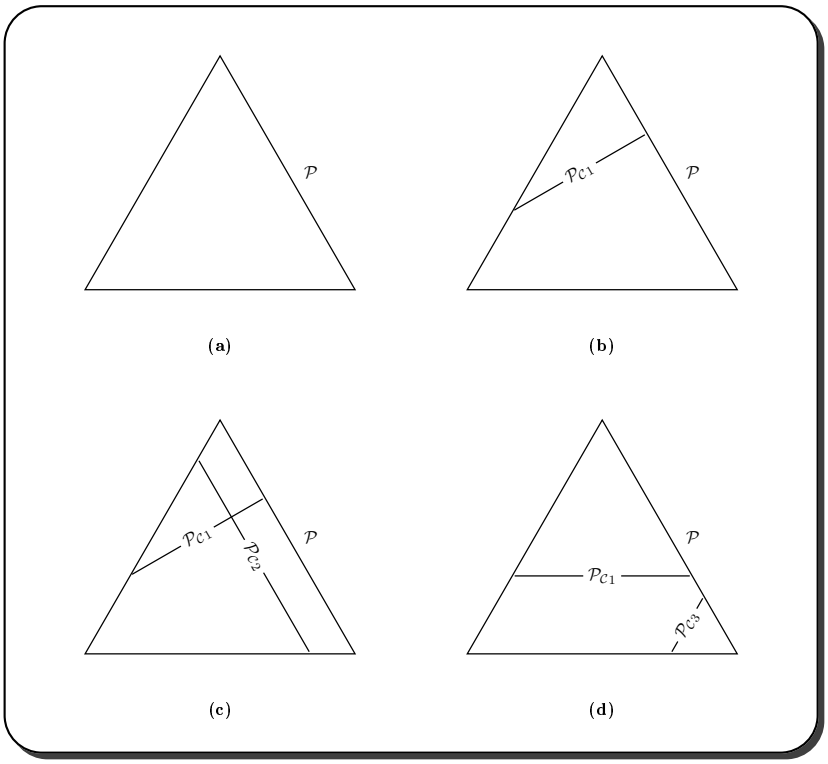


FIG. 3.1 – Différents problèmes d'optimisation sous contraintes. Chaque point de la surface $\mathcal{P}$ d'un triangle représente une distribution de probabilités sur 3 points. Les segments $\mathcal{P}_{\mathcal{C}1}$, $\mathcal{P}_{\mathcal{C}2}$ et $\mathcal{P}_{\mathcal{C}3}$ représentent des ensembles de distributions contraints. Si l'on n'impose aucune contrainte (cas (a)), alors tous les modèles sont autorisés. En imposant une contrainte linéaire $\mathcal{P}_{\mathcal{C}1}$, on se restreint aux $p \in \mathcal{P}$ situés sur le segment $\mathcal{P}_{\mathcal{C}1}$, comme indiqué en (b). Une seconde contrainte linéaire peut déterminer p exactement, si les deux contraintes sont satisfiables. C'est le cas en (c), où l'intersection de $\mathcal{P}_{\mathcal{C}1}$ et $\mathcal{P}_{\mathcal{C}2}$ est non vide. Alternativement, une seconde contrainte linéaire pourrait être inconsistante avec la première (par exemple, la première pourrait exiger que la probabilité du premier point est $1/3$ et la seconde que la probabilité du troisième point est $3/4$), comme en (d).

L'entropie est une valeur comprise entre 0 et $\ln|\mathcal{Y}|$. Elle est nulle lorsque le modèle n'a aucune incertitude, c'est-à-dire quand, étant donné un contexte x, la distribution de probabilité $p(Y|x)$ est concentrée sur une seule valeur y_0 ($p(y_0|x) = 1$ et $\forall y \in \mathcal{Y} \setminus \{y_0\},\ p(y|x) = 0$). Elle est maximale ($\ln|\mathcal{Y}|$) quand la distribution $p(Y|x)$ est uniforme sur $\mathcal{Y}$ (i.e. quand $\forall y \in \mathcal{Y},\ p(y|x) = \frac{1}{|\mathcal{Y}|}$). À l'aide de cette définition, le principe d'entropie maximum peut maintenant être présenté.

Principe d'entropie maximum

Pour sélectionner un modèle parmi un ensemble $\mathcal{P}_\mathcal{C}$ de distributions de probabilités, choisir le modèle $p_* \in \mathcal{P}_\mathcal{C}$ d'entropie $H(p)$ maximale :

$$p_* = \underset{p \in \mathcal{P}_\mathcal{C}}{\text{Argmax}}\, H(p) \tag{3.4}$$

On peut montrer [Berger 96]que p_* est toujours défini, c'est-à-dire qu'il existe toujours un modèle unique p_* d'entropie maximale dans n'importe quel ensemble contraint $\mathcal{P}_\mathcal{C}$, lorsque les contraintes sont des équations linéaires (ce qui est le cas ici avec les E_p).

3.1.4 Forme paramétrique

Le principe du maximum d'entropie fournit un problème d'optimisation sous contraintes : trouver $p_* \in \mathcal{P}_\mathcal{C}$ qui maximise $H(p)$. Dans les cas simples, ce problème peut être résolu analytiquement, mais dans le cas général, la solution ne peut être explicitée, et l'on a besoin d'une approche plus indirecte.

Pour s'attaquer au problème, on utilise la méthode des multiplicateurs de Lagrange, issue de la théorie de l'optimisation sous contraintes (voir [Della Pietra 97] pour une plus ample étude sur l'optimisation sous contraintes appliquée au maximum d'entropie).

- Le problème original d'optimisation sous contrainte est le problème **primal** :

$$\text{trouver } p_* = \underset{p \in \mathcal{P}_\mathcal{C}}{\text{Argmax}}\, H(p)$$

- On introduit un paramètre (ou multiplicateur de Lagrange) pour chaque contrainte :
  - un paramètre μ_x pour chaque contrainte stochastique :
    $$\textstyle\sum_y p(y|x) = 1\,;$$
  - un paramètre λ_i pour les contraintes $E_p(f_i) = E_{\tilde{p}}(f_i)$.

On définit le lagrangien $\Lambda(p, \lambda, \mu)$ par :

$$\Lambda(p, \lambda, \mu) = H(p) + \sum_i \lambda_i \left(E_p(f_i) - E_{\tilde{p}}(f_i)\right) + \sum_x \mu_x \left(\sum_y p(y|x) - 1\right) \tag{3.5}$$

– En gardant λ, μ fixes, on calcule le maximum (non contraint) du lagrangien $\Lambda(p, \lambda, \mu)$ parmi tous les $p \in \mathcal{F}$, où $\mathcal{F}$ représente l'ensemble des fonctions de $\mathcal{X} \times \mathcal{Y}$ dans $[0, 1]$, en annulant les dérivées partielles sur $p(y|x)$. Il vient :

$$\begin{aligned}
\frac{\partial \Lambda(p, \lambda, \mu)}{\partial p(y|x)} &= \frac{\partial H(p)}{\partial p(y|x)} + \sum_i \lambda_i \frac{\partial E_p(f_i)}{\partial p(y|x)} + \mu_x \\
&= \tilde{p}(x)(-1 - \ln p(y|x)) + \left[\sum_i \lambda_i \tilde{p}(x) f_i(x, y)\right] + \mu_x \\
&= 0 \\
\Rightarrow \quad & \ln p(y|x) = \sum_i \lambda_i f_i(x, y) + \frac{\mu_x}{\tilde{p}(x)} - 1 \\
\Rightarrow \quad & p(y|x) = \exp\left(\sum_i \lambda_i f_i(x, y)\right) \exp\left(\frac{\mu_x}{\tilde{p}(x)} - 1\right)
\end{aligned} \tag{3.6}$$

On note qu'on peut fixer les μ_x en fonction de λ de façon à ce que la solution soit un élément de $\mathcal{P}$, en injectant l'expression de $p(y|x)$ obtenue dans les contraintes $\sum_y p(y|x) = 1$, ce qui revient à poser :

$$Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right) \tag{3.7}$$

et

$$\mu_x(\lambda) = \tilde{p}(x)\left(1 - \ln Z_\lambda(x)\right)$$

.

– On note p_λ le p pour lequel $\Lambda(p, \lambda, \mu(\lambda))$ atteint son maximum, et $\Psi(\lambda)$ la valeur de ce maximum.

$$p_\lambda \overset{\text{Déf.}}{=} \operatorname*{Argmax}_{p \in \mathcal{P}} \Lambda(p, \lambda, \mu(\lambda)) \tag{3.8}$$

$$\Psi(\lambda) \overset{\text{Déf.}}{=} \Lambda(p_\lambda, \lambda, \mu(\lambda)) \tag{3.9}$$

$\Psi(\lambda)$ est appelée fonction **duale**. D'après l'équation (3.6), les fonctions p_λ et $\Psi(\lambda)$ valent :

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right) \tag{3.10}$$

$$\Psi(\lambda) = -\sum_x \tilde{p}(x) \ln Z_\lambda(x) + \sum_i \lambda_i E_{\tilde{p}}(f_i) \tag{3.11}$$

où $Z_\lambda(x)$ est la constante de normalisation donnée par la formule (3.7).

– Finalement, on pose le problème **dual** d'optimisation non-contrainte :

$$\text{trouver } \lambda^* = \operatorname*{Argmax}_\lambda \Psi(\lambda)$$

À première vue, l'intérêt de poser ce problème dual n'apparaît pas clairement. Cependant, un principe fondamental de la théorie des multiplicateurs de Lagrange, le théorème de Kuhn-Tucker, affirme que sous certaines conditions, les problèmes primal et dual sont étroitement liés[3]. C'est le cas dans la situation présente. Le résultat final de ce théorème est le suivant : si λ^* est la solution du problème dual, alors $p_{\lambda^*} = p_*$. En d'autres termes :

> *Le modèle d'entropie maximale soumis aux contraintes $\mathcal{P}_\mathcal{C}$ a la forme paramétrique de (3.10), les valeurs des paramètres λ^* pouvant être déterminées en maximisant la fonction duale $\Psi(\lambda)$.*

En pratique, la conséquence la plus importante de ce résultat est que tout algorithme trouvant le maximum λ^* de $\Psi(\lambda)$ peut être utilisé pour trouver le maximum p_* de $H(p)$ pour $p \in \mathcal{P}_\mathcal{C}$.

3.1.5 Relation avec le Maximum de Vraisemblance

La **log-vraisemblance** $\mathcal{L}_{\tilde{p}}(p)$ d'une distribution empirique $\tilde{p}$ par rapport à un modèle p est définie par :

$$\mathcal{L}_{\tilde{p}}(p) \overset{\text{Déf.}}{=} \ln \prod_{x,y} p(y|x)^{\tilde{p}(x,y)} = \sum_{x,y} \tilde{p}(x,y) \ln p(y|x) \tag{3.12}$$

On peut vérifier[4] que la fonction duale $\Psi(\lambda)$ de la section précédente est

[3] le théorème de Kuhn-Tucker concerne des problèmes d'optimisation sous contrainte en général ; on peut démontrer le lien entre les modèles à maximum d'entropie et les modèles exponentiels à maximum de vraisemblance peut être démontré sans faire appel à ce théorème (voir [Della Pietra 97]).

[4] Le premier terme de $\Psi(\lambda)$ se récrit ainsi :

$$\begin{aligned}
\sum_x \tilde{p}(x) \ln Z_\lambda(x) &= \sum_x \sum_y \tilde{p}(x,y) \ln Z_\lambda(x) \\
&= \sum_x \sum_y \tilde{p}(x,y) \left[\sum_i \lambda_i f_i(x,y) - \ln p_\lambda(y|x)\right] \\
&= \sum_i \lambda_i E_{\tilde{p}}(f_i) - \mathcal{L}_{\tilde{p}}(p)
\end{aligned}$$

en fait la log-vraisemblance du modèle exponentiel p_λ :

$$\Psi(\lambda) = \mathcal{L}_{\tilde{p}}(p_\lambda) \tag{3.13}$$

Avec cette interprétation, le résultat de la section précédente peut être réexprimé ainsi :

> *Le modèle $p_* \in \mathcal{P}_\mathcal{C}$ d'entropie maximale est le modèle dans la famille paramétrique (3.10) qui maximise la vraisemblance de la base d'apprentissage $\tilde{p}$.*

Ce résultat fournit une justification supplémentaire au principe d'entropie maximale : si sélectionner un modèle p_* sur la base du maximum d'entropie n'est pas assez intuitif, il se trouve que le même modèle p_* est aussi celui qui rend le mieux compte de la base d'apprentissage, parmi tous les modèles ayant la forme paramétrique (3.10).

	Primal	**Dual**
problème	$\text{Argmax}_{p \in \mathcal{P}_\mathcal{C}} H(p)$	$\text{Argmax}_\lambda \Psi(\lambda)$
description	maximum d'entropie	maximum de vraisemblance
type	optimisation sous contraintes	optimisation sans contraintes
domaine	$p \in \mathcal{P}_\mathcal{C}$	vecteurs de réels $\{\lambda_1, \lambda_2, \dots\}$
solution	p_*	λ_*
Théorème de Kuhn-Tucker : $p_* = p_{\lambda^*}$		

TAB. 3.1 – La dualité du maximum d'entropie et du maximum de vraisemblance est un exemple du phénomène plus général de dualité en optimisation sous contraintes.

3.1.6 Calcul des paramètres

Pour la plupart des problèmes, hormis les plus simples, le λ^* qui maximise $\Psi(\lambda)$ ne peut être déterminé analytiquement. Il faut alors recourir à des méthodes numériques. Du point de vue de l'optimisation numérique, la fonction $\Psi(\lambda)$ se comporte bien, du fait qu'elle est C^∞, et concave en λ. Par conséquent, une variété de méthodes numériques peuvent être employées pour calculer λ^*. Une méthode simple consiste à calculer λ^* en maximisant itérativement $\Psi(\lambda)$, coordonnée par coordonnée. Appliquée au problème de maximum d'entropie, cette technique mène à l'algorithme de Brown ([Brown 59]). D'autres méthodes telles que la descente de gradient peuvent être utilisées.

Une méthode d'optimisation spécifique au problème de maximum d'entropie a été développée dans [Darroch 72] et est connue sous le nom « Iterative Scaling Algorithm » (GIS). La version présentée ici a été développée plus particulièrement pour le problème présent, et la démonstration de sa

monotonicité et de sa convergence est donnée dans [Della Pietra 97]. Cet algorithme peut s'appliquer si les fonctions indicatrices $f_i(x, y)$ sont positives ou nulles :

$$f_i(x, y) \geq 0 \quad \text{pour tous } i,\ x,\ \text{et } y \tag{3.14}$$

Cet algorithme est donc plus général que celui de Darroch-Ratcliff, qui nécessite en plus de cette contrainte que les fonctions satisfassent $\sum_i f_i(x, y) = 1$ pour tous x, y.

Algorithme « Improved Iterative Scaling » (IIS)

`Entrée` : Fonctions indicatrices $f_1, f_2, \ldots, f_n$; distribution empirique $\tilde{p}(x, y)$
`Sortie` : Paramètres optimaux λ_i^* ; modèle optimal p_{λ^*}.

1. Initialiser $\lambda_i = 0$ pour tout $i \in \{1, 2, \ldots, n\}$
2. Pour chaque $i \in \{1, 2, \ldots, n\}$:
   (a) Soit $\Delta\lambda_i$ la solution de :

$$\sum_{x,y} \tilde{p}(x) p_\lambda(y|x) f_i(x, y) \exp\left(\Delta\lambda_i f^\#(x, y)\right) = \tilde{p}(f_i) \tag{3.15}$$

   avec

$$f^\#(x, y) \overset{\text{Déf.}}{=} \sum_{i=1}^{n} f_i(x, y) \tag{3.16}$$

   (b) Mettre à jour la valeur de λ_i par : $\lambda_i \leftarrow \lambda_i + \Delta\lambda_i$
3. Aller à l'étape 2 si les λ_i n'ont pas tous convergé.

Le point-clef dans cet algorithme est le calcul des incréments $\Delta\lambda_i$ qui résolvent (3.15). Si $f^\#(x, y)$ est constante ($f^\#(x, y) = M$ pour tous x, y, par exemple), alors $\Delta\lambda_i$ est donné explicitement par :

$$\Delta\lambda_i = \frac{1}{M} \ln \frac{E_{\tilde{p}}(f_i)}{E_{p\lambda}(f_i)}$$

Si $f^\#(x, y)$ n'est pas constante, alors les $\Delta\lambda_i$ doivent être calculés numériquement. La méthode de Newton permet de résoudre le problème efficacement. Cette méthode calcule la solution α_* d'une équation $g(\alpha_*) = 0$ itérativement par la relation de récurrence

$$\alpha_{n+1} = \alpha_n - \frac{g(\alpha_n)}{g'(\alpha_n)} \tag{3.17}$$

avec un choix approprié pour α_0, en fonction du domaine de définition de g et du domaine de convergence de cette suite.

3.1.7 Modèle SCFG : un exemple trivial de modèle à maximum d'entropie

Rappelons la définition d'une SCFG : un quintuplet $(\mathcal{T}, \mathcal{NT}, S, \mathcal{R}, p)$, avec

- $\mathcal{T}$ un ensemble de symboles terminaux,
- $\mathcal{NT} = \{N_1, N_2, \ldots, N_m\}$ un ensemble de symboles non-terminaux,
- $S \in \mathcal{NT}$ le symbole de départ,
- $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ un ensemble de règles de réécriture, de la forme $X \rightarrow Y_1\ Y_2\ \ldots$, où X est un non-terminal ($X \in \mathcal{NT}$) et les Y sont dans $\mathcal{NT} \cup \mathcal{T}$. On note $T(R)$ le symbole de tête de la règle R.
- p une fonction de probabilités conditionnelles sur les règles de $\mathcal{R}$, respectant les contraintes :

$$p(r_i|N_j) = 0 \text{ si } T(r_i) \neq N_j$$

et

$$\sum_i p(r_i|N_j) = \begin{cases} 1 & \text{s'il existe au moins une règle de tête } N_j, \\ 0 & \text{sinon.} \end{cases}$$

3.1.7.1 Modèle exponentiel

Une SCFG peut être utilisée comme modèle génératif de production d'arbres syntaxiques. Le processus stochastique en cause est une succession de réécritures d'une chaîne de symboles, en choisissant X le symbole non-terminal de la chaîne courante le plus à gauche, en tirant aléatoirement une règle de réécriture, selon la probabilité $p(r_i|X)$, et en remplaçant dans la chaîne le symbole X par les symboles de queue de la règle r tirée.

L'étape aléatoire du processus est donc le choix d'une règle r parmi les règles de tête X. Lorsqu'on fait l'apprentissage des paramètres d'une SCFG, on cherche donc à déterminer les distributions de probabilités conditionnelles $p(r|T)$ des règles r connaissant le symbole à récrire T.

Il est facile d'écrire ce modèle sous la forme d'un modèle exponentiel, en posant :

- des fonctions indicatrices f_i définies par :

$$f_i(r, T) = \begin{cases} 1 & \text{, si } r = r_i \text{ et } T = T(r_i), \\ 0 & \text{sinon.} \end{cases}$$

- des paramètres λ_i définis par :

$$\lambda_i = \ln p\left(r_i|T(r_i)\right)$$

On a bien alors, pour tout (r, T) la relation :

$$p(r|T) = \exp(\sum_i \lambda_i f_i(r, T))$$

Pour déterminer les paramètres $p(r|T)$ d'une SCFG à partir d'un corpus d'arbres syntaxiques, on utilise en général le principe de maximum de vraisemblance. On pourrait aussi, avec cette mise sous forme de modèle exponentiel, considérer le principe de maximum d'entropie, pour tomber sur le même résultat, comme suit.

3.1.7.2 Maximum de vraisemblance

On dispose pour l'apprentissage d'une collection d'arbres syntaxiques respectant la structure des règles hors-contextes. Chaque arbre peut être considéré lui-même comme une collection de non-terminaux auxquels on a appliqué une règle de réécriture. Donc, on peut considérer le corpus comme une collection de N exemples de non-terminaux auxquels on a appliqué une règle de réécriture. Nommons $\tilde{p}(r_i, T(r_i))$ la probabilité empirique de l'application de la règle r_i sur le symbole $T(r_i)$. La log-vraisemblance du corpus s'écrit :

$$\mathcal{L}_{\tilde{p}}(p) = \sum_i \tilde{p}(r_i, T(r_i)) \ln p(r_i|T(r_i))$$

La solution du maximum de vraisemblance se calcule aisément à l'aide de la méthode des coefficients de Lagrange, qui donne pour résultat :

$$p(r_i|T(r_i)) = \frac{\tilde{p}(r_i, T(r_i))}{\sum_k \tilde{p}(r_k|T(r_i))}$$

3.1.7.3 Maximum d'entropie

D'après la section 3.1.5, on devrait retrouver le même résultat en cherchant le modèle de maximum d'entropie parmi les modèles respectant les contraintes :

$$\forall i,\ E_p(f_i) = E_{\tilde{p}}(f_i)$$

En fait, il n'existe qu'un seul modèle probabiliste respectant ces contraintes, qui est donc par défaut celui ayant l'entropie maximale ! En effet, la résolution d'une contrainte donne directement la probabilité cherchée :

$$
\begin{array}{lcl}
 & E_p(f_i) & = E_{\tilde{p}}(f_i) \\
\Rightarrow & \sum_k \tilde{p}(T(r_k))p(r_k|T(r_k))f_i(r_k,T(r_k)) & = \sum_k \tilde{p}(r_k,T(r_k))f_i(r_k,T(r_k)) \\
\Rightarrow & \tilde{p}(T(r_i))p(r_i|T(r_i)) & = \tilde{p}(r_i,T(r_i)) \\
\Rightarrow & \tilde{p}(T(r_i))p(r_i|T(r_i)) & = \tilde{p}(T(r_i))\tilde{p}(r_i|T(r_i)) \\
\Rightarrow & p(r_i|T(r_i)) & = \tilde{p}(r_i|T(r_i)) \\
\Rightarrow & p(r_i|T(r_i)) & = \dfrac{\tilde{p}(r_i,T(r_i))}{\sum_k \tilde{p}(r_k,T(r_i))}
\end{array}
$$

(à condition que le non-terminal $T(r_i)$ apparaisse une fois au moins dans le corpus, de façon à ce que la simplification par $\tilde{p}(T(r_i))$ soit possible).

On a donc bien la même solution. En résumé, une SCFG apprise sur un corpus d'arbres est :

- un modèle d'entropie maximum respectant les contraintes :

$$E_p(f_i) = E_{\tilde{p}}(f_i)$$

  où les f_i sont des fonctions indicatrices de la présence d'une règle de réécriture ;
- un modèle exponentiel de maximum de vraisemblance.

3.1.8 Synthèse

Les modèles à maximum d'entropie se distinguent finalement par :

- le processus probabilisé, c'est-à-dire les distributions de probabilités conditionnelles $p(y|x)$ qui modélisent le processus ; le choix de ces distributions passe par le choix des éléments de $\mathcal{Y}$, qui sont *produits* à chaque étape du processus, ainsi que par le choix des éléments de $\mathcal{X}$, qui constituent le *contexte* dans lequel y est produit ;
- les fonctions indicatrices $f_i(x,y)$ que l'on choisit d'observer dans un corpus d'apprentissage, et dont on veut rendre égales les espérances dans le modèle et dans le corpus d'apprentissage.

Dans la plupart des cas, on trouve la solution au problème en posant :

$$p_\lambda(y|x) = \frac{1}{Z_x}\exp(\sum_i \lambda_i f_i(x,y))$$

avec

$$Z_x = \sum_{y' \in \mathcal{Y}} \exp(\sum_i \lambda_i f_i(x,y'))$$

puis en résolvant le problème dual, qui consiste à maximiser sur λ la vraisemblance de p_λ par rapport au corpus d'apprentissage.

Dans la suite de ce chapitre est exposé notre modèle GCFG, modèle à maximum d'entropie orienté vers l'analyse. On peut remarquer que cette orientation n'est pas imposée par les modèles à maximum d'entropie : les SCFG sont des modèles orientés vers la génération, et leurs paramètres sont solution d'un problème de maximisation d'entropie. [Lafferty 96] donne aussi un modèle à maximum d'entropie qui enrichit les modèles de bigrammes standards par des probabilités sur les étiquettes morphosyntaxiques des mots. Ce modèle reste, comme celui des bigrammes, tourné vers la génération, en ce sens qu'il construit des probabilités de produire des mots. Mais s'ils n'*imposent* pas une conception tournée vers l'analyse, les modèles à maximum d'entropie la *permettent*, comme nous allons le voir.

3.2 Principe et définition de la GCFG (Gibbsian CFG)

Le modèle exposé ici s'inspire directement du modèle SCFG. La grammaire est composée d'un ensemble de N_r règles de réécriture $X \rightarrow Y_1...Y_i$, de N_s symboles terminaux et non-terminaux, les symboles terminaux n'apparaissant que dans les membres de droite des règles. De plus, à chaque règle r_i est associé un potentiel λ_i. Ce potentiel remplace la probabilité d'une règle SCFG, les probabilités des règles de même membre de gauche devant alors sommer à 1 (contrainte stochastique).

3.2.1 Potentiel d'un arbre, et probabilité conditionnelle

Contrairement à une SCFG, on va construire un modèle non génératif. On ne cherche donc pas à définir la probabilité de *production* d'un arbre. En revanche, on définit le **potentiel d'un arbre** d'analyse t comme la somme des potentiels des règles qui le constituent. Il s'agit donc du produit scalaire $\lambda \cdot f(t)$ de deux vecteurs de taille M (nombre de règles de la grammaire), la i^e composante de λ étant le potentiel λ_i de la règle r_i, et la i^e composante de $f(t)$ étant le nombre d'occurrence(s) de la règle r_i dans l'arbre t, à quelque niveau que ce soit.

On définit de plus la probabilité d'un arbre d'analyse t conditionnellement à ses feuilles $w = (w_1...w_n)$ (où w représente la phrase analysée) par la formule :

$$p(t|w) = \frac{e^{\lambda \cdot f(t)}}{\sum_{y \rightarrow w} e^{\lambda \cdot f(y)}} \tag{3.18}$$

où $\sum_{y \rightarrow w}$ est une somme sur tous les arbres y de la grammaire qui ont pour feuilles w.

En réalité, le modèle que l'on cherche est un modèle à maximum d'entropie qui soit en accord avec un corpus d'apprentissage sur les statistiques

d'occurrences des règles *conditionnellement* aux feuilles. Il doit donc respecter les contraintes (3.1) (voir p.49), en considérant :

- w = phrase à analyser (le x de la formule (3.1)) ;
- t = arbre d'analyse d'une phrase (le y de la formule (3.1)) ;
- $f_i(w,t) = \begin{cases} \text{nombre d'occurrences de la règle hors-contexte } r_i \text{ dans l'arbre } t & \text{si } t \to w, \\ 0 & \text{sinon.} \end{cases}$

Le modèle que l'on trouve a d'après la section précédente la forme que nous venons de poser. Les paramètres y apparaissant sont obtenus par maximisation de la vraisemblance conditionnelle du corpus.

3.2.2 Analyse syntaxique à l'aide d'une GCFG

L'analyse syntaxique d'une phrase w consiste à déterminer l'arbre $\bar{t}$ de plus fort potentiel parmi les arbres d'analyse possibles. Au vu de la formule (3.18), cela revient à trouver l'arbre de probabilité maximale, conditionnellement à la phrase analysée :

$$\begin{aligned} \bar{t} &= \operatorname*{Argmax}_{t \to w} \lambda \cdot f(t) \\ &= \operatorname*{Argmax}_{t \to w} \prod_{r_i \in t} e^{\lambda_i f_i(t)} \quad \text{car} \sum_{y \to w} e^{\lambda \cdot f(y)} \\ &= \operatorname*{Argmax}_{t \to w} p_\lambda(t|w) \end{aligned} \tag{3.19}$$

L'expression (3.19) montre à quel point ce modèle est proche d'un modèle SCFG pour effectuer l'analyse syntaxique. En effet, avec une SCFG, la solution est donnée par :

$$\bar{t} = \operatorname*{Argmax}_{t \to w} p(t) = \operatorname*{Argmax}_{t \to w} \prod_{r_i \in t} p(r_i)^{f_i(t)}$$

L'utilisation d'un modèle GCFG ne nécessite donc pas le développement d'un algorithme d'analyse spécifique : on peut réutiliser tel quel tout algorithme d'analyse de SCFG[5], à condition que celui-ci ne requière pas la condition $p(r_i) \leq 1$. En particulier, les algorithmes de type $A*$, qui nécessitent un score variant monotonement avec le nombre d'étapes de l'analyse, doivent être évités. Nous avons utilisé ici un analyseur tabulaire ascendant dérivé de CYK [Chappelier 98][6] pour effectuer les tests d'analyse, en remplaçant

[5] comme par exemple les algorithmes CYK [Lari 90] ou Earley [Earley 86]

[6] L'algorithme en question est simplement un algorithme CYK adapté aux grammaires hors-contextes stochastiques qui ne sont pas sous forme normale de Chomsky (CNF). La binarisation de la grammaire est effectuée dynamiquement (i.e. pendant le processus d'analyse) comme décrit à l'annexe B.

simplement les probabilités des règles $p(r_i)$ par les valeurs e^{λ_i}.

3.3 Apprentissage des potentiels (paramètres λ_i)

3.3.1 Principe

Étant donné un corpus d'apprentissage, constitué d'un ensemble W de phrases w et d'arbres d'analyse $t \in \mathcal{C}$ de ces phrases, on cherche à calculer les paramètres du modèle λ de façon à maximiser la probabilité des arbres conditionnellement aux phrases :

$$\bar{\lambda} = \underset{\lambda}{\text{Argmax}}\, p_\lambda(\mathcal{C}|W) = \underset{\lambda}{\text{Argmax}} \prod_{t \in \mathcal{C}} p_\lambda(t|w(t))$$

(en notant $w(t)$ la phrase analysée par l'arbre t, i.e. les feuilles de t).

$$\bar{\lambda} = \underset{\lambda}{\text{Argmax}} \ln \sum_{t \in \mathcal{C}} p_\lambda(t|w(t))$$

$$\bar{\lambda} = \underset{\lambda}{\text{Argmax}} \sum_{t \in \mathcal{C}} \ln p_\lambda(t|w(t)) = \underset{\lambda}{\text{Argmax}}\, \mathcal{A}(\lambda)$$

$\mathcal{A}(\lambda)$ est la « log-vraisemblance conditionnelle » du corpus ; l'apprentissage du modèle consiste en la maximisation de ce critère. On note ici l'une des différences fondamentales de ce modèle avec une SCFG, pour laquelle on cherche en général à maximiser la probabilité du corpus ($\prod_{t \in \mathcal{C}} p_\lambda(t)$), ce qui se résout facilement en affectant à chaque règle une probabilité proportionnelle à sa fréquence dans le corpus[7].

3.3.2 Improved Iterative Scaling Algorithm

La méthode utilisée pour le calcul des paramètres s'inspire directement de la méthode IIS (Improved Iterative Scaling) exposée dans
[Berger 97, Della Pietra 97, Lafferty 96]. Nous en donnons une rapide explication, instanciée sur le cas particulier du modèle GCFG.

Plutôt que de chercher à maximiser directement le critère $\mathcal{A}(\lambda)$, ce qui se révèle trop ardu, la méthode améliore itérativement le modèle, à partir d'un modèle initial λ_0. Une itération consiste à passer d'un modèle λ à un modèle λ', en tentant de maximiser sur λ' le critère :

$$\mathcal{A}_\lambda(\lambda') = \sum_{t \in \mathcal{C}} \ln \frac{p_{\lambda'}(t|w(t))}{p_\lambda(t|w(t))}$$

[7] On considère évidemment que l'on dispose d'un corpus d'arbres syntaxiques, construits à partir des règles de la grammaire hors-contexte.

(ce qui est pour le moment équivalent au critère initial, à la constante $\left(-\sum_{t\in\mathcal{C}} \ln p_\lambda(t|w(t))\right)$ près, qui est bien une constante dès lors que l'on cherche λ' à partir de λ).

Posons $Z_{\lambda,w} = \sum_{t\to w} e^{\lambda\cdot f(t)}$ la constante de normalisation associée à la phrase w dans le modèle λ. On peut alors écrire $p_\lambda(t|w) = Z_{\lambda,w}^{-1} e^{\lambda\cdot f(t)}$, et :

$$\mathcal{A}_\lambda(\lambda') = \sum_{t\in\mathcal{C}} (\lambda'-\lambda)\cdot f(t) - \sum_{t\in\mathcal{C}} \ln \frac{Z_{\lambda',w(t)}}{Z_{\lambda,w(t)}}$$

En majorant le logarithme par la formule $\ln\alpha \leq \alpha - 1$, on peut minorer $\mathcal{A}_\lambda(\lambda')$ par :

$$\mathcal{A}_\lambda(\lambda') \geq \sum_{t\in\mathcal{C}} \Delta\lambda\cdot f(t) - \sum_{t\in\mathcal{C}} \frac{Z_{\lambda',w(t)}}{Z_{\lambda,w(t)}} + |\mathcal{C}| = \mathcal{B}_\lambda(\lambda')$$

(avec $\Delta\lambda = \lambda' - \lambda$, et $|\mathcal{C}|$ étant le nombre d'arbres de la base d'apprentissage.)

On obtient ainsi un critère intermédiaire $\mathcal{B}_\lambda(\lambda')$, dont le maximum est forcément positif (car $\mathcal{B}_\lambda(\lambda) = 0$). Un modèle λ' qui maximise $\mathcal{B}_\lambda(\lambda')$ rend donc forcément positif le critère $\mathcal{A}_\lambda(\lambda')$. Hélas, les termes $Z_{\lambda',w}$ rendent le calcul du maximum de $\mathcal{B}_\lambda(\lambda')$ délicat. On continue donc à développer les calculs ainsi :

$$\frac{Z_{\lambda',w}}{Z_{\lambda,w}} = \frac{\sum_{y\to w} e^{\lambda'\cdot f(y)}}{Z_{\lambda,w}} = \sum_{y\to w} \frac{e^{\lambda\cdot f(y)}}{Z_{\lambda,w}} \frac{e^{\lambda'\cdot f(y)}}{e^{\lambda\cdot f(y)}} = \sum_{y\to w} p_\lambda(y|w) e^{\Delta\lambda\cdot f(y)}$$

On note $f^\#(y) = \sum_i f_i(y)$ le nombre de règles utilisées dans un arbre y.
On a alors : $\Delta\lambda\cdot f(y) = \sum_i \frac{f_i(y)}{f^\#(y)}(f^\#(y)\Delta\lambda_i)$.
Du fait de la convexité de l'exponentielle :

$$e^{\Delta\lambda\cdot f(y)} \leq \sum_i \frac{f_i(y)}{f^\#(y)} e^{f^\#(y)\Delta\lambda_i}$$

Injectant ces deux derniers résultats dans l'expression de $\mathcal{B}_\lambda(\lambda')$:

$$\begin{aligned}
\mathcal{B}_\lambda(\lambda') &\geq \mathcal{C}_\lambda(\lambda') \\
\mathcal{C}_\lambda(\lambda') &\overset{\text{Déf.}}{=} \sum_{t\in\mathcal{C}} \Delta\lambda\cdot f(t) - \sum_{t\in\mathcal{C}} \sum_{y\to w(t)} p_\lambda(y|w(t)) \sum_i \frac{f_i(y)}{f^\#(y)} e^{\Delta\lambda_i f^\#(y)} \\
&\quad + |\mathcal{C}|
\end{aligned}$$

On obtient ainsi le dernier critère $\mathcal{C}_\lambda(\lambda')$, dont le maximum en λ' est encore forcément positif, et qui est toujours inférieur à $\mathcal{A}_\lambda(\lambda')$. Pour aller au

plus vite vers le maximum de $\mathcal{A}_\lambda(\lambda')$, on va donc chercher à maximiser $\mathcal{C}_\lambda(\lambda')$ à chaque itération de l'algorithme. Il suffit pour cela d'annuler les dérivées partielles en $\Delta\lambda_i$, donc de résoudre pour chaque règle r_i de la grammaire l'équation suivante :

$$0 = -\sum_{t\in\mathcal{C}} f_i(t) + \sum_{t\in\mathcal{C}} \sum_{y\to w(t)} p_\lambda(y|w(t)) f_i(y) (e^{\Delta\lambda_i})^{f^\#(y)}$$

Il s'agit d'un polynôme en $\alpha = e^{\Delta\lambda_i}$, de coefficients tous positifs, qui est donc facilement annulé, par exemple par la méthode de Newton.

3.3.3 Algorithme Inside-Outside

Le premier terme du polynôme $-\sum_{t\in\mathcal{C}} f_i(t)$ est facilement obtenu : il représente la fréquence de la règle r_i dans le corpus arborisé.

Reste le problème du calcul des coefficients de degrés supérieurs : le terme $\sum_{y\to w}$ ... implique une sommation sur toutes les analyses de la phrase w. C'est l'une des difficultés majeures de ce modèle, le nombre d'analyses pouvant croître exponentiellement avec la longueur de la phrase. Cette étape de calcul nécessite dans certains modèles de Markov-Gibbs d'employer une méthode approchée par échantillonnage [Lafferty 96]. Ici, une factorisation du calcul peut être effectuée à l'aide d'un algorithme Inside-Outside, comme nous allons le montrer (cf. Annexe B)[8].

On récrit la somme par :

$$\mathcal{S}_{w,\lambda}(\alpha) = \sum_{y\to w} p_\lambda(y|w) f_i(y) \alpha^{f^\#(y)} = Z_{\lambda,w}^{-1} \sum_{y\to w} e^{\lambda\cdot f(y)} f_i(y) \alpha^{f^\#(y)}$$

$f_i(y)$ représente le nombre d'occurrences de la règle r_i dans l'arbre y. Notons $C(y,[j,r_i,k])$ le nombre d'occurrences dans l'arbre y de la règle r_i en position (j,k), c'est-à-dire lorsque r_i domine $w_j ... w_k$. En pratique, comme nous utilisons une grammaire sans boucle[9], une règle peut apparaître 0 ou 1 fois dans une position (j,k) donnée et dans un arbre donné. La somme se récrit alors ainsi :

$$\begin{aligned}
\mathcal{S}_{w,\lambda}(\alpha) &= Z_{\lambda,w}^{-1} \sum_{1\le j\le k\le|w|} \sum_{y\to w} e^{\lambda\cdot f(y)} \alpha^{f^\#(y)} C(y,[j,r_i,k]) \\
&= Z_{\lambda,w}^{-1} \sum_{1\le j\le k\le|w|} \sum_{y\to w} V(y) C(y,[j,r_i,k])
\end{aligned}$$

[8] L'algorithme Inside-Outside est peut être utilisé pour des buts différents. Classiquement, il sert à évaluer la probabilité des règles d'une SCFG dans un apprentissage EM (Expectation Maximization), avec un corpus de phrases (apprentissage non supervisé). Ici, l'apprentissage est supervisé, et l'algorithme sert uniquement à factoriser les calculs.

[9] Le terme « grammaire sans boucle » désigne une grammaire dans laquelle un non-terminal ne peut pas être récrit en lui-même *et seulement lui-même* après un nombre quelconque de réécritures.

où $V(y) = e^{\lambda \cdot f(y)} \alpha^{f^{\#}(y)} = \prod_{r_i \in y} (f_i \alpha)^{f_i(y)}$: V(y) est le produit des polynômes $P_i(\alpha) = (\lambda_i \alpha)$ associés aux règles qui constituent y.

D'après [Goodman 98] (pp.26-57) $\sum_{y \to w} V(y) C(y, [j, r_i, k])$ peut alors être calculé pour tout r_i, j et k à l'aide d'un algorithme Inside-Outside implémenté dans un parser de type CYK.

Preuve : Les conditions définies par Goodman sont bien réunies :

- $< \mathbb{R}_0^{+\infty}[X], +, *, 0, 1 >$[10] est un semi-anneau commutatif.
- La valeur $V(y)$ associée à un arbre est le produit des polynômes $P_i(\alpha)$ associés aux règles de y.
- La grammaire considérée est une grammaire hors-contexte sans boucle.
- Une dérivation dans une table CYK est associée uniquement à un arbre d'analyse (et réciproquement), et les règles de grammaire apparaissant dans la dérivation et l'arbre correspondant sont les mêmes.

La figure 3.2 (page 66) détaille l'algorithme Inside-Outside utilisé. Elle appelle quelques remarques :

- la grammaire utilisée est sans boucle, ce qui garantit la terminaison de la boucle `Répéter ... jusqu'à...` concernant les règles d'arité 1.
- la grammaire est mise sous forme normale de Chomsky comme suit : une règle $X \to Y_1 ... Y_n$, (avec $n \geq 3$) est remplacée par la règle binaire $X \to (Y_1 ... Y_{n-1}) Y_n$, et sont ajoutées les règles $(Y_1 ... Y_{n-1}) \to (Y_1 ... Y_{n-2}) Y_{n-1}$, ... , $(Y_1 Y_2) \to Y_1 Y_2$ (il faut aussi pour cela ajouter les symboles $(Y_1 ... Y_{n-1})$, ..., $(Y_1 Y_2)$). N'_r et N'_s représentent le nombre de règles et de symboles dans la grammaire ainsi binarisée.
- $P_r(\alpha) = \lambda_r \alpha$ est le polynôme élémentaire associé à la règle r. Lors de la binarisation d'une règle $r = X \to Y_1 ... Y_n$, la règle $X \to (Y_1 ... Y_{n-1}) Y_n$ est associée à $P_r(\alpha)$; les autres sont associées au polynôme 1.
- le résultat recherché $(\sum_{y \to w} V(y) C(y, [j, r_i, k]))$ est égal au produit $inside_r[j, r_i, k] * outside[j, t(r_i), k]$ (en notant $t(r)$ le symbole de tête de la règle r)

3.3.4 Résumé de l'algorithme d'apprentissage

Finalement, l'apprentissage se résume par l'algorithme de la figure 3.3.

[10] l'ensemble des polynômes de coefficients positifs, sur lequel sont définies les opérations d'addition et de multiplication habituelles, et leurs éléments neutres respectifs 0 et 1

table de polynômes $inside_s[1..n,1..|N_s|,1..n+1]:=0$
table de polynômes $inside_r[1..n,1..|N'_r|,1..n+1]:=0$
table de polynômes $outside[1..n,1..|N_s|,1..n+1]:=0$
/* calcul des polynômes inside */
pour chaque i
  $inside_s[i,w_i,i+1]=1$
pour chaque longueur l, de la plus courte à la plus longue
  pour chaque départ i
    pour chaque longueur de coupure t
      pour chaque règle $A \rightarrow BC$ de la grammaire
        $inside_r[i,A \rightarrow BC,i+l] := inside_r[i,A \rightarrow BC,i+l]$
          $+inside_s[i,B,i+t]*inside_s[i+t,C,i+l]*P_{A\rightarrow BC}(\alpha)$
        $inside_s[i,A,i+l] := inside_s[i,A,i+l]$
          $+inside_s[i,B,i+t]*inside_s[i+t,C,i+l]*P_{A\rightarrow BC}(\alpha)$
    répéter :
      pour chaque règle $A \rightarrow B$ de la grammaire
        $inside_r[i,A \rightarrow B,i+l] := inside_r[i,A \rightarrow B,i+l]$
          $+inside_s[i,B,i+l]*P_{A\rightarrow B}(\alpha)$
        $inside_s[i,A,i+1] := inside_s[i,A,i+1]$
          $+inside_s[i,B,i+l]*P_{A\rightarrow B}(\alpha)$
    jusqu'à ce que la table n'en soit plus modifiée.

/* calcul des polynômes outside */
pour chaque longueur l, de la plus longue à la plus courte
  pour chaque départ i
    pour chaque longueur de coupure t
      pour chaque règle $A \rightarrow BC$ de la grammaire
        $outside[i,B,i+t] := outside[i,B,i+t]$
          $+outside[i,A,i+l]*inside_s[i+t,C,i+l]*P_{A\rightarrow BC}(\alpha)$
        $outside[i+t,C,i+l] := outside[i+t,C,i+l]$
          $+outside[i,A,i+l]*inside_s[i,B,i+t]*P_{A\rightarrow BC}(\alpha)$
    pour chaque règle $A \rightarrow B$ de la grammaire
      $outside[i,B,i+l] := outside[i,B,i+l]$
      $+outside[i,A,i+l]*P_{A\rightarrow B}(\alpha)$

FIG. 3.2 – Algorithme CYK Inside-Outside

Définir un modèle initial λ'
Répéter :
$\lambda \leftarrow \lambda'$.
/* passage de λ à λ' */
Mise à zéro des polynômes $S^r(\alpha)$ associés aux règles r.
Pour chaque exemple t de la base d'apprentissage :
Analyser $w(t)$ par l'algorithme CYK Inside-Outside.
Calculer $Z_{\lambda,w(t)}$ comme la somme des coefficients du polynôme $\sum_r inside_r[1,r,n]$.
Pour chaque élément $[j,r,k]$ apparaissant dans la table CYK
$S^r(\alpha) := S^r(\alpha) + Z^{-1}_{\lambda,w(t)} \sum_{y \to w} V(y) C(y,[j,r_i,k])$
$(= S^r(\alpha) + Z^{-1}_{\lambda,w(t)} inside_r[j,r,k] * outside[j,t(r),k])$
Pour chaque règle r_i de la grammaire :
Résoudre : $S^{r_i}(\alpha) = -sum_{t \in \mathcal{C}} f_i(t)$
Calculer le paramètre du modèle λ' par : $\lambda'_i = \lambda_i + \ln \alpha$
jusqu'à la convergence du critère $\mathcal{A}(\lambda)$.

Remarque : le critère $\mathcal{A}(\lambda)$ est facile à obtenir à la fin d'une passe, du fait que les constantes de normalisation $Z_{\lambda,w(t)}$ sont calculées lors de cette passe. C'est pourquoi le test d'arrêt se base sur le modèle précédent λ plutôt que sur le nouveau modèle λ', quitte à faire une itération inutile : cela évite le recalcul des constantes de normalisation, qui demanderait une nouvelle analyse par table CYK de tous les exemples de la base à chaque passe.

FIG. 3.3 – Algorithme « Improved Iterative Scaling »

3.4 Expériences

La première motivation de ce travail était de pallier les faiblesses apparentes des modèles SCFG pour la reconnaissance automatique de la parole. Cependant, rien que dans le strict cadre de l'analyse syntaxique, il semble que ce modèle se comporte de façon plus conforme à l'intuition qu'une SCFG. Nous allons le montrer en reprenant l'exemple théorique de M. Johnson [Johnson 98]. Nous donnerons ensuite les résultats expérimentaux dont nous disposons, qui semblent confirmer que ce modèle « colle » mieux aux données d'apprentissage qu'une SCFG.

3.4.1 Un petit exemple théorique

Cet exemple est tiré d'une étude de M. Johnson [Johnson 98], et illustre un comportement des SCFG qui peut sembler paradoxal. Supposons que l'on dispose d'un corpus $\tilde{\tau}_1$ constitué de deux arbres A_1 et B_1 seulement (figure 3.4), l'arbre A_1 apparaissant avec une fréquence relative f (i.e. son nombre d'occurrences divisé par la taille du corpus). On entraîne une SCFG sur ce corpus selon la méthode habituelle, qui consiste à affecter aux règles une probabilité proportionnelle à leur fréquence en corpus.

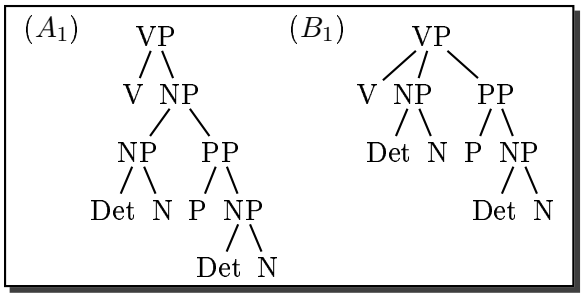


FIG. 3.4 – Corpus d'apprentissage $\tilde{\tau}_1$. Ce corpus contient l'arbre A_1 avec une fréquence relative f, et B_1 avec une fréquence relative $1-f$. A_1 correspond par exemple à la phrase « Regarde le blond avec la chaussure », et A_2 à « Regarde les étoiles avec la lunette ».

Les probabilités $(\hat{P}_1)$ des règles induites par ce corpus sont les suivantes :

R	Compte(R)	$\hat{P}_1(R)$
VP $\rightarrow$ V NP	f	f
VP $\rightarrow$ V NP PP	$1-f$	$1-f$
NP $\rightarrow$ Det N	2	$2/(2+f)$
NP $\rightarrow$ NP PP	f	$f/(2+f)$
PP $\rightarrow$ P NP	1	1

Lors d'une analyse syntaxique de la « phrase » `V Det N P Det N`, les structures (A_1) et (B_1) se voient affectées des probabilités $\hat{P}_1(A_1) = 4f^2/(2+f)^3$ et $\hat{P}_1(B_1) = 4(1-f)/(2+f)^2$. Ces deux probabilités ne somment pas à 1, ce qui n'est pas étonnant du fait que la SCFG $\hat{P}_1$ assigne une probabilité non nulle à des arbres n'apparaissant pas dans le corpus. Cela étant, dans une tâche d'analyse syntaxique, on est de toute façon plus intéressé par l'estimation de la fréquence relative des arbres pouvant correspondre à une même production que par leur probabilité absolue. La fréquence relative estimée $\hat{f}_1$ de (A_1), pour la phrase `V Det N P Det N`, est : $\hat{f}_1 = \hat{P}_1(\tau = A_1 | \tau \in \{A_1, B_1\}) = \hat{P}_1(A_1)/(\hat{P}_1(A_1) + \hat{P}_1(B_1)) = f^2/(2-f)$. Idéalement, cette fréquence relative estimée $\hat{f}_1$ devrait être proche de sa fréquence relative réelle f dans le corpus d'apprentissage. La relation entre les deux est tracée dans la figure 3.7 (page 71). On voit que les deux fonctions peuvent différer substantiellement. Par exemple, à $f = 0.48$, $\hat{f}_1 = 0.15$.

De plus, ce comportement contre-intuitif diffère selon la forme de la grammaire. M. Johnson poursuit son étude en utilisant des représentations d'arbres différentes. Le corpus $\tilde{\tau}_2$ (figure 3.5) est obtenu en représentant l'arbre (B) sous forme adjonctive de Chomsky (A l'étant déjà).

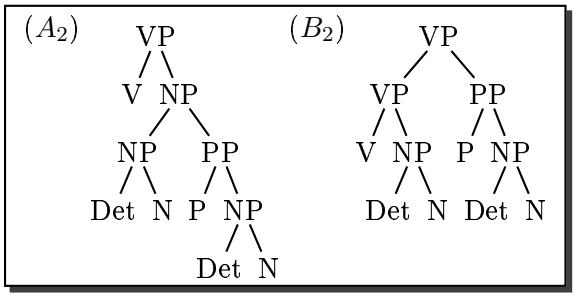


FIG. 3.5 – Corpus d'apprentissage $\tilde{\tau}_2$. Ce corpus contient l'arbre A_2 avec une fréquence relative f, et B_2 avec une fréquence relative $1-f$.

L'apprentissage d'une SCFG $\hat{P}_2$ sur ce corpus $\tilde{\tau}_2$ conduit à la fréquence relative estimée de $\hat{f}_2 = \hat{P}_2(\tau = A_2 | \tau \in \{A_2, B_2\}) = (f^2 - 2f)/(2f^2 - f - 2)$, dont la valeur, tracée sur la figure 3.7, diffère de f, quoique de façon moins significative que $\hat{f}_1$.

Le corpus $\tilde{\tau}_3$ (figure 3.6) est obtenu en aplanissant les règles de têtes NP et VP.

L'apprentissage d'une SCFG $\hat{P}_3$ sur ce corpus $\tilde{\tau}_3$ conduit à la fréquence relative estimée de $\hat{f}_3 = \hat{P}_3(\tau = A_3 | \tau \in \{A_3, B_3\}) = f^2/(2 - 3f + 2f^2)$, dont la valeur, tracée sur la figure 3.7, diffère de f de façon toujours moins significative que $\hat{f}_1$. Elle est plus proche de f que $\hat{f}_2$ seulement quand f est supérieur à environ 0.7.

Lorsque l'on entraîne un modèle GCFG sur chacun de ces corpus, les

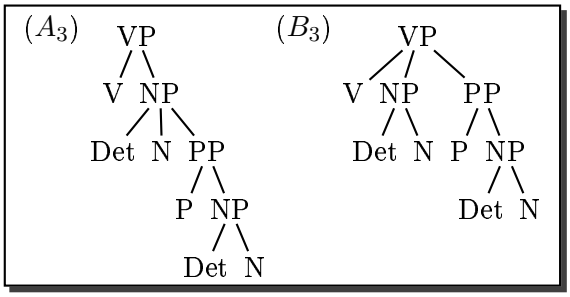


FIG. 3.6 – Corpus d'apprentissage $\tilde{\tau}_3$. Ce corpus contient l'arbre A_3 avec une fréquence relative f, et B_3 avec une fréquence relative $1 - f$.

résultats sont au contraire conformes à l'intuition : les deux arbres se voient affectés d'une fréquence relative estimée égale à leur fréquence relative observée en corpus. Les courbes des fréquences estimées pour chacun des corpus se confondent et sont représentées par le segment f sur la figure 3.7. Cela montre que la GCFG « colle » aux données d'apprentissage d'une façon plus intuitive qu'une SCFG, et moins dépendante de la forme de la grammaire hors-contexte sous-jacente.

3.4.2 Résultats expérimentaux

Le modèle GCFG a été testé en grandeur réelle sur un corpus dérivé du corpus SUSANNE#3[Sampson 94], comptant 4 292 arbres d'analyses, 1 920 non-terminaux dont 395 préterminaux, 11 935 terminaux, 17 669 règles grammaticales. Certaines règles unaires (ou filaires, par exemple $X \rightarrow Y$) ont été manuellement retirées du corpus original de façon à ce que la grammaire obtenue soit sans boucle.

Le premier test consiste en l'apprentissage du modèle à partir du corpus complet, puis en l'analyse syntaxique des phrases du corpus, et enfin en la comparaison des arbres ainsi obtenus avec les arbres du corpus. Les résultats sont regroupés sous le label $Test = Apprentissage$ de la figure 3.8 p.73.

- `Tx(Ana)` y représente le taux des phrases recevant au moins une analyse ;
- `Tx(Jus)` représente le taux des phrases correctement analysées parmi celles recevant au moins une analyse ;
- les taux de précision et de rappel (colonnes `Pré` et `Rap`) sont obtenus en considérant la séquence des arbres $\tilde{\tau}$ produits par l'analyseur comme un ensemble $E(\tilde{\tau})$ de triplets $< N, g, d >$, où N est un non-terminal et g et d sont les positions dans le corpus du premier et dernier mot de la chaîne analysée par N.
  En comparaison avec une séquence d'arbres de référence $\tilde{\tau}'$, la précision

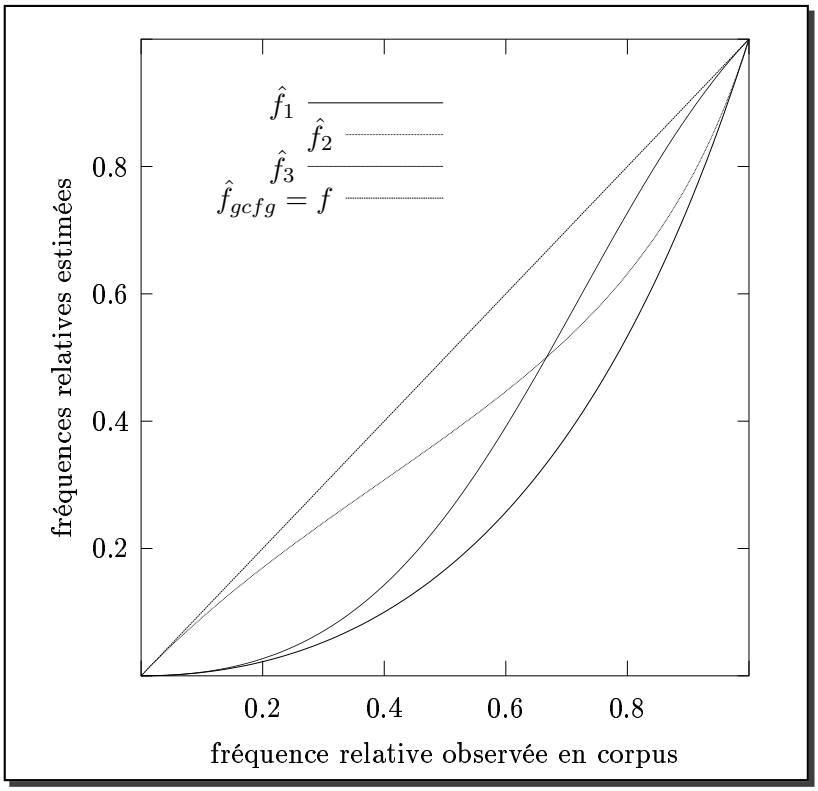


FIG. 3.7 – Fréquences relatives estimées de l'attachement de PP à NP, tracées en fonction de la fréquence relative f de cet attachement observée dans les données d'apprentissage des divers modèles étudiés. f_i est la fréquence relative estimée par une SCFG de l'arbre A_i du corpus τ_i (figures 3.4, 3.5 et 3.6) lorsque sa fréquence observée est f. f_{gcfg} est la fréquence relative de l'arbre A_i estimée par une GCFG pour $i \in \{1, 2, 3\}$.

et le rappel sont calculés comme :

$$Tx(Pre)(\tilde{\tau}) = \frac{|E(\tilde{\tau}) \cap E(\tilde{\tau}')|}{|E(\tilde{\tau})|}, \;\; Tx(Rap)(\tilde{\tau}) = \frac{|E(\tilde{\tau}) \cap E(\tilde{\tau}')|}{|E(\tilde{\tau}')|}$$

.

Le second test est identique au premier, sinon que l'apprentissage s'effectue sur 9 dixièmes du corpus, tirés aléatoirement, et le test sur le dixième restant. Les résultats présentés sont des moyennes des résultats obtenus avec 10 tirages aléatoires initiaux. La précision et le rappel sont calculés sur les seuls arbres recevant au moins une analyse[11].

Le modèle SCFG a été testé sur les mêmes bases, et l'on indique la *diminution du taux d'erreur* que la GCFG offre en comparaison avec la SCFG. Cette valeur est calculée par : $1 - \frac{1-Tx[GCFG]}{1-Tx[SCFG]}$.

Enfin, le corpus a été simplifié pour une deuxième série de tests, en réduisant les labels des non-terminaux à leur première lettre, et en supprimant les règles unaires, de façon à obtenir une grammaire nettement plus ambiguë sur laquelle l'utilisation de statistiques semble plus pertinente, les règles apparaissant alors plus souvent dans le corpus d'apprentissage.

3.4.3 Discussion

L'exemple de la section 3.4.1 montre qu'une SCFG dont les paramètres sont appris depuis un corpus arborisé peut se comporter en analyse de façon inattendue, affectant des probabilités plus grandes à des formes rencontrées moins souvent dans le corpus. Sur le même exemple, une GCFG se comporte en revanche conformément à l'intuition. Cette différence est principalement due aux critères d'apprentissage des modèles : les SCFG étant usuellement considérées comme des modèles génératifs, l'apprentissage de leurs paramètres se fait en maximisant la probabilité d'engendrer un corpus par un processus stochastique produisant un arbre depuis sa racine. Le critère d'apprentissage est donc[12] $p_\lambda(\mathcal{C})$, maximisé en affectant **aux règles** des probabilités proportionnelles à leur fréquence en corpus. Un tel modèle fait apparemment l'hypothèse que le langage est *engendré par un processus* grammatical.

Les GCFG en revanche sont pensées comme des modèles pour l'analyse, d'où leur critère d'apprentissage $p_\lambda(\mathcal{C}|W)$, qui correspond intuitivement à la probabilité d'engendrer un corpus d'arbres à partir de leurs feuilles. Si possible, ce critère sera maximisé lorsque les probabilités affectées **aux arbres**

[11] La couverture de la grammaire est seulement d'environ $13,5\%$ sur les bases de test, du fait que de nombreuses règles n'apparaissent qu'une fois dans la base : les règles apprises dans le corpus d'apprentissage ne suffisent pas à donner une analyse à $86,5\%$ des phrases de la base de test. D'autres résultats des GCFG sont donnés dans le chapitre 5 concernant les GTSG. Ils sont obtenus sur un corpus permettant l'apprentissage de grammaires ayant une meilleure couverture que ne le permet SUSANNE.

[12] v.section 3.2 pour les notations

Test = Apprentissage

Modèle	Tx(Ana)	Tx(Jus)	P	R
Corpus Susanne				
SCFG	1	0,782	0,989	0,988
GCFG	1	0,845	0,994	0,993
Diminution du Tx d'erreur		29%	43%	42%
Corpus simplifié				
Modèle	Tx(Ana)	Tx(Jus)	P	R
SCFG	1	0.568	0.964	0.962
GCFG	1	0.605	0.968	0.966
Diminution du Tx d'erreur		8,6%	11,4%	10,3%

Test ≠ Apprentissage

Modèle	Tx(Ana)	Tx(Jus)	P	R
Corpus Susanne				
SCFG	0,135	0,472	0,911	0,927
GCFG	0,135	0,462	0,908	0,924
Diminution du Tx d'erreur		−2%	−3%	−4%
Corpus simplifié				
Modèle	Tx(Ana)	Tx(Jus)	P	R
SCFG	1	0.860	0.983	0.982
GCFG	1	0.866	0.984	0.983
Diminution du Tx d'erreur		4,1%	4,5%	5,6%

FIG. 3.8 – Résultats comparés des modèles SCFG et GCFG, sur une tâche d'analyse syntaxique.

de mêmes feuilles seront proportionnelles à leur fréquence en corpus. Comme il y a suffisamment de paramètres dans l'exemple pour atteindre ce résultat, cela explique l'adéquation parfaite des fréquences relatives observées et attribuées.

La section 3.4.1 relate aussi le fait que représenter les structures (VP $\rightarrow$ V NP PP) et (NP $\rightarrow$ Det NP PP) de façon similaire dans le corpus (i.e. ou toutes deux aplanies, comme ici, ou toutes deux sous forme adjonctive de Chomsky) augmente les performances des SCFG apprises. Il n'en est rien pour les GCFG, qui se comportent parfaitement dans tous les cas. Cependant, des tests sur des corpora plus réalistes [Johnson 98] montrent que seul l'aplanissement augmente véritablement les performances des SCFG. M.Johnson soupçonne que la raison en est l'affaiblissement de l'hypothèse d'indépendance des règles provoqué par cet aplanissement. Il serait intéressant de mesurer ainsi l'effet d'un aplanissement des structures sur l'apprentissage d'une GCFG, qui ne fait pas cette hypothèse d'indépendance : le potentiel d'une règle y est calculé par rapport aux potentiels de toutes les règles de la grammaire, plutôt que par rapport aux seules règles de même tête.

Les performances d'une GCFG sont très satisfaisantes en auto-test (colonne $Test = Apprentissage$ de la figure 3.8) : avec le même nombre de paramètres qu'une SCFG apprise dans les mêmes conditions, elle réduit d'un tiers le taux de phrases mal analysées, et de moitié le *manque* en précision et en rappel (au niveau des labels) ; elle « colle » indiscutablement mieux aux données, ce qui montre que le critère d'apprentissage est adapté à l'analyse syntaxique.

En généralisation (colonne $Test \neq Apprentissage$), les GCFG ne semblent utiles que lorsque les règles qui apparaissent dans le corpus de test sont souvent représentées dans le corpus d'apprentissage. Dans le cas contraire, elles sont équivalentes aux SCFG. Cela montre peut-être les limites d'une approche statistique des grammaires hors-contextes : pour apprendre un modèle pertinent, il faut suffisamment d'exemples pour chacun de ses paramètres, ce qui n'est pas le cas avec les corpus SUSANNE ; ainsi, avec ce corpus, en généralisation, on peut se demander si l'utilisation de la SCFG est plus pertinente qu'un tirage aléatoire parmi les analyses obtenues avec une CFG non probabiliste.

Les résultats obtenus sur le corpus ATIS, présentés dans le chapitre 5 avec les résultats des GTSG, confirment cette impression : le corpus ATIS est nettement plus redondant que SUSANNE, chaque règle y apparaissant plus souvent ; dans ce cas, la GCFG se compare favorablement à la SCFG, que ce soit en auto-test ou en généralisation.

3.5 Conclusion

On a présenté dans ce chapitre une méthode de valuation des grammaires hors-contextes qui diffère de celle des SCFG par son critère d'apprentissage, adapté à une tâche d'analyse syntaxique, et par l'absence de contraintes stochastiques ($p_i \leq 1$, et $\sum p = 1$) sur les valeurs de ses paramètres. La forme des grammaires obtenues étant sensiblement la même que celle des SCFG, on dispose d'algorithmes standards permettant leur utilisation en analyse syntaxique. On a aussi détaillé un algorithme pour l'apprentissage de leurs paramètres, qui factorise suffisamment les calculs pouvoir être exécuté en un temps raisonnable. Le modèle GCFG présenté est fondé sur la modélisation à maximum d'entropie, exposée à la section 3.1.

Les études expérimentales montrent qu'en analyse, les paramètres obtenus collent mieux aux données d'apprentissage et à l'intuition que ceux d'une SCFG.

Chapitre 4

Grammaires à substitution d'arbres (TSG)

On introduit dans ce chapitre le modèle DOP de grammaires à substitution d'arbres. Il consiste à considérer un corpus d'arbres syntaxiques comme une base d'arbres élémentaires d'une STSG, les probabilités de ces arbres étant posées comme proportionnelles à leurs fréquences dans le corpus. En suivant l'étude de K. Sima'an [Sima'an 99], on étudie certaines propriétés de ce modèle qui peuvent réduire ses performances en analyse. On donne également la méthode de probabilisation alternative des règles proposée par K. Sima'an pour répondre aux problèmes posés par ces propriétés. Une autre méthode est étudiée, basée sur le critère de maximum de vraisemblance, mais dans le cas des STSG, ce critère empêche malheureusement la grammaire de généraliser son apprentissage à de nouvelles données.

Dans une seconde partie est évoqué le problème de la complexité de l'analyse syntaxique posé par les grammaires à substitution d'arbres. On passe en revue quelques travaux relatifs, qui cherchent à résoudre le problème par des méthodes d'échantillonnage, ou en changeant les critères d'analyse.

4.1 Probabilisation des TSG

4.1.1 Data Oriented Parsing (DOP)

Une STSG (grammaire à substitution d'arbres stochastique) est une grammaire pour laquelle les règles sont des arbres élémentaires que l'on peut combiner à l'aide de l'opérateur de substitution pour obtenir les dérivations des arbres d'analyse.

DOP [Scha 90] est une approche probabiliste de l'interprétation et de la désambiguïsation du langage naturel. Elle définit le langage comme un processus stochastique qui recombine des structures extraites d'une représentation de l'expérience linguistique passée (« l'expérience linguistique »

étant sous la forme d'un corpus de phrases annotées syntaxiquement et/ou sémantiquement).

Le traitement du langage « orienté par les données » est habituellement implémenté comme un modèle STSG extrêmement redondant : dans sa définition la plus générale, le modèle DOP utilise comme arbres élémentaires l'ensemble de *tous* les sous-arbres des arbres d'analyse contenus dans un corpus annoté (treebank) disponible pour l'entraînement du modèle [Bod 93]. Chaque arbre élémentaire τ de racine A est associé à une probabilité élémentaire $p(\tau)$ proportionnelle au nombre d'occurrences de τ dans le corpus d'apprentissage. Cette probabilité est définie sur l'ensemble des arbres élémentaires de même racine, et l'on a la propriété :

$$\sum_{\tau \text{t.q.} r(\tau)=A} p(\tau) = 1$$

en notant $r(\tau)$ la racine de l'arbre élémentaire τ [Bod 93, Bod 95].

On appellera **fragment** un sous-arbre de la base d'apprentissage. Lorsqu'un sous-arbre est lexicalisé (lorsque toutes ses feuilles sont des terminaux de la grammaire, donc des feuilles des arbres du corpus d'apprentissage), on l'appellera un **constituant**.

On dira que τ' est un fragment initial de τ si et seulement si l'on peut dériver τ à partir de τ', c'est-à-dire s'il existe des fragments $\tau_1, \ldots, \tau_m$ tels que $\tau = \tau' \circ \tau_1 \circ \cdots \circ \tau_m$. Pour un arbre τ, on définit $\sigma(\tau)$ comme l'ensemble de tous les fragments initiaux de τ.

Pour deux arbres syntaxiques τ et τ', on définit $f(\tau', \tau)$ le nombre d'occurrences de τ' dans τ. Si τ' n'apparaît pas dans τ, on pose $f(\tau', \tau) = 0$. Pour un arbre τ, on note $r(\tau)$ le non-terminal associé à sa racine. On notera également $N(\tau)$ le nombre de nœuds non-terminaux de τ qui n'en sont pas racine.

Dans la suite, on suppose que l'on dispose d'un corpus $\mathcal{C}$ constitué d'une collection d'arbres $t_1, \ldots, t_n$. Le terme *collection* est utilisé pour souligner que l'on peut avoir $t_i = t_j$ pour $i \neq j$. Étant donné un tel corpus, on définit Ω l'ensemble des fragments apparaissant dans $\mathcal{C}$. On utilise $\sigma[\Omega] = \cup_{i=1}^{n} \sigma(t_i)$ pour désigner l'ensemble des fragments initiaux de $\mathcal{C}$. On utilise l'indice A pour noter la restriction à l'ensemble des fragments de racine A. Ainsi $\sigma[\Omega_A]$ désigne l'ensemble de tous les fragments de $\mathcal{C}$ de racine A, qui sont aussi tous les fragments initiaux de Ω_A ($\sigma(\Omega_A) = \Omega(A)$).

Le nombre d'occurrences d'un fragment τ dans un corpus $\mathcal{C}$ est noté $f(\tau) = \sum_{i=1}^{n} f(\tau, t_i)$. Enfin, étant donné un fragment τ de racine $r(\tau)$, la *fréquence relative* $F(\tau)$ de τ dans $\mathcal{C}$ est définie par $F(\tau) = \frac{f(\tau)}{f(r(\tau))}$.

Pour produire une nouvelle phrase à partir des arbres élémentaires du modèle, on utilise l'opération de *substitution du non-terminal le plus à gauche*, qui est une fonction partielle sur les paires de fragments. Cette opération de substitution est définie pour deux arbres α et β si et seulement si la racine

de β est identique au non-terminal le plus à gauche des feuilles de α. Une *dérivation* (ou *dérivation gauche*) $\alpha_1 \circ \cdots \circ \alpha_n$ d'un arbre t part d'un arbre initial $\alpha_1 \in \sigma(t)$ et continue en substituant itérativement l'arbre α_{k+1} à la feuille non-terminale[1] la plus à gauche de l'arbre $(((\alpha_1 \circ \alpha_2) \dots) \circ \alpha_k)$.

Dans le modèle DOP classique, comme le définit Bod, la probabilité de substituer l'arbre α de racine A à un non-terminal A est le nombre d'occurrences de α dans le corpus d'apprentissage, divisé par le nombre total des occurrences des arbres de racine A :

$$p(\alpha) = \frac{f(\alpha)}{\sum_{\alpha' \in \sigma[\Omega_A]} f(\alpha')} \tag{4.1}$$

La probabilité $p(d)$ d'une dérivation $d = \alpha_1 \circ \cdots \circ \alpha_n$ est alors définie par[2]

$$p(d) = \prod_{\alpha \in d} p(\alpha) = p(\alpha_1) \dots p(\alpha_n)$$

La probabilité d'un arbre est égale à la probabilité pour qu'une quelconque de ses dérivations soit produite, ce qui correspond à la somme des probabilités de ses dérivations. Soient τ un arbre et $d_1, \dots, d_n$ ses différentes dérivations possibles[3], où chaque d_j est constitué d'arbres élémentaires α_{ij} de la grammaire tels que $\alpha_{1j} \circ \cdots \circ \alpha_{m_j j} = \tau$. α_{ij} est ainsi le $i^{\text{ème}}$ arbre élémentaire de la dérivation d_j. Alors la DOP-probabilité $P_{DOP}(\tau)$ de τ est donnée par :

$$P_{DOP}(T) = \sum_{d \Rightarrow T} p(d) = \sum_{d \Rightarrow T} \prod_{t \in d} p(t) = \sum_{j=1}^{n} \prod_{i=1}^{m_j} p(\alpha_{ij}) \tag{4.2}$$

où « $d \Rightarrow T$ » signifie « pour toute dérivation d produisant l'arbre d'analyse T »[4].

4.1.2 Propriétés du modèle DOP

R.Bonnema a mis en évidence certaines faiblesses du modèle DOP « classique », i.e. tel que le définit Bod [Bonnema 99], que l'on va détailler ici.

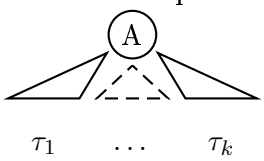
Le modèle DOP est un modèle *ad hoc* en ce sens que la probabilisation des sous-arbres élémentaires ne repose que sur une heuristique d'apprentissage, et non sur un critère mathématique.

[1] on entend par *feuille non-terminale* une feuille d'un arbre dont le label est un non-terminal de la grammaire.

[2] $t \in d$ représente le fait que le fragment t participe à la dérivation d.

[3] Dans une grammaire à substitution d'arbres, au contraire des grammaires hors-contextes, un arbre peut avoir plusieurs dérivations, comme le montre par exemple le tableau de la figure 4.3

[4] Un même arbre d'analyse peut en effet posséder plusieurs dérivations différentes (et ce malgré la convention de réécriture de la feuille non-terminale la plus à gauche en premier).

L'innovation la plus importante apportée par le modèle DOP à l'analyse syntaxique probabiliste est le fait qu'il choisit d'utiliser *tous* les fragments du corpus d'apprentissage comme base de la STSG. On peut se demander cependant s'il est pertinent de définir la probabilité d'un fragment par sa fréquence relative dans le corpus d'apprentissage, parmi les fragments de même racine. En particulier, cela implique une normalisation qui dépend très fortement du nombre de fragments apparaissant dans le corpus avec cette racine. Pour un fragment $\tau = A(\tau_1, \ldots, \tau_k) =$ 

,

la taille de l'ensemble de ses fragments initiaux est donnée par l'équation récursive :

$$|\sigma(\tau)| = \prod_{i=1}^{k} (|\sigma(\tau_i)| + 1)$$

La nature exponentielle de l'opération d'extraction de fragments implique que les gros arbres du corpus apportent une contribution disproportionnellement grande à la masse des probabilités. Le plus gros constituant de racine A des données détermine l'ordre de grandeur de la probabilité de tous les fragments de racine A.

On peut illustrer cet effet par un simple calcul : supposons, pour faire simple, que la base de données soit constituée d'arbres binaires équilibrés ; supposons de plus que l'un des constituants τ_1 de racine A soit de profondeur 6, et que tous les n autres constituants $\tau_2, \ldots, \tau_{n+1}$ de racine A soit de profondeur 5. Pour la profondeur 5, la valeur de $|\sigma(\tau_i)|$ est égale à 458329, et pour la profondeur 6, $|\sigma(\tau_1)|$ vaut environ $2,1.10^{11}$. Cela signifie que la part de la masse de probabilités des fragments de racine A absorbée par les fragments de profondeur 6 est de $(2,1.10^{11} - 45329)/(2,1.10^{11} + n.458329)$. Pour $n = 999$, c'est-à-dire que 1 constituant parmi 1000 est de profondeur 6, cela signifie que $99,8\%$ de la probabilité des fragments de racine A va aux fragments de profondeur 6.

4.1.2.1 Comportement sur des exemples

Comme il n'est pas évident de voir ce que cette propriété implique quant au comportement du modèle en génération d'arbres, prenons deux exemples : le premier exemple concerne une base hypothétique d'arbres, dans laquelle on n'observerait aucune dépendance entre les règles de réécriture hors-contexte qui la constituent. Il s'agit donc d'une base parfaitement décrite par une

SCFG. La définition formelle d'une telle grammaire est donnée ci-dessous :

Soit G une grammaire hors-contexte stochastique. Au regard d'une treebank $\mathcal{C} = \tau_1, \ldots, \tau_n$, les probabilités de production de G sont données par l'estimateur standard de leur fréquence relative : $\hat{p}(A \rightarrow \alpha) = \frac{f(A \rightarrow \alpha)}{f(A)}$. La probabilité assignée par G à un arbre τ est donnée par $p_G(\tau) = \prod_{(A \rightarrow \alpha) \in G} p_G(A \rightarrow \alpha)^{f(A \rightarrow \alpha; \tau)}$. Supposons que $\mathcal{C}$ ait la propriété que pour tous les fragments β y apparaissant :

$$\frac{f(\beta)}{f(r(\beta))} = \prod_{(A \rightarrow \alpha) \in G} p_G(A \rightarrow \alpha)^{f(A \rightarrow \alpha; \beta)} \tag{4.3}$$

Cette équation exprime le fait que les applications de règles de réécriture de G dans $\mathcal{C}$ sont des évènements indépendants. On l'appelle la *contrainte d'indépendance*. La figure 4.1 montre une instanciation possible de $\mathcal{C}$, où G contient les règles :

règle r	$p_G(r)$
$S \rightarrow A$	1/4
$S \rightarrow B$	1/4
$S \rightarrow A\ B$	1/2
$A \rightarrow 0$	1/2
$A \rightarrow 1$	1/2
$B \rightarrow 0$	1/2
$B \rightarrow 1$	1/2

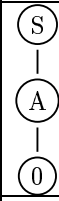
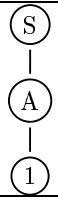
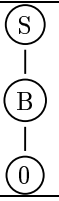
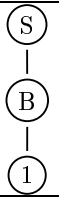
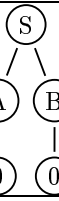
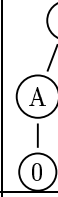
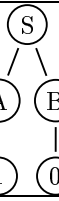
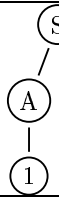
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
τ								
$f(\tau)$	1	1	1	1	1	1	1	1
$F(\tau)$	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

FIG. 4.1 – Base d'arbres $\mathcal{C}_1$ respectant la condition d'indépendance des règles hors-contextes, fréquences en corpus, fréquences relatives.

La figure 4.2 indique pour la base $\mathcal{C}_1$ les valeurs des probabilités élémentaires affectées aux fragments dans le modèle DOP. La probabilité d'un fragment est calculée par la formule (4.1).

Les DOP-probabilités des arbres sont calculées à l'aide de la formule (4.2). Pour chacun des huit arbres considérés dans les bases d'exemples de cette section, le calcul est explicité dans la figure 4.3.

$p(\alpha)$	$a0$	$a1$	$b0$	$b1$	sa	sb	$sa0$	$sa1$	$sb0$	$sb1$
α										
$f(\alpha)$	3	3	3	3	2	2	1	1	1	1
DOP	1/2	1/2	1/2	1/2	2/24	2/24	1/24	1/24	1/24	1/24

$p(\alpha)$	sab	$sa0b$	$sa1b$	$sab0$	$sab1$	$sa0b0$	$sa0b1$	$sa1b0$	$sa1b1$
α									
$f(\alpha)$	4	2	2	2	2	1	1	1	1
DOP	4/24	2/24	2/24	2/24	2/24	1/24	1/24	1/24	1/24

FIG. 4.2 – Probabilités élémentaires des fragments DOP calculées sur la base $\mathcal{C}_1$

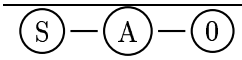
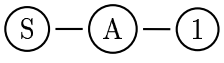
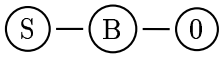
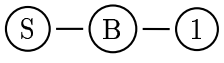
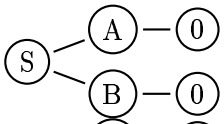
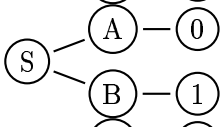
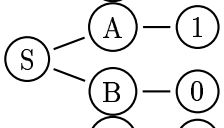
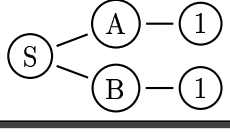
τ	DOP-probabilité
	$sa \cdot a0 + sa0$
	$sa \cdot a1 + sa1$
	$sb \cdot b0 + sb0$
	$sb \cdot b1 + sb1$
	$sab \cdot a0 \cdot b0 + sa0b \cdot b0 + sab0 \cdot a0 + sa0b0$
	$sab \cdot a0 \cdot b1 + sa0b \cdot b1 + sab1 \cdot a0 + sa0b1$
	$sab \cdot a1 \cdot b0 + sa1b \cdot b0 + sab0 \cdot a1 + sa1b0$
	$sab \cdot a1 \cdot b1 + sa1b \cdot b1 + sab1 \cdot a1 + sa1b1$

FIG. 4.3 – Calcul des DOP-probabilités des arbres ; notation : $sa0$ représente la probabilité élémentaire du segment $S - A - 0$.

La figure 4.4 donne les probabilités affectées par les modèles SCFG et DOP aux arbres de la base $\mathcal{C}_1$ lorsque cette base a servi à l'apprentissage des probabilités élémentaires des deux modèles.

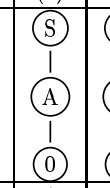
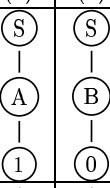
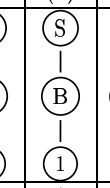
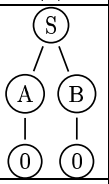
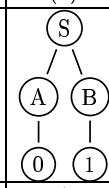
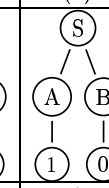
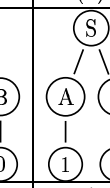
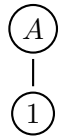
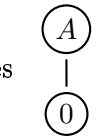
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
τ								
$f(\tau)$	1	1	1	1	1	1	1	1
$F(\tau)$	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{SCFG}	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{DOP}	1/12	1/12	1/12	1/12	1/6	1/6	1/6	1/6

FIG. 4.4 – Probabilités affectées par les modèles SCFG et DOP sur la base $\mathcal{C}_1$ en auto-apprentissage.

Étant donné la contrainte d'indépendance, on peut demander à un modèle de langage sans biais d'affecter aux arbres apparaissant dans $\mathcal{C}_1$ des probabilités égales à leur fréquences relatives. Par définition de $\mathcal{C}_1$, une SCFG dont les paramètres sont calculés à l'aide de l'estimateur standard de la fréquence relative assigne des probabilités correctes à chacun des arbres de $\mathcal{C}_1$. On constate que le modèle DOP semble fortement biaisé en faveur des gros arbres.

Dans le deuxième exemple, voyons ce qui se passe si l'on omet l'hypothèse d'indépendance. On construit un nouveau corpus $\mathcal{C}_2$ en échangeant les  des instances de (2) avec les  des instances de (5). Les arbres (2) deviennent alors égaux aux arbres (1) et les arbres (5) deviennent égaux aux arbres (7), sans que les fréquences relatives des règles hors-contextes soient modifiées. On peut vérifier que l'hypothèse d'indépendance, formulée par l'équation (4.3), ne s'applique plus. Le nouveau corpus est constitué des 6 arbres de la figure 4.5.

La figure 4.6 indique pour la base $\mathcal{C}_2$ les valeurs des probabilités élémentaires affectées aux fragments dans le modèle DOP.

La figure 4.7 donne les probabilités affectées par les modèles SCFG et DOP aux arbres de la base $\mathcal{C}_2$ lorsque cette base a servi à l'apprentissage des probabilités élémentaires des deux modèles.

L'opération effectuée sur le corpus indique que la règle $A \rightarrow 0$ doit être

favorisée par rapport à $A \rightarrow 1$ lorsque la première production est $S \rightarrow A$. Au contraire, $A \rightarrow 0$ a tendance à être évitée lorsque $S \rightarrow A\ B$ est sélectionnée en premier. Le modèle DOP reflète légèrement ces dépendances, ce que ne fait évidemment pas le modèle SCFG. En revanche, le modèle DOP reste fortement biaisé en faveur des gros arbres : l'arbre (6), bien qu'apparaissant deux fois moins souvent que l'arbre (1) dans la base d'apprentissage, se voit affecté d'une DOP probabilité moitié plus grande !

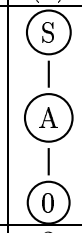
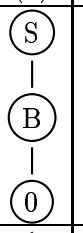
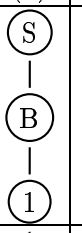
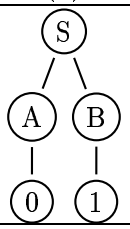
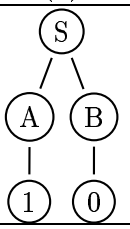
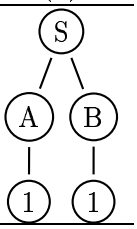
	(1)	(2)	(3)	(4)	(5)	(6)
τ						
$f(\tau)$	2	1	1	1	2	1
$F(\tau)$	1/4	1/8	1/8	1/8	1/4	1/8

FIG. 4.5 – Base d'arbres $\mathcal{C}_2$ ne respectant pas la condition d'indépendance des règles hors-contextes, fréquences en corpus, fréquences relatives.

$p(\alpha)$	$a0$	$a1$	$b0$	$b1$	sa	sb	$sa0$	$sb0$	$sb1$
α									
$f(\alpha)$	3	3	3	3	2	2	2	1	1
DOP	1/2	1/2	1/2	1/2	2/24	2/24	2/24	1/24	1/24

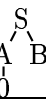
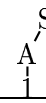
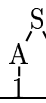
$p(\alpha)$	sab	$sa0b$	$sa1b$	$sab0$	$sab1$	$sa0b1$	$sa1b0$	$sa1b1$
α								
$f(\alpha)$	4	1	3	2	2	1	2	1
DOP	4/24	1/24	3/24	2/24	2/24	1/24	2/24	1/24

FIG. 4.6 – Probabilités élémentaires des fragments DOP dans l'exemple 2

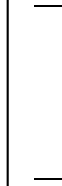
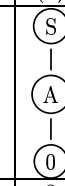
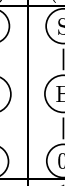
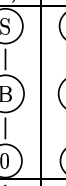
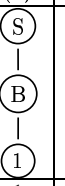
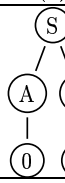
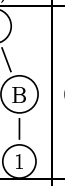
	(1)	(2)	(3)	(4)	(5)	(6)	(2) de $\mathcal{C}_1$	(5) de $\mathcal{C}_1$
τ								
$f(\tau)$	2	1	1	1	2	1	0	0
$F(\tau)$	1/4	1/8	1/8	1/8	1/4	1/8	0	0
P_{SCFG}	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{DOP}	6/48	4/48	4/48	7/48	11/48	9/48	2/48	5/48

FIG. 4.7 – Probabilités affectées par les modèles SCFG et DOP sur la base $\mathcal{C}_2$ en auto-apprentissage, ainsi que sur les 2 arbres de $\mathcal{C}_1$ absents de $\mathcal{C}_2$.

4.1.3 Une probabilisation alternative

Dans [Bonnema 99] une autre paramétrisation a été proposée pour le modèle DOP, qui repose également sur une heuristique, et non sur un critère d'apprentissage. Cette paramétrisation donne la priorité à l'estimation des probabilités des constituants des arbres de la base d'apprentissage, là où le modèle original de Bod ne considère que les fragments[5] ; comme exposé dans la section précédente, on remarque en effet que le modèle DOP affecte à un fragment une probabilité de substitution qui n'est pas proportionnelle à sa fréquence dans le corpus d'apprentissage, mais qui dépend essentiellement de sa taille. Il emploie donc une mesure de probabilité peu conforme au principe qui consiste à préférer des structures apparaissant souvent aux structures apparaissant moins souvent.

Le principe de Bonnema et Scha est le suivant : on dispose d'une collection d'arbres syntaxiques, que l'on ne considère plus comme une collection de fragments. À la place, un arbre doit être transformé en hypothèses sur les fragments utilisés pour le créer. Pour mesurer la probabilité d'un fragment, il faudrait mesurer le nombre de fois où ce fragment est effectivement *utilisé* dans le corpus. Ceci peut être obtenu en considérant chaque arbre du corpus comme l'ensemble de toutes ses dérivations, chaque dérivation consistant en une séquence de substitutions. La part de probabilité apportée à un fragment par un constituant unique est alors donnée par deux facteurs : la fréquence relative du constituant, et la proportion des dérivations de ce constituant qui font appel à ce fragment.

Étant donné un fragment α et un constituant τ, on définit $\phi(\alpha, \tau)$ comme la proportion des dérivations de τ qui commencent avec α. Si $\delta(\tau)$ désigne l'ensemble des dérivations de τ, α_{di} le $i^{\text{ème}}$ fragment de la dérivation d, et

[5] Rappel : les constituants sont les fragments complètement lexicalisés de la base d'apprentissage : leurs feuilles sont des éléments terminaux de la grammaire.

n_d le nombre de fragments de la dérivation d, alors :

$$\phi(\alpha,\tau) = \frac{|\{d = (\alpha_{d1},\ldots,\alpha_{dn_d}) \in \delta(\tau) \text{ t.q. } \alpha_{d1} = \alpha\}|}{|\{\delta(\tau)\}|}$$

On peut noter que les éléments de $\sigma(\tau)$ définissent une partition sur l'ensemble $\delta(\tau)$, c'est-à-dire qu'à chaque fragment initial $\alpha \in \sigma(\tau)$ de τ correspond un sous-ensemble des dérivations de $\delta(\tau)$, et que ces ensembles sont disjoints deux-à-deux, et que leur réunion est $\delta(\tau)$. Ainsi, on a $\sum_{\alpha\in\sigma(\tau)} \phi(\alpha,\tau) = 1$. **Pour chaque constituant τ ayant la même racine que le fragment α, on calcule la probabilité pour que α soit utilisé dans une dérivation de τ, multipliée par la probabilité que le constituant τ soit sélectionné dans le corpus**. La probabilité d'un fragment de racine A exprime donc la probabilité jointe de ces deux évènements. Pour calculer la probabilité de substitution d'un fragment à partir du corpus complet, on prend la somme de ce produit sur l'ensemble des constituants de racine A du corpus :

$$p'(\alpha) = \sum_{r\in\Omega_A} F(\tau)\phi(\alpha,\tau)$$

Il est de plus relativement aisé de calculer la valeur de $\phi(\alpha,\tau)$: la taille de l'ensemble des dérivations du constituant τ est en effet $|\delta(\tau)| = 2^{N(\tau)}$, $N(T)$ étant le nombre de nœuds internes de τ (i.e. les nœuds qui ne sont ni racine, ni feuille de τ). En effet, il y a une bijection entre l'ensemble des ensembles de ces nœuds et l'ensemble des dérivations de τ ; choisir une dérivation est équivalent à choisir l'ensemble des « nœuds de substitution », qui sont les nœuds sur lesquels on effectue une opération de substitution. Le nombre de dérivations de τ commençant par α se calcule de la même façon, et vaut : $2^{N(\tau)-N(\alpha)}$, ce qui donne pour $\phi(\alpha,\tau)$:

$$\phi(\alpha,\tau) = \begin{cases} \frac{2^{N(\tau)-N(\alpha)}}{2^{N(\tau)}} = 2^{-N(\alpha)} & \text{si } \alpha \in \sigma(\tau) \\ 0 & \text{sinon.} \end{cases}$$

En notant Ω_α l'ensemble des constituants de Ω qui commencent par α, on établit facilement que $\sum_{\tau\in\Omega_\alpha} f(\tau) = f(\alpha)$ en donc que $\sum_{\tau\in\Omega_\alpha} F(\tau) = F(\alpha)$. Finalement, on peut poser le principe d'apprentissage de Bonnema et Scha ainsi :

> Étant donné une base d'arbres $\mathcal{C}$ et l'ensemble des fragments Ω apparaissant dans $\mathcal{C}$, on définit la probabilité d'un fragment α par :
>
> $$p'(\alpha) = 2^{-N(\alpha)}F(\alpha)$$
>
> où $N(\alpha)$ est le nombre de nœuds internes de α, et $F(\alpha) = \frac{f(\alpha)}{f(r(\alpha))}$ est la fréquence en corpus de α divisée par la fréquence de sa racine $r(\alpha)$.

4.1.3.1 Comportement sur des exemples

$p(\alpha)$	$a0$	$a1$	$b0$	$b1$	sa	sb	$sa0$	$sa1$	$sb0$	$sb1$
α										
$f(\alpha)$	3	3	3	3	2	2	1	1	1	1
DOP	1/2	1/2	1/2	1/2	2/24	2/24	1/24	1/24	1/24	1/24
$F(\alpha)$	1/2	1/2	1/2	1/2	1/4	1/4	1/8	1/8	1/8	1/8
$N(\alpha)$	0	0	0	0	1	1	1	1	1	1
$2^{-N(\alpha)}$	1	1	1	1	1/2	1/2	1/2	1/2	1/2	1/2
DOP'	1/2	1/2	1/2	1/2	1/8	1/8	1/16	1/16	1/16	1/16

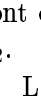
$p(\alpha)$	sab	$sa0b$	$sa1b$	$sab0$	$sab1$	$sa0b0$	$sa0b1$	$sa1b0$	$sa1b1$
α									
$f(\alpha)$	4	2	2	2	2	1	1	1	1
DOP	4/24	2/24	2/24	2/24	2/24	1/24	1/24	1/24	1/24
$F(\alpha)$	4/8	2/8	2/8	2/8	2/8	1/8	1/8	1/8	1/8
$N(\alpha)$	2	2	2	2	2	2	2	2	2
$2^{-N(\alpha)}$	1/4	1/4	1/4	1/4	1/4	1/4	1/4	1/4	1/4
DOP'	1/8	1/16	1/16	1/16	1/16	1/32	1/32	1/32	1/32

FIG. 4.8 – Probabilités élémentaires des fragments dans le modèle DOP' calculées à partir du corpus $\mathcal{C}_1$

On peut comparer le comportement du modèle de Bonnema et Scha, appelé ici *DOP'*, aux modèles DOP et SCFG classiques sur les mêmes exemples que précédemment.

Le différentes étapes du calcul des probabilités élémentaires des fragments sont détaillées dans les figures 4.8 et 4.9 respectivement pour les bases $\mathcal{C}_1$ et $\mathcal{C}_2$.

La figure 4.10 montre le comportement sur le premier exemple, concernant un corpus d'apprentissage respectant la contrainte d'indépendance de l'application des règles hors-contextes (corpus $\mathcal{C}_1$).

Le modèle DOP' affecte sur cet exemple des probabilités correctes à tous les arbres, c'est-à-dire égales à leurs fréquences relatives dans le corpus. Bonnema et Scha ont d'ailleurs démontré que le modèle DOP' est identique à une SCFG lorsque le corpus respecte la contrainte d'indépendance, et ce quelle que soit la grammaire hors-contexte G.

Dans le deuxième exemple (figure 4.11), le modèle DOP' semble refléter les dépendances entre les règles hors-contextes à l'intérieur des arbres. Considérons les arbres (2), (3), (4) et (6), qui ont la même fréquence relative dans le corpus. On voit que l'arbre (5) a une forte dépendance interne. Le fait que les règles $S \rightarrow A\ B$ et $A \rightarrow 0$ tendent à s'éviter se traduit par les probabilités de (4) et (6), qui sont respectivement inférieure et supérieure de

$p(\alpha)$	$a0$	$a1$	$b0$	$b1$	sa	sb	$sa0$	$sb0$	$sb1$
α	A 0	A 1	B 0	B 1	S A	S B	S A 0	S B 0	S B 1
$f(\alpha)$	3	3	3	3	2	2	2	1	1
DOP	1/2	1/2	1/2	1/2	2/24	2/24	2/24	1/24	1/24
$F(\alpha)$	1/2	1/2	1/2	1/2	1/4	1/4	1/4	1/8	1/8
$N(\alpha)$	0	0	0	0	1	1	1	1	1
$2^{-N(\alpha)}$	1	1	1	1	1/2	1/2	1/2	1/2	1/2
DOP'	1/2	1/2	1/2	1/2	1/8	1/8	1/8	1/16	1/16

$p(\alpha)$	sab	$sa0b$	$sa1b$	$sab0$	$sab1$	$sa0b1$	$sa1b0$	$sa1b1$
α	S A B	S A B 0	S A B 1	S A B 0	S A B 1	S A B 0 1	S A B 1 0	S A B 1 1
$f(\alpha)$	4	1	3	2	2	1	2	1
DOP	4/24	1/24	3/24	2/24	2/24	1/24	2/24	1/24
$F(\alpha)$	4/8	1/8	3/8	2/8	2/8	1/8	2/8	1/8
$N(\alpha)$	2	2	2	2	2	2	2	2
$2^{-N(\alpha)}$	1/4	1/4	1/4	1/4	1/4	1/4	1/4	1/4
DOP'	1/8	1/32	3/32	1/16	1/16	1/32	1/16	1/32

FIG. 4.9 – Probabilités élémentaires des fragments dans le modèle DOP' calculées à partir du corpus $\mathcal{C}_2$

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
τ	(S (A 0))	(S (A 1))	(S (B 0))	(S (B 1))	(S (A 0) (B 0))	(S (A 0) (B 1))	(S (A 1) (B 0))	(S (A 1) (B 1))
$f(\tau)$	1	1	1	1	1	1	1	1
$F(\tau)$	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{SCFG}	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{DOP}	1/12	1/12	1/12	1/12	1/6	1/6	1/6	1/6
$P_{DOP'}$	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

FIG. 4.10 – Probabilités affectées par les modèles SCFG et DOP et DOP' sur la base $\mathcal{C}_1$ en auto-apprentissage.

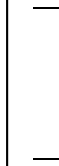
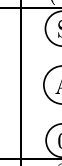
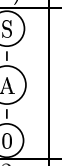
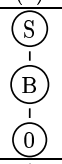
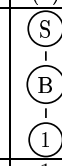
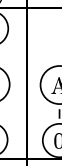
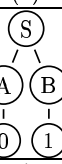
	(1)	(2)	(3)	(4)	(5)	(6)	(2) de $\mathcal{C}_1$	(5) de $\mathcal{C}_1$
τ								
$f(\tau)$	2	1	1	1	2	1	0	0
$F(\tau)$	1/4	1/8	1/8	1/8	1/4	1/8	0	0
P_{SCFG}	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{DOP}	6/48	4/48	4/48	7/48	11/48	9/48	2/48	5/48
$P_{DOP'}$	12/64	8/64	8/64	7/64	11/64	9/64	4/64	5/64

FIG. 4.11 – Probabilités affectées par les modèles SCFG, DOP et DOP' sur la base $\mathcal{C}_2$ en auto-apprentissage, ainsi que sur les 2 arbres de $\mathcal{C}_1$ absents de $\mathcal{C}_2$.

1/64 à la probabilité de modèle SCFG.

Les constituants (2) et (3) ne montrent pas de dépendance interne entre leurs règles. Leur probabilité assignée par DOP' est de fait égale à la probabilité assignée par la SCFG, et tombe entre les valeurs de (4) et (6).

Le modèle DOP' est en réalité meilleur que le modèle DOP surtout du fait qu'il ne favorise pas exagérément les gros constituants par rapport aux petits. Il reste cependant un modèle biaisé en ce sens que la probabilité affectée aux arbres du corpus d'apprentissage n'est pas proportionnelle aux fréquences de ces arbres dans le corpus. L'exemple de la figure 4.11 le montre pour les arbres (4) et (6). On peut trouver ce comportement acceptable, voire y être favorable, car il semble refléter certaines dépendances internes aux arbres entre les règles hors-contextes qui y apparaissent. Mais on peut aussi regretter de ne pas disposer d'un modèle non biaisé. En fait, un contre-argument de celui exposé au paragraphe précédent serait que les règles $S \rightarrow A\ B$ et $A \rightarrow 0$ tendent à s'éviter, *sauf en présence de la règle* $B \rightarrow 1$, ce qui alors n'est pas reflété par le modèle DOP', ni par le modèle DOP.

Une *bonne* probabilisation du modèle DOP, qui n'apparaisse pas biaisée, reste donc à définir. Dans la suite, voyons ce qui se passe si l'on utilise un critère d'apprentissage classique des SCFG pour calculer les probabilités des arbres élémentaires de la grammaire DOP : le critère de la **vraisemblance maximale**.

4.1.4 Maximisation de la vraisemblance

4.1.4.1 Principe

On considère maintenant la grammaire à substitution d'arbres comme un processus stochastique génératif, et le corpus d'arbres $\mathcal{C} = (t_1, \ldots, t_n)$ dont on dispose comme d'un ensemble de productions de ce processus. La probabilité de produire le corpus est donnée par :

$$p(\mathcal{C}) = \prod_{i=1}^{n} p(t_i)$$

On note $\tilde{p}$ la distribution empirique de probabilité, observée sur le corpus $\mathcal{C}$: par définition, $\tilde{p}(t) = \frac{N(t)}{n}$, où $N(t)$ représente le nombre d'occurrences de t dans le corpus. La *log-vraisemblance* de p, relativement à la distribution de probabilité $\tilde{p}$, vaut :

$$\mathcal{L}_{\tilde{p}}(p) = \sum_{t \in \mathcal{C}} \tilde{p}(t) \log p(t) \tag{4.4}$$

Cette log-vraisemblance est proportionnelle au logarithme de la probabilité du corpus. En effet :

$$\begin{aligned}
\log p(\mathcal{C}) &= \log \prod_{i=1}^{n} p(t_i) \\
&= \sum_{i=1}^{n} \log p(t_i) \\
&= \sum_{t \in \mathcal{C}} N(t) \log p(t) \\
&= \sum_{t \in \mathcal{C}} n\tilde{p}(t) \log p(t) \\
&= n \sum_{t \in \mathcal{C}} \tilde{p}(t) \log p(t) \\
&= n\mathcal{L}_{\tilde{p}}(p)
\end{aligned}$$

Trouver les paramètres du modèle qui maximisent la probabilité de produire le corpus $\mathcal{C}$ est donc équivalent à maximiser la log-vraisemblance de ces paramètres, donc à trouver :

$$\hat{p} = \operatorname*{Argmax}_{p} \mathcal{L}_{\tilde{p}}(p) \tag{4.5}$$

Ce critère d'apprentissage est souvent utilisé pour déterminer les paramètres des modèles stochastiques tels que les HMM ou les SCFG. Que peut-il apporter aux grammaires à substitution d'arbres ?

4.1.4.2 Méthode : Expectation Maximization (EM)

Lorsque l'on peut observer dans le corpus d'apprentissage toutes les étapes du processus stochastique, la solution à l'équation (4.5) est triviale : la méthode des multiplicateurs de Lagrange conduit à affecter à chaque transition une probabilité proportionnelle à sa fréquence dans le corpus. C'est par exemple le cas lorsqu'on fait l'apprentissage des probabilités des règles d'une grammaire hors-contexte à partir d'un corpus d'arbres : chaque règle se voit affectée d'une probabilité proportionnelle à sa fréquence dans le corpus. [6]

Ici en revanche, les étapes du processus ne sont pas observables dans le corpus $\mathcal{C}$: un même arbre t possède souvent plusieurs dérivations dans une STSG, et il y a donc plusieurs façons pour le produire à l'aide du processus de génération, sans qu'on puisse dire laquelle est utilisée dans le corpus.

$$\mathcal{L}_{\tilde{p}}(p) = \sum_{t \in \mathcal{C}} \tilde{p}(t) \log \sum_{d \Rightarrow t} p(d) \tag{4.6}$$

La somme sur toutes les dérivations conduisant à l'arbre t, apparaissant dans l'expression de la log-vraisemblance, pose problème lorsqu'il s'agit de maximiser cette valeur. On peut heureusement utiliser un algorithme EM (pour Expectation Maximization [Dempster 77]), qui est un algorithme itératif : plutôt que de maximiser la valeur de $\mathcal{L}_{\tilde{p}}(p)$ directement, on calcule une série de modèles $p_0, p_1, \ldots$ dont la log-vraisemblance va croissant. On ne peut garantir ainsi dans le cas général que l'on tend vers un maximum global de la fonction, mais seulement vers un maximum local.

Une étape de l'algorithme consiste à passer d'un modèle p à un modèle p', en tentant d'obtenir une valeur $\mathcal{L}_{\tilde{p}}(p') - \mathcal{L}_{\tilde{p}}(p)$ aussi grande que possible. On peut écrire :

[6] Il semble que cette méthode ait inspiré le modèle DOP, dans lequel chaque arbre élémentaire, ou fragment, de la grammaire est associé à une probabilité proportionnelle à sa fréquence d'apparition dans le corpus. Cependant, il apparaît que dans le cas d'une grammaire à substitution d'arbres, cela ne maximise plus la vraisemblance du modèle obtenu. Plutôt que de partir de la méthode d'apprentissage des SCFG, on préfère ici s'inspirer de leur critère d'apprentissage, afin d'en dériver une méthode propre aux STSG.

$$\begin{aligned}
\mathcal{L}_{\tilde{p}}(p') - \mathcal{L}_{\tilde{p}}(p) &= \sum_{t\in\mathcal{C}} \tilde{p}(t) \log \frac{p'(t)}{p(t)} \\
&= \sum_{t\in\mathcal{C}} \tilde{p}(t) \log \sum_{d\Rightarrow t} \frac{p'(d)}{p(t)} \\
&= \sum_{t\in\mathcal{C}} \tilde{p}(t) \log \sum_{d\Rightarrow t} \frac{p'(d)p(d|t)}{p(d)} \\
&\geq \sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p(d|t) \log \frac{p'(d)}{p(d)} \\
&\overset{\text{Déf.}}{=} Q(p'|p)
\end{aligned}$$

L'inéquation finale vient de la concavité de la fonction logarithme : $\sum_x p(x) \log q(x) \geq \log \sum_x p(x)q(x)$, quand $\sum_x p(x) = 1$ et $\forall x, p(x) \geq 0$.

$Q(p'|p)$ est une fonction intermédiaire, que l'on cherche à maximiser maintenant.

$$\begin{aligned}
Q(p'|p) &= \sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p(d|t) \log \frac{\prod_{\alpha\in d} p'(\alpha)^{n(\alpha,d)}}{\prod_{\alpha\in d} p(\alpha)^{n(\alpha,d)}} \\
&= \sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p(d|t) \sum_{\alpha\in d} n(\alpha,d) \log \frac{p'(\alpha)}{p(\alpha)} \\
&= \sum_{\alpha\in\mathcal{G}} \left[\sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p(d|t) n(\alpha,d) \right] \log \frac{p'(\alpha)}{p(\alpha)} \\
&= \sum_{\alpha\in\mathcal{G}} \left[\sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p(d|t) n(\alpha,d) \right] \log p'(\alpha) - \Gamma(p)
\end{aligned}$$

où $\mathcal{G}$ désigne l'ensemble des fragments de la grammaire, et Γ est une fonction de p, ne dépendant pas de p'.

D'après ce qui précède, on maximise à chaque étape $Q(p^{(i+1)}|p^{(i)})$ en résolvant :

$$p^{(i+1)} = \underset{p'}{\text{Argmax}} \sum_{\alpha\in\mathcal{G}} \left[\sum_{t\in\mathcal{C}} \tilde{p}(t) \sum_{d\Rightarrow t} p^{(i)}(d|t) n(\alpha,d) \right] \log p'(\alpha) \tag{4.7}$$

sous les contraintes stochastiques :

$$\forall A, \sum_{\alpha\in\Omega_A} p'(\alpha) = 1 \tag{4.8}$$

A nouveau, la solution est donnée par la méthode des multiplicateurs de Lagrange :

$$\tilde{p} \propto \sum_{t \in \mathcal{C}} \tilde{p}(t) \sum_{d \Rightarrow t} p(d|t) n(\alpha, d) \tag{4.9}$$

Le signe « $\propto$ » signifie « proportionnel à ». Le facteur de normalisation est obtenu par l'application des contraintes stochastiques (4.8).

4.1.4.3 Contre-arguments

Si le critère de la vraisemblance maximale semble naturel pour l'apprentissage des modèles statistiques en général, il se révèle assez inefficace dans le cadre des grammaires à substitution d'arbres, du moins lorsque **tous** les fragments de la base d'apprentissage sont extraits et introduits dans la grammaire, comme c'est le cas dans le modèle DOP.

Pour le montrer, reprenons l'exemple du corpus d'apprentissage $\mathcal{C}_2$, dont les arbres apparaissent dans la figure 4.5 p.84.

La log-vraisemblance de la base, lorsque p est la probabilité induite par le modèle, s'écrit[7] :

$$\begin{aligned}
\mathcal{L}_{\tilde{p}}(p) &= \sum_{t \in \mathcal{C}} \tilde{p}(t) \log \sum_{d \Rightarrow t} p(d) \\
&= \frac{1}{4}\log(sa0 + sa \cdot a0) + \frac{1}{8}\log(sb0 + sb \cdot b0) + \frac{1}{8}\log(sb1 + sb \cdot b1) \\
&\quad + \frac{1}{8}\log(sa0b1 + sa0b \cdot b1 + sab1 \cdot a0 + sab \cdot a0 \cdot b1) \\
&\quad + \frac{1}{4}\log(sa1b0 + sa1b \cdot b0 + sab0 \cdot a1 + sab \cdot a1 \cdot b0) \\
&\quad + \frac{1}{8}\log(sa1b1 + sa1b \cdot b1 + sab1 \cdot a1 + sab \cdot a1 \cdot b1)
\end{aligned}$$

En suivant la méthode d'Expectation Maximization, on obtient les équations de mise-à-jour de la figure 4.12.

Quelques itérations sont données dans la figure 4.13.

Les probabilités des arbres du corpus avec le modèle obtenu après convergence de l'algorithme EM sont données dans la figure 4.14, sous l'appellation DEM.

Les distributions de départ $p^{(0)}$ sont des distributions uniformes : les fragments de racine X sont affectés de la probabilité initiale $|\sigma[\Omega_X]|^{-1}$.

On constate que les arbres de la base d'apprentissage obtiennent avec ce modèle des probabilités exactement égales à leur probabilité empirique $\tilde{p}$. Le

[7] Les probabilités des fragments élémentaires sont notées par le nom des fragments en minuscule. Par exemple, $p\left(\textcircled{S}\text{–}\textcircled{A}\text{–}\textcircled{0} \right) = sa0$

$$
\begin{aligned}
a0' &\propto \frac{1}{4}\cdot\frac{sa{\cdot}a0}{sa{\cdot}a0+sa0} + \frac{1}{8}\cdot\frac{a0.(sab.b1+sab1)}{a0.(sab.b1+sab1)+sa0b.b1+sa0b1}\\
a1' &\propto \frac{1}{4}\cdot\frac{a1.(sab.b0+sab0)}{a1.(sab.b0+sab0)+sa1b.b0+sa1b0}\\
&\quad + \frac{1}{8}\cdot\frac{a1.(sab.b1+sab1)}{a1.(sab.b1+sab1)+sa1b.b1+sa1b1}\\
b0' &\propto \frac{1}{8}\cdot\frac{sb.b0}{sb.b0+sb0} + \frac{1}{4}\cdot\frac{b0.(sab.a1+sa1b)}{b0.(sab.a1+sa1b)+sa1b0}\\
b1' &\propto \frac{1}{8}\cdot\frac{sb.b1}{sb.b1+sb1} + \frac{1}{8}\cdot\frac{b1.(sab.a0+sa0b)}{b1.(sab.a0+sa0b)+sab1.a0+sa0b1}\\
&\quad + \frac{1}{8}\cdot\frac{b1.(sab.a1+sa1b)}{b1.(sab.a1+sa0b)+sab1.a1+sa1b1}\\
sa' &\propto \frac{1}{4}\cdot\frac{sa.a0}{sa.a0+sa0}\\
sb' &\propto \frac{1}{8}\cdot\frac{sb.b0}{sb.b0+sb0} + \frac{1}{8}\cdot\frac{sb.b1}{sb.b1+sb1}\\
sa0' &\propto \frac{1}{4}\cdot\frac{sa0}{sa.a0+sa0}\\
sb0' &\propto \frac{1}{8}\cdot\frac{sb0}{sb.b0+sb0}\\
sb1' &\propto \frac{1}{8}\cdot\frac{sb1}{sb.b1+sb1}\\
sab' &\propto \frac{1}{8}\frac{sab.a0.b1}{sab.a0.b1+sa0b.b1+sab1.a0+sa0b1} + \frac{1}{4}\frac{sab.a1.b0}{sab.a1.b0+sa1b.b0+sab0.a1+sa1b0}\\
&\quad + \frac{1}{8}\frac{sab.a1.b1}{sab.a1.b1+sa1b.b1+sab1.a1+sa1b1}\\
sa0b' &\propto \frac{1}{8}\cdot\frac{sa0b.b1}{sab.a0.b1+sa0b.b1+sab1.a0+sa0b1}\\
sa1b' &\propto \frac{1}{4}\frac{sa1b.b0}{sab.a1.b0+sa1b.b0+sab0.a1+sa1b0} + \frac{1}{8}\frac{sa1b.b1}{sab.a1.b1+sa1b.b1+sab1.a1+sa1b1}\\
sab0' &\propto \frac{1}{4}\cdot\frac{sab0.a1}{sab.a1.b0+sa1b.b0+sab0.a1+sa1b0}\\
sab1' &\propto \frac{1}{8}\frac{sab1.a0}{sab.a0.b1+sa0b.b1+sab1.a0+sa0b1} + \frac{1}{8}\frac{sab1.a1}{sab.a1.b1+sa1b.b1+sab1.a1+sa1b1}\\
sa0b1' &\propto \frac{1}{8}\cdot\frac{sa0b1}{sab.a0.b1+sa0b.b1+sab1.a0+sa0b1}\\
sa1b0' &\propto \frac{1}{4}\cdot\frac{sa1b0}{sab.a1.b0+sa1b.b0+sab0.a1+sa1b0}\\
sa1b1' &\propto \frac{1}{8}\cdot\frac{sa1b1}{sab.a1.b1+sa1b.b1+sab1.a1+sa1b1}
\end{aligned}
$$

FIG. 4.12 – Équations de mise-à-jour de l'algorithme EM sur la base $\mathcal{C}_2$

α		A0	A1	B0	B1	Sa	Sb	Sa0	Sb0	Sb1	Sab	Sa0b	Sa1b	Sab0	Sab1	Sa0b1	Sa1b0	Sa1b1	Sa1	Sa0b0	$\mathcal{L}_{\tilde{p}}(p)$
	i																				
$p^{(i)}(\alpha)$	0	0.5	0.5	0.5	0.5	0.083	0.083	0.083	0.041	0.041	0.166	0.041	0.125	0.083	0.083	0.041	0.083	0.041			
$P^{(i)}_{DOP}(\alpha)$		0.5	0.5	0.5	0.5	0.083	0.083	0.125	0.083	0.083	0.166	0.125	0.208	0.166	0.166	0.145	0.229	0.187	0.042	0.104	-1.959
Espérance		1.238	1.171	1.409	1.484	0.666	1	1.333	0.5	0.5	0.871	0.142	0.878	0.363	0.507	0.285	0.727	0.222			
$p^{(i)}(\alpha)$	1	0.513	0.486	0.487	0.512	0.083	0.125	0.166	0.062	0.062	0.108	0.017	0.109	0.045	0.063	0.035	0.090	0.027			
$P^{(i)}_{DOP}(\alpha)$		0.513	0.486	0.487	0.512	0.083	0.125	0.209	0.123	0.126	0.108	0.073	0.162	0.098	0.119	0.106	0.192	0.142	0.041	0.059	-1.846
Espérance		0.986	0.906	1.318	1.450	0.408	0.999	1.591	0.506	0.493	0.729	0.086	0.952	0.229	0.524	0.336	0.945	0.195			
$p^{(i)}(\alpha)$	2	0.521	0.478	0.476	0.523	0.051	0.124	0.198	0.063	0.061	0.091	0.010	0.119	0.028	0.065	0.042	0.118	0.024			
$P^{(i)}_{DOP}(\alpha)$		0.521	0.478	0.476	0.523	0.051	0.124	0.225	0.122	0.127	0.091	0.058	0.162	0.072	0.113	0.106	0.209	0.141	0.024	0.042	-1.807
Espérance		0.789	0.714	1.224	1.405	0.236	0.999	1.763	0.515	0.485	0.594	0.052	0.983	0.131	0.542	0.393	1.128	0.173			
$p^{(i)}(\alpha)$	3	0.524	0.475	0.465	0.534	0.029	0.124	0.220	0.064	0.060	0.074	0.006	0.122	0.016	0.067	0.049	0.141	0.021			
$P^{(i)}_{DOP}(\alpha)$		0.524	0.475	0.465	0.534	0.029	0.124	0.235	0.122	0.127	0.074	0.045	0.158	0.051	0.107	0.109	0.222	0.138	0.014	0.029	-1.780
Espérance		0.648	0.586	1.136	1.358	0.131	0.998	1.868	0.525	0.476	0.474	0.032	0.989	0.070	0.558	0.450	1.267	0.156			
$p^{(i)}(\alpha)$	4	0.524	0.475	0.455	0.544	0.016	0.124	0.233	0.065	0.059	0.059	0.004	0.123	0.008	0.069	0.056	0.158	0.019			
$P^{(i)}_{DOP}(\alpha)$		0.524	0.475	0.455	0.544	0.016	0.124	0.242	0.122	0.127	0.059	0.035	0.151	0.035	0.102	0.112	0.231	0.135	0.008	0.020	-1.763
										...											
Espérance		0.266	0.351	0.782	1.126	0.000	0.983	1.999	0.596	0.420	0.010	0.000	0.916	0.000	0.608	0.733	1.620	0.111			
$p^{(i)}(\alpha)$	22	0.431	0.568	0.409	0.590	0.000	0.122	0.249	0.074	0.052	0.001	0.000	0.114	0.000	0.076	0.091	0.202	0.013			
$P^{(i)}_{DOP}(\alpha)$		0.431	0.568	0.409	0.590	0.000	0.122	0.249	0.124	0.125	0.001	0.000	0.115	0.000	0.076	0.124	0.249	0.125	0.000	0.000	-1.733
Espérance		0.265	0.351	0.781	1.125	0.000	0.983	1.999	0.596	0.420	0.008	0.000	0.915	0.000	0.608	0.734	1.621	0.111			
$p^{(i)}(\alpha)$	23	0.430	0.569	0.409	0.590	0.000	0.122	0.249	0.074	0.052	0.001	0.000	0.114	0.000	0.076	0.091	0.202	0.013			
$P^{(i)}_{DOP}(\alpha)$		0.430	0.569	0.409	0.590	0.000	0.122	0.249	0.124	0.125	0.001	0.000	0.115	0.000	0.076	0.124	0.249	0.125	0.000	0.000	-1.733
Espérance		0.264	0.350	0.780	1.124	0.000	0.983	1.999	0.596	0.419	0.006	0.000	0.915	0.000	0.608	0.735	1.622	0.111			
$p^{(i)}(\alpha)$	24	0.429	0.570	0.409	0.590	0.000	0.122	0.249	0.074	0.052	0.000	0.000	0.114	0.000	0.076	0.091	0.202	0.013			
$P^{(i)}_{DOP}(\alpha)$		0.429	0.570	0.409	0.590	0.000	0.122	0.249	0.124	0.125	0.000	0.000	0.114	0.000	0.076	0.124	0.249	0.125	0.000	0.000	-1.733

FIG. 4.13 – Itérations EM à partir de la base $\mathcal{C}_2$.

modèle « apprend » la base parfaitement, mais on voit immédiatement ce que cela a de fâcheux : *les probabilités associées aux arbres n'apparaissant pas dans la base d'apprentissage sont nulles, et par conséquent, le modèle a une capacité nulle de généralisation.* Si l'on veut utiliser un tel modèle pour une application d'analyse syntaxique, alors toute phrase n'apparaissant pas dans la base d'apprentissage recevra zéro analyse, et la probabilité d'une analyse y apparaissant sera simplement proportionnelle à sa fréquence dans le corpus.

Ce comportement, qui ressemble à un comportement de « sur-apprentissage », n'est pas propre à cet exemple. Revenons en effet au critère d'apprentissage du modèle, i.e. la log-vraisemblance du corpus, donnée par l'équation (4.6) (p.91). Quelles que soient les probabilités affectées aux fragments, il est clair que la somme des probabilités affectées aux arbres du langage (l'ensemble des arbres que la grammaire peut produire) est inférieure ou égale à 1.[8] On est donc amené à trouver le maximum de $\sum_{t\in\mathcal{C}} \tilde{p}(t) \log p(t)$ sous la contrainte $\sum_{t\in\mathcal{C}} p(t) \leq 1$. La méthode des multiplicateurs de Lagrange donne le résultat : $\forall t \in \mathcal{C},\ p(t) = \tilde{p}(t)$. Bien sûr, cet optimum théorique ne peut pas être atteint avec n'importe quel modèle, car chaque modèle impose des contraintes supplémentaires sur p. Cependant, dans le cas du modèle DOP, c'est-à-dire lorsque tous les fragments du corpus d'apprentissage entrent dans la grammaire, et à condition que tous les arbres du corpus aient le même symbole S en leur racine, on peut atteindre cet optimum très facilement, en posant :

$$p(\alpha) = \begin{cases} \tilde{p}(\alpha) & \text{si } \alpha \in \mathcal{C} \\ 0 & \text{si } \alpha \in \Omega_S \setminus \mathcal{C} \\ |\sigma[\Omega_S]|^{-1} & \text{si } \alpha \in \Omega \setminus \Omega_S \end{cases}$$

En effet, avec cette instanciation, le processus de génération d'arbres, qui commence par choisir un fragment de racine S, est forcé de choisir un fragment correspondant à un arbre complet de la base d'apprentissage, avec la probabilité empirique de cet arbre. Comme le premier fragment choisi est un arbre complet, toutes ses feuilles sont des terminaux de la grammaire, et le processus est terminé. Seuls les arbres de la base d'apprentissage peuvent donc être produits, et ils le sont avec une probabilité correspondant à leur probabilité empirique : ce modèle maximise ainsi la vraisemblance du corpus, mais interdit, comme on l'a vu sur l'exemple, toute généralisation.

La table 4.1 indique pour quatre modèles TSG la valeur de la log-vraisemblance du corpus $\mathcal{C}_2$. Les trois modèles étudiés précédemment sont considérés, ainsi que le modèle uniforme.

On remarque que le modèle DOP original produit la log-vraisemblance la plus basse, même comparé au modèle uniforme. La méthode d'apprentissage de ses probabilités élémentaires joue sans doute un rôle secondaire dans le

[8] si la grammaire est *consistante*, la somme des probabilités vaut 1.

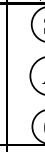
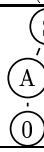
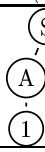
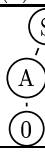
	(1)	(2)	(3)	(4)	(5)	(6)	(2) de $\mathcal{C}_1$	(5) de $\mathcal{C}_1$
τ								
$f(\tau)$	2	1	1	1	2	1	0	0
$\tilde{p}(\tau)$	1/4	1/8	1/8	1/8	1/4	1/8	0	0
P_{SCFG}	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
P_{DOP}	6/48	4/48	4/48	7/48	11/48	9/48	2/48	5/48
$P_{DOP'}$	12/64	8/64	8/64	7/64	11/64	9/64	4/64	4/64
P_{DEM}	1/4	1/8	1/8	1/8	1/4	1/8	0	0

FIG. 4.14 – Probabilités affectées par les modèles SCFG, DOP, DOP' et DEM sur la base $\mathcal{C}_2$ en auto-apprentissage, ainsi que sur les 2 arbres de $\mathcal{C}_1$ absents de $\mathcal{C}_2$.

Modèle	probabilité des fragments	log-vraisemblance
Uniforme	$\lvert\sigma[\Omega_{r(\alpha)}]\rvert^{-1}$	$-1,957$
DOP	$\frac{f(\alpha)}{\sum_{\alpha' \in \sigma[\Omega_A]} f(\alpha')}$	$-1,959$
DOP'	$2^{-N(\alpha)} F(\alpha)$	$-1,900$
DEM	obtenu par algorithme EM	$-1,733$

TAB. 4.1 – Log-vraisemblances de divers modèles sur le corpus $\mathcal{C}_2$.

succès de ce modèle en analyse syntaxique. Le modèle DOP' produit une log-vraisemblance plus grande, et décrit donc mieux le corpus d'apprentissage que le modèle DOP. Ses propriétés semblent adaptées à la généralisation. Enfin, le modèle DEM produit une log-vraisemblance du corpus d'apprentissage nettement plus grande que les autres, réduisant du même coup à zéro la probabilité des arbres n'apparaissant pas dans le corpus d'apprentissage.

Ce tableau montre bien que la méthode de probabilisation du modèle DOP originale est loin d'être la meilleure du point de vue statistique. Les bonnes performances obtenues par DOP viennent essentiellement du mécanisme de production des analyses syntaxiques, qui affecte des scores plus élevés aux arbres les plus gros ayant été vus dans la base d'apprentissage. La log-vraisemblance obtenue avec le modèle DEM laisse à penser qu'il reste de la place pour affiner l'apprentissage des paramètres. L'algorithme EM seul n'est cependant pas la réponse à ce problème, puisqu'il conduit à une grammaire ayant un pouvoir nul de généralisation. La bonne solution est peut-être celle exposée au chapitre 5 : un modèle maximisant la vraisemblance *conditionnelle* du corpus, sans faire tomber à zéro les probabilités des arbres n'y apparaissant pas.

4.2 Complexité de l'analyse

Un apport important de la désambiguïsation probabiliste est qu'elle permet la sélection d'une analyse unique d'une phrase. L'analyse, et la désambiguïsation d'une phrase, avec le modèle DOP, peut être réalisée en maximisant une entité donnée, qui peut être la probabilité de l'analyse, la probabilité de la dérivation... Une question centrale dans la recherche sur le modèle DOP est de savoir quelle entité choisir pour cette maximisation. Dans sa présentation du modèle DOP, Bod [Bod 95] soutient que le critère de « l'analyse la plus probable » (MPP) est le meilleur choix pour la désambiguïsation syntaxique. Du moins théoriquement, ce critère est supérieur à celui de la « dérivation la plus probable » (MPD), ce qui est confirmé par quelques études empiriques sur la petite base d'arbres de la Penn-Treebank (750 arbres au total) [Bod 95, Bod 96].

Le choix du MPP, plutôt que du MPD, comme principe de maximisation est unique parmi les modèles d'analyse probabiliste. Apparemment, ceci est dû au rôle majeur que les SCFG ont joué dans ces modèles : dans les SCFG, il n'existe pas de différence entre MPD et MPP, du fait que chaque analyse est générée par exactement une dérivation. Sur le plan théorique, le MPP est supérieur au MPD dans le sens où il minimise la probabilité de commettre des erreurs. Cependant, comme l'a prouvé K.Sima'an [Sima'an 96a, Sima'an 99], le problème du calcul de l'analyse la plus probable est NP-difficile dans le

cadre des PTSG[9]. Divers travaux ont été entrepris pour contourner ce problème, afin de rendre les STSG exploitables en pratique.

Méthodes d'échantillonnage Monte-Carlo

L'algorithme de Monte-Carlo de Bod [Bod 92] calcule des *approximations* de la dérivation la plus probable et de l'analyse la plus probable.

Le principe de l'échantillonnage est le suivant : la première étape est la construction de l'ensemble des analyses d'une phrase sous la forme d'une *forêt de dérivations*, qui correspond à peu près au contenu d'une table CYK, ou de la table utilisée dans l'algorithme Inside-Outside (voir Annexe B). La construction de cette forêt peut se faire en temps polynomial (par rapport à la taille de la phrase analysée). La deuxième étape est une série de tirages aléatoires parmi les dérivations présentes dans la forêt. Chaque tirage se fait également en un temps polynomial. On compte le nombre de tirages de chaque dérivation, ou de chaque arbre, et l'on en déduit au bout d'un « certain » temps le plus probable, comme celui qui est sorti le plus souvent.

Pour obtenir de bonnes approximations, il est nécessaire d'appliquer la procédure d'échantillonnage itérativement, le résultat étant censé approcher le MPP/MPD lorsque le nombre d'itérations croît.

Les méthodes de Monte-Carlo ont également été étudiées par Goodman [Goodman 98], qui en a donné une implémentation efficace (en moyenne, l'implémentation de Bod nécessitait trois heures trente de calcul par analyse). Goodman a de plus donné des arguments qui laissent à penser que le nombre de tirages nécessaires pour obtenir un taux d'erreur constant est exponentiel en raison de la taille de la phrase analysée.

Enfin, dans [Chappelier 00], J.-C. Chappelier et M. Rajman ont corrigé les fonctions de tirage aléatoire utilisées par Bod et Goodman, en montrant que ces dernières conduisent à une estimation biaisée des probabilités des dérivations. Ils ont également proposé des méthodes de contrôle du nombre d'itérations nécessaires en fonction de la probabilité d'erreur admissible.

Du fait que les algorithmes de Monte-Carlo obtiennent de bonnes approximations seulement lorsque le nombre d'itérations est relativement grand, ils apparaissent plus comme des outils de prototypage que comme des solutions pratiques. Le développement de technologies futures en calculs parallèles et rapides pourrait modifier cela, jusqu'à un certain point. Mais étant donné que la taille des domaines et des applications ira toujours croissante, il sera toujours nécessaire de développer des algorithmes ou des modèles alternatifs pour répondre au problème de la complexité du MPP.

[9] la démonstration en est donnée à l'annexe A.

Extraction de la dérivation la plus probable

K.Sima'an note [Sima'an 99] qu'il peut y avoir des situations où l'extraction de la dérivation la plus probable suffit. En effet, la quantité à maximiser doit être choisie de façon à obtenir le moins d'erreurs possibles. Les taux d'erreur sont mesurés par rapport à une *interprétation* de l'analyse, ou autrement dit, par l'utilisation que l'on veut en faire ; cette utilisation détermine le type d'information que l'analyse doit contenir, et cette information détermine à son tour le type de mesure du taux d'erreur (par exemple le taux d'analyses fausses, de labels faux, de parenthésages faux). Pour l'analyse syntaxique d'une phrase, l'extraction du MPP est celle qui minimise le nombre d'erreurs. Mais pour d'autres tâches, lorsque les mesures d'erreurs reposent sur des paramètres moins contraignants que le taux d'arbres exacts, d'autres quantités peuvent permettre d'obtenir d'aussi bons résultats. Par exemple, on peut considérer la tâche consistant à affecter un parenthésage des phrases, sans se soucier des labels associés aux groupes parenthésés. A cette fin, il est suffisant de maximiser une quantité qui minimise les chances de créer un mauvais parenthésage ; le MPP remplit parfaitement cette tâche, mais des quantités moins complexes à calculer la remplissent aussi bien.

K.Sima'an a ainsi développé un système d'analyse extrayant la dérivation la plus probable dans une grammaire STSG quelconque [Sima'an 99]. Son principal souci est de réduire l'encombrement en mémoire que nécessite cette analyse, en le rendant linéaire sur la taille de la grammaire ; car même si le MPD se fait en temps polynomial par rapport à la taille de la phrase, il requiert la construction de forêts de dérivations qui peuvent saturer par leur taille la mémoire disponible. La méthode qu'il propose consiste en deux passes d'analyses : la première passe utilise une CFG qui est une approximation de la STSG, dans le sens où les analyses qu'elle produit sont un sur-ensemble des analyses obtenues avec la STSG. Cette CFG agit comme un filtre qui produit une première forêt d'analyses, aussi petite que possible. La seconde passe utilise la STSG pour extraire de cette forêt la dérivation la plus probable.

DOP SCFG et extraction de l'analyse de GLRP maximum

Une autre approche originale du modèle DOP est celle proposée par Goodman [Goodman 98]. Plutôt que de projeter une STSG à partir d'un corpus arborisé, il en projette une SCFG. La SCFG est équivalente à la STSG de ce corpus dans le sens où les deux produisent le même espace d'analyses, avec les mêmes probabilités. Cependant, les espaces de dérivations de la STSG et de la SCFG diffèrent. L'implémentation SCFG de DOP ne peut prendre en compte la notion de sous-arbre et interdit le calcul de quantités

qui en dépendent, comme le MPD (de la STSG). De plus, le calcul du MPP, comme prescrit par R.Bod a été prouvé dans [Sima'an 96a] comme étant NP-difficile ; passer au modèle SCFG n'y change évidemment rien. Cependant l'implémentation par une SCFG autorise une analyse syntaxique rapide par la maximisation d'autres quantités que la probabilité d'une analyse.

Outre le mécanisme de projection d'une SCFG, Goodman présente des algorithmes efficaces pour l'analyse. La principale quantité qu'il maximise pour ces analyses est le General Labeled Recall Parse (GLRP) ; le GLRP est l'arbre d'analyse qui maximise l'espérance du *taux de rappel en labels* :

- un *constituant labellisé* est un triplet $\langle i, A, j \rangle$, où $w_i \dots w_j$ sont les mots dominés par le constituant et A est le label du constituant ;
- Si T est un arbre d'analyse d'une phrase et cT l'arbre *correct* pour cette phrase, le *taux de rappel en labels* est le rapport entre le nombre de constituants communs dans T et cT sur le nombre total de constituants de cT ;
- Goodman affecte à chaque constituant apparaissant dans une table d'analyse (ou forêt de dérivations) un poids, calculé comme le rapport entre *le produit de ses probabilités inside et outside* et *la probabilité totale de la phrase*[10] ;
- le GLRP est ensuite déterminé comme l'analyse de la phrase ayant la somme des poids de ses constituants labellisés la plus grande.

Goodman montre que sur le corpus utilisé par Bod, le GLRP obtient des résultats semblables au MPP du modèle DOP STSG.

Une des propriétés séduisantes de l'instanciation de Goodman du modèle DOP est qu'il emploie la base d'apprentissage *telle quelle* pour l'analyse. La taille de la grammaire SCFG est égale au nombre de nœuds de la base, alors que le nombre de sous-arbres d'une grammaire STSG extraite du corpus peut être substantiellement plus grand. Cette propriété peut également apparaître comme un défaut, dans le cas d'un très grand corpus, extrêmement redondant : alors que le nombre de sous-arbres de la STSG peut être limité du fait de la redondance, le nombre de règles de la SCFG croît linéairement avec la taille du corpus (elle compte environ $8 * N * T$ règles, où T est le nombre d'arbres du corpus, et N le nombre moyen de mots par phrase). Goodman n'offre aucun moyen de limiter le nombre de sous-arbres de la grammaire, car la SCFG ne repose pas sur des sous-arbres ; son modèle ne s'applique de plus qu'au modèle DOP, c'est-à-dire avec des probabilités définies par la méthode de Bod, et non aux STSG en général. Il ne peut s'appliquer à une

[10] Comme montré en annexe B, ces probabilités sont calculables en temps polynomial, par factorisation.

autre probabilisation des arbres élémentaires extraits d'un corpus, comme la probabilisation proposée dans cette thèse, ou celle définie par [Bonnema 99].

Chapitre 5

GTSG : modèle de Gibbs pour les grammaires à substitution d'arbres

Ce chapitre décrit nos modèles de grammaires à substitution d'arbres probabilistes orientés pour l'analyse. Comme pour les grammaires GCFG décrites à la section 3.2, cette orientation est réalisée en associant à chaque arbre élémentaire un potentiel, et non une probabilité conditionnée par sa racine. Les probabilités des arbres syntaxiques *conditionnellement à leurs feuilles* sont alors directement définies à partir de ces potentiels, sans passer par la définition d'un processus stochastique génératif.

Dans une première partie, nous envisageons plusieurs modèles pour la description de grammaires orientées vers l'analyse. Ils diffèrent d'abord dans la façon dont ils définissent la probabilité conditionnelle d'un arbre à partir des potentiels de ses fragments[1] : on qualifie de « modèle d'arbres » (section 5.1.1) un modèle définissant cette probabilité *directement*, et de « modèle de dérivations » (section 5.1.2) un modèle définissant cette probabilité comme la somme des probabilités des dérivations de l'arbre[2].

La méthode d'apprentissage considérée pour ces deux types de modèles est basée sur le principe du Maximum d'Entropie (voir chapitre 3). Le « modèle de dérivations » mène à la présentation d'un troisième modèle, dont la méthode d'apprentissage est cette fois-ci basée sur le principe du Maximum de Vraisemblance (section et 5.1.3). En effet, dans ce cas, contrairement à ce qui se passe dans le cas des grammaires GCFG, les deux méthodes d'apprentissage ne conduisent pas au même modèle.

La suite du chapitre est focalisée sur le troisième modèle, qui constitue notre GTSG. Nous définissons les algorithmes permettant la mise en œuvre de l'apprentissage de ses paramètres dans les sections 5.2 et 5.3.1. Nous

[1] Les fragments d'un arbre sont ses sous-arbres.

[2] les exemples de la section 4.1.2.1 p.80 dont de tels « modèles de dérivations »

comparons ensuite les performances des GTSG en analyse avec celles de STSG composées des mêmes arbres élémentaires et entraînées sur les mêmes bases d'apprentissage, dans la section 5.4.

5.1 Modèles envisagés

5.1.1 Un modèle d'arbres à maximum d'entropie

Dans ce modèle, on suit les conventions des modèles à maximum d'entropie définies à la section 3.1, en considérant :

- $x = w$: les variables x représentent les phrases à analyser, ou encore les feuilles des arbres d'analyse,
- $y = t$: les variables y représentent les arbres d'analyse,
- $f_i(y) = f_i(t) = f_{\alpha_i}(t) = \#(\alpha_i, t)$: les fonctions indicatrices représentent le nombre d'occurrences d'un fragment α_i de la grammaire dans un arbre d'analyse t.

Le modèle est défini par un ensemble de distributions de probabilités conditionnelles : $p(t|w)$.

Le corpus d'apprentissage est constitué d'une base d'arbres $\mathcal{C}$ que l'on peut résumer par la distribution de probabilités empiriques :

$$\tilde{p}(w,t) = \begin{cases} \tilde{p}(t) & \text{si } w \text{ sont les feuilles de } t \\ 0 & \text{sinon.} \end{cases}$$

Pour déterminer les distributions $p(t|w)$ à l'aide d'un corpus d'apprentissage, en respectant le principe de maximum d'entropie, on le cherche parmi l'ensemble contraint $\mathcal{P}_\mathcal{C}$:

$$\mathcal{P}_\mathcal{C} = \{p \in \mathcal{P} \text{ t.q. } E_p(f_i) = E_{\tilde{p}}(f_i) \text{ pour } i \in \{1, 2, ..., n\}\} \tag{5.1}$$

Rappelons que les espérances sont définies par :

$$\begin{aligned} E_{\tilde{p}}(f) &= \sum_{w,t} \tilde{p}(w,t) f(t) \\ E_p(f) &= \sum_{w,t} \tilde{p}(w) p(t|w) f(t) \end{aligned}$$

On cherche le modèle $p \in \mathcal{P}_\mathcal{C}$ qui maximise la pseudo-entropie conditionnelle :

$$H_p(T|W) = -\sum_{w,t} \tilde{p}(w) p(t|w) \log p(t|w)$$

5.1.1.1 Résolution analytique

On considère les contraintes associées aux fonctions indicatrices f_{t^0}, c'est-à-dire aux fragments de la grammaire correspondant à des arbres complets t^0 du corpus d'apprentissage. Pour chacun de ces arbres t^0, il faut vérifier : $E_p(f_{t^0}) = E_{\tilde{p}}(f_{t^0})$:

$$\begin{aligned}
& E_p(f_{t^0}) && = E_{\tilde{p}}(f_{t^0}) \\
\Rightarrow\ & \textstyle\sum_{w,t} \tilde{p}(w)p(t|w)f_{t^0}(t) && = \displaystyle\sum_{w,t} \tilde{p}(w,t)f_{t^0}(t) && (5.2)\\
\Rightarrow\ & \textstyle\sum_{w} \tilde{p}(w)p(t^0|w) && = \displaystyle\sum_{w} \tilde{p}(w,t^0) && (5.3)\\
\Rightarrow\ & \tilde{p}(w(t^0))p(t^0|w(t^0)) && = \tilde{p}(t^0) && (5.4)\\
\Rightarrow\ & p(t^0|w(t^0)) && = \frac{\tilde{p}(t^0)}{\tilde{p}(w(t^0))}
\end{aligned}$$

Justification des formules :
(5.3) est justifiée par le fait que $f_{t^0}(t)$ vaut 0 pour $t \neq t^0$ et $f_{t^0}(t^0) = 1$.
(5.4) vient du fait que $p(t^0|w) = 0$ si $w \neq w(t^0)$, c'est-à-dire si la chaîne w ne correspond pas aux feuilles de l'arbre t^0 (notées $w(t^0)$).

Ainsi, pour chaque phrase du corpus d'apprentissage, la répartition de probabilités conditionnelles du modèle est entièrement connue à partir des contraintes d'espérance des fonctions indicatrices. On constate de plus que la pseudo-entropie conditionnelle ne dépend que des probabilités conditionnelles des arbres associés aux phrases du corpus, et est donc invariante par rapport aux arbres correspondant à des phrases hors corpus d'apprentissage.

Si l'on passe au modèle dual, basé sur le maximum de vraisemblance, on cherche à décrire la probabilité conditionnelle $p(t|w)$ sous la forme :

$$p_\lambda(t|w) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_\alpha \lambda_\alpha f_\alpha(t)\right)$$

... et à maximiser en λ la vraisemblance conditionnelle :

$$\mathcal{L}_{\tilde{p}}(p_\lambda) \overset{\text{Déf.}}{=} \log \prod_{w,t} p_\lambda(t|w)^{\tilde{p}(w,t)} = \sum_{w,t} \tilde{p}(w,t) \log p_\lambda(t|w)$$

Une solution de ce problème est :

$$\lambda_\alpha = \begin{cases} \log\gamma \cdot \tilde{p}(t|w) & \text{si } t \in \mathcal{C} \text{ et } \alpha = t \\ 0 & \text{sinon} \end{cases}$$

en faisant tendre γ vers l'infini, de façon à s'assurer que les arbres absents du corpus d'apprentissage aient une probabilité nulle lorsque la phrase

à analyser peut l'être par un des arbres du corpus. En fait, il existe une multitude de solutions au problème : on peut garder n'importe quelle valeur de λ_α pour les fragments qui ne sont pas des arbres complets du corpus d'apprentissage, du moment qu'on s'autorise à faire tendre les autres vers l'infini, dans un rapport constant, pour avoir une solution. Cela vient du fait que les critères auxquels on fait appel ne reposent que sur les phrases du corpus d'apprentissage.

D'autre part, si l'on n'autorise pas les infinis comme valeurs des λ_α, il y a de très fortes chances pour qu'aucune solution n'existe, dans la mesure où n'importe quel arbre issu de la composition de fragments portera une probabilité non nulle. Il faudrait pour avoir une solution que toutes les phrases du corpus d'apprentissage ne puissent être analysées que par des arbres de ce corpus avec la grammaire hors-contexte sous-jacente.

Cette grammaire semble donc mal adaptée à un apprentissage de ce type, car cet apprentissage ne fixe rien sur les paramètres associés aux fragments qui ne sont pas des arbres complets du corpus d'apprentissage. Sur des phrases n'appartenant pas à ce corpus, elle est capable de produire des analyses associées à des probabilités non nulles, mais ces probabilités sont aléatoires, puisqu'elles n'influencent pas le critère d'apprentissage.

On pourrait envisager de contourner le problème :

- en excluant les arbres complets du corpus des arbres élémentaires α de la grammaire, c'est-à-dire en forçant $\lambda_\alpha = 0$ pour $\alpha \in \mathcal{C}$;
- et/ou en réalisant un apprentissage lissé, comme on le fait à la section 5.3.2 pour les GTSG.

Cependant, ces options n'ont pas été explorées plus avant, du fait que l'on se heurte avec ce type de modèle à un problème de complexité de l'analyse syntaxique.

5.1.1.2 Complexité de l'analyse

Un autre argument allant à l'encontre de ce type de modèle est en effet la complexité de son utilisation en analyse syntaxique. Comme signalé à la section 6, l'analyse syntaxique en temps polynomial, à l'aide des modèles à substitution d'arbres, est un problème difficile. La solution à ce problème proposée au chapitre 6 contraint en particulier à se restreindre à des grammaires pour lesquelles le nombre de dérivations d'un arbre d'analyse influe positivement sur sa probabilité, c'est-à-dire que plus un arbre compte de dérivations, plus sa probabilité est grande. Ou encore, plus un arbre compte de fragments de la grammaire, plus sa probabilité est grande. Intuitivement, cette « contrainte » peut sembler raisonnable : si l'on peut construire un même arbre par différents « chemins », en utilisant différentes structures, il a plus de chances de se produire.

Mais dans le cas du modèle d'arbres à maximum d'entropie, rien ne garantit un tel comportement de la grammaire. En effet, le paramètre associé à

un fragment peut prendre une valeur négative (rien ne l'interdit), tendrait à affaiblir la probabilité d'un arbre contenant un fragment donné. On n'a alors pas trouvé dans le cadre de ce travail une grammaire équivalente dont les dérivations les plus probables soient égales aux analyses les plus probables de la grammaire originale, comme on l'a fait pour les GTSG au chapitre 6.

Cet argument final pousse à renoncer à ce type de modèle, plutôt qu'à chercher à déterminer par d'autres critères les paramètres indéterminés auxquels on est confronté. Dans la suite, on s'intéresse à des modèles de dérivations, où la probabilité d'un arbre apparaît comme la somme des probabilités de ses dérivations, ce qui élimine automatiquement ce dernier problème.

5.1.2 Un modèle de dérivations à maximum d'entropie

Dans ce modèle, on suit les conventions des modèles à maximum d'entropie définies en 3.1, en considérant :

- $x = w$: les variables x représentent les phrases à analyser, ou encore les feuilles des arbres d'analyse,
- $y = d$: les variables y représentent les **dérivations** des arbres d'analyse,
- $f_i(y) = f_i(d) = f_{\alpha_i}(d) = \#(\alpha_i, d)$: les fonctions indicatrices représentent le nombre d'occurrences d'un fragment α_i de la grammaire **dans une dérivation** d.

Le modèle est défini par un ensemble de distributions de probabilités conditionnelles : $p(d|w)$.

Le corpus d'apprentissage est constitué d'une base d'arbres $\mathcal{C}$ que l'on peut résumer par la distribution de probabilités empiriques :

$$\tilde{p}(w,t) = \begin{cases} \tilde{p}(t) & \text{si } w \text{ sont les feuilles de } t, \\ 0 & \text{sinon.} \end{cases}$$

En respectant le principe de maximum d'entropie, on cherche les distributions $p(d|w)$ à partir d'un corpus d'apprentissage, parmi l'ensemble contraint $\mathcal{P}_\mathcal{C}$:

$$\mathcal{P}_\mathcal{C} = \{p \in \mathcal{P} \text{ t.q. } E_p(f_i) = E_{\tilde{p}}(f_i) \text{ pour } i \in \{1, 2, ..., n\}\} \tag{5.5}$$

Rappelons que les espérances sont définies par :

$$\begin{aligned} E_{\tilde{p}}(f) &= \sum_{w,d} \tilde{p}(w,d) f(d) \\ E_p(f) &= \sum_{w,d} \tilde{p}(w) p(d|w) f(d) \end{aligned}$$

On cherche le modèle $p \in \mathcal{P}_\mathcal{C}$ qui maximise la pseudo-entropie conditionnelle :

$$H_p(T|W) = -\sum_{w,d} \tilde{p}(w)p(d|w)\log p(d|w)$$

Parties cachées du modèle

Le principal problème de cette approche vient de ce que les dérivations n'apparaissent pas dans le corpus d'apprentissage : ce sont des parties cachées. Or, l'expression de l'espérance empirique des fonctions indicatrices, $E_{\tilde{p}}(f)$, repose sur les probabilités empiriques des dérivations $\tilde{p}(w,d)$, qui ne sont pas observables, le corpus étant uniquement décrit par les probabilités empiriques $\tilde{p}(w,t)$ des arbres.

De fait, dans les modèles à maximum d'entropie tels qu'ils ont été définis à la section 3.1, le corpus d'apprentissage contient des instances observables des évènements que l'on cherche à probabiliser. Ici, les évènements sont des dérivations, qui ne sont pas observables, et la méthode d'apprentissage évoquée à la section 3.1.5 ne peut pas s'appliquer.

Il serait intéressant de chercher à résoudre le problème en utilisant une probabilité pseudo-empirique pour les dérivations : $\tilde{p}'(d|w) = p(d|t) * \tilde{p}(t|w)$. On se trouve alors avec un modèle à maximum d'entropie dont les contraintes sont :

$$\mathcal{P}_{\mathcal{C}} = \left\{ p \in \mathcal{P} / \left[\begin{array}{l} \sum_{w,d} \tilde{p}(w)p(d|w)f_\alpha(d) \\ = \sum_{w,d} \tilde{p}(w,t(d)) * p(d|t(d))f_\alpha(d) \text{ pour } \alpha \in \mathcal{R} \end{array} \right. \right\} \quad (5.6)$$

Ce problème reste pour le moment un problème ouvert. En revanche, on sait résoudre le problème du modèle exponentiel à maximum de vraisemblance, explicité dans la suite. Il faut noter que si les « modèles exponentiels à maximum de vraisemblance » peuvent apparaître comme les problèmes duaux des modèles à maximum d'entropie, ce n'est que lorsque les contraintes de ces modèles sont linéaires. Dans l'équation (5.6), le terme $p(d|t(d))$ dépend du modèle et vaut $\frac{p(d|w)}{\sum_{d'/t(d')=t(d)} p(d'|w)}$, ce qui donne des contraintes couplées non-linéaires sur les probabilités $p(d|w)$. Il n'y a alors pas de dualité entre le modèle présenté dans la suite et celui à entropie maximale qui vient d'être exposé, et dont la solution reste à découvrir (ou du moins, s'il existe un lien entre ces deux modèles, celui-ci reste à définir).

5.1.3 GTSG : Modèle exponentiel de dérivations, maximum de vraisemblance

Le modèle Gibbsien de Grammaires à Substitution d'Arbres (GTSG[3]) proposé dans cette thèse s'inspire du modèle STSG classique et des modèles

[3]GTSG = Gibbsian Tree Substitution Grammar.

à maximum d'entropie. Comme dans un modèle STSG, la probabilité d'*un arbre* d'analyse est posée comme la somme des probabilités des dérivations qu'il admet. L'expression gibbsienne de la probabilité d'*une dérivation* est quant-à-elle issue des modèles à maximum d'entropie. Le mélange des deux types de modèles dans une GTSG s'inspire directement du travail de J.D. Lafferty et de ses modèles de Gibbs-Markov ([Lafferty 96]).

Dans le modèle GTSG, on pose *a priori* la forme des probabilités conditionnelles des dérivations connaissant les phrases par la formule :

$$p_\lambda(d|w) = \frac{1}{Z_\lambda(w)} e^{\sum_{\alpha\in\mathcal{R}} \lambda_\alpha f_\alpha(d)} \tag{5.7}$$

où[4]

$$\begin{cases} Z_\lambda(w) = & \sum_{d\Rightarrow w} e^{\sum_{\alpha\in\mathcal{R}} \lambda_\alpha f_\alpha(d)} \\ f_\alpha(d) = & \text{nombre d'occurrences du fragment } \alpha \text{ dans la dérivation } d \end{cases}$$

Dans la suite, on notera λ et $f(d)$ les vecteurs $(\lambda_{\alpha_1}, \ldots, \lambda_{\alpha_n})$ et $(f_{\alpha_1}(d), \ldots, f_{\alpha_n}(d))$ respectivement, en considérant que les fragments de la grammaire sont identifiés par leur indice $i \in \{1, \ldots, n\}$. La probabilité d'une dérivation s'écrit avec cette convention[5] :

$$p_\lambda(d|w) = \frac{1}{Z_\lambda(w)} e^{\lambda\cdot f(d)}$$

La probabilité conditionnelle d'un arbre d'analyse est définie comme la somme des probabilités conditionnelles de ses dérivations[6] :

$$p_\lambda(t|w) = \sum_{d\Rightarrow t} p_\lambda(d|w)$$

On voit que ce modèle est très proche des modèles DOP, de la même façon que le modèle GCFG est proche du modèle SCFG : la probabilité d'une dérivation apparaît comme un produit des valeurs e^{λ_α} associées à ses fragments (à une constante multiplicative près qui ne dépend que de la phrase w) ; dans le modèle DOP, ces valeurs ne sont autres que les probabilités des fragments conditionnées par leur racine : $p(\alpha|r(\alpha))$. En outre, les probabilités des arbres sont définies de la même manière comme la somme des probabilités des dérivations. Ces similitudes ont pour principal avantage qu'elles permettent

[4] L'indice $d \Rightarrow w$ signifie : pour toutes les dérivations d menant à des arbres dont les feuilles sont w, ou encore, pour toutes les dérivations d produisant la phrase w, si l'on considère la phrase comme le produit d'une dérivation.

[5] $\lambda \cdot f(d)$ est le produit scalaire de λ et $f(d)$.

[6] Une dérivation produit un arbre unique : donc $p(d,t|w) = p(t|d,w)p(d|w) = p(t|d)p(d|w)$ vaut $p(d|w)$ si d est une dérivation de t, et 0 sinon.

d'utiliser les mêmes algorithmes pour l'analyse syntaxique dans les deux cas, lorsqu'il s'agira d'extraire l'arbre le plus probable parmi les arbres possibles pour une phrase.

5.2 Apprentissage des paramètres

Pour l'apprentissage des paramètres du modèle GTSG, on utilise le critère du maximum de vraisemblance appliqué aux probabilités $p_\lambda(w,t)$ des *arbres*. La log-vraisemblance de la probabilité empirique $\tilde{p}$ selon les probabilités p_λ s'écrit :

$$L_{\tilde{p}}(p_\lambda) \stackrel{\text{Déf.}}{=} \log \prod_{w,t} p_\lambda(t|w)^{\tilde{p}(w,t)} = \sum_{w,t} \tilde{p}(w,t) \log p_\lambda(t|w) \tag{5.8}$$

On cherche les paramètres λ qui maximisent cette log-vraisemblance :

$$\lambda* = \operatorname*{Argmax}_\lambda \sum_{w,t} \tilde{p}(w,t) \log p_\lambda(t|w)$$

5.2.1 Dérivation de l'algorithme Improved Iterative Scaling (IIS)

Pour résoudre le problème, on suit la même méthode que celle utilisée avec les grammaires GCFG ; celle-ci est proche de l'algorithme Improved Iterative Scaling présenté page 62. Ce qui diffère ici est la forme de la probabilité des arbres, qui apparaît comme une somme de produits de probabilités, et non plus comme un produit simple.

Plutôt que de chercher à maximiser directement le critère $L_{\tilde{p}}(p_\lambda)$, on améliore itérativement le modèle, à partir d'un modèle initial λ_0. Une itération consiste à passer d'un modèle λ à un modèle λ', en tentant de maximiser sur λ' le critère :

$$\mathcal{A}_\lambda(\lambda') = L_{\tilde{p}}(p_{\lambda'}) - L_{\tilde{p}}(p_\lambda)$$

(ce qui est pour le moment équivalent au critère initial, à une constante près).

En utilisant la formule 5.8 :

$$\begin{aligned}
\mathcal{A}_\lambda(\lambda') &= \sum_{w,t} \tilde{p}(w,t) \log \frac{p_{\lambda'}(t|w)}{p_\lambda(t|w)} \\
&= \sum_{w,t} \tilde{p}(w,t) \log \frac{\sum_{d \Rightarrow t} p_{\lambda'}(d,t|w)}{p_\lambda(t|w)} \\
&= \sum_{w,t} \tilde{p}(w,t) \log \sum_{d \Rightarrow t} \frac{p_{\lambda'}(d,t|w)}{p_\lambda(t|w)} \\
&= \sum_{w,t} \tilde{p}(w,t) \log \sum_{d \Rightarrow t} \frac{p_{\lambda'}(d,t|w)}{\frac{p_\lambda(d,t|w)}{p_\lambda(d|t,w)}} \\
&= \sum_{w,t} \tilde{p}(w,t) \log \sum_{d \Rightarrow t} p_\lambda(d|t,w) \frac{p_{\lambda'}(d,t|w)}{p_\lambda(d,t|w)}
\end{aligned}$$

Du fait de la concavité de la fonction logarithme, et compte tenu du fait que
$\sum_{d \Rightarrow t} p_\lambda(d|t,w) = 1$ pour tous les arbres du corpus (i.e. pour tous les couples (w,t) tels que $\tilde{p}(w,t) \neq 0$) :

$$\begin{aligned}
\mathcal{A}_\lambda(\lambda') \geq \mathcal{B}_\lambda(\lambda') &\overset{\text{Déf.}}{=} \sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t,w) \log \frac{p_{\lambda'}(d,t|w)}{p_\lambda(d,t|w)} \\
&= \sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \log \frac{p_{\lambda'}(d|w)}{p_\lambda(d|w)}
\end{aligned}$$

On obtient ainsi un critère intermédiaire $\mathcal{B}_\lambda(\lambda')$. On sait déjà que son maximum est forcément positif ou nul, car $\mathcal{B}_\lambda(\lambda) = 0$. Si on trouve un λ' pour lequel $\mathcal{B}_\lambda(\lambda')$ est strictement positif, on sait que le modèle en λ' aura une plus grande log-vraisemblance que le modèle en λ. Pour aller au plus vite vers le résultat, on cherche donc à maximiser $\mathcal{B}_\lambda(\lambda')$.

En exprimant les probabilités conditionnelles du modèle à l'aide des potentiels λ_i, il vient :

$$\begin{aligned}
& \mathcal{B}_\lambda(\lambda') \\
= & \sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \log \frac{Z_{\lambda'}(w)^{-1} e^{\lambda' \cdot f(d)}}{Z_\lambda(w)^{-1} e^{\lambda \cdot f(d)}} \\
= & \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) (\lambda' - \lambda) \cdot f(d) \right] - \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \log \frac{Z_{\lambda'}(w)}{Z_\lambda(w)} \right] \\
= & \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) (\lambda' - \lambda) \cdot f(d) \right] - \left[\sum_{w,t} \tilde{p}(w,t) \log \frac{Z_{\lambda'}(w)}{Z_\lambda(w)} \right] \\
& \text{car } \sum_{d \Rightarrow t} p_\lambda(d|t) = 1
\end{aligned}$$

En posant $\Delta\lambda = \lambda' - \lambda$ pour le premier terme, et en considérant l'inégalité de Jensen[7] pour le deuxième terme, il vient :

$$\begin{aligned}
& \mathcal{B}_\lambda(\lambda') \\
\geq\ & \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t)\left(\frac{Z_{\lambda'}(w)}{Z_\lambda(w)} - 1\right)\right] \\
\geq\ & \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t) \frac{Z_{\lambda'}(w)}{Z_\lambda(w)}\right] + 1 \\
\overset{\text{Déf.}}{=}\ & \mathcal{C}_\lambda(\lambda') + 1
\end{aligned}$$

On obtient un deuxième critère intermédiaire $\mathcal{C}_\lambda(\lambda')$, qui minore le premier, et que l'on cherche de nouveau à maximiser (ça devient une habitude)[8]. Pour cela, on développe d'abord l'expression de $Z_{\lambda'}(w)$, puis celle de $Z_\lambda(w)$, en utilisant les formules :

$$\begin{aligned}
Z_{\lambda'}(w) &= \sum_{d \Rightarrow w} e^{\lambda' \cdot f(d)} \\
\frac{1}{Z_\lambda(w)} &= p_\lambda(d|w) \frac{1}{e^{\lambda \cdot f(d)}}
\end{aligned}$$

On obtient :

$$\begin{aligned}
\mathcal{C}_\lambda(\lambda') &= \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t) \frac{\sum_{d \Rightarrow w} e^{\lambda' \cdot f(d)}}{Z_\lambda(w)}\right] \\
&= \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow w} \frac{e^{\lambda' \cdot f(d)}}{Z_\lambda(w)}\right] \\
&= \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow w} p_\lambda(d|w) \frac{e^{\lambda' \cdot f(d)}}{e^{\lambda \cdot f(d)}}\right] \\
&= \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow t} p_\lambda(d|t) \Delta\lambda \cdot f(d)\right] - \left[\sum_{w,t} \tilde{p}(w,t) \sum_{d \Rightarrow w} p_\lambda(d|w) e^{\Delta\lambda \cdot f(d)}\right]
\end{aligned}$$

Ce n'est pas fini (mais presque). Pour pouvoir maximiser facilement $\mathcal{C}_\lambda(\lambda')$, il faut encore découpler les paramètres λ_α apparaissant en argument de l'exponentielle. Pour cela, on pose $f^\#(d) \overset{\text{Déf.}}{=} \sum_{\alpha \in \mathcal{R}} f_\alpha(d)$: $f^\#(d)$ représente simplement le nombre de fragments utilisés dans la dérivation d.

En utilisant la convexité de la fonction exponentielle, combinée à la relation $\sum_{\alpha \in \mathcal{R}} \frac{f_\alpha(d)}{f^\#(d)} = 1$, on obtient l'inégalité suivante :

[7] $\log x \leq x - 1$

[8] On constate qu'à nouveau, $\mathcal{C}_\lambda(\lambda) + 1$, qui minore $\mathcal{B}_\lambda(\lambda)$ vaut zéro, ce qui garantit que son maximum est positif ou nul, et qu'en le trouvant on améliore effectivement $\mathcal{B}_\lambda(\lambda')$.

$$\begin{aligned}\exp\left(\Delta\lambda \cdot f(d)\right) &= \exp\left(\sum_{\alpha\in\mathcal{R}} \Delta\lambda_\alpha f_\alpha(d)\right) \\ &= \exp\left(\sum_{\alpha\in\mathcal{R}} \frac{f_\alpha(d)}{f^\#(d)}\left(\Delta\lambda_\alpha f^\#(d)\right)\right) \\ &\leq \sum_{\alpha\in\mathcal{R}} \frac{f_\alpha(d)}{f^\#(d)} \exp\left(\Delta\lambda_\alpha f^\#(d)\right)\end{aligned}$$

En injectant cette inégalité dans l'expression de $\mathcal{C}_\lambda(\lambda')$, on obtient :

$$\begin{aligned}&\mathcal{C}_\lambda(\lambda') \\ \geq\ &\left[\sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow t} p_\lambda(d|t)\Delta\lambda\cdot f(d)\right] - \left[\sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow w} p_\lambda(d|w)\sum_{\alpha\in\mathcal{R}}\frac{f_\alpha(d)}{f^\#(d)}e^{\Delta\lambda_\alpha f^\#(d)}\right] \\ \overset{\text{Déf.}}{=}\ &\mathcal{D}_\lambda(\lambda')\end{aligned}$$

L'expression $\mathcal{D}_\lambda(\lambda')+1$, qui minore $\mathcal{C}_\lambda(\lambda')+1$, donc $\mathcal{B}_\lambda(\lambda')$, donc $\mathcal{A}_\lambda(\lambda')$ a un maximum positif ou nul, car à nouveau $\mathcal{D}_\lambda(\lambda)+1=0$. On a enfin obtenu un critère facile à maximiser, les exposants en λ_α étant découplés. Il suffit maintenant d'annuler les dérivées partielles en $\Delta\lambda_{\alpha_0}$ pour tout $\alpha_0\in\mathcal{R}$ pour trouver le maximum de $\mathcal{D}_\lambda(\lambda')$. La solution est donnée pour chaque α_0 par l'équation :

$$\begin{aligned}&\frac{\delta\mathcal{D}_\lambda(\lambda')}{\delta\Delta\lambda_{\alpha_0}} = 0 \\ \Leftrightarrow\ &\begin{cases}\sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow t} p_\lambda(d|t) f_{\alpha_0}(d) \\ = \sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow w} p_\lambda(d|w) f_{\alpha_0}(d)\left(e^{\Delta\lambda\alpha_0}\right)^{f^\#(d)}\end{cases}\end{aligned}$$

Pour tout α_0, on est ainsi amené à chercher la solution $\hat{x}$ de l'équation polynomiale en x :

$$\sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow t} p_\lambda(d|t) f_{\alpha_0}(d) = \sum_{w,t}\tilde{p}(w,t)\sum_{d\Rightarrow w} p_\lambda(d|w) f_{\alpha_0}(d) x^{f^\#(d)} \quad (5.9)$$

puis à passer au modèle suivant en affectant à λ_{α_0} la nouvelle valeur $\lambda_{\alpha_0}+\log\hat{x}$. La solution de l'équation polynomiale est facilement obtenue par la méthode de Newton, compte tenu du fait que les coefficients sont tous positifs ou nuls. Il reste cependant à pouvoir déterminer ces coefficients en un temps raisonnable.

5.2.2 Algorithme Inside-Outside

5.2.2.1 Membre de droite

Le membre de droite de la formule (5.9) de ré-estimation des potentiels λ_{α_0} repose sur une double somme : une somme sur les exemples du corpus d'apprentissage, et une somme sur toutes les dérivations possibles d'une phrase. Cette dernière nécessite une factorisation pour pouvoir être effectivement réalisée, le nombre de dérivations d'un seul arbre pouvant être très grand[9], et une phrase pouvant être analysée par de nombreux arbres.

On peut utiliser un algorithme *Inside-Outside* pour factoriser les calculs. Pour cela, on commence par récrire le terme délicat du membre de droite de l'équation de ré-estimation (5.9) sous une forme adéquate :

$$\begin{aligned}
\sum_{d\Rightarrow w} p_\lambda(d|w) f_{\alpha_0}(d) x^{f^\#(d)} &= \frac{1}{Z_\lambda(w)} \sum_{d\Rightarrow w} e^{\sum_{\alpha\in\mathcal{R}} \lambda_\alpha f_\alpha(d)} f_{\alpha_0}(d) x^{f^\#(d)} \\
&= \frac{1}{Z_\lambda(w)} \sum_{d\Rightarrow w} f_{\alpha_0}(d) x^{f^\#(d)} \prod_{\alpha\in d} \left(e^{\lambda_\alpha}\right)^{f_\alpha(d)} \\
&= \frac{1}{Z_\lambda(w)} \sum_{d\Rightarrow w} f_{\alpha_0}(d) \prod_{\alpha\in d} \left(e^{\lambda_\alpha} x\right)^{f_\alpha(d)} \\
&\qquad \text{et en posant } v(\alpha) = e^{\lambda_\alpha} x : \\
&= \frac{1}{Z_\lambda(w)} \sum_{d\Rightarrow w} f_{\alpha_0}(d) \prod_{\alpha\in d} v(\alpha)^{f_\alpha(d)}
\end{aligned}$$

Le terme $\sum_{d\Rightarrow w} f_{\alpha_0}(d) \prod_{\alpha\in d} v(\alpha)^{f_\alpha(d)}$ peut être calculé à l'aide de l'algorithme *Inside-Outside* décrit à l'annexe B (p.177).

De plus, par définition :

$$Z_\lambda(w) = \sum_{d\Rightarrow w} \prod_{\alpha\in d} \left(e^{\lambda_\alpha}\right)^{f_\alpha(d)}$$

On peut donc également calculer $Z_\lambda(w)$ par un algorithme *Inside-Outside*, en posant[10] $v(\alpha) = e^{\lambda_\alpha}$, et $\xi = S$ (S étant le symbole initial de la grammaire, de sorte que pour toute dérivation d conduisant à w, $f_S(d) = 1$). On peut aussi remarquer que la valeur de $Z_\lambda(w)$ est la somme des coefficients des polynômes calculés précédemment pour les α_0 correspondant à des règles de tête S, et ainsi éviter de répéter l'algorithme *Inside-Outside*.

[9] Comme signalé p.86, dans le modèle DOP, si $N(t)$ représente le nombre de nœuds internes de l'arbre t, le nombre de dérivations compatibles avec cet arbre vaut $2^{N(t)}$.

[10] Les notations sont celles définies à l'annexe B.

5.2.2.2 Membre de gauche

On récrit de la même façon la partie « difficile » du membre de gauche de l'équation (5.9).

$$\begin{aligned}\sum_{d\Rightarrow t} p_\lambda(d|t) f_{\alpha_0}(d) &= \frac{1}{Z_\lambda(t)} \sum_{d\Rightarrow t} f_{\alpha_0}(d) e^{\sum_{\alpha\in\mathcal{R}} \lambda_\alpha f_\alpha(d)} \\ &\quad \text{et en posant } v(\alpha) = e^{\lambda_\alpha} : \\ &= \frac{1}{Z_\lambda(t)} \sum_{d\Rightarrow t} f_{\alpha_0}(d) \prod_{\alpha\in d} v(\alpha)^{f_\alpha(d)}\end{aligned}$$

Le terme $\sum_{d\Rightarrow t} f_{\alpha_0}(d) \prod_{\alpha\in d} v(\alpha)^{f_\alpha(d)}$ peut être obtenu par l'algorithme *Inside-Outside*, à condition de filtrer la table d'analyse créée par l'algorithme de la figure B.4(p.191) avant de passer aux étapes suivantes. Ce filtrage peut se réaliser ainsi :

1. établir une correspondance entre les nœuds-têtes de la table et les nœuds de l'arbre t en fonction de leur position par rapport aux mots et de leur label ;
2. supprimer de la table les nœuds n'ayant pas de correspondance ;
3. supprimer de la table les liens (dans *liste_paires*) *associés*[11] à des règles, lorsque la structure interne de la règle correspondante ne se trouve pas dans l'arbre à la position correspondante au père du lien,
4. supprimer de la table les nœuds n'étant pas atteints lors d'un parcours descendant à partir du symbole initial de la cellule $cell[1,n]$

On peut montrer que la table obtenue décrit toutes les dérivations menant à l'arbre t, et uniquement ces dérivations. Les valeurs calculées alors par les algorithmes B.5, B.6, B.7 et B.8 (pp. 192 à 195) sont alors le pendant de celles des formules B.9 (p.184), B.6 (p.184) et B.1 (p.177) restreintes aux dérivations menant à t.

Pour $Z_\lambda(t)$, on remarque comme précédemment que :

$$Z_\lambda(t) = \sum_{d\Rightarrow t} f_S(d) \prod_{\alpha\in d} v(\alpha)^{f_\alpha(d)}$$

car pour toute dérivation, $f_S(d) = 1$. La valeur de $Z_\lambda(t)$ est donc la valeur Externe du symbole S de la cellule $cell[1,n]$, obtenue par l'algorithme B.7 dans la table filtrée.

[11] Les liens de *liste_paires* sont des règles binaires, obtenues par binarisation de la TSG. La première des règles R_b obtenues par binarisation d'une règle R reste « associée » à R. Les autres sont libres. Cette première règle partage en particulier la valeur $v(R)$ de R, alors que les autres ont pour valeur 1, élément neutre pour la multiplication.

5.3 Apprentissage par profondeur croissante

5.3.1 Capacités de généralisation du modèle

Le modèle appris par la méthode précédente souffre en partie du même défaut de sur-apprentissage que le modèle décrit à la section 4.1.4.3 : dans la majeure partie des cas, les valeurs affectées aux arbres ne seront pertinentes que pour des phrases du corpus d'apprentissage. Pour le montrer, prenons par exemple une base d'apprentissage constituée des deux arbres de la figure 5.1 :

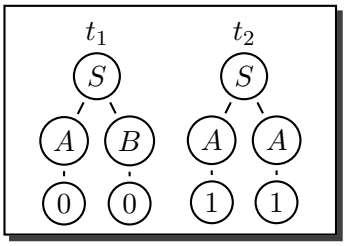


FIG. 5.1 – Base d'apprentissage

Les fragments d'une grammaire d'arbres extraits de cette base sont représentés dans la figure 5.2.

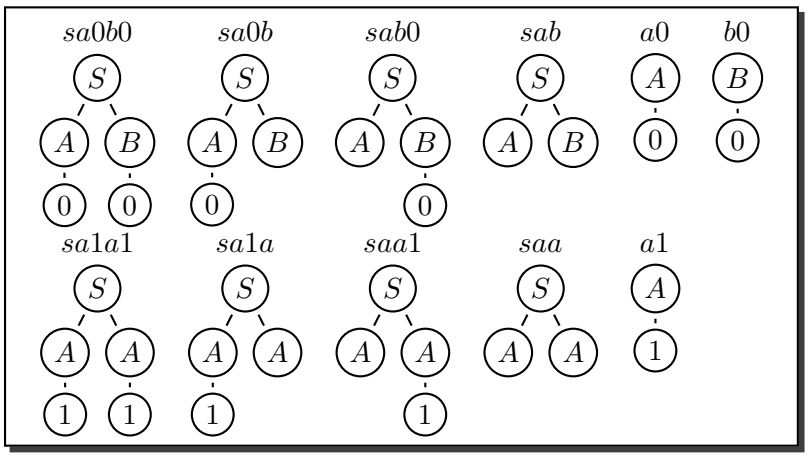


FIG. 5.2 – Fragments de la base de la figure 5.1.

Outre les arbres de la base d'apprentissage, la grammaire de la figure 5.2 peut produire les quatre arbres de la figure 5.3.

L'apprentissage des paramètres selon la méthode précédente tend à maximiser le critère de vraisemblance conditionnelle :

$$p_\lambda(t_1|00) * p_\lambda(t_2|11)$$

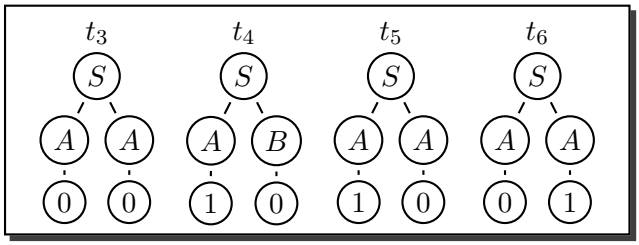


FIG. 5.3 – Arbres pouvant être produits par la TSG déduite de la base 5.1.

On voit que $p_\lambda(t_2|11) = 1$ quelles que soient les valeurs des paramètres, car t_2 est le seul arbre d'analyse de la phrase "11".

L'apprentissage consiste donc en la maximisation de :

$$p_\lambda(t_1|00) = \frac{e^{\lambda_{sa0b0}} + e^{\lambda_{sa0b}+\lambda_{b0}} + e^{\lambda_{sab0}+\lambda_{a0}} + e^{\lambda_{sab}+\lambda_{a0}+\lambda_{b0}}}{\begin{array}{l} e^{\lambda_{sa0b0}} + e^{\lambda_{sa0b}+\lambda_{b0}} + e^{\lambda_{sab0}+\lambda_{a0}} + e^{\lambda_{sab}+\lambda_{a0}+\lambda_{b0}} \\ +e^{\lambda_{sa0a}+\lambda_{a0}} + e^{\lambda_{saa}+\lambda_{a0}+\lambda_{a0}} \end{array}}$$

Le maximum, 1, peut être approché en faisant tendre $(e^{\lambda_{sa0b0}}+e^{\lambda_{sa0b}+\lambda_{b0}}+e^{\lambda_{sab0}+\lambda_{a0}} + e^{\lambda_{sab}+\lambda_{a0}+\lambda_{b0}})$ vers $+\infty$, ou en faisant tendre $(e^{\lambda_{sa0a}+\lambda_{a0}} + e^{\lambda_{saa}+\lambda_{a0}+\lambda_{a0}})$ vers 0. Plusieurs solutions existent, équivalentes pour le critère. Considérons les deux suivantes :

- **solution (1)** : $\lambda_{sa0b0} = +\infty$, les autres paramètres ont des valeurs finies aléatoires.
- **solution (2)** : $\lambda_{a0} = -\infty$, les autres paramètres ont des valeurs finies aléatoires.

Dans les deux cas, les modèles obtenus analysent les phrases "00" et "11" avec les arbres de la base d'apprentissage. En revanche, ils se comportent différemment avec la phrase "10" qui n'y apparaît pas. La solution (2) « bloque » l'arbre t_5 en empêchant l'utilisation du fragment $a0$, alors que la solution (1) ne le bloque pas : Selon la valeur des paramètres laissés libres, il se peut que t_5 soit préféré à t_4 lors d'une analyse syntaxique.

Ainsi, l'apprentissage avec ce critère n'apprend rien sur les analyses des phrases n'apparaissant pas dans la base d'apprentissage. D'une façon générale, si les arbres complets de cette base font partie de la grammaire en tant qu'arbres élémentaires, leurs potentiels associés λ ont tendance à croître vers l'infini avec l'algorithme IIS décrit précédemment, alors que les autres fragments gardent des valeurs insignifiantes, voire tendent vers $-\infty$ dans certains cas. Le critère d'apprentissage est moins restrictif que celui de maximum de vraisemblance dans un modèle génératif, qui pousse les arbres hors-base à avoir une probabilité nulle : ici, les phrases n'apparaissant pas dans la base peuvent recevoir des analyses de probabilités non nulles, le problème étant

que les valeurs de ces probabilités ne sont absolument pas significatives, car elles ne modifient pas le critère d'apprentissage.

5.3.2 Algorithme d'apprentissage par profondeur croissante

Soit $\mathcal{R}$ l'ensemble des fragments d'une TSG. On peut séparer les éléments de $\mathcal{R}$ en deux groupes : $\mathcal{R}_a$, le groupe des fragments qui apparaissent dans d'autres fragments de $\mathcal{R}$, et $\mathcal{R}_b$, ceux qui n'apparaissent que dans eux-mêmes. D'après ce qui précède, et dans le cas où tous les fragments de la base d'apprentissage constituent la grammaire, le critère de la vraisemblance conditionnelle peut être maximisé en fixant les paramètres associés aux fragments de $\mathcal{R}_b$ seulement, laissant ainsi inspécifiés ceux associés aux fragments de $\mathcal{R}_a$.

De façon à ce que les paramètres appris soient plus significatifs sur les exemples n'appartenant pas à la base d'apprentissage, on peut modifier l'algorithme d'apprentissage en considérant des modèles intermédiaires. On considère :

- les sous-ensembles $\mathcal{R}_p$ des fragments de $\mathcal{R}$ de profondeur maximale p, de telle sorte que $\mathcal{R}_p \subset \mathcal{R}_{p+1}$
- les sous-ensembles $\lambda_{\mathcal{R}_p}$ des paramètres associés aux fragments de $\mathcal{R}_p$.
- les grammaires $\mathcal{G}_p = \{\mathcal{T}, \mathcal{NT}, \mathcal{R}_p, \lambda_{\mathcal{R}_p}, S\}$ constituées des règles de profondeur maximale p ($\mathcal{R}_p$), associées à leurs paramètres.

L'apprentissage par profondeur croissante réside dans le calcul des paramètres $\lambda_{\mathcal{R}_1}$ de $\mathcal{G}_1$, puis, par ordre des p croissants, des paramètres $\lambda_{\mathcal{R}_p} \backslash \lambda_{\mathcal{R}_{p-1}}$ du modèle $\mathcal{G}_p$, *en gardant* $\lambda_{\mathcal{R}_{p-1}}$ *constant.*
Les paramètres $\lambda_{\mathcal{R}_p} \backslash \lambda_{\mathcal{R}_{p-1}}$ sont déterminés en maximisant la vraisemblance conditionnelle $L_{\tilde{p}}(p_{\lambda_{\mathcal{R}_p}})$ du corpus selon le modèle $\mathcal{G}_p$, à $\lambda_{\mathcal{R}_{p-1}}$ fixé.

En pratique, l'apprentissage par profondeur croissante ne demande que peu de modifications des algorithmes. Seules les étapes d'initialisation et de mise-à-jour des paramètres sont modifiées dans l'algorithme IIS. Pour apprendre les paramètres $\lambda_{\mathcal{R}_p}$:

- Initialisation :
  - les paramètres $\lambda_{\mathcal{R}_{pmax}} \backslash \lambda_{\mathcal{R}_p}$ sont initialisés à $-\infty$;
  - les paramètres $\lambda_{\mathcal{R}_p} \backslash \lambda_{\mathcal{R}_{p-1}}$ sont initialisés à 0 ;
  - les paramètres $\lambda_{\mathcal{R}_{p-1}}$ conservent leur valeur.
- Mise-à-jour : Seuls les paramètres de $\lambda_{\mathcal{R}_p}$ sont mis à jour.

Évidemment, comme il faut répéter IIS $pmax$ fois [12], l'apprentissage devient relativement long. On pourrait cependant l'optimiser en remarquant

[12] $pmax$ est la profondeur maximale des arbres du corpus d'apprentissage.

que les paramètres convergent plus vite pour les modèles de profondeur plus élevée : le nombre de passes de IIS pourrait être ainsi adapté en fonction de p. À titre évocatif, les tests d'apprentissage, qui ont été réalisés sans optimisation, prennent de un à deux jours de calcul sur une Sun Ultra 10, avec une base de 3000 arbres de profondeur maximum $pmax = 16$, et en effectuant 200 passes IIS par profondeur.

5.4 Expériences

5.4.1 Extraction des grammaires

Le principe d'apprentissage exposé précédemment repose sur la maximisation de la probabilité des arbres d'un corpus conditionnellement à leurs feuilles. Pour rester cohérent avec ce principe, l'objet de cette thèse étant en effet de proposer un système où apprentissage et analyse fonctionnent sur le même modèle, l'analyse syntaxique doit donc s'appuyer sur la probabilité des arbres conditionnellement à leurs feuilles, que constitue la phrase à analyser.

Nous avons choisi d'extraire uniquement l'analyse la plus probable comme solution. Ce problème, MPP (Most Probable Parse), est un problème NP-difficile, dans le cas général, lorsqu'on utilise des grammaires à substitution d'arbres. Cela est démontré en annexe A. Cela signifie qu'on ne connaît aucun algorithme déterministe qui puisse résoudre le problème en un temps polynomial.

Nous nous sommes donc restreints à l'emploi de grammaires pPTSG[13], qui permettent de résoudre le problème MPP en temps polynomial. Ces grammaires sont décrites au chapitre 6, ainsi qu'une méthode pour les extraire à partir d'un corpus d'arbres.

5.4.1.1 pPTSG Min-max

Les grammaires pPTSG Min-max sont obtenues en extrayant du corpus deux types de sous-arbres comme arbres élémentaires : les sous-arbres minimaux, de profondeur 1, correspondant à des règles de grammaire hors-contextes, et les sous-arbres maximaux, qui sont tous les sous-arbres du corpus dont toutes les branches sont étendues jusqu'aux feuilles. Ces grammaires sont obtenues par l'algorithme 8-1.2[14], en utilisant pour chaque arbre de profondeur 1 le schéma d'expansion complet, comprenant toutes ses feuilles. Elles sont donc polynomiales, et l'analyse syntaxique peut être menée par l'extraction de la dérivation la plus probable[15] dans une grammaire équivalente.

[13] pPTSG : polynomial Probabilistic Tree Substitution Grammar
[14] voir p.157
[15] MPD : Most Probable Derivation

5.4.1.2 pPTSG « par têtes »

Les pSTSG « par têtes » sont obtenues de façon similaire : tous les sous-arbres de profondeur 1 sont sélectionnés du corpus, les autres arbres élémentaires étant ceux dont la « branche de tête » est étendue autant que possible. Une « branche de tête » est une branche dont tous les nœuds, associés à des catégories syntaxiques, sont étendus seulement s'ils correspondent à la tête lexicale du nœud qui les domine. Ces grammaires ont également été extraites par l'algorithme 8-1.2 , en utilisant pour chaque arbre de profondeur 1 le schéma d'expansion constitué de la tête lexicale de la règle hors-contexte correspondante. Les têtes lexicales sont définies comme dans [Collins 99].

5.4.2 Résultats expérimentaux

La version de Bod du corpus ATIS a été utilisée pour cette évaluation. Ce corpus (désigné par ATIS-750) consiste en 750 arbres syntaxiques délexicalisés, c'est-à-dire auxquels on a enlevé les feuilles terminales. Les « phrases » analysées sont donc des suites de labels morpho-syntaxiques. La grammaire TSG Min-max extraite compte 2434 arbres élémentaires, dont 381 sont des arbres de profondeur 1, correspondant aux règles hors-contextes, et 2053 arbres « lexicalisés »[16] La pSTSG « par têtes » compte seulement 930 arbres élémentaires, dont 381 de profondeur 1, et 549 « branches de têtes ».

Deux sortes d'expériences ont été réalisées :
- « auto-test » : apprentissage et analyse sur la base d'apprentissage complète.
- test 10% : apprentissage sur 90% de la base, et test sur les 10% restants.

Les tableaux de résultats (5.5, 5.6, 5.4) indiquent les valeurs suivantes :
- Couverture : nombre de phrases recevant au moins une analyse (correcte ou non).
- Justes : nombre de phrases correctement analysées (par référence au corpus).
- Croisement : nombre de phrases recevant un parenthésage croisé avec le parenthésage de la base d'apprentissage.
- les taux de précision et de rappel *en labels* (colonnes `PrécisonL` et `RappelL`) sont obtenus en considérant la séquence des arbres $\tilde{\tau}$ produits par l'analyseur comme un ensemble $E(\tilde{\tau})$ de triplets $< N, g, d >$, où N est un non-terminal et g et d sont les positions dans le corpus du premier et dernier mot de la chaîne analysée par N.
  En comparaison avec une séquence d'arbres de référence $\tilde{\tau}'$, la précision et le rappel sont calculés comme :

$$Tx(Pre)(\tilde{\tau}) = \frac{|E(\tilde{\tau}) \cap E(\tilde{\tau}')|}{|E(\tilde{\tau})|}, \quad Tx(Rap)(\tilde{\tau}) = \frac{|E(\tilde{\tau}) \cap E(\tilde{\tau}')|}{|E(\tilde{\tau}')|}.$$

[16] le lexique étant en fait constitué de labels morpho-syntaxiques.

Dans chaque expérience, le modèle GTSG est comparé au modèle obtenu par l'apprentissage « classique » (DOP), où les arbres élémentaires sont paramétrés par une probabilité proportionnelle à leur fréquence dans le corpus.

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (STSG)					
Test 10%					
	75	37	15	0.927	0.936
	74	38	11	0.936	0.941
	75	44	12	0.955	0.954
	72	40	9	0.948	0.948
	75	49	10	0.953	0.964
Moyenne					
	74.2	41.6	11.4	0.9444	0.9491
		56%	15.3%		
Auto-test					
	750	691	0	0.9966	0.9942
		92.1%	0%		
Modèle GTSG					
Test 10%					
	75	41	10	0.945	0.956
	74	33	14	0.932	0.942
	75	44	11	0.959	0.963
	72	41	9	0.953	0.957
	75	44	15	0.934	0.952
Moyenne					
	74.2	40.6	11.8	0.9448	0.9547
		54.7%	15.9%		
Auto-test					
	750	691	0	0.9968	0.9940
		92.1%	0%		

FIG. 5.4 – Résultats d'analyse avec une TSG Min-max. Le corpus ATIS-750 a été modifié par l'ajout d'une racine commune unique X à chaque arbre.

La figure 5.9 donne également à titre de comparaison les résultats obtenus sur la base ATIS-750 avec des grammaires hors-contextes, dans leur déclinaison stochastique (SCFG) et non-générative (GCFG).

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (STSG)					
Test 10%					
1	74	34	5	0.953	0.951
2	75	43	12	0.949	0.942
3	75	38	12	0.938	0.938
4	75	33	11	0.930	0.932
5	75	28	17	0.918	0.923
6	75	39	10	0.939	0.945
7	73	38	11	0.946	0.948
8	74	28	17	0.920	0.925
9	74	39	11	0.949	0.946
10	74	35	10	0.942	0.944
Moyenne					
	74.4	35.5	11.6	0.9389	0.9398
		47.7%	15.6%		
Auto-test					
	750	664	0	0.9965	0.9918
		88.5%	0%		
Modèle GTSG					
Test 10%					
1	74	39	6	0.950	0.958
2	75	47	8	0.963	0.959
3	75	36	10	0.936	0.947
4	75	32	11	0.926	0.936
5	75	31	15	0.928	0.943
6	75	37	12	0.930	0.944
7	73	39	11	0.947	0.954
8	74	25	15	0.918	0.938
9	74	38	5	0.951	0.958
10	74	34	10	0.929	0.945
Moyenne					
	74.4	35.8	10.3	0.9384	0.9488
		48.1%	13.8%		
Auto-test					
	750	691	0	0.9962	0.9939
		92.1%	0%		

FIG. 5.5 – Résultats d'analyse avec une TSG Min-max apprise sur le corpus ATIS-750. Apprentissage de la GTSG par profondeur croissante, avec 20 passes « par profondeur ».

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (STSG)					
Test 10%					
1	74	31	16	0.917	0.926
2	75	37	13	0.936	0.939
3	74	39	8	0.952	0.953
4	75	41	14	0.930	0.942
5	74	38	5	0.957	0.959
6	72	33	13	0.925	0.948
7	74	34	18	0.917	0.935
8	74	31	9	0.940	0.934
9	73	34	9	0.936	0.951
10	74	38	11	0.939	0.952
Moyenne					
	73.9	35.6	11.6	0.9355	0.9443
		48.2%	15.7%		
Auto-test					
	750	664	0	0.9965	0.9918
		88.5%	0%		
Modèle GTSG					
Test 10%					
1	74	33	14	0.916	0.941
2	75	40	13	0.942	0.958
3	74	41	5	0.957	0.968
4	75	40	13	0.920	0.950
5	74	40	8	0.948	0.956
6	72	34	10	0.932	0.963
7	74	34	12	0.922	0.956
8	74	28	6	0.937	0.941
9	73	38	8	0.938	0.960
10	74	36	8	0.933	0.963
Moyenne					
	73.9	36.4	9.7	0.9349	0.9561
		49.3%	13.1%		
Auto-test					
	750	691	0	0.9962	0.9939
		92.1%	0%		

FIG. 5.6 – Résultats d'analyse avec une TSG Min-max apprise sur le corpus ATIS-750. Apprentissage de la GTSG par profondeur croissante, avec 200 passes « par profondeur ».

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (STSG)					
Test 10%					
1	74	30	15	0.925	0.929
2	75	30	11	0.936	0.938
3	74	27	6	0.955	0.937
4	75	34	8	0.943	0.951
5	74	33	9	0.946	0.940
6	72	32	6	0.942	0.959
7	74	29	13	0.926	0.931
8	74	28	5	0.946	0.937
9	73	31	12	0.929	0.937
10	74	34	9	0.933	0.938
Moyenne					
	73.9	30.8	9.4	0.938	0.940
		41.7%	12.7%		
Auto-test					
	750	414	44	0.971	0.956
		55.2%	5.9%		
Modèle GTSG (ML simple)					
Test 10%					
1	74	31	18	0.915	0.939
2	75	38	9	0.950	0.956
3	73	34	9	0.945	0.954
4	75	40	9	0.941	0.960
5	73	35	7	0.950	0.955
6	72	35	9	0.939	0.958
7	73	38	11	0.939	0.955
8	73	31	5	0.946	0.942
9	73	32	9	0.944	0.953
10	74	32	11	0.933	0.958
Moyenne					
	73.5	34.6	9.7	0.940	0.953
		47.1%	13.2%		
Modèle GTSG (ML itératif)					
Test 10%					
1	74	30	15	0.924	0.926
2	75	32	15	0.938	0.938
3	74	30	6	0.952	0.939
4	75	44	8	0.950	0.954
5	74	31	9	0.939	0.933
6	72	38	11	0.942	0.950
7	74	34	11	0.939	0.945
8	74	30	5	0.952	0.935
9	73	33	10	0.940	0.944
10	74	39	10	0.946	0.948
Moyenne					
	73.9	34.1	10	0.942	0.941
		46.1%	13.5%		
Auto-test					
	750	471	34	0.976	0.971
		62.8%	4.5%		

FIG. 5.7 – Résultats d'analyse avec une TSG « par tête » apprise sur le corpus ATIS-750. Extraction de la *dérivation* la plus probable.

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (STSG)					
Test 10%					
1	74	30	13	0.929	0.934
2	75	30	11	0.940	0.939
3	74	28	7	0.953	0.936
4	75	34	9	0.941	0.947
5	74	31	10	0.940	0.932
6	72	34	10	0.938	0.949
7	74	28	14	0.928	0.931
8	74	26	7	0.941	0.930
9	73	31	12	0.931	0.933
10	74	32	11	0.928	0.933
Moyenne					
	73.9	30.4	10.4	0.937	0.936
		41.1%	14.1%		
Auto-test					
	750	412	57	0.969	0.952
		54.9%	7.6%		
Modèle GTSG (ML simple)					
Test 10%					
1	74	35	16	0.923	0.940
2	75	39	9	0.954	0.955
3	73	35	10	0.948	0.951
4	75	44	8	0.948	0.962
5	73	35	8	0.951	0.954
6	72	38	11	0.940	0.958
7	73	36	10	0.937	0.952
8	73	30	6	0.944	0.940
9	73	33	10	0.941	0.951
10	74	32	12	0.931	0.953
Moyenne					
	73.5	35.7	10	0.942	0.951
		48.5%	13.6%		
Modèle GTSG (ML itératif)					
Test 10%					
1	74	34	13	0.933	0.937
2	75	38	12	0.945	0.949
3	74	33	10	0.948	0.938
4	75	42	9	0.948	0.960
5	74	35	8	0.947	0.948
6	72	35	11	0.934	0.951
7	74	40	12	0.945	0.953
8	74	27	7	0.940	0.940
9	73	33	9	0.941	0.952
10	74	35	11	0.943	0.946
Moyenne					
	73.9	35.2	10.2	0.942	0.948
		47.6%	13.8%		
Auto-test					
	750	479	43	0.976	0.970
		63.9%	5.7%		

FIG. 5.8 – Résultats d'analyse avec une TSG « par tête » apprise sur le corpus ATIS-750. Extraction de l'*analyse* la plus probable.

	Couverture	Justes	Croisés	PrécisionL	RappelL
Modèle classique (SCFG)					
Test 10%					
1	74	29	16	0.922	0.921
2	75	27	17	0.930	0.918
3	74	29	9	0.951	0.933
4	75	35	13	0.941	0.940
5	74	30	9	0.946	0.934
6	72	35	14	0.937	0.941
7	74	30	14	0.933	0.933
8	74	26	7	0.939	0.925
9	73	31	14	0.932	0.928
10	74	34	10	0.935	0.932
Moyenne					
	73.9	30.6	12.3	0.937	0.930
		41.4%	16.6%		
Auto-test					
	750	379	84	0.9655	0.9409
		50.5%	11.2%		
Modèle GCFG					
Test 10%					
1	74	33	14	0.932	0.932
2	75	32	16	0.943	0.937
3	74	29	6	0.960	0.942
4	75	44	9	0.954	0.956
5	74	36	6	0.965	0.953
6	72	37	13	0.936	0.944
7	74	39	12	0.947	0.948
8	74	24	7	0.941	0.929
9	73	35	11	0.945	0.944
10	74	37	11	0.945	0.944
Moyenne					
	73.9	34.6	10.5	0.947	0.943
		46.8%	14.2%		
Auto-test					
	750	413	64	0.9729	0.9541
		55.1%	8.5%		

FIG. 5.9 – Résultats d'analyse avec une CFG apprise sur le corpus ATIS-750. Apprentissage de la GCFG sur 200 passes.

5.4.3 Discussion

Ces expériences amènent quelques remarques :

- Dans la première expérience (figure 5.4), la base a été modifiée par l'ajout d'une même racine X commune à chaque arbre. D'une part, cela rend la base compatible avec les grammaires stochastiques hors contextes ou à substitution d'arbres, qui nécessitent par définition un symbole initial[17]. D'autre part, cela ajoute des paramètres aux modèles obtenus, qui comptent des règles supplémentaires, tout en contraignant les analyses admissibles d'une phrase : on sélectionne en analyse uniquement les arbres ayant pour racine X. En somme, on obtient des grammaires moins ambiguës avec plus de paramètres.
  Pour rester fidèle au concept d'« apprentissage adapté à l'analyse », les grammaires GTSG apprises sur ce corpus ont pris en compte cette contrainte dans leur algorithme d'apprentissage : les coefficients de normalisation $Z_\lambda(w)$ ont été calculés comme une somme sur tous les arbres admissibles analysant la phrase w, **à l'exception** de ceux n'ayant pas X pour racine.
  Malgré cette précaution, les GTSG apprises sur ces bases ne se comportent pas mieux, voire produisent de moins bons résultats en moyenne, que les STSG « classiques », les écarts sur cinq expériences n'apparaissant pas être véritablement significatifs. En *auto-test*, les deux types de modèles obtiennent les scores maximaux.
- La tendance s'inverse lorsque l'on supprime la contrainte du symbole initial unique, et que l'on diminue par là-même le nombre de paramètres des modèles, comme le montrent les figures 5.5 et 5.6. Les GTSG semblent donc plus utiles lorsque les grammaires sont moins contraintes. L'écart se creuse lorsque l'on augmente le nombre de passes d'apprentissage des GTSG, mais l'amplitude du phénomène n'est là encore pas significative.
- Les résultats les plus concluants en faveur des grammaires GTSG, sont obtenus en auto-test : elles produisent le score maximal de $92,1\%$ de phrases correctement analysées[18], l'apprentissage classique n'atteignant que $88,1\%$. C'est là l'illustration du biais théorique constaté par

[17] ce symbole initie le processus aléatoire de génération d'arbre associé à ces grammaires stochastiques

[18] Il peut paraître étonnant que $92,1\%$ soit le score maximal, et non 100%. Cela vient de ce que la base compte en réalité seulement 555 arbres distincts et 512 « phrases » distinctes sur les 750 échantillons. Cela s'explique par le fait que les « phrases » analysées sont en réalité des suites de symboles désignant les catégories morphosyntaxiques des mots : si des suites de symboles identiques apparaissent dans le corpus avec des arbres d'analyse différents, ce n'est pas parce que les annotateurs ont changé d'avis d'une phrase à l'autre, mais parce que des phrases différentes ont produit la même suite de catégories morphosyntaxiques.

Bonnema et Scha[19] [Bonnema 99].

Pour conclure, l'apprentissage non génératif développé dans ce document semble être d'autant plus intéressant que les grammaires comptent moins de paramètres, comme le montre la comparaison entre les modèles SCFG et GCFG de la figure 5.9, qui possèdent 381 paramètres chacun. On peut s'étonner dans cette expérience du score particulièrement élevé du modèle GCFG, qui est le meilleur de toute la batterie de tests sur la partie « test 10% ». Il ne faut cependant pas comparer les expériences entre elles, car chacune repose sur des partitionnements différents de la base originale : on a uniquement cherché à comparer, sur les mêmes partitionnements, les résultats des grammaires non-génératives avec les grammaires stochastiques de même structure.

5.5 Conclusion

Dans ce chapitre, on a exposé un nouveau modèle de grammaire probabiliste non-générative : le modèle Gibbsien de Grammaire à Substitution d'Arbres, ou GTSG. On a donné une méthode complète pour réaliser l'apprentissage de ses paramètres à l'aide d'un corpus arborisé. Cette méthode est complétée par un mécanisme de lissage, exposé sous la dénomination « Apprentissage par profondeur croissante ». Des expériences ont été réalisées sur le corpus ATIS, et l'on a également résultats obtenus avec différents types de GTSG, ainsi que les résultats obtenus sur ce corpus avec une GCFG, qui viennent compléter ceux donnés en section 3.4.

Ces deux types de grammaires non-génératives (GTSG et GCFG) prennent clairement l'avantage sur leurs homologues stochastiques dans les cas où le corpus d'apprentissage et le corpus de test sont confondus. Ils profitent dans ce cas de la suppression des différents « biais » constatés pour les grammaires stochastiques, dus pour les SCFG à la différence entre un apprentissage conditionné par la racine et une utilisation en analyse conditionnée par les feuilles des arbres, et dus dans le cas des STSG à l'absence de critère d'apprentissage. La suppression de ces biais se fait également sentir dans le cas de corpus d'apprentissage et de test séparés, où les grammaires non-génératives obtiennent également les meilleurs résultats. L'amélioration est cependant moins nette qu'on pourrait l'espérer, comme si elle était diluée dans différents facteurs d'erreur qui surviennent alors, comme la trop grande variabilité des corpus utilisés.

On n'a hélas pas mis en évidence dans ce type de tests l'avantage que l'on pourrait tirer de l'autre particularité de la paramétrisation des modèles non-génératifs, à savoir l'absence de normalisation : cette normalisation, dans les modèles basés sur des processus stochastiques, est nécessaire du fait que les paramètres de ces modèles sont des probabilités. Elle pose un problème non

[19] voir p.79.

négligeable lorsqu'on essaie de mélanger ces modèles à d'autres, comme on fait couramment en reconnaissance de la parole. Il serait alors très intéressant de pouvoir mettre en place des modèles non-génératifs tels que ceux exposés ici dans un tel contexte.

Chapitre 6

Grammaires à substitution d'arbres polynomiales

Comme il est démontré dans l'annexe A, trouver l'arbre d'analyse le plus probable dans une grammaire à substitution d'arbres est un problème NP-difficile, ce qui signifie que l'on ne *sait* pas le résoudre de façon déterministe en un temps polynomial par rapport à la taille des données, que sont la grammaire et la phrase à analyser. Cependant, ce constat s'applique aux TSG *en général*, dans le sens où *il existe des TSG pour lesquelles ce problème est NP-complet*; en particulier, les TSG construites par réduction du problème 3SAT sont évidemment dans ce cas.

Dans une application de langage naturel, on peut contourner ce problème en utilisant une TSG qui ne soit pas dans ce cas, par le choix adéquat des arbres élémentaires qui la composent. Les grammaires que nous allons définir dans cette section diffèrent du modèle DOP par le fait que les fragments de la base d'apprentissage ne sont pas tous choisis comme arbres élémentaires. On remarque que de telles restrictions sont fréquemment employées, par exemple par la limitation de la profondeur des fragments, dans le but de limiter la taille de la grammaire. L'originalité de la présente approche est que l'on cherche ici à limiter la complexité de l'analyse par rapport à la taille de la grammaire, et non cette taille elle-même.

6.1 Principe

L'idée générale de cette approche consiste à limiter l'ensemble des arbres élémentaires constituant la grammaire $\mathcal{G}$ de telle sorte que l'on puisse définir une PTSG *équivalente* $\mathcal{G}_{equiv}$ telle que la propriété suivante soit vraie **pour tout arbre d'analyse** A de $\mathcal{G}$:

Condition 1.

Il existe au moins une **dérivation** *dans $\mathcal{G}_{equiv}$ dont le potentiel*
1. *est égal au potentiel dans $\mathcal{G}$ de cet* **arbre** *A ;*
2. *est maximal parmi les potentiels dans $\mathcal{G}_{equiv}$ des dérivations menant à A.*

Notons que dans ce cas, la complexité algorithmique liée à la sommation des probabilités sur les diverses dérivations d'un même arbre d'analyse disparaît, du fait que l'on utilise la grammaire équivalente pour extraire la dérivation la plus probable, et non plus l'arbre le plus probable : avec une telle grammaire, l'analyse syntaxique d'une phrase[1] peut se faire en recherchant dans $\mathcal{G}_{equiv}$ la dérivation de plus fort potentiel produisant cette phrase, qui correspond automatiquement à l'arbre de plus fort potentiel dans $\mathcal{G}$.

Une PTSG pour laquelle une grammaire équivalente peut être produite sera appelée *« grammaire probabiliste à substitution d'arbre polynomiale »*, et notée pPTSG.

Tout l'enjeu de l'approche proposée ici est ainsi de trouver les diverses restrictions des modèles PTSG correspondant à des grammaires polynomiales. Un exemple trivial d'une telle restriction est donné par le modèle où les arbres élémentaires sont limités aux sous-arbres de profondeur 1, grammaire dans laquelle tout arbre d'analyse possède une seule dérivation. La grammaire équivalente est alors elle-même.

Dans la suite, on propose d'examiner les conditions que des PTSG doivent remplir pour être polynomiales, puis de définir des algorithmes pour extraire de telles grammaires à partir d'un corpus d'apprentissage.

6.2 Restrictions successives

Dans le cas général, l'existence d'une grammaire équivalente répondant aux conditions requises pour que $\mathcal{G}$ soit polynomiale est très complexe à déterminer, et sa recherche d'autant plus. On se restreint donc à un sous-ensemble des grammaires polynomiales pour lequel on peut espérer trouver des caractérisations, en fixant la forme de $\mathcal{G}_{equiv}$:

- $\mathcal{G}_{equiv}$ et $\mathcal{G}$ doivent partager les mêmes arbres élémentaires,
- les pseudo-probabilités des arbres élémentaires de $\mathcal{G}_{equiv}$ sont posées comme leur DOP-(pseudo-)probabilités dans $\mathcal{G}$: $P_{\mathcal{G}_{equiv}}(t) = P_{\mathcal{G}_{DOP}}(t)$.

Remarque 1. Pour une STSG, la DOP-probabilité d'un arbre d'analyse est la somme des probabilités de ses dérivations. Le préfixe (pseudo-) rappelle que dans le cadre de cette thèse, les valeurs associées à des arbres ne sont pas forcément normalisées de façon adéquate pour pouvoir être appelées

[1] du moins le problème de la recherche de l'arbre d'analyse de plus fort potentiel

probabilités. La probabilité d'une dérivation, pseudo ou pas, reste le produit de celles des arbres élémentaires qui la constituent. Dans le cas des GTSG, la pseudo-probabilité d'une règle est définie comme l'exponentielle de son potentiel λ.

Remarque 2. Le choix de la forme de $\mathcal{G}_{equiv}$ n'est évidemment pas innocent. On cherche en effet un moyen facile de factoriser le calcul de la DOP-probabilité des arbres d'analyse ; il est donc naturel de penser à utiliser les DOP-probabilités des arbres élémentaires qui apparaissent dans ses dérivations dans $\mathcal{G}$.

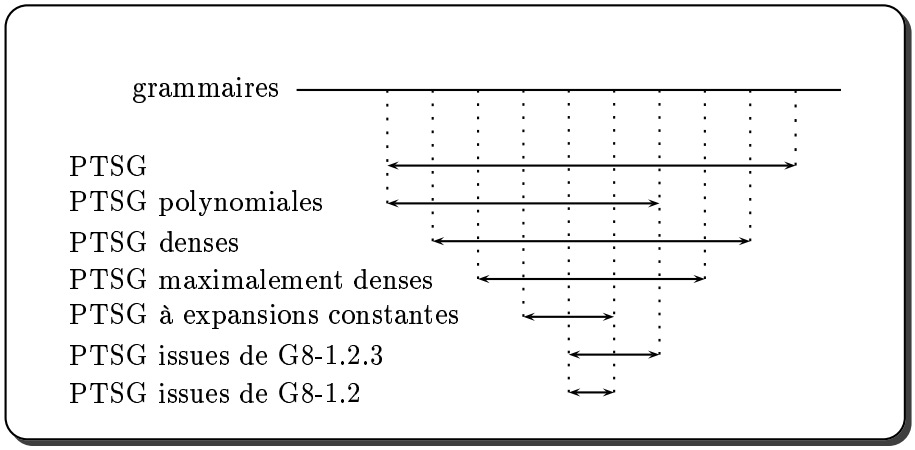
Les méthodes d'extraction de grammaires polynomiales développées nécessitent d'autres restrictions. On définit dans la suite plusieurs types de grammaires :

- **Les grammaires polynomiales** proprement dites sont définies avec la forme de $\mathcal{G}_{equiv}$ précédente, et respectent des conditions de polynomialité plus fortes que la condition 1 définie ci-dessus. On demande en effet que la propriété soit vraie non seulement pour les arbres d'analyses, mais aussi pour les arbres *incomplets*, c'est-à-dire dont les feuilles ne sont pas toutes des terminaux.
- **Les grammaires denses** vérifient la propriété suivante : lorsque deux arbres élémentaires t_1 et t_2 peuvent se superposer partiellement, et que le résultat de cette superposition est un arbre t, alors les parties de t_1 et t_2 qui ne sont pas dans la partie superposée ont des dérivations (dans la grammaire considérée).
- **Les grammaires maximalement denses** sont des grammaires dans lesquelles tous les sous-arbres de profondeur 1 de leurs arbres élémentaires[2] sont eux-mêmes des arbres élémentaires.
- **Les grammaires à expansions constantes** sont des grammaires maximalement denses pour lesquelles chaque arbre élémentaire de profondeur 1 a la même expansion dans les arbres élémentaires qui le contiennent[3].
- **Les grammaires G8-1.2 et G8-1.2.3** respectivement obtenues par l'algorithme du théorème 8 p.157, sans et avec l'étape 3.

Le graphique suivant illustre les restrictions successives auxquelles les grammaires considérées sont soumises : les ensembles de grammaires y sont représentés par des segments sur une droite. Il exprime les différentes inclusions de ces ensembles ; par exemple, on y lit qu'il existe des grammaires denses non polynomiales, que les grammaires maximalement denses sont denses, ou encore que les grammaires G8-1.2.3 à expansions constantes sont les grammaires G8-1.2.

[2] i.e. les sous-arbres représentant des règles de grammaire CFG.

[3] L'expansion d'un arbre t dans un arbre T désigne la liste des feuilles de t auxquelles il faut substituer un sous-arbre pour obtenir T.



Ces différentes inclusions ne disent rien sur les facultés de modélisation du langage des différentes grammaires. Si les grammaires denses sont un sous-ensemble des PTSG, on peut en effet malgré tout transformer n'importe quelle PTSG en une grammaire (maximalement) dense équivalente, c'est-à-dire qui produise les mêmes arbres avec les mêmes potentiels : il suffit de lui ajouter tous les sous-arbres de profondeur 1 de ses arbres élémentaires, en leur associant une pseudo-probabilité nulle. En particulier, c'est le cas pour les grammaires polynomiales. La « restriction » de la recherche de grammaires polynomiales parmi les grammaires maximalement denses polynomiales n'introduit donc pas une perte de généralité dans les langages modélisables, ce qui est en revanche le cas pour les grammaires à expansions constantes, G8-1.2.3 et G8-1.2.

Le tableau suivant récapitule la complexité de certaines opérations pour les différentes grammaires étudiées. La colonne *Caractérisation* donne une idée de la difficulté de caractériser une grammaire comme étant polynomiale, par rapport à sa taille. Dans le cas général, il faut regarder tous les arbres pouvant être produits par cette grammaire ($\mathcal{C}(\mathcal{G})$) et pour chacun trouver s'il existe plusieurs décompositions maximales. Pour les grammaires denses, il faut vérifier que la superposition de deux quelconques de leurs arbres élémentaires ($\mathcal{G}$) est aussi un arbre élémentaire. Les grammaires à expansions constantes sont polynomiales par construction.

La colonne *Calcul* $\mathcal{G}_{equiv}$ indique les algorithmes (connus) nécessaires à l'obtention de la DOP-pseudo-probabilité d'un arbre élémentaire de la grammaire, étape nécessaire de la création d'une grammaire équivalente. L'algorithme *Inside* est de complexité polynomiale par rapport à la taille de l'arbre (cubique dans une implémentation basique). L'algorithme *linéaire* est donné par le lemme 5 de la page 160.

La colonne *Analyse* concerne l'extraction de l'arbre d'analyse de plus fort potentiel.

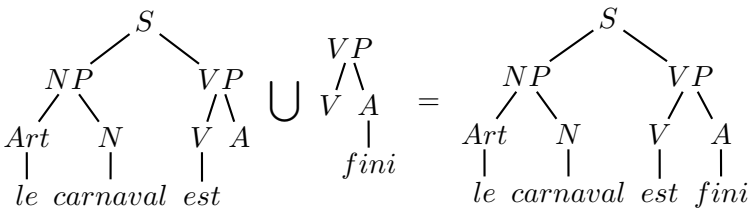
Type	Caractérisation	Calcul $\mathcal{G}_{equiv}$	Analyse
PTSG	$\lvert\mathcal{C}(\mathcal{G})\rvert$	*Inside*	*Inside*
denses	$\lvert\mathcal{G}\rvert^2$	*Inside*	*Inside*
à expansions constantes	0	linéaire	*Inside*

Dans la suite, les différentes grammaires sont détaillées, avec les démonstrations nécessaires. La section 6.3 détaille la façon dont on a représenté les arbres syntaxiques dans les démonstrations suivantes. La section 6.4 donne une caractérisation des PTSG polynomiales. Les grammaires denses sont exposées dans la section 6.5, avec une condition nécessaire et suffisante de polynomialité. Enfin, la section 6.6 traite des grammaires maximalement denses, et donne des algorithmes pour extraire des grammaires maximalement denses polynomiales à partir d'un corpus arborisé d'apprentissage ; ces algorithmes mènent aux grammaires désignées ci-dessus par G8-1.2 et G8-1.2.3.

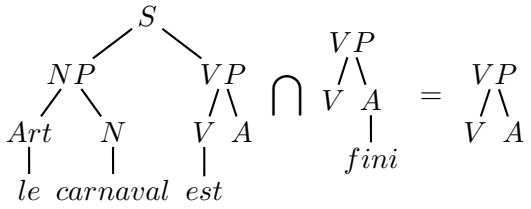
6.3 Représentation des arbres d'analyses

Dans les démonstrations qui suivent, on utilise des notions de réunion, d'intersection et de complémentaire d'arbres ou de sous-arbres syntaxiques. En voici des exemples, avant de passer à des définitions plus formelles :

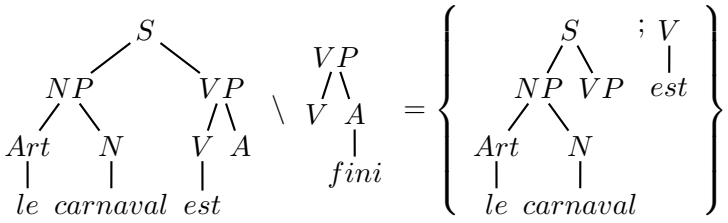
– Union de deux arbres :



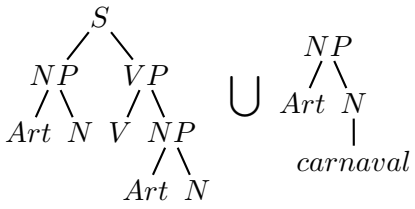
– Intersection :



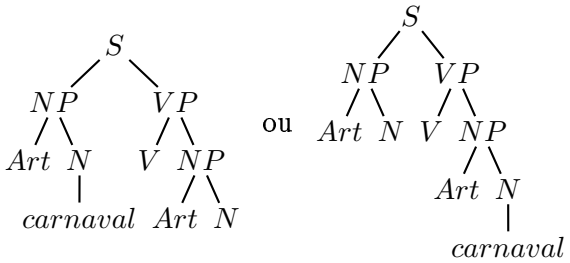
– Complémentaire :



Pour les rendre utilisables, ces notions doivent être définies plus précisément. On remarque en effet que l'opération d'union peut être confuse dans certains cas :



peut vouloir signifier l'un des deux arbres suivants :



De plus, le « complémentaire » d'un arbre dans un autre apparaît comme un ensemble d'arbres, qui n'est donc pas un objet de la même nature, ce qui ne correspond pas à la notion habituelle du complémentaire. Dans les pages qui suivent, pour pouvoir définir précisément ces opérations, on cherche à représenter les arbres syntaxiques **sous une forme ensembliste**.

6.3.1 Définition des représentations d'arbres syntaxiques

Grammaire CFG

Soit R_{CFG} l'ensemble des règles d'une grammaire CFG, dont les éléments seront notés r_i. On notera $ST(r)$ le symbole tête (ou partie gauche) de la règle r, et $SQ(r,i)$ son i^e symbole de queue (ou partie droite). Par exemple, $ST(X \rightarrow Y\ Z) = X$, $SQ(X \rightarrow Y\ Z,\ 1) = Y$ et $SQ(X \rightarrow Y\ Z,\ 2) = Z$.

Représentation

Une *représentation* est un graphe orienté dans lequel les nœuds sont labellisés par des règles de R_{CFG} **et** identifiés par un indice, et où les liens sont ordonnés :

- On définit une représentation T comme la donnée de :
  $T = (N(T), L(T))$, où $N(T)$ est un ensemble de nœuds et $L(T)$ un ensemble de liens entre ces nœuds.
- $N(T) \subset R_{CFG} \otimes X$, où X est un ensemble infini. On notera $n_i = (r_i, x_i)$ un élément de $N(T)$. n_i est composé d'une règle de grammaire r_i et d'une « situation » x_i. Si X est l'ensemble des points d'un plan, x_i se conçoit comme le point où se situe r_i dans la représentation T, qui est alors une représentation graphique dans un plan.
- $L(T) \subset N(T) \otimes \mathbb{N}^* \otimes N(T)$, où $\mathbb{N}^*$ est l'ensemble des entiers naturels strictement positifs. Un élément $l = (n_i, k, n_j)$ de $L(T)$ représente un arc orienté du nœud n_i vers le nœud n_j. k est un numéro d'ordre ; il indique que c'est $SQ(r_i, k)$ (le k^e symbole de queue de r_i) qui est récrit par la règle r_j, ou encore que n_i est lié à n_j par sa k^e « sortie ». On dira de plus que n_i est le père de n_j.

Une « représentation » est destinée à représenter sous forme ensembliste un arbre syntaxique. Un exemple de représentations associées à un arbre est donné par la figure 6.1.

Représentation d'un arbre
La donnée de $T = (N, L)$ ne suffit pas à affirmer que T est la représentation d'un arbre. Pour que ce soit le cas, il faut que chaque nœud de N sauf un ait un père unique dans N et que le graphe soit connexe (lorsque les liens sont considérés comme des relations non-orientées). Le nœud sans père de N est la racine de la représentation de l'arbre.

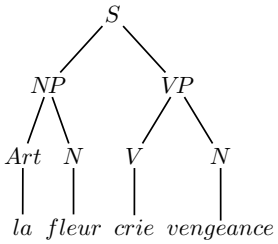
Représentation d'un arbre syntaxique
Enfin, pour que T représente un arbre syntaxique, il faut de plus que pour tout lien $l = (n_i, k, n_j)$ de $L(T)$, on ait : $SQ(n_i, k) = ST(n_j)$.

6.3.2 Passage d'une représentation à une autre

Les objets définis précédemment sont des représentations ensemblistes de graphes. Un arbre syntaxique apparaît comme une classe d'équivalence sur ses différentes représentations, en remarquant que deux représentations d'un arbre diffèrent uniquement par les index associés à ses nœuds. Pour formaliser cela, on va définir dans cette section une relation d'équivalence $\mathcal{S}$ à partir des fonctions permettant de passer d'une représentation à l'autre.

Définition 1.
Soit f_X une injection de X dans lui-même.
Soit f_N la fonction de $R_{CFG} \otimes X$ dans lui-même, qui à (r, x) fait correspondre $(r, f_X(x))$.
Soit f_L la fonction de $N(T) \otimes \mathbb{N}^* \otimes N(T)$ dans lui-même, qui à (r_p, k, r_f) fait correspondre $(f_N(r_p), k, f_N(r_f))$.

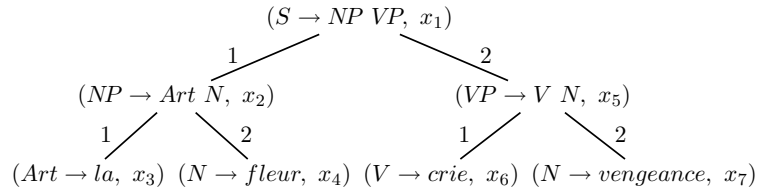
Arbre syntaxique



Une de ses représentations possibles

$$N(T) = \left\{ \begin{array}{l} n_1 = (S \rightarrow NP\ VP,\ x_1); \\ n_2 = (NP \rightarrow Art\ N,\ x_2); \\ n_3 = (Art \rightarrow la,\ x_3); \\ n_4 = (N \rightarrow fleur,\ x_4); \\ n_5 = (VP \rightarrow V\ N,\ x_5); \\ n_6 = (V \rightarrow crie,\ x_6); \\ n_7 = (N \rightarrow vengeance,\ x_7) \end{array} \right\} \quad L(T) = \left\{ \begin{array}{l} (n_1, 1, n_2); (n_1, 2, n_5); \\ (n_2, 1, n_3); (n_2, 2, n_4); \\ (n_5, 1, n_6); (n_5, 2, n_7) \end{array} \right\}$$

Interprétation graphique de la représentation



Une autre de ses représentations possibles

$$N'(T) = \left\{ \begin{array}{l} n'_1 = (S \rightarrow NP\ VP,\ x_4); \\ n'_2 = (NP \rightarrow Art\ N,\ x_3); \\ n'_3 = (Art \rightarrow la,\ x_6); \\ n'_4 = (N \rightarrow fleur,\ x_2); \\ n'_5 = (VP \rightarrow V\ N,\ x_8); \\ n'_6 = (V \rightarrow crie,\ x_{100}); \\ n'_7 = (N \rightarrow vengeance,\ x_{89}) \end{array} \right\} \quad L'(T) = \left\{ \begin{array}{l} (n'_1, 1, n'_2); (n'_1, 2, n'_5); \\ (n'_2, 1, n'_3); (n'_2, 2, n'_4); \\ (n'_5, 1, n'_6); (n'_5, 2, n'_7) \end{array} \right\}$$

FIG. 6.1 – Un arbre syntaxique et deux de ses « représentations » possibles.

Soit f_T la fonction de $(R_{CFG} \otimes X) \otimes (N(T) \otimes \mathbb{N}^* \otimes N(T))$ qui à $T = (N, L)$ fait correspondre $f_T(T) = (f_N(N), f_L(L))$.

Propriétés :

- l'image d'une représentation par f_T est une représentation. En effet, les nœuds apparaissant dans $f_L(L)$ sont images par f_N de ceux apparaissant dans L. Or ces derniers appartenant à N, leurs images appartiennent à $f_N(N)$. Donc $(f_N(N), f_L(L))$ est bien une représentation.
- l'image de la représentation d'un arbre est la représentation d'un arbre. En effet :
  - Soit n' un nœud de $f_T(T)$, et (n'_i, k, n') les liens de $f_L(L)$ qui pointent vers n'. f_X étant injectif, f_N et f_L le sont aussi. f_N est donc une bijection de $N(T)$ dans $f_N(N(T))$, et f_L est une bijection de $L(T)$ dans $f_L(L(T))$. Si n' possède p pères dans $f_T(T)$, cela signifie qu'il existe exactement p liens distincts (n'_i, k, n') dans $f_L(L(T))$. Comme f_L est une bijection de $L(T)$ dans $f_L(L(T))$, les antécédents de ces p liens par f_L sont p liens distincts de $L(T)$, de fils $f_N^{-1}(n')$. De plus tous les liens de $L(T)$ ayant $f_N^{-1}(n')$ comme fils ont pour image par f_L un lien de $f_L(L(T))$ de fils n'. Donc n' a le même nombre de liens entrants dans $f_T(T)$ que $f_N^{-1}(n')$ en a dans T. Du fait de la bijection induite par f_N de $N(T)$ dans $f_N(N(T))$, tous les nœuds de $f_N(N(T))$ sauf un ont un unique lien dans $f_L(L(T))$ où ils apparaissent en tant que fils.
  - Soient deux nœuds distincts n'_1 et n'_2 de $f_N(N(T))$. Leurs antécédents dans $N(T)$ sont encore deux nœuds distincts, et ils sont liés par un chemin représenté par un ensemble connexe d'arcs de $L(T)$. L'image de ces arcs par f_L est encore un ensemble connexe d'arcs, qui relient donc n'_1 et n'_2. Donc le graphe $f_T(T)$ est connexe.
- L'image de la représentation d'un arbre syntaxique est une représentation d'un arbre syntaxique. En effet, les règles grammaticales restent inchangées par f_N, et donc, si $l' = (n'_i, k, n'_j)$ est un lien de $f_L(L(T))$, on a :
  - $SQ(n'_i, k) = SQ(f_N^{-1}(n'_i), k)$
  - et $ST(n'_j) = ST(f_N^{-1}(n'_j))$.

  Comme $(f_N^{-1}(n'_i), k, f_N^{-1}(n'_j))$ fait partie de $L(T)$, et qu'on suppose T « syntaxique », $ST(f_N^{-1}(n'_j)) = SQ(f_N^{-1}(n'_i), k)$, et donc $SQ(n'_i, k) = ST(n'_j)$. Donc $f_T(T)$ est la représentation d'un arbre syntaxique.

Définition 2. Soit $\mathcal{S}$ la relation sur l'ensemble des représentations telle que : $T_1 \mathcal{S} T_2 \Leftrightarrow \exists f_X$ une injection de X dans X telle que $f_T(T_1) = T_2$.

$\mathcal{S}$ est symétrique, transitive, réflexive.

Démonstration de la symétrie de $\mathcal{S}$. Supposons $T_1 \mathcal{S} T_2$, la relation étant établie par une injection donnée f_X. Soit $X(T_1)$ l'ensemble des « positions »

des nœuds de $N(T_1)$, et $X(T_2)$ l'ensemble des « positions » des nœuds de $N(T_2)$. Comme f_X est une injection, et que $X(T_2) = f_X(X(T_1))$, $X(T_1)$ et $X(T_2)$ ont le même nombre d'éléments. Soit f' la fonction de X dans X telle que :

- si $x \in X(T_2)$, $f'(x) = f_X^{-1}(x)$
- f' restreint à $X(T_1)\backslash(X(T_1) \cap X(T_2))$, est une bijection dans $X(T_2)\backslash(X(T_1) \cap X(T_2))$ (ce qui est possible car les deux ensembles ont même nombre d'éléments)
- si $x \notin X(T_1) \cup X(T_2)$, $f'(x) = x$

Alors f' est une injection de X dans X et $f'_T(T_2) = T_1$. Donc $T_2 \mathcal{S} T_1$. □

La transitivité de $\mathcal{S}$ se démontre en utilisant le fait que la composition de deux injections est une injection. La réflexivité est triviale en prenant pour injection f la fonction identité.

6.3.3 Arbre syntaxique

Un arbre syntaxique apparaît comme une classe d'équivalence par la relation $\mathcal{S}$ de représentations syntaxiques.

Dans la suite, on assimilera une représentation à un arbre. Par exemple, si t est une représentation d'un arbre élémentaire d'une TSG $\mathcal{G}$, on notera : $t \in \mathcal{G}$.

Définition 3. On appelle **représentation canonique** d'un arbre sa représentation dans laquelle les nœuds sont indexés dans $\mathbb{N}^*$ par leur ordre de parcours dans un parcours en profondeur d'abord, de la gauche vers la droite, le nœud racine ayant l'index 1.

Par exemple, la première représentation de la figure 6.1 est une représentation canonique si $X = \mathbb{N}^*$ et $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 2, 3, 4, 5, 6, 7)$.

6.3.4 Opérateurs sur les représentations

Définition 4. On définit l'union, l'intersection et l'inclusion de deux représentations ainsi :

- $T_1 \cup T_2 = (N(T_1) \cup N(T_2), L(T_1) \cup L(T_2))$
- $T_1 \cap T_2 = (N(T_1) \cap N(T_2), L(T_1) \cap L(T_2))$
- $T_1 \subset T_2 \Leftrightarrow N(T_1) \subset N(T_2)$ et $L(T_1) \subset L(T_2)$

Remarque 3. Lorsque T_1 et T_2 sont des représentations, si $T_1 \cup T_2$ est une représentation syntaxique, alors $T_1 \cap T_2$ en est une également.

Remarque 4. Lorsque T_1, T_2 et $T_1 \cup T_2$ sont des représentations syntaxiques, alors T_1 (ou respectivement T_2) a la même racine que $T_1 \cup T_2$, et T_2 (respectivement T_1) a la même racine que $T_1 \cap T_2$.

Définition 5. On définit le complémentaire d'une représentation dans une autre par :

$$T_1 \setminus T_2 = connexes\left((N(T_1) \cup N(T_2)) \setminus N(T_2), (L(T_1) \cup L(T_2)) \setminus L(T_2)\right)$$

L'opérateur $connexes(N, T)$ désigne l'ensemble $\{(N_1, L_1), \ldots, (N_k, L_k)\}$ des sous-graphes connexes maximaux de (N, L).

Remarque 5. Si T_1 et T_2 sont des représentations d'arbres, alors les éléments de $T_1 \setminus T_2$ sont aussi des représentations d'arbres.

6.3.5 Décompositions

Définition 6. $\langle t_1, \ldots, t_k \rangle$ est une **décomposition** d'une représentation syntaxique T ssi :

- t_1,...t_k sont des représentations syntaxiques,
- $i \neq j \Rightarrow N(t_i) \cap N(t_j) = \emptyset$
- $N(t_1) \cup \cdots \cup N(t_k) = N(T)$
- $L(t_1) \cup \cdots \cup L(t_k) \subset L(T)$

La notation $\langle t_1, \ldots, t_k \rangle$ rappelle que l'on a affaire à une décomposition. Cependant, il ne s'agit que d'un ensemble de représentations, qui pourrait donc être noté : $\{t_1; \ldots; t_k\}$.

Exemple :

$$\left\langle \begin{array}{c} A, x_1 \\ | \\ 0, x_2 \end{array} \;,\; \begin{array}{ccc} & S, x_0 & \\ \swarrow & & \searrow \\ A, x_1 & & B, x_3 \\ & & | \\ & & 1, x_4 \end{array} \right\rangle \text{ est une décomposition de } \begin{array}{ccc} & S, x_0 & \\ \swarrow & & \searrow \\ A, x_1 & & B, x_3 \\ | & & | \\ 0, x_2 & & 1, x_4 \end{array} .$$

Notations 1. L'ensemble des décompositions d'une représentation t sera noté $\boldsymbol{\Delta(t)}$.

Notations 2. L'ensemble des décompositions $\langle t_1, \ldots, t_k \rangle$ d'une représentation t dont les éléments sont des arbres élémentaires d'une TSG $\mathcal{G}$ sera noté $\boldsymbol{\Delta_{\mathcal{G}}(t)}$:

$$\Delta_{\mathcal{G}}(t) = \{\langle t_1, \ldots, t_k \rangle \in \Delta(t) / t_1, \ldots, t_k \in \mathcal{G}\}$$

Notations 3. Pour une PTSG $\mathcal{G}$, l'ensemble des représentations syntaxiques admettant au moins une décomposition par $\mathcal{G}$ (i.e. l'ensemble des représentations t / $\Delta_{\mathcal{G}}(t) \neq \emptyset$) sera noté $\boldsymbol{\mathcal{C}(\mathcal{G})}$.

Définition 7. Soient $\delta = \langle t_1, \ldots, t_n \rangle$ et $\delta' = \langle t'_1, \ldots, t'_n \rangle$ deux décompositions de t (i.e. $\delta, \delta' \in \Delta(t)$).

δ' est dite **plus fine** que δ si chacun de ses éléments t'_i est inclus dans un élément t_j de δ ; on note alors $\delta' \leq \delta$:

$$\forall (\delta, \delta') \in \Delta(t)^2,$$
$$(\delta' \leq \delta) \Leftrightarrow \forall t'_i \in \delta', \exists t_j \in \delta / t'_i \subset t_j$$

La relation « **strictement plus fine** » est définie par :

$$\forall (\delta, \delta') \in \Delta(t)^2,$$
$$(\delta' < \delta) \Leftrightarrow (\delta' \leq \delta) \text{ et } (\delta' \neq \delta)$$

Comme la relation de finesse est définie à partir d'inclusion d'ensembles, on peut utiliser le fait que c'est une relation d'ordre partielle. On dira que deux décompositions sont **comparables** si l'une est plus fine que l'autre.

Définition 8. Une décomposition δ dans $\Delta_{\mathcal{G}}(t)$ est dite **maximale** si toute décomposition de t dans $\Delta_{\mathcal{G}}(t)$ comparable à δ est plus fine que δ :

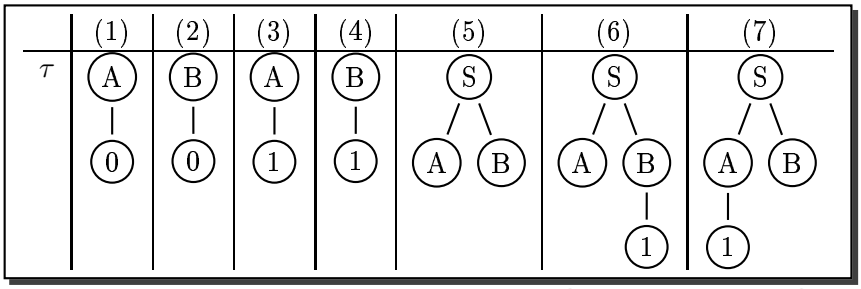
$$\forall \delta \in \Delta_{\mathcal{G}}(t), \big(\delta \text{ maximale } \Leftrightarrow \forall \delta' \in \Delta_{\mathcal{G}}(t) (\delta' \text{ comparable à } \delta \Rightarrow \delta' \leq \delta)\big)$$

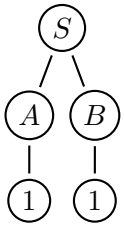
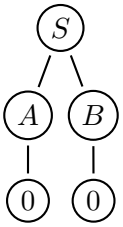
Définition 9. Pour tout arbre $t \in \mathcal{C}(\mathcal{G})$, on appellera **sous-arbre initial** de t un sous-arbre de t dont la racine est la racine de t.

Définition 10. Un sous-arbre initial t' de t sera dit **maximal dans** $\mathcal{G}$ si

1. $t' \in \mathcal{G}$
2. tout sous-arbre initial de t appartenant à $\mathcal{G}$ et contenant t' est égal à t' (un sous-arbre initial maximal dans $\mathcal{G}$ n'est pas nécessairement unique).

Exemple : Supposons une grammaire contenant les arbes élémentaires suivants :



Soient t_1 et t_2 des arbres de $\mathcal{C}(\mathcal{G})$: $t_1 =$  , $t_2 =$ 

Alors :
- les arbres élémentaires (6) et (7) sont les deux sous-arbres initiaux de t_1 maximaux dans $\mathcal{G}$;
- t_2 possède un seul sous-arbre initial maximal dans $\mathcal{G}$: (5).

Définition 11. Un sous-arbre initial τ commun à deux arbres τ_1 et τ_2 sera dit **maximal pour** τ_1 **et** τ_2 si tout sous-arbre initial commun à τ_1 et τ_2 est aussi sous-arbre initial de τ.

Un sous-arbre initial commun maximal est nécessairement unique.

On considère dans cette dernière définition les classes d'arbres. Pour revenir à des représentations, on peut dire : soient $\{(t_1, t_2)_i\}$ l'ensemble des couples de représentants de τ_1 et τ_2 ayant la même racine, et $\{(t_1 \cup_{init} t_2)_i\}$ chacune des composantes connexes maximales de leurs intersections contenant leur racine commune. Le sous-arbre initial commun à τ_1 et τ_2 est la classe des plus grandes de ces composantes.

6.4 Grammaires à substitution d'arbres polynomiales

Définition 12.

Une PTSG $\mathcal{G}$ est dite polynomiale si

$$\forall t \in \mathcal{C}(\mathcal{G}),\ \exists \langle t_1,\ \ldots, t_k \rangle \in \Delta_{\mathcal{G}}(t) / P_{DOP}(t) = \prod_i P_{DOP}(t_i)$$

Rappel : $P_{DOP}(t)$ désigne la DOP-pseudo-probabilité de l'arbre t, égale à la somme des pseudo-probabilités des décompositions de t. La pseudo-probabilité d'une décomposition est le produit des pseudo-probabilités de

ses éléments. La pseudo-probabilité d'un arbre élémentaire est une valeur réelle positive ou nulle. Dans le cadre des GTSG, les arbres élémentaires sont associés à des potentiels λ, et leur pseudo-probabilité vaut e^{λ}. Pour une STSG (par exemple pour le modèle DOP initial), la pseudo-probabilité est simplement la probabilité de l'arbre élémentaire[4]. Par abus de notation, on note dans la suite les pseudo-probabilités par la lettre p, et les DOP-pseudo-probabilités par P_{DOP}.

Théorème 1.

$\forall t \in \mathcal{C}(\mathcal{G}),\ \forall \langle t_1,\ \ldots, t_k\rangle \in \Delta_{\mathcal{G}}(t)$, *on a :*
(1) $P_{DOP}(t) = \prod_i P_{DOP}(t_i)$
ssi
(2) $\langle t_1,\ \ldots, t_k\rangle$ *est l'unique décomposition maximale de* t

Démonstration du théorème 1 :

Condition suffisante : (2)⇒(1)

Si $\delta = \langle t_1,\ \ldots, t_k\rangle \in \Delta_{\mathcal{G}}(t)$ est maximale et unique, on a :

$$\{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta\} = \Delta_{\mathcal{G}}(t)$$

En effet, soit $\delta' \in \Delta_{\mathcal{G}}(t)$, et $\delta \ >>\in \Delta_{\mathcal{G}}(t)\ /\ \delta' \leq \delta \ >>$ et $\delta \ >>$ maximal. Comme δ est l'unique décomposition maximale de t, $\delta \ >>= \delta$, et donc $\delta' \leq \delta$.

Du fait de l'égalité précédente,

$$\sum_{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta} p(\delta') = \sum_{\delta' \in \Delta_{\mathcal{G}}(t)} p(\delta') \overset{\text{Déf.}}{=} P_{DOP}(t)$$

Or d'après le lemme 1 (voir plus bas),

$$\sum_{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta} p(\delta') = \prod_i P_{DOP}(t_i),$$

donc $P_{DOP}(t) = \prod_i P_{DOP}(t_i)$.

Condition nécessaire : (1)⇒(2)

Démontrons la propriété par l'absurde. Si $\delta = \langle t_1,\ \ldots, t_k\rangle \in \Delta_{\mathcal{G}}(t)$ n'est pas maximale ou est maximale et pas unique, alors :

$$E = \Delta_{\mathcal{G}}(t) \setminus \{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta\} \neq \emptyset$$

[4] d'où le choix du terme de « pseudo-probabilité »

et donc

$$\begin{aligned} P_{DOP}(t) &= \sum_{\delta' \in \Delta_{\mathcal{G}}(t)} p(\delta') = \sum_{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta} p(\delta') + \sum_{\delta' \in E} p(\delta') \\ &> \sum_{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta} p(\delta') = \prod_i P_{DOP}(t_i) \end{aligned}$$

d'où $P_{DOP}(t) \neq \prod_i P_{DOP}(t_i)$. □

Lemme 1.

Si $\delta = \langle t_1, \ldots, t_k \rangle$ est une décomposition de t, alors

$$\sum_{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta} p(\delta') = \prod_i P_{DOP}(t_i)$$

Démonstration du lemme 1.
Par définition, une décomposition $\delta' = \langle t'_1, \ldots, t'_{k'} \rangle$ de t est plus fine que δ si et seulement si :

$$\forall t'_i \in \delta', \exists t_j \in \delta / t'_i \subset t_j$$

De plus, si $t'_i \subset t_{j0}$, du fait que les sous-arbres de δ sont disjoints deux-à-deux par la définition 6, t'_i est disjoint des arbres de δ autres que t_{j0}, et n'est donc inclus que dans t_{j0}. Ainsi, la définition précédente est équivalente à :

Une décomposition $\delta' = \langle t'_1, \ldots, t'_{k'} \rangle$ de t est plus fine que δ si et seulement si :

$$\forall t'_i \in \delta', \exists ! t_j \in \delta / t'_i \subset t_j$$

On peut alors, pour une décomposition δ' plus fine que δ, définir l'application $map_{\delta'\delta}$ de δ' dans δ :

$$\forall (i,j) \in [\![1, k']\!] \otimes [\![1, k]\!], map_{\delta'\delta}(t'_i) = t_j \Leftrightarrow t'_i \subset t_j$$

.

Montrons qu'alors : $\forall j_0 \in [\![1, k]\!],\ map^{-1}_{\delta'\delta}(t_{j0}) \in \Delta_{\mathcal{G}}(t_{j0})$.

Soit n un nœud de t_{j0}, et t'_i l'arbre de δ' contenant n (t'_i existe et est unique car (1) les nœuds de t sont la réunion des nœuds des arbres de δ', et (2) les arbres de δ' sont deux-à-deux disjoints ; tout ceci par la définition 6). Comme $\delta' \leq \delta$, $\exists ! t_j \in \delta / t'_i \subset t_j$. Et comme les sous-arbres de δ sont deux-à-deux disjoints, t'_i est disjoint des sous-arbres de δ dans lesquels il

n'est pas inclus. Comme il n'est pas disjoint de t_{j0}, il lui est inclus. Donc $t'_i \in map^{-1}_{\delta'\delta}(t_{j0})$. Or $n \in N(t'_i)$, donc : $n \in \cup_{t'_i \in map^{-1}_{\delta'\delta}(t_{j0})} N(t'_i)$.

Inversement, supposons n un nœud de $\cup_{t'_i \in map^{-1}_{\delta'\delta}(t_{j0})} N(t'_i)$. Il existe donc un sous-arbre t'_i de $map^{-1}_{\delta'\delta}(t_{j0})$ contenant n. Or $t'_i \subset t_{j0}$, par définition de la fonction $map_{\delta'\delta}$, donc $n \in N(t_{j0})$.

Ainsi, un nœud n est dans t_{j0} si et seulement si il est dans un arbre de $map^{-1}_{\delta'\delta}(t_{j0})$, i.e. :

$$N(t_{j0}) = \cup_{t'_i \in map^{-1}_{\delta'\delta}(t_{j0})} N(t'_i)$$

.

De plus, un lien de $\cup_{t'_i \in map^{-1}_{\delta'\delta}(t_{j0})} L(t'_i)$ appartient forcément à l'un des t'_i de $map^{-1}_{\delta'\delta}(t_{j0})$, et est donc inclus dans t_{j0} (toujours par définition de $map_{\delta'\delta}$). Donc :

$$\cup_{t'_i \in map^{-1}_{\delta'\delta}(t_{j0})} L(t'_i) \subset L(t_{j0})$$

$map^{-1}_{\delta'\delta}(t_{j0})$ vérifie donc toutes les conditions de la définition 6, et est une décomposition de t_{j0}.

Ainsi est bien définie l'application :

$$\begin{array}{rcl} \mathcal{M} \ : \ \{\delta' \in \Delta_{\mathcal{G}}(t) | \delta' \leq \delta\} & \longrightarrow & \Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k) \\ \delta' & \longmapsto & (map^{-1}_{\delta'\delta}(t_1), \ldots, map^{-1}_{\delta'\delta}(t_k)) \end{array}$$

Pour s'en convaincre, il faut noter l'unicité de la fonction $map_{\delta'\delta}$, δ et δ' étant connus.

Montrons que $\mathcal{M}$ est bijective.

Soit $(\delta_1, \ldots, \delta_k) \in \Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k)$. Montrons que $(\delta_1, \ldots, \delta_k)$ est l'image par $\mathcal{M}$ de $\delta_1 \cup \cdots \cup \delta_k$. On remarque d'abord que $\delta_1 \cup \cdots \cup \delta_k$ **est une décomposition de** t, car :

$$N(t) = \cup_{t_j \in \delta} N(t_j) = \cup_{t_j \in \delta} \cup_{t'_i \in \delta_j} N(t'_i) = \cup_{t'_i \in \delta_1 \cup \cdots \cup \delta_k} N(t'_i)$$
$$\cup_{t'_i \in \delta_1 \cup \cdots \cup \delta_k} L(t'_i) = \cup_{t_j \in \delta} \cup_{t'_i \in \delta_j} L(t'_i) \subset \cup_{t_j \in \delta} L(t_j) \subset L(t)$$

Soit maintenant t'_i un élément de $\delta_1 \cup \cdots \cup \delta_k$. Il existe donc au moins un δ_j tel que $t'_i \in \delta_j$. Or δ_j est une décomposition de t_j, donc $N(t'_i) \subset N(t_j)$ et $L(t'_i) \subset L(t_j)$, donc $t'_i \subset t_j$. Ainsi :

$$\forall t'_i \in \delta_1 \cup \cdots \cup \delta_k, \exists t_j \in \delta / t'_i \subset t_j$$

ce qui prouve que $\delta_1 \cup \cdots \cup \delta_k$ **est une décomposition de** t **plus fine que** δ . Nommons-la δ'. On va maintenant montrer qu'avec cette définition de δ', on a pour tout j : $map^{-1}_{\delta'\delta}(t_j) = \delta_j$.

L'application $map_{\delta'\delta}$ est bien définie, car δ' est plus fine que δ.

Soit t_j un élément de δ. $\forall t'_i \in \delta_j,\ t'_i \in \delta'$ et $map_{\delta'\delta}(t'_i) = t_j$ Donc $\delta_j \subset map^{-1}_{\delta'\delta}(t_j)$.

Inversement, soit t'_i un élément de $map^{-1}_{\delta'\delta}(t_j)$. t'_i appartient forcément à un δ_k, puisqu'il est dans leur union δ'. Or s'il appartient à un δ_k, il est inclus dans le t_k correspondant, qui ne peut être alors que t_j puisque $map_{\delta'\delta}(t'_i) = t_j$. Donc $t'_i \in \delta_j$. Donc : $map^{-1}_{\delta'\delta}(t_j) \subset \delta_j$. Il vient avec les deux inclusions précédentes : $map^{-1}_{\delta'\delta}(t_j) = \delta_j$

On vient de prouver :

$$\forall(\delta_1, \ldots, \delta_k) \in \Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k),$$
$$\exists \delta' \in \Delta_{\mathcal{G}}(t) / \delta' \leq \delta \ et\ \mathcal{M}(\delta') = (\delta_1, \ldots, \delta_k)$$

Or

$$\forall \delta' \in \Delta_{\mathcal{G}}(t)\ / \delta' \leq \delta,\ \ \cup_{t_j \in \delta} map^{-1}_{\delta'\delta}(t_j) = \delta'$$

(du fait que chaque t'_i de δ' a une image par $map_{\delta'\delta}$.) Donc si deux décompositions δ' et δ $>>$ plus fines que δ ont la même image par $\mathcal{M}$, cela entraîne :

$$(map^{-1}_{\delta'\delta}(t_1), \ldots, map^{-1}_{\delta'\delta}(t_k)) = (map^{-1}_{\delta\ >>\delta}(t_1), \ldots, map^{-1}_{\delta\ >>\delta}(t_k))$$

$$\Rightarrow \cup_{t_j \in \delta} map^{-1}_{\delta'\delta}(t_j) = \cup_{t_j \in \delta} map^{-1}_{\delta\ >>\delta}(t_j)$$

$$\Rightarrow \delta' = \delta\ >>$$

Donc tout élément de $\Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k)$ a **un antécédent unique** par $\mathcal{M}$ dans $\{\delta' \in \Delta_{\mathcal{G}}(t) / \delta' \leq \delta\}$. $\mathcal{M}$ **est une bijection** de $\{\delta' \in \Delta_{\mathcal{G}}(t)$ telle que $\delta' \leq \delta\}$ dans $\Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k)$.

On a alors :

$$\prod_{j=1}^{k} P_{DOP}(t_j) = \prod_{j=1}^{k} \sum_{\delta_j \in \Delta_{\mathcal{G}}(t_j)} p(\delta_j) = \sum_{(\delta_1, \ldots, \delta_k) \in \Delta_{\mathcal{G}}(t_1) \otimes \cdots \otimes \Delta_{\mathcal{G}}(t_k)} \prod_{j=1}^{k} p(\delta_j)$$

Or $\mathcal{M}$ est bijective, donc :

$$\prod_{j=1}^{k} P_{DOP}(t_j) = \sum_{\{\delta' \in \Delta_{\mathcal{G}}(t) / \delta' \leq \delta\}} \prod_{j=1}^{k} p(map^{-1}_{\delta'\delta}(t_j))$$

$$\prod_{j=1}^{k} P_{DOP}(t_j) = \sum_{\{\delta' \in \Delta_{\mathcal{G}}(t)/\delta' \leq \delta\}} \prod_{j=1}^{k} \prod_{t'_i \in map^{-1}_{\delta'\delta}(t_j)} p(t'_i)$$

Or les ensembles $map^{-1}_{\delta'\delta}(t_j)$ sont deux-à-deux disjoints quand j varie, et leur réunion vaut δ', donc pour tout δ' plus fin que δ :

$$\prod_{j=1}^{k} \prod_{t'_i \in map^{-1}_{\delta'\delta}(t_j)} p(t'_i) = \prod_{t'_i \in \delta'} p(t'_i) = p(\delta')$$

D'où, avec l'équation précédente :

$$\prod_{j=1}^{k} P_{DOP}(t_j) = \sum_{\{\delta' \in \Delta_{\mathcal{G}}(t)/\delta' \leq \delta\}} p(\delta')$$

... ce qui termine la démonstration du lemme 1. □

A partir du théorème 1, nous pouvons dériver une première condition nécessaire et suffisante pour qu'une PTSG soit polynomiale :

Théorème 2.

Une PTSG $\mathcal{G}$ est polynomiale ssi $\forall t \in \mathcal{C}(\mathcal{G})$, t admet une unique décomposition maximale.

Cette condition nécessaire et suffisante n'est cependant pas très utilisable en pratique. En particulier, du fait de sa forme très générale (elle implique l'ensemble des éléments de $\mathcal{C}(\mathcal{G})$), elle ne se prête pas de façon évidente à servir comme test pour vérifier si une PTSG donnée est polynomiale.

6.5 PTSG denses

Afin d'obtenir une condition nécessaire et suffisante de polynomialité plus opérationnelle, on s'intéresse maintenant à une sous-famille des PTSG, que l'on a choisi de nommer PTSG « dense ».

Définition 13.

Une PTSG $\mathcal{G}$ sera dite **dense** si elle vérifie la propriété suivante :
$\forall t_1, t_2 \in \mathcal{G}/(t_1 \cup t_2)$ est un arbre, $(t_1 \setminus t_2) \subset \mathcal{C}(\mathcal{G})$

Remarque 6. Notons que toute PTSG contenant parmi ses arbres élémentaires tous les sous-arbres de ces arbres élémentaires constitue un exemple simple d'une PTSG dense (qui sera dite maximalement dense, voir la définition 14).

Corollaire.

(1) Une PTSG est dense
ssi
(2) $\forall t_1, t_2 \in \mathcal{G}/(t_1 \cup t_2)$ *est un arbre,* $(t_1 \cup t_2)$ *admet une décomposition dans* $\Delta_\mathcal{G}$*, dont* t_1 *fait partie.*

Démonstration du corollaire.

Condition nécessaire : (1) $\Rightarrow$ (2)

Les éléments $\{t'_1, \ldots, t'_n\}$ de $(t_2 \setminus t_1)$ sont par définition les parties connexes maximales de $(N(t_1) \cup N(t_2)) \setminus N(t_1), L(t_1) \cup L(t_2)) \setminus L(t_1)$.

Par hypothèse, $\mathcal{G}$ est dense, donc $t'_1, \ldots, t'_n$ appartiennent à $\mathcal{C}(\mathcal{G})$, et admettent des décompositions $\delta_1, \ldots, \delta_n$ dans $\mathcal{G}$. Les relations d'inclusion sur les nœuds et les liens qui en découlent entraînent :

$$N(t_1 \cup t_2) \quad = N(t_1) \cup \bigcup_{1 \leq j \leq n} N(t'_j) \quad = N(t_1) \cup \bigcup_{1 \leq j \leq n} \bigcup_{t^i_j \in \delta_j} N(t^i_j)$$

$$L(t_1 \cup t_2) \quad \supset L(t_1) \cup \bigcup_{1 \leq j \leq n} L(t'_j) \quad \supset L(t_1) \cup \bigcup_{1 \leq j \leq n} \bigcup_{t^i_j \in \delta_j} L(t^i_j)$$

Enfin t_1 est disjoint de chaque t^i_j, et ceux-ci sont disjoints entre eux, étant des sous-ensembles disjoints de sous-ensembles disjoints t'_j. Donc, par définition, $\langle t_1, t^1_1, \ldots, t^{|\delta_1|}_1, \ldots, t^1_n, \ldots, t^{|\delta_n|}_n \rangle$ est une décomposition de $t_1 \cup t_2$. Les éléments de cette décomposition appartiennent bien à $\mathcal{G}$.

Condition suffisante : : (2) $\Rightarrow$ (1)

Soit $\delta = \langle t_1, t'_1, \ldots, t'_n \rangle$ une décomposition dans $\Delta_\mathcal{G}$ de $t_1 \cup t_2$, et $t >>$ un élément de $(t_2 \setminus t_1)$. Montrons que $t >> \in \mathcal{C}(\mathcal{G})$.

Soit $\mathcal{I}_{t >>}(\delta)$ l'ensemble des éléments de δ ayant une intersection non-vide avec $t >>$.

- **Les éléments de $\mathcal{I}_{t>>}(\delta)$ sont disjoints deux-à-deux** (car ils sont dans δ).
- **La réunion de leurs nœuds contient** $N(t >>)$, parce que $t >>$ est inclus dans $t_1 \cup t_2$, que chaque nœud de $t_1 \cup t_2$ est couvert par un élément de δ, et qu'on garde dans $\mathcal{I}_{t>>}(\delta)$ tous les éléments de δ ayant une intersection (en particulier au niveau des nœuds) avec $t >>$.
- **La réunion de leurs nœuds ne contient que** $N(t >>)$. En effet, s'il y avait un nœud n'appartenant pas à $N(t >>)$ dans un arbre t'_i de $\mathcal{I}_{t>>}(\delta)$. Alors :
  - $t >> \cup t'_i$ serait une composante connexe de $t_1 \cup t_2$, car t'_i aurait une intersection avec $t >>$ par définition de $\mathcal{I}_{t>>}(\delta)$;
  - on aurait $N(t >> \cup t'_i) \in N(t_2 \cup t_1) \setminus N(t_1)$ et $L(t >> \cup t'_i) \in L(t_2 \cup t_1) \setminus L(t_1)$ car ni $t >>$ ni t'_i n'auraient d'intersection avec t_1 ;
  - $t >> \cup t'_i$ contiendrait strictement $t >>$, car t'_i possèderaient un nœud n'appartenant pas à $t >>$.

  En résumé, $t >> \cup t'_i$ serait une composante connexe appartenant au complémentaire de t_1 dans $t_1 \cup t_2$, qui contiendrait strictement $t >>$. $t >>$ ne pourrait alors pas être une composante connexe maximale, ce qui nierait la proposition initiale : $t >> \in (t_2 \setminus t_1)$..
- **La réunion de leurs liens est incluse dans** $L(t >>)$. En effet, les liens des éléments de $\mathcal{I}_{t>>}(\delta)$ sont des liens de $t_1 \cup t_2$ entre des nœuds de $t >>$ (d'après le point précédent). Or, comme $t >>$ est un ensemble connexe maximal, il contient **tous** les liens de $t_1 \cup t_2$ qui sont entre des nœuds de $t >>$.

Ces conditions suffisent à conclure que $\mathcal{I}_{t>>}(\delta)$ est une décomposition de $t >>$. Comme ses éléments sont dans $\mathcal{G}$, $t >>$ est dans $\mathcal{C}(\mathcal{G})$. Comme c'est vrai pour tout $t >> \in (t_2 \setminus t_1)$, $\mathcal{G}$ est dense. □

A l'aide du lemme 2 qui sera démontré ci-après, nous allons maintenant établir le théorème central suivant :

Théorème 3.

Pour toute PTSG $\mathcal{G}$ dense, on a :
(1) $\forall t \in \mathcal{C}(\mathcal{G})$, t admet une décomposition maximale unique
ssi
(2) $\forall t_1, t_2 \in \mathcal{G}$ / $t_1 \cup t_2$ est un arbre, $(t_1 \cup t_2) \in \mathcal{G}$.

Démonstration du théorème 3 :

Condition nécessaire : (1)⇒(2)

Soient $t_1, t_2 \in \mathcal{G}$ /$(t_1 \cup t_2)$ est un arbre.

Du fait de la densité de $\mathcal{G}$, $t_1 \cup t_2 \in \mathcal{C}(\mathcal{G})$ et donc, d'après (1), $(t_1 \cup t_2)$ admet une décomposition maximale unique.

Le lemme 2 ci-après montre qu'alors $(t_1 \cup t_2) \in \mathcal{G}$ et (2) est donc bien vérifiée.

Condition suffisante : (2)⇒(1)

Soient δ_1 et δ_2 deux décompositions maximales de t. Montrons que si (2) est vérifiée, alors δ_1 et δ_2 sont confondues.

Notons : $\delta_1 = \{t_1^1, \ldots t_{n_1}^1\}$ et $\delta_2 = \{t_1^2, \ldots t_{n_2}^2\}$.

Soit $\mathcal{I}_{t_i^1}(\delta_2) = \{t_j^2 \in \delta_2 | t_j^2 \cap t_i^1 \neq \emptyset\}$, l'ensemble des éléments de δ_2 ayant une intersection non-vide avec t_i^1. Notons ses éléments $\tau_1^i, \ldots, \tau_{|\mathcal{I}_{t_i^1}(\delta_2)|}^i$.

Montrons par récurrence sur k la relation :

$$t_i^1 \cup \bigcup_{1 \leq j \leq k} \tau_j^i \in \mathcal{G}, \text{ pour tout } 1 \leq k \leq |\mathcal{I}_{t_i^1}(\delta_2)|$$

- La propriété est vraie pour $k = 0$, car $t_i^1 \in \mathcal{G}$.
- Supposons-la vraie pour $k-1$. Cela veut dire que :

$$t_i^1 \cup \bigcup_{1 \leq j \leq k-1} \tau_j^i \in \mathcal{G}$$

  Or

$$t_i^1 \cup \bigcup_{1 \leq j \leq k} \tau_j^i = (t_i^1 \cup \bigcup_{1 \leq j \leq k-1} \tau_j^i) \cup \tau_k^i$$

  τ_k^i a une intersection avec $(t_i^1 \cup \bigcup_{1 \leq j \leq k-1} \tau_j^i)$, car il a une intersection avec t_i^1. Comme ce sont tous les deux des sous-arbres de t, d'intersection non-vide, leur réunion est un sous-arbre de t. Comme ils appartiennent tous deux à $\mathcal{G}$, leur réunion appartient à $\mathcal{G}$ (d'après l'hypothèse initiale (2)). La propriété est donc vraie à l'ordre k.
- La propriété est donc vraie pour $k = |\mathcal{I}_{t_i^1}(\delta_2)|$.

Nommons U la réunion $\left(t_i^1 \cup \bigcup_{1 \leq j \leq |\mathcal{I}_{t_i^1}(\delta_2)|} \tau_j^i\right)$.

U appartient à $\mathcal{G}$ d'après ce qui précède. Les arbres de $\mathcal{I}_{t_i^1}(\delta_2)$ sont évidemment inclus dans U. Quant aux arbres de δ_2 qui ne font pas partie de $\mathcal{I}_{t_i^1}(\delta_2)$, ils n'ont aucune intersection avec t_i^1 (car ils ne sont pas dans $\mathcal{I}_{t_i^1}(\delta_2)$), ni avec les arbres de $\mathcal{I}_{t_i^1}(\delta_2)$ (car ceux-ci sont des arbres de δ_2, et par définition d'une décomposition). Ils n'ont donc aucune intersection avec U. On vérifie alors facilement que $(\delta_2 \setminus \mathcal{I}_{t_i^1}(\delta_2)) \cup \{U\}$ **est une décomposition de** t.

δ_2 est strictement plus fine que $(\delta_2 \setminus \mathcal{I}_{t_i^1}(\delta_2)) \cup \{U\}$ ssi $\mathcal{I}_{t_i^1}(\delta_2)$ contient plus d'un élément. Or δ_2 est maximale, donc $\mathcal{I}_{t_i^1}(\delta_2)$ possède au plus un

élément. Il ne peut être vide car les nœuds de t_i^1 se retrouvent forcément dans δ_2. Donc $\mathcal{I}_{t_i^1}(\delta_2)$ possède un et un seul élément τ^i. Les nœuds de t_i^1 font partie des nœuds de τ^i. Les liens de t_i^1 étant tous les liens de t situés entre des nœuds de t_i^1, les liens de τ^i étant tous les liens de t situés entre des nœuds de τ^i, et les nœuds de t_i^1 faisant partie des nœuds de τ^i, les liens de t_i^1 font partie des liens de τ^i. Donc :$t_i^1 \subset \tau^i$.

On a montré : $\forall t_i^1 \in \delta_1, \exists! \tau^i \in \delta_2 | t_i^1 \subset \tau^i$. Cela prouve que : $\delta_1 \leq \delta_2$. Comme δ_1 est maximale, $\delta_1 = \delta_2$. □

Lemme 2.

Pour toute PTSG $\mathcal{G}$ dense, et tout couple t_1, t_2 de $\mathcal{G}$ tel que $t_1 \cup t_2$ est un arbre, on a :
(1) $t_1 \cup t_2$ admet une décomposition maximale unique
ssi
(2) $t_1 \cup t_2 \in \mathcal{G}$.

Démonstration du lemme 2.

Condition nécessaire : (1)⇒(2)

Soit $t_1, t_2 \in \mathcal{G} / t_1 \cup t_2$ est un arbre. Comme $\mathcal{G}$ est dense, $t_1 \cup t_2$ admet une décomposition δ_1 dans $\mathcal{G}$ dont t_1 fait partie, et une décomposition δ_2 dans $\mathcal{G}$ dont t_2 fait partie. Comme $t_1 \cup t_2$ admet une décomposition maximale unique δ, δ_1 et δ_2 sont plus fines que δ. Cela implique :

$$\exists (t_1', t_2') \in \delta^2 | t_1 \subset t_1',\ t_2 \subset t_2'$$

Comme t_1' et t_2' font partie de δ, ils sont soit disjoints, soit identiques. Ils ne peuvent être disjoints car t_1 et t_2 ont une intersection non-vide (car leur union est un arbre) et sont respectivement inclus dans t_1' et t_2'. Donc t_1' et t_2' sont identiques, et : $t_1 \subset t_1',\ t_2 \subset t_1' \Rightarrow t_1 \cup t_2 \subset t_1'$. Or t_1' est un élément de δ, donc un sous-ensemble de $t_1 \cup t_2$. Donc $t_1' = t_1 \cup t_2$. Comme $t_1' \in \mathcal{G}$, on a finalement : $t_1 \cup t_2 \in \mathcal{G}$.

Condition suffisante : (2)⇒(1)

Si (2) est vérifiée, alors $t_1 \cup t_2 \in \mathcal{G}$ et donc $t_1 \cup t_2$ admet $\langle t_1 \cup t_2 \rangle$ comme décomposition dans $\mathcal{G}$. Toutes ses autres décompositions étant plus fines que $\langle t_1 \cup t_2 \rangle$, $\langle t_1 \cup t_2 \rangle$ est la décomposition maximale unique de $t_1 \cup t_2$. □

En combinant les théorèmes 2 et 3, on a donc la condition nécessaire et suffisante suivante pour caractériser les grammaires denses polynomiales :

Théorème 4.

Pour toute PTSG dense $\mathcal{G}$, on a :
(1) $\mathcal{G}$ est polynomiale
ssi
(2) $\forall t_1, t_2 \in \mathcal{G} / t_1 \cup t_2$ est un arbre, $t_1 \cup t_2 \in \mathcal{G}$.

6.6 Vers une condition suffisante de polynomialité

Pour une famille plus restreinte de PTSG, les PTSG *maximalement denses*, on peut de plus démontrer une condition suffisante additionnelle de polynomialité (voir théorème 7 ci-dessous) qui sera utile pour l'extraction de PTSG polynomiales à partir de corpus (voir le théorème 8 ci-dessous).

6.6.1 PTSG maximalement denses

Notations 4. Pour toute PTSG $\mathcal{G}$, on notera $\mathcal{G}_1$ l'ensemble des éléments (arbres) de $\mathcal{G}$ de profondeur 1 (donc réduits à un nœud), et $\Omega_1(\mathcal{G})$ l'ensemble des sous-arbres de profondeur 1 des éléments de $\mathcal{G}$ (i.e. l'ensemble des nœuds de $\mathcal{G}$, considérés comme des sous-arbres).

Définition 14. Toute PTSG $\mathcal{G}$ vérifiant la propriété $\Omega_1(\mathcal{G}) = \mathcal{G}_1$ sera dite **maximalement dense**.

Lemme 3.

Toute PTSG maximalement dense est dense.

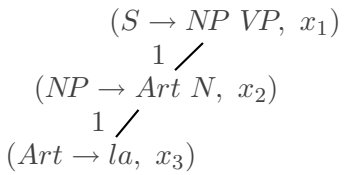
6.6.2 Expansion d'un sous-arbre

Définition 15.

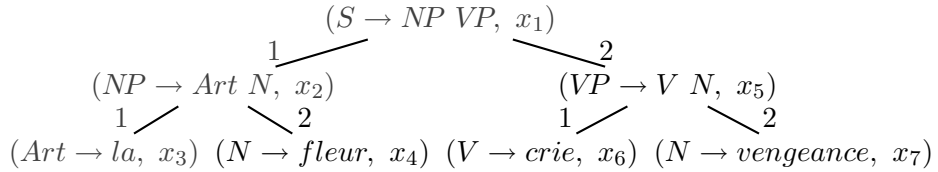
Soit t' un sous-arbre d'un arbre t. On appelle **expansion** de t' dans t l'ensemble des couples (np, i) tels que : $\exists (np, i, nf) \in L(t)$ et $\nexists\ (np, i, nf) \in L(t')$. On appelle « expansion canonique » de t' dans t l'expansion de t' dans t lorsque les indices de t sont choisis pour que t' ait sa forme canonique. On passe d'une expansion de t' à son expansion canonique en appliquant aux nœuds de l'expansion la même transformation que celle qui permet de passer de la représentation de t' à sa représentation canonique.

Remarque 7. Intuitivement, l'expansion d'un sous-arbre t' de t représente l'ensemble des liens de t « accrochés » à des feuilles de t'.

Exemple : L'expansion canonique de



dans



est $\{((NP \rightarrow Art\ N, 2), 2)\,; ((S \rightarrow NP\ VP, 1), 2)\}$.

6.6.3 Condition suffisante de polynomialité

Théorème 5.

Pour toute PTSG $\mathcal{G}$ maximalement dense, on a :
Si
(1) l'expansion de tout élément de $\mathcal{G}_1$ est constante sur $\mathcal{G} \setminus \mathcal{G}_1$
alors
(2) $\forall t_1, t_2 \in \mathcal{G} / t_1 \cup t_2$ est un arbre, $t_1 \cup t_2 \in \mathcal{G}$.

Démonstration. Soient $t_1, t_2 \in \mathcal{G} / t_1 \cup t_2$ est un arbre. Notons : $I = t_1 \cap t_2$ et $t = t_1 \cup t_2$.

Quitte à inverser les indices, on considère que la racine de t est la racine de t_1, et que la racine de I est la racine de t_2.

Comme les sous-arbres de profondeur 1 de I sont dans $t_2 \in \mathcal{G}$, ils sont dans $\Omega_1(\mathcal{G})$. Comme $\mathcal{G}$ est maximalement dense, ils sont dans $\mathcal{G}_1$. Par hypothèse, ils ont donc une expansion constante dans tout arbre de $\mathcal{G} \setminus \mathcal{G}_1$.

1er cas : $t_1, t_2 \in \mathcal{G} \setminus \mathcal{G}_1$

On considère **l'expansion de** $I = t_1 \cap t_2$ **dans** t_2. Montrons par l'absurde qu'elle est vide.

Supposons donc qu'elle n'est pas vide. Cela implique qu'il existe un lien (np, i, nf) dans t_2, tel qu'aucun lien $(np, i, .)$ n'est présent dans I. Or $(\{np\}, \emptyset)$ est un sous-arbre de profondeur 1 de I, donc admet par hypothèse une expansion canonique constante dans tout arbre de $\mathcal{G} \setminus \mathcal{G}_1$, donc constante dans t_1 et t_2. Comme (np, i) est dans l'expansion dans t_2 de $(\{np\}, \emptyset)$, il est dans l'expansion dans t_1 de $(\{np\}, \emptyset)$. Donc il existe dans t_1 un lien (np, i, nf'). Si nf et nf' étaient différents, les liens (np, i, nf) et (np, i, nf') de $t_1 \cup t_2$ seraient distincts, et la réunion ne serait pas un arbre syntaxique. Or on se place dans le cas où la réunion est un arbre syntaxique. Donc $nf' = nf$. Mais alors, le lien (np, i, nf) apparaissant dans t_1 et t_2, il apparaît dans $I = t_1 \cap t_2$, ce qui nie la deuxième phrase de ce paragraphe. Donc l'expansion de I dans t_2 est vide.

On vient de démontrer qu'aucun nœud de $t_2 \setminus I$ n'apparaît comme fils d'un nœud de I. De plus, on a fixé t_2 de sorte qu'il ait même racine que I. Donc aucun nœud de $t_2 \setminus I$ n'apparaît comme père d'un nœud de I. Donc aucun nœud de $t_2 \setminus I$ n'est lié à un nœud de I. Or t_2 doit être un ensemble connexe de nœuds et $I \subset t_2$. Si $N(t_2 \setminus I)$ était non-vide, t_2 ne serait plus connexe car il n'y aurait pas de chemin entre un nœud de $t_2 \setminus I$ et un nœud de I. Donc $N(t_2 \setminus I)$ est vide. Si $L(t_2 \setminus I)$ était non-vide, cela voudrait dire qu'il existe un lien, dans t_2, qui n'existe pas dans $t_1 \cap t_2$, donc pas dans t_1, mais qui se trouve entre deux nœuds de $t_1 \cap t_2$ (d'après ce qui précède :$N(t_2 \setminus I) = \emptyset$), donc entre deux nœuds de t_1. On aurait donc deux nœuds, qui appartiendraient tous deux à t_1 et à t_2, et tels que le chemin qui les relieraient serait différent dans t_1 et dans t_2 (dans t_2, ils seraient reliés par un lien de $L(t_2 \setminus I)$, et dans t_1 par autre chose, car t_1 est connexe). On aurait donc deux chemins distincts qui les relierait dans $t_1 \cup t_2$, ce qui nierait la qualité d'arbre de $t_1 \cup t_2$.

On vient finalement de démontrer que $t_2 \setminus I$ est vide, d'où :

$$t_2 \setminus (t_1 \cap t_2) = \emptyset \Rightarrow t_2 = t_1 \cap t_2 \Rightarrow t_1 = t_1 \cup t_2$$

2ème cas : $t_1 \in \mathcal{G}_1,\ t_2 \in \mathcal{G} \setminus \mathcal{G}_1$

On peut alors écrire : $t_1 = (\{(g, x)\}, \emptyset)$. Comme $t_1 \cup t_2$ est un arbre, (g, x) y est lié aux autres nœuds (qui sont un ou plus car $t_2 \in \mathcal{G} \setminus \mathcal{G}_1$). Or $L(t_1)$ est vide, donc les liens concernant (g, x) sont dans $L(t_2)$, ce qui suppose que $(g, x) \in N(t_2)$. D'où : $t_1 \subset t_2 \Rightarrow t_1 \cup t_2 = t_2$.

3ème cas : $t_1 \in \mathcal{G} \setminus \mathcal{G}_1,\ t_2 \in \mathcal{G}_1$

La démonstration du cas précédent amène de la même façon : $t_2 \subset t_1 \Rightarrow t_1 \cup t_2 = t_1$.

4ème cas : $t_1, t_2 \in \mathcal{G}_1$

$L(t_1 \cup t_2) = L(t_1) \cup L(t_1) = \emptyset \cup \emptyset = \emptyset$. Pour que $t_1 \cup t_2$ soit un arbre, il faut que ce soit un graphe connexe ; ce graphe n'a pas d'arcs, donc il ne peut qu'être réduit à un sommet, d'où : $N(t_1 \cup t_2) = N(t_1) = N(t_2)$. Donc : $t_1 = t_2$. □

En conclusion : On a prouvé un théorème un peu plus fort en réalité que le théorème 5 :

Théorème 6.

Pour toute PTSG $\mathcal{G}$ maximalement dense, on a :
Si
(1) l'expansion de tout élément de $\mathcal{G}_1$ est constante sur $\mathcal{G} \setminus \mathcal{G}_1$
alors
(2) $\forall t_1, t_2 \in \mathcal{G} / t_1 \cup t_2$ est un arbre, $t_1 \subset t_2$ **et/ou** *$t_2 \subset t_1$.*

Comme toute PTSG maximalement dense est dense (lemme 3), d'après les théorèmes 2 et 3, la condition (2) du théorème 5 est équivalente à la polynomialité de $\mathcal{G}$ et on a donc :

Théorème 7.

Pour toute PTSG $\mathcal{G}$ maximalement dense, on a :
Si
(1) l'expansion de tout élément de $\mathcal{G}_1$ est constante sur $\mathcal{G} \setminus \mathcal{G}_1$
alors
(2) $\mathcal{G}$ est polynomiale.

Remarque 8. La condition (1) des théorèmes 5 et 7 n'est bien qu'une condition suffisante car il existe des PTSG polynomiales qui ne vérifient pas (1). Par exemple, la PTSG

$$\mathcal{G} = \left\{ \begin{array}{c} S \\ | \\ a \end{array} , \begin{array}{c} S \\ /\backslash \\ S\ S \end{array} , \begin{array}{c} S \\ /\backslash \\ S\ S \\ \quad | \\ \quad a \end{array} , \begin{array}{c} S \\ /\backslash \\ S\ S \\ |\ \ | \\ a\ \ a \end{array} , \begin{array}{c} S \\ /\backslash \\ S\ S \\ |\ \ | \\ a\ \ a \end{array} \right\}$$

est bien polynomiale (on peut par exemple vérifier que la condition (2) du théorème 3 est vérifiée) mais elle ne vérifie pas la condition (1) du théorème 7 car l'arbre $\begin{array}{c} S \\ /\backslash \\ S\ S \end{array}$ n'est pas d'expansion constante dans $\mathcal{G} \setminus \mathcal{G}_1$.

6.7 Méthode d'extraction des PTSG polynomiales

L'ensemble des résultats démontrés ci-dessus peut être mis à profit pour extraire des PTSG polynomiales à partir de corpus arborisés. Une méthode possible est donnée par l'algorithme d'extraction suivant, donné sous forme de théorème :

6.7.1 Extraction des sous-arbres élémentaires

Théorème 8.

Soit $\mathcal{C}$ un corpus arborisé et soit la PTSG $\mathcal{G}$ construite selon la procédure suivante :

1. *inclure dans $\mathcal{G}$ tous les sous-arbres de profondeur 1 présents dans $\mathcal{C}$;*
2. *pour chacun des arbres dans $\mathcal{G}_1$, définir une expansion et ajouter à $\mathcal{G}$ tous les sous-arbres présents dans $\mathcal{C}$ au sein desquels les expansions des éléments de $\mathcal{G}_1$ sont constantes ;*
3. *parcourir les sous-arbres restants t_0 du corpus $\mathcal{C}$ et ajouter à la grammaire $\mathcal{G}$ ceux qui vérifient :*
   $\forall t' \in \mathcal{G} \cup \{t_0\} / t_0 \cup t'$ est un arbre, $t_0 \cup t' \in \mathcal{G} \cup \{t_0\}$
   ($\mathcal{G}$ devient ainsi $\mathcal{G} \cup \{t_0\}$).

Alors $\mathcal{G}$ est polynomiale.

Démonstration. La PTSG issue de l'étape 1 est trivialement polynomiale.

Comme, du fait de l'étape 1, toutes les grammaires produites sont nécessairement maximalement denses, la polynomialité de la grammaire issue de l'étape 2 est garantie par le théorème 7.

Comme toutes les grammaires produites sont denses, la polynomialité de la grammaire issue de l'étape 3 est garantie par le théorème 7.

Pour ce qui est de l'étape 3, comme la grammaire $\mathcal{G}$ est maximalement dense, sa polynomialité est garantie par le théorème 4. □

Remarque 9. La PTSG $\mathcal{G}$ obtenue sera fortement conditionnée par :

1. la définition des expansions des éléments de $\mathcal{G}_1$ lors de l'étape 2 de la procédure de construction : parmi les schémas d'expansion systématiques, on peut par exemple envisager l'expansion systématique par la (feuille) tête, ou encore l'expansion systématique de toutes les feuilles non terminales ;

2. l'ordre de parcours des t lors de l'étape 3 de la procédure de construction : on peut par exemple envisager un parcours par profondeur croissante, par profondeur décroissante, ...

Exemple : Soit le corpus arborisé $\mathcal{C}$ suivant :

$$\mathcal{C} = \left\{ \begin{array}{ccc} S & S & S \\ /\backslash & /\backslash & /\backslash \\ S\ S & S\ S & S\ S \\ |\ | & |\ /\backslash & /\backslash\ | \\ a\ a & a\ S\ S & S\ S\ a \\ & |\ | & |\ |\ \\ & a\ a & a\ a \end{array} \right\}$$

En appliquant la procédure d'extraction du théorème 8 (avec comme règle d'expansion l'expansion systématique de toutes les feuilles non terminales et comme ordre de parcours l'ordre des profondeurs croissantes), on obtient par exemple la PTSG polynomiale constituée des arbres élémentaires suivants :

	Arbres élémentaires ajoutés
Étape 1	$\begin{array}{cc} S & S \\ /\backslash & \| \\ S\ S & a \end{array}$
Étape 2	$\begin{array}{ccccc} S & S & S \\ /\backslash & /\backslash & /\backslash \\ S\ S & S\ S & S\ S \\ |\ | & |\ /\backslash & /\backslash\ | \\ a\ a & a\ S\ S & S\ S\ a \\ & |\ | & |\ |\ \\ & a\ a & a\ a \end{array}$
Étape 3	$\begin{array}{ccccc} S & S & S & S \\ /\backslash & /\backslash & /\backslash & /\backslash \\ S\ S & S\ S & S\ S & S\ S \\ | & | & /\backslash & \| \quad /\backslash \\ a & a & S\ S & a \quad S\ S \\ & & |\ | & \quad\quad | \\ & & a\ a & \quad\quad a \end{array}$

6.7.2 Calcul des DOP-pseudo-probabilités

Les grammaires obtenues par l'algorithme décrit par le théorème 8 sont polynomiales. Pour utiliser efficacement cette propriété, il faut pouvoir calculer la DOP-pseudo-probabilité de chacun de leurs arbres élémentaires. En utilisant la définition de la polynomialité, il suffit pour connaître la DOP-pseudo-probabilité d'un arbre élémentaire t de connaître celles des arbres

élémentaires de sa décomposition maximale dans $\mathcal{G} \setminus \{t\}$, d'en faire le produit, et d'y ajouter la pseudo-probabilité de t. On peut alors calculer incrémentalement les paramètres de la grammaire équivalente par taille croissante d'arbres. Si l'on n'utilise pas cette propriété, on est forcé de calculer la DOP-pseudo-probabilité de chacun des arbres élémentaires par un algorithme *inside-outside* au mieux, ce qui alourdit considérablement la tâche.

On démontre pour finir deux lemmes utiles qui permettent de déterminer rapidement la décomposition maximale d'un arbre élémentaire t dans $\mathcal{G} \setminus \{t\}$. Il faut noter que cela n'est possible qu'avec les grammaires obtenues par les étapes 1 et 2 de l'algorithme 8. En effet, rien ne garantit que t possède une seule décomposition maximale dans $\mathcal{G} \setminus \{t\}$, qui n'est pas dans le cas général une grammaire polynomiale (même si $\mathcal{G}$ l'est).

Lemme 4.

Pour toute PTSG $\mathcal{G}$, on a :
Si
(1) l'expansion de tout élément de $\mathcal{G}_1$ est constante sur $\mathcal{G} \setminus \mathcal{G}_1$
alors
(2) tout arbre t de $\mathcal{C}(\mathcal{G}) \setminus \mathcal{G}_1$ possède un unique sous-arbre initial maximal dans $\mathcal{G} \setminus \{t\}$.

Démonstration. Supposons par l'absurde qu'on a (1) et qu'il existe $t \in \mathcal{C}(\mathcal{G}) \setminus \{t\}$ ne possédant pas un unique sous-arbre initial maximal dans $\mathcal{G}$. Comme tout arbre de $\mathcal{C}(\mathcal{G}) \setminus \mathcal{G}_1$ possède par construction au moins un sous-arbre initial maximal dans $\mathcal{G} \setminus \{t\}$, t possède $n > 1$ sous-arbre initiaux distincts $t_1, \ldots, t_n$ maximaux dans $\mathcal{G} \setminus \{t\}$. Considérons alors deux tels sous-arbres t_1 et t_2.

t_1 et t_2 étant tous deux des sous-arbres initiaux de t, ils possèdent au moins un sous-arbre initial commun. Soit t_0 leur sous-arbre initial commun *maximal*. Par construction, t_0 est un sous-arbre initial strict de t_1 et de t_2 (sinon t_1 et t_2 ne pourraient être maximaux et distincts). Ses expansions dans t_1 et dans t_2 sont donc non vides et ne peuvent être identiques car sinon t_0 ne serait pas maximal pour t_1 et t_2. Comme t_1 et t_2 sont des arbres élémentaires de $\mathcal{G}$, l'expansion de t_0 n'est alors pas *constante dans* $\mathcal{G}$, c'est-à-dire qu'il existe une nœud de t_0 qui n'a pas la même expansion dans t_1 et t_2. Un tel nœud est un élément de $\mathcal{G}_1$. (1) n'est donc pas vérifiée ce qui conclut la démonstration par l'absurde. □

On peut remarquer ici que si l'on utilise l'étape 3 de l'algorithme 8, cette propriété n'est plus vraie. Sur l'exemple précédent, le premier arbre de l'étape 2 possède les deux premiers arbres de l'étape 3 comme sous-arbres initiaux maximaux et le deuxième arbre de l'étape 2 possède les deux derniers arbres de l'étape 3 comme sous-arbres initiaux maximaux.

Lemme 5.

Pour toute PTSG $\mathcal{G}$ telle que l'expansion de tout élément de $\mathcal{G}_1$ est constante sur $\mathcal{G} \setminus \mathcal{G}_1$ et tout arbre t dans $\mathcal{C}(\mathcal{G})$, la décomposition δ de t obtenue
–en sélectionnant tout d'abord l'unique sous-arbre t_0 de t maximal dans $\mathcal{G} \setminus \{t\}$ (dont l'existence est garantie par le lemme 4),
–puis en itérant récursivement cette procédure de sélection pour chacun des sous-arbres de t dominés par les feuilles de t_0
est maximale.

Démonstration. Soit $t \in \mathcal{C}(\mathcal{G})$ et $\delta \in \Delta_{\mathcal{G}}(t)$ la décomposition construite par la procédure indiquée ci-dessus et supposons par l'absurde qu'il existe $\delta' \in \Delta_{\mathcal{G}}(t)/\delta < \delta'$. Comme $\delta < \delta'$, il existe, par définition, au moins un sous-arbre t' de δ' appartenant à $\mathcal{G}$ et admettant une décomposition dont les (au moins 2) éléments sont des arbres présents dans δ. En particulier, l'arbre présent dans δ correspondant au sous-arbre initial de t' dans cette décomposition est un arbre δ qui ne peut être maximal car le sous-arbre initial maximal, à la position qu'il occupe, contient t'. Cela est impossible du fait de la construction de δ.

En conséquence, δ est bien maximale. □

6.8 Conclusion

Nous avons donné dans ce chapitre tous les moyens nécessaires pour construire des grammaires à substitution d'arbres à partir d'un corpus d'arbres d'apprentissage, permettant d'effectuer l'analyse syntaxique d'une phrase en un temps polynomial. Ces moyens passent par :

- la sélection des arbres élémentaires pouvant, ou devant faire partie de la grammaire, parmi les sous-arbres d'un corpus ;
- la définition d'une grammaire équivalente dont les arbres élémentaires sont les mêmes que ceux la grammaire initiale mais associés à des probabilités élémentaires égales à la DOP-probabilité des arbres élémentaires de la grammaire initiale ;
- l'utilisation de la grammaire équivalente en analyse pour extraire la *dérivation la plus probable.*

Nous avons montré que la dérivation la plus probable obtenue avec la grammaire équivalente est aussi l'*analyse la plus probable* obtenue avec la grammaire initiale.

Dans cette thèse, les PTSG polynomiales ont vu leur application dans la comparaison des modèles STSG et GTSG, en permettant l'analyse syntaxique en un temps fini des bases de tests.

D'autres travaux rendent compte de la performance de ces modèles comparés au modèle DOP complet, i.e. où tous les sous-arbres apparaissant dans le corpus d'apprentissage sont utilisés comme arbres élémentaires de la grammaire. Ils ont montré que certaines instances (le modèle Min-max en particulier) peuvent obtenir des résultats en analyse équivalents aux résultats du modèle complet, tout en permettant une diminution considérable de la complexité de tâche d'analyse (voir [Chappelier 01b, Chappelier 01a, Chappelier 02b, Chappelier 02a]).

Chapitre 7

Conclusions et perspectives

7.1 Conclusions

Dans ce travail ont été développés deux modèles originaux de grammaires stochastiques non-génératifs : le modèle Gibbsien de Grammaire Hors-Contexte (GCFG), adapté du modèle de Grammaire Hors-Conexte Stochastique (SCFG), et le modèle Gibbsien de Grammaire à Substitution d'Arbres (GTSG), adapté du modèle de Grammaire à Substitution d'Arbres Stochastique (STSG).

Pour ces deux modèles, nous avons développé tout le processus permettant leur utilisation : nous avons exposé comment extraire ces modèles à partir d'un corpus d'arbres d'analyse, comment calculer leurs paramètres selon des critères discriminants, et comment les utiliser de façon efficace dans une tâche d'analyse syntaxique.

Deux principaux thèmes ont été soulignés : d'une part le thème des modèles non-génératifs, dont GCFG et GTSG font partie ; d'autre part le thème de la complexité algorithmique de l'analyse syntaxique à l'aide de grammaires à substitution d'arbres, qui a donné lieu à la présentation des grammaires à substitution d'arbres polynomiales.

Modèles non-génératifs

Une nouvelle méthode de valuation des grammaires hors-contextes a été présentée dans cette thèse. Elle diffère de celle des SCFG par son critère d'apprentissage, adapté à une tâche d'analyse syntaxique, et par l'absence de contraintes stochastiques ($p(r) \leq 1$, et $\sum_{r_X=(X\rightarrow\ldots)} p(r_X) = 1$) sur les valeurs de ses paramètres. La forme des grammaires obtenues étant sensiblement la même que celle des SCFG, on dispose d'algorithmes standards pouvant être utilisés en analyse syntaxique. On a aussi détaillé un algorithme d'apprentissage des paramètres, qui factorise suffisamment les calculs pour qu'ils soient exécutés en un temps raisonnable.

Les études expérimentales montrent qu'en analyse, les GCFG obtiennent d'excellents résultats, puisqu'elles obtiennent le comportement le meilleur de ce qu'on peut espérer de grammaires hors-contexte. Les paramètres obtenus collent mieux aux données d'apprentissage et à l'intuition que ceux d'une SCFG. Ils se comportent également mieux en généralisation, l'amélioration étant d'autant plus nette que le corpus d'apprentissage et la grammaire sont « adaptés » à un traitement statistique : chaque règle de la grammaire doit apparaître suffisamment souvent dans le corpus pour que l'on puisse en extraire des statistiques pertinentes. La difficulté de trouver de tels corpus met en évidence les limites d'une probabilisation des grammaires hors-contextes : celles-ci sont en général trop ou pas assez ambiguës ; les grammaires les plus ambiguës étudiées le sont trop pour qu'une simple probabilisation de leurs règles hors-contextes mène à des analyses satisfaisantes ; pour réduire leur ambiguïté, on est amené à intégrer dans leurs non-terminaux des informations additionnelles à la nature de groupes syntaxiques, ce qui a pour effet de multiplier le nombre de règles hors-contextes à probabiliser. Le biais dû à l'utilisation des paramètres génératifs en analyse est alors souvent négligeable comparé au biais dû au faible rapport du volume des données d'apprentissage au nombre de paramètres à évaluer. Le gain constaté en généralisation par l'emploi d'une grammaire non-générative reste alors relativement faible sur de tels corpus.

Le principe d'une GCFG repose sur l'utilisation de ses paramètres directement dans la définition de la probabilité conditionnelle $p(a|w)$ d'un arbre a syntaxique connaissant ses feuilles w, que sont les mots de la phrase à analyser. Ses paramètres sont calculés de façon à maximiser la probabilité des arbres du corpus d'apprentissage, connaissant les phrases. Ce modèle s'oppose ainsi à un modèle génératif tel qu'une SCFG, dont les paramètres servent initialement à définir la probabilité $p(a)$ des arbres, et sont calculés de façon à maximiser la probabilité des arbres du corpus, comme produits à partir de leur racine par un processus génératif.

Le même principe a été appliqué aux grammaires à substitution d'arbre : une version probabiliste non-générative (GTSG) a été développée, et comparée à leur version générative (STSG). Les résultats obtenus sont là encore en faveur du nouveau modèle, qui surpasse les STSG sur la tâche d'analyse syntaxique, avec le même bémol que pour les GCFG : l'amélioration se fait surtout sentir lorsque la base est suffisamment grande et redondante.

Grammaires à substitution d'arbres polynomiales

Une nouvelle approche du modèle DOP a été présentée : sa restriction à des grammaires probabilistes à substitution d'arbres polynomiales (pPTSG), c'est-à-dire des PTSG pour lesquelles la recherche de l'arbre d'analyse le plus probable peut s'effectuer de façon exacte en un temps polynomial (par oppo-

sition au caractère NP-difficile de cette recherche dans le cas général). Une caractérisation complète des pPTSG a été donnée, ainsi que des conditions suffisantes de polynomialité, qui permettent l'extraction efficace de telles grammaires à partir d'un corpus arborisé. Les modèles obtenus constituent alors un compromis intéressant entre le modèle DOP général et les grammaires hors-contextes : il est aussi « simple » à utiliser en analyse qu'une grammaire hors-contexte (complexité cubique) tout en permettant une probabilisation plus riche. La réduction de la complexité de l'analyse vient du fait que l'on peut associer à une pPTSG une grammaire PTSG pour laquelle la recherche de la dérivation la plus probable est équivalente à la recherche de l'analyse la plus probable dans la grammaire d'origine. Un cas particulier de la restriction des arbres élémentaires aux seuls arbres de profondeur 1 et aux arbres totalement ancrés (principe de sélection maximale-minimale) constitue une telle pPTSG. Elle correspond à l'extraction d'arbres élémentaires du corpus par le schéma d'expansion sur toutes les feuilles des arbres de profondeur 1. Un second cas particulier consiste en l'expansion systématique (à chaque niveau) de la tête lexicale de chaque sous-arbre du corpus. Bien que ses performances sur le corpus ATIS ne soient pas aussi bonnes que celles obtenues avec la première méthode, cette approche devrait encore être testée, par exemple sur des corpus où la notion de tête lexicale est plus pertinente.

7.2 Perspectives

En ce qui concerne les perspectives ouvertes par les résultats obtenus dans le cadre de cette recherche, le point le plus pertinent paraît être la prise en compte des principes présentés dans les cas où des modèles génératifs ne sont pas employés dans une tâche de génération :

- d'une part, les paramètres des modèles développés dans le cadre des modèles non génératifs ne requièrent aucune normalisation, car ils ne correspondent pas (directement) à des valeurs de probabilités. Ils sont donc moins contraints, et les scores obtenus ne dépendent pas d'un présupposé sur la façon dont on les utilise (i.e. un processus stochastique génératif). On a montré que ce relâchement de contraintes qui peuvent sembler artificielles ne pénalise de fait pas les résultats des quelques modèles d'analyse syntaxique considérés.
- d'autre part, les critères d'apprentissage des modèles développés sont conçus de façon à « coller » à la tâche à laquelle ces modèles seront affectés, plutôt que de correspondre aux tâches pour lesquelles ils ont initialement été conçus : on peut directement probabiliser l'analyse syntaxique plutôt que de probabiliser un processus génératif intermédiaire.

Ces propriétés peuvent être exploitées à divers niveaux dans le traitement du langage, depuis le balisage de texte par des catégories morpho-syntaxiques (les modèles gibbsiens s'intègrant très facilement à des algorithmes de type Viterbi) à la traduction automatique. R. Besançon a par exemple commencé à intégrer de tels modèles pour réaliser des associations sens-mots dans la partie sémantique d'un système de recherche documentaire [Besançon 02]. De façon générale, chaque fois que l'on utilise un processus génératif de type markovien pour évaluer le score d'une hypothèse, on peut se poser la question de savoir s'il ne serait pas préférable d'estimer ce score par un modèle de type non-génératif.

Un autre aspect intéressant des modèles présentés est la procédure de lissage proposée à la section 5.3.2 sous la dénomination « apprentissage par profondeur croissante ». Cette procédure pourrait être envisagée dans le cadre de modèles de type « N-grammes non-génératifs », qui seraient aux N-grammes ce que les GTSG sont aux STSG.

Les propriétés des modèles non-génératifs pourraient également être exploitées en reconnaissance de la parole. Elles le sont d'ailleurs déjà en partie, avec l'utilisation de modèles neuronaux [Boite 00], qui implémentent à leur façon des critères d'apprentissage des probabilités des phonèmes *connaissant le signal*, et viennent remplacer dans les systèmes les plus récents des modèles de multi-gaussiennes, qui modélisent quant à eux la probabilité d'émettre un signal *connaissant le phonème*, et qui sont ensuite utilisés « à l'envers », la tâche de reconnaissance partant du signal : les critères d'apprentissage des réseaux neuronaux sont alors bien définis par rapport à la tâche pour laquelle il seront utilisés.

Les systèmes de reconnaissance de la parole font largement appel à des modèles de langage génératifs probabilistes, et cela semble peu dérangeant du fait qu'il « génèrent » effectivement des phrases en sortie. Cependant, l'apprentissage de ces modèles correspond en général à la tâche de génération de phrases *sorties de tout contexte*, et non pas à cette génération *à partir du signal*. De plus, les processus stochastiques sous-tendant ces modèles imposent une normalisation de leurs paramètres, qui les rend mal adaptés à leur mariage avec des probabilités phonétiques à l'aide de formules probabilistes pourtant classiques (Bayes). Cela apparaît de deux façons : d'une part ils ont tendance à prendre trop ou pas assez d'importance par rapport aux probabilités phonétiques (problème de l'apprentissage décorrélé du contexte applicatif), d'autre part ils favorisent excessivement les interprétations qui nécessitent le moins d'étapes du processus pour être générées (problème de la normalisation des probabilités).

- La réponse habituelle à ces problèmes passe par l'ajout de paramètres α et β, l'un en facteur, l'autre en exposant, aux probabilités associées à une étape du processus stochastique. Ces paramètres, déterminés par

des moyens empiriques, sont alors intégrées aux formules de Bayes pour extraire les interprétations d'un signal.

- D'autres approches tentent de se passer du facteur, par exemple en remplaçant la probabilité de l'émission d'une phrase par sa moyenne géométrique sur le nombre d'étapes nécessaires à sa génération (V. Annexe C)
- L'exemple des modèles syntaxiques présentés dans ce travail pourrait être suivi, en supprimant d'abord les contraintes stochastiques appliquées aux paramètres des processus en cause (donc en enlevant à ceux-ci leur caractère « procédural »), et en réalisant leur apprentissage dans un schéma de reconnaissance de la parole, en conformité avec leur tâche. Il se peut qu'on supprime ainsi la nécessité d'avoir à déterminer des paramètres empiriques correctifs. On aurait surtout alors un cadre mathématique plus clair et uniforme tout le long du processus de reconnaissance, et pourquoi pas ?, les modèles obtenus pourraient mener à des résultats meilleurs...

Annexe A

NP-difficulté du problème MPP avec une STSG

Trouver l'arbre d'analyse le plus probable (MPP) selon une grammaire à substitution d'arbres est dans le cas général[1] un problème NP-difficile. C'est ce qu'a prouvé K. Sima'an dans [Sima'an 96a], à qui l'on doit la démonstration présentée dans cette annexe.

On peut prouver que le problème MPP est NP-difficile en réduisant toute instance du problème 3SAT à un problème de seuil que l'on nomme sMPP, dont MPP est le problème d'optimisation associé, cette réduction se faisant avec une complexité polynomiale.

Le problème 3SAT est le suivant :

- Étant donné une formule booléenne sous forme normale 3CNF, sur les variables $u_1, \ldots, u_n$:

  $$F = (d_{11} \vee d_{12} \vee d_{13}) \wedge (d_{21} \vee d_{22} \vee d_{23}) \wedge \cdots \wedge (d_{m1} \vee d_{m2} \vee d_{m3})$$

  où $m \geq 1$ et d_{ij} est un littéral parmi $\{u_1 \ldots, u_n\}$, pour tout $1 \leq i \leq m$ et tout $1 \leq j \leq 3$. Cette formule est aussi notée $C_1 \wedge C_2 \wedge \cdots \wedge C_m$, où C_i représente $(d_{i1} \vee d_{i2} \vee d_{i3})$, pour tout $1 \leq i \leq m$.
- La formule F est-elle satisfiable, i.e. est-elle vraie pour au moins une affectation des variables $u_1 \ldots u_n$?

Le problème sMPP est quant à lui :

- Étant donnée une STSG $\mathcal{G}$, une phrase w et un réel s compris entre 0 et 1,
- existe-t-il une analyse de w dans $\mathcal{G}$ dont la probabilité, assignée par $\mathcal{G}$, est supérieure ou égale à s ?

[1] avec une forme quelconque de STSG

La réduction de 3SAT à une instance de sMPP doit construire une STSG, une phrase, et une valeur de seuil, de façon à ce que la réponse au problème sMPP corresponde à la réponse à 3SAT. On prend pour exemple la formule 3SAT [Barton 87] :

$$(u_1 \vee \overline{u_2} \vee u_3) \wedge (\overline{u_1} \vee u_2 \vee \overline{u_3})$$

où u_1, u_2 et u_3 sont des variables booléennes.

Une instance 3SAT est satisfiable si et seulement si au moins un littéral de chaque disjonction reçoit la valeur $vrai$. Les différentes occurrences d'une même variable doivent évidemment recevoir la même valeur. Ces deux observations constituent la base de la réduction : la STSG va être construite de façon à ce qu'une solution de F soit représentée par un arbre admettant deux sortes de dérivations : les unes prennent en charge la consistance des valeurs des variables le long de la formule, les autres assignent une valeur $vrai$ à une variable de chaque disjonction. Les dérivations auront des probabilités qui permettent de dire si un arbre est solution de F en regardant si sa probabilité est supérieure à un seuil donné. La probabilité d'un arbre indique s'il possède suffisamment de dérivations de chaque sorte pour être solution de F.

Réduction : La réduction construit une STSG et une phrase. La STSG a pour symbole initial S, un terminal 1, et les non-terminaux

$$\{V, F, C_1, C_2, , \ldots, C_m, u_1, \overline{u_1}, u_2, \overline{u_2}, \ldots, u_n, \overline{u_n}\}$$

La phrase à analyser est $\underbrace{11\ldots1}_{\text{longueur : } 3m}$. Les arbres élémentaires sont construits comme suit :

1. Pour chaque variable booléenne u_i ($1 \leq i \leq n$), on construit deux arbres élémentaires, l'un assignant la valeur $vrai$ et l'autre la valeur $faux$ à u_i, sur toute la formule. Chacun de ces arbres a pour racine S, avec pour enfants $C_1, \ldots, C_m$, les enfants de C_k étant les non-terminaux correspondant à ses variables d_{k1}, d_{k2} et d_{k3}. L'arbre correspondant à l'affectation de la valeur $vrai$ (resp. $faux$) à u_i est modélisé par l'ajout d'un enfant V (resp. F) à chaque non-terminal u_i et d'un enfant F (resp. V) à chaque non-terminal $\overline{u_i}$. Les arbres élémentaires pour u_1, u_2 et u_3 apparaissent en haut à gauche de la figure A.1.
2. Trois arbres élémentaires sont construits pour chaque disjonction C_k, chacun ayant pour racine C_k avec comme enfants les non-terminaux correspondants à leurs variables d_{k1}, d_{k2} et d_{k3}. Chaque arbre est créé par l'ajout d'un enfant V en position fils de l'un des d_{ki}, et vérifie ainsi qu'un littéral de la formule est vrai. Les arbres élémentaires correspondants dans l'exemple apparaissent en haut à droite de la figure A.1.

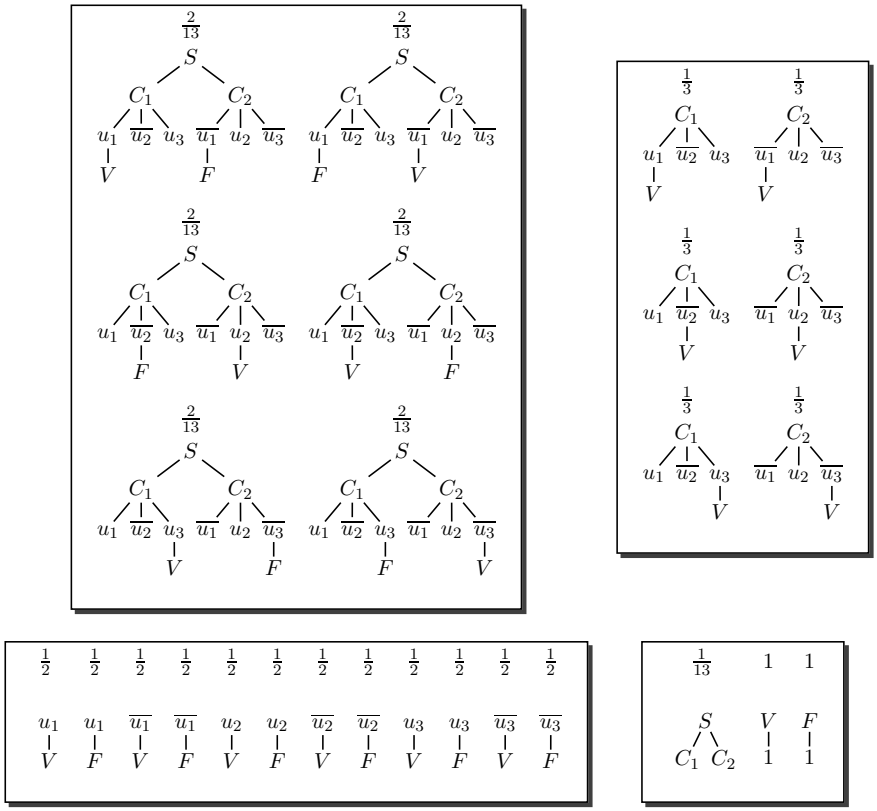


FIG. A.1 – Arbres élémentaires de l'exemple de la réduction de 3SAT à sMPP.

3. Pour chaque littéral de chaque variable u_i, deux arbres élémentaires sont construits, l'un assignant V, l'autre F en fils du littéral. Ces arbres apparaissent en bas à gauche de la figure A.1 pour l'exemple.
4. Un arbre de racine S et de fils $C_1, \ldots, C_m$, ainsi que les arbres de racine V et F et de fils 1 sont ajoutés à la grammaire (voir en bas à droite de la figure A.1).
5. Des probabilités sont associées aux arbres élémentaires ainsi construits. Les probabilités des arbres élémentaires ayant la même racine doivent sommer à 1. La probabilité d'un arbre élémentaire de racine S construit à l'étape 1 de la réduction est p_i, où u_i est la seule variable dont les littéraux ont un fils dans l'arbre en question. Soit n_i le nombre d'occurrences des deux littéraux u_i et $\overline{u_i}$ dans F. On pose $p_i = \theta(\frac{1}{2})^{n_i}$, où θ est un réel devant vérifier certaines conditions dérivées dans la suite. La probabilité de l'arbre de racine S construit à l'étape 4 doit donc être $p_0 = [1 - 2\sum_{i=1}^{n} p_i]$. La probabilité des arbres élémentaires de racine c_k construits à l'étape 2 est $(\frac{1}{3})$, celle des arbres de racine u_i et $\overline{u_i}$ est $(\frac{1}{2})$, et celle des arbres de racine V et F est (1).

Soit Q le seuil de probabilité défini plus bas. L'instance du problème sMPP produit par cette réduction est la suivante :

- Étant donné la STSG produite par la réduction (les probabilités sont définies plus bas), la phrase $\underbrace{11\ldots1}_{\text{longueur : } 3m}$, et le seuil Q,
- existe-t-il un arbre d'analyse dont la probabilité est supérieure ou égale à Q ?

Recherche des probabilités et du seuil En observant la STSG résultant de cette réduction, on voit qu'un arbre d'analyse possède deux types de dérivations :

1. Le premier type utilise un des $2n$ arbres élémentaires de racine S construits à l'étape 1 de la réduction. Ce type de dérivation correspond à l'affectation de valeurs aux littéraux d'une variable u_i d'une manière consistante. Pour chaque $1 \leq i \leq n$, la probabilité d'une telle dérivation est :
$$p_i \left(\frac{1}{2}\right)^{3m-n_i} = \theta \left(\frac{1}{2}\right)^{3m}$$
$3m - n_i$ est en effet le nombre de sites de substitution auxquels fixer des arbres élémentaires construits à l'étape 3 sur l'arbre élémentaire correspondant à la variable u_i.
2. Le second type de dérivations substitue les arbres élémentaires de racine C_k dans l'arbre de racine S construit à l'étape 4, puis les arbres de racines u_i et $\overline{u_i}$, puis de racine F et V aux sites de substitution

correspondants. Ce type de dérivation correspond à l'affectation de la valeur *vrai* à au moins un littéral de chaque disjonction. La probabilité d'une telle dérivation est :

$$p_0 \left(\frac{1}{2}\right)^{2m} \left(\frac{1}{3}\right)^{m} = \left[1 - 2\theta \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{n_i}\right] \left(\frac{1}{2}\right)^{2m} \left(\frac{1}{3}\right)^{m}$$

On cherche maintenant des valeurs au seuil Q et au paramètre θ. Tout arbre remplissant la condition de consistance des affectations et la condition de vérité de chaque disjonction doit avoir n dérivations du premier type et au moins une dérivation du second type. On peut noter qu'un arbre ne peut pas avoir plus de n dérivations du premier type. De ce fait, le seuil Q doit être fixé à :

$$Q = n\theta \left(\frac{1}{2}\right)^{3m} + \left[1 - 2\theta \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{n_i}\right] \left(\frac{1}{2}\right)^{2m} \left(\frac{1}{3}\right)^{m}$$

θ doit de plus remplir certaines conditions pour que la réduction soit acceptable :

1. Pour tout i : $0 \leq p_i \leq 1 \Rightarrow 0 \leq \theta \left(\frac{1}{2}\right)^{n_i} \leq 1 \Rightarrow 0 \leq \theta \leq 2^{n_i}$. De plus, pour p_0, il faut vérifier : $0 \leq p_0 \leq 1 \Rightarrow 0 \leq 2\theta \sum_{i=1}^{n} (\frac{1}{2})^{n_i} \leq 1$. Cela mène à la condition (plus forte que les n précédentes) :

$$0 \leq \theta \leq \frac{1}{2\sum_{i=1}^{n} (\frac{1}{2})^{n_i}}$$

2. Comme on veut être capable de dire si un arbre possède le second type de dérivations uniquement en regardant sa probabilité, la probabilité du second type de dérivations doit se distinguer de celles du premier type. De plus, si un arbre possède plusieurs dérivations du deuxième type, il ne faut pas que la somme des probabilités de ces dérivations soit confondue avec la probabilité d'une dérivation du premier type. Tout arbre possède au plus 3^m dérivations du second type. On demande pour cette raison à ce que :

$$3^m \left[1 - 2\theta \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{n_i}\right] \left(\frac{1}{2}\right)^{2m} \left(\frac{1}{3}\right)^{m} < \theta \left(\frac{1}{2}\right)^{3m}$$

ce qui équivaut à :

$$\theta > \frac{1}{2\sum_{i=1}^{n} (\frac{1}{2})^{n_i} + (\frac{1}{2})^{m}}$$

Existence de θ : Il existe un θ satisfaisant ces conditions du fait que la borne inférieure $\frac{1}{2\sum_{i=1}^{n} (\frac{1}{2})^{n_i} + (\frac{1}{2})^{m}}$ est strictement inférieure à la borne supérieure $\frac{1}{2\sum_{i=1}^{n} (\frac{1}{2})^{n_i}}$.

Polynomialité de la réduction : La réduction se fait en temps polynomial en m et n. Elle ne construit pas plus de $2n + 3m + 4n + 1 + 2$ arbres élémentaires, chacun composé d'au plus $7m + 1$ nœuds. Le calcul des probabilités élémentaires de ces arbres et du seuil Q se fait aussi en temps polynomial.

La réduction préserve les réponses : Examinons les deux réponses au problème F :

- F **est satisfiable** : Si telle est la réponse, alors il existe une affectation aux variables qui soit consistant et pour lequel chaque disjonction a au moins un littéral $vrai$. Chaque affectation possible est représentable par un arbre de la STSG. Un arbre correspondant à une affectation « satisfaisante » doit avoir n dérivations du premier type et au moins une dérivation du second type. La probabilité d'un tel arbre doit donc être supérieure ou égale à Q.
- F **n'est pas satisfiable :** toutes les affectations possibles aux variables u_i sont soit non-consistantes, soit impliquent au moins une disjonction fausse. Les arbres correspondant à des affectations non-consistantes ont des probabilités inférieures au seuil Q, car ils ont moins de n dérivations du premier type, et les dérivations du second type ne peuvent pas compenser la probabilité manquante. Pour les affectations consistantes, comme elles impliquent au moins une disjonction C_k fausse, les arbres correspondants ne peuvent avoir le second type de dérivation, car celles-ci font appel à un arbre élémentaire pour tout k de racine C_k vérifiant que C_k est vrai.

En résumé, on a une réduction polynomiale du problème 3SAT au problème de la recherche de l'existence d'un arbre de probabilité supérieure à un seuil donné dans une grammaire à substitution d'arbre. On remarque qu'on peut de plus déterminer si un arbre donné est solution du problème en un temps polynomial, en calculant la valeur Interne de sa racine S, selon un algorithme tel que celui présenté page 115. Le problème sMPP est donc NP-complet.

Or le problème MPP avec une STSG est le problème d'optimisation associé au problème sMPP : il consiste à *trouver l'arbre le plus probable* dans une grammaire $\mathcal{G}$, alors que sMPP consiste à *déterminer s'il existe un arbre de probabilité supérieure à un seuil.* Si l'on trouve la réponse à MPP, on a automatiquement la réponse à sMPP, donc le problème MPP est au moins aussi difficile que sMPP : MPP est NP-difficile avec une STSG. De plus, une STSG est une instance particulière d'une PTSG (les GTSG présentées dans cette thèse étant une autre instance). Donc le problème MPP est NP-difficile avec une PTSG.

Remarque 10. Cette propriété se traduit par le fait qu'*on peut trouver* une PTSG et une phrase pour lesquels MPP est NP-difficile, et non pas que ce

problème est NP-difficile *pour toute PTSG et pour tout énoncé*. Par exemple, les grammaires hors-contextes probabilistes, qui sont des instances de PTSG, permettent de résoudre MPP en un temps polynomial. C'est cette remarque qui est à la base de la caractérisation et du développement de grammaires PTSG polynomiales (chapitre 6).

Annexe B

Algorithme Inside-Outside

Dans cette partie est explicité l'algorithme Inside-Outside[1], utilisé pour factoriser le calcul des expressions du type :

$$\sum_{\delta \Rightarrow w} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(\delta)} \right] \tag{B.1}$$

où :

- $w = w_1 \dots w_n$ est une suite de mots, ou terminaux ;
- $\sum_{\delta \Rightarrow w}$ représente une somme sur toutes les dérivations qui produisent la chaîne w selon une grammaire hors-contexte $\mathcal{G}$;
- les α sont des éléments de δ, i.e. des règles de $\mathcal{G}$ intervenant dans la dérivation ;
- $v(\alpha)$ est une fonction des α, dont l'image se trouve dans un semi-anneau commutatif $\mathcal{A}$. $\sum$ et $\prod$ sont les opérations *somme* et *produit* de $\mathcal{A}$, ayant respectivement 0 et 1 comme éléments neutres ;
- $f_\xi(\delta)$ est le nombre d'occurrences de ξ dans l'arbre syntaxique associé à la dérivation δ. ξ peut être un symbole ou une règle hors-contexte. $f_\xi(\delta)$ est un entier [2].

Note : Dans cette thèse, les valeurs que l'on calcule sont dans des réels positifs, ou des polynômes à coefficients réels positifs.

B.1 Représentation des dérivations

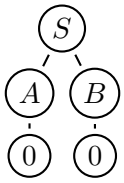
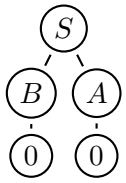
Soit $\mathcal{T}$ un ensemble de symboles, dits symboles terminaux, $\mathcal{NT}$ un ensemble de symboles « non-terminaux », $\mathcal{R} \subset \mathcal{I} \times \mathcal{NT} \times (\mathcal{T} \cup \mathcal{NT})^*)$ un

[1] L'algorithme Inside-Outside est décrit (encore plus) en détail dans [Goodman 98], dont cette annexe s'inspire librement, et qu'elle développe dans le cas particulier des grammaires à substitution d'arbre.

[2] La multiplication d'une valeur z par un entier, dans $\mathcal{A}$, est simplement définie récursivement par « $0 * z = 0$ et $\forall n \in \mathbb{N}, (n+1) * z = (n * z) + z$ ».

ensemble de règles de réécriture, $S \in \mathcal{NT}$ un « symbole initial ». Le quadruplet $(\mathcal{T}, \mathcal{NT}, \mathcal{R}, S) = \mathcal{G}$ est la grammaire hors-contexte.

Une règle α de $\mathcal{R}$ s'écrit : (id, X, y), ou encore $(id) : X \to y$, où $id \in \mathcal{I}$ est un identifiant de la règle, $X \in \mathcal{NT}$ est le symbole récrit, ou *symbole de tête* de la règle, et $y \in (\mathcal{T} \cup \mathcal{NT})^*$) est une suite de symboles, résultat de la réécriture de X par la règle α et appelés symboles *de queue* . L'identificateur sert à pouvoir distinguer deux règles ayant même tête et feuilles. Par exemple, si l'on utilise cette grammaire hors-contexte comme intermédiaire d'une TSG, comme expliqué plus bas, les fragments :

$\alpha_{TSG1} =$  et $\alpha_{TSG2} =$ 

sont traduits par $(\alpha_{TSG1}) : S \to 0\ 0$ et $(\alpha_{TSG2}) : S \to 0\ 0$ afin d'être distingués. Dans la suite, cet identifiant sera noté par un entier, et l'on garde la notation α pour désigner les règles de la grammaire dans laquelle on travaille, fût-elle hors-contexte. Lorsqu'il n'est pas nécessaire, on omettra de noter l'identifiant.

On définit une opération de composition notée « $\circ$ » de $(\mathcal{T} \cup \mathcal{NT})^* \times \mathcal{R}$ dans $(\mathcal{T} \cup \mathcal{NT})^*$ ainsi :

$w \circ \alpha = w'$ si et seulement si

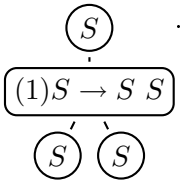
- le premier symbole non-terminal de w est le symbole de tête de α (l'opération n'est pas définie dans le cas contraire) ;
- w' est la chaîne obtenue en remplaçant dans w le premier symbole non-terminal par les symboles de queue de α.

Si $(\ldots(S \circ \alpha_1) \circ \ldots) \circ \alpha_n = w$, alors $\delta = (\alpha_1, \ldots, \alpha_n)$ est appelée une *dérivation* de w. On peut la noter : $\delta = \alpha_1 \circ \ldots \circ \alpha_n$. Par exemple : $(S \to A\ B) \circ (A \to 0) \circ (B \to 0)$ est une dérivation de "0 0", obtenue par la succession de réécritures :

$$\text{"}S\text{"} \overset{S \to A\ B}{\longrightarrow} \text{"}A\ B\text{"} \overset{A \to 0}{\longrightarrow} \text{"}0\ B\text{"} \overset{B \to 0}{\longrightarrow} \text{"}0\ 0\text{"}$$

Sous cette forme, connue en théorie des langages formels, les dérivations sont peu pratiques à utiliser pour la démonstration qui vient. On a par exemple du mal à définir la notion de dérivation pour une chaîne partiellement lexicalisée : comment dériver "$S\ a$" avec les règles $S \to S\ S$ et $S \to a$? Si l'on compose "$S\ S$" avec $S \to a$, on ne peut obtenir que "$a\ S$" car seul le premier non-terminal de la chaîne est pris en compte par l'opération.

On va donc utiliser une représentation alternative pour les dérivations, plus adaptée aux calculs que l'on souhaite mener à bien : une représentation en arbre. On associe maintenant à chaque règle α de la grammaire un arbre

élémentaire $t(\alpha)$, de profondeur 2, dont la racine est la tête de α, ayant pour fils unique α lui-même, et pour petits-fils les symboles de queue de α. Par exemple, $(1)S \to S\ S$ est associé à 

.

Une dérivation δ est alors représentée par un arbre $d(\delta)$, ainsi défini :
- une dérivation de longueur 1 ($\delta = \alpha$) est représentée par $d(\delta) = t(\alpha)$.
- $d(\delta \circ \alpha)$ est obtenu en substituant à la première feuille non-terminale de $d(\delta)$ l'arbre $t(\alpha)$.

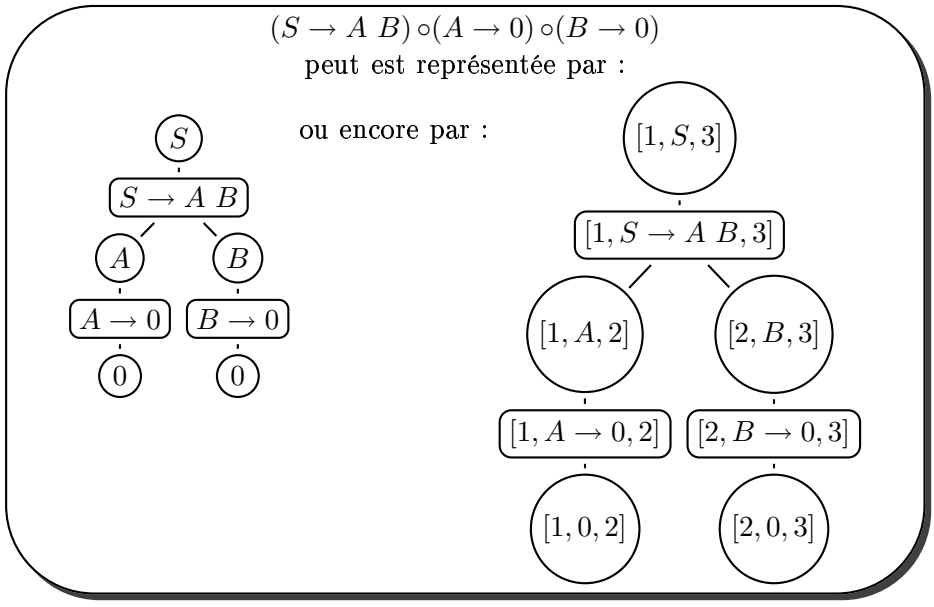


FIG. B.1 – Exemple de représentation en arbre d'une dérivation

Si δ récrit S en w, l'arbre obtenu $d(\delta)$ est par construction un arbre d'analyse de w dans une grammaire à substitution d'arbres $\mathcal{G}$ dont les fragments sont les arbres $t(\alpha)$, et le symbole initial est S.

Soit $\Delta(w)$ l'ensemble des dérivations de w ; soit $\mathcal{D}(w)$ l'ensemble des arbres d'analyse de w dans $\mathcal{G}$. L'opération exposée précédemment, qui associe à une dérivation de $\Delta(w)$ un arbre de $\mathcal{D}(w)$, est clairement bijective : pour trouver l'antécédent d'un arbre, il suffit d'en effectuer un parcours gauche-droite, en profondeur d'abord. Si la suite de fragments ainsi trouvée est $t_1, \ldots, t_n$, la dérivation associée est $t^{-1}(t_1) \circ \ldots \circ t^{-1}(t_n)$.

De plus, chaque nœud d'une dérivation de $\mathcal{D}(w)$ sera étiqueté par sa position par rapport aux terminaux. Ainsi $\nu = [i, \alpha, j]$ représente un nœud contenant α et dominant les mots $w_i \ldots w_{j-1}$. Enfin, on affecte une valeur aux nœuds :

$$v([i,\xi,j]) = \begin{cases} 1 & \text{si } \xi \text{ est un symbole, terminal ou non ;} \\ v(\xi) & \text{si } \xi \text{ est une règle.} \end{cases}$$

Par construction, la valeur d'une dérivation est égale au produit des valeurs associées aux nœuds de l'arbre qui la représente[3]. De plus, ξ apparaît autant de fois dans l'arbre syntaxique produit par δ qu'il y a de nœuds labellisés par ξ dans $d(\delta)$:

$$\sum_{1 \leq i < j \leq n} f_{[i,\xi,j]}(d(\delta)) = f_\xi(\delta) \tag{B.2}$$

Ainsi :

$$\begin{aligned}
&\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)} \right] \\
&= \sum_{\delta \in \Delta(w)} \sum_{1 \leq i < j \leq n} f_{[i,\xi,j]}(d(\delta)) \left[\prod_{\nu \in d(\delta)} v(\nu)^{f_\nu(d(\delta))} \right] && \text{(B.3)} \\
&= \sum_{d \in \mathcal{D}(w)} \sum_{1 \leq i < j \leq n} f_{[i,\xi,j]}(d) \left[\prod_{\nu \in d} v(\nu)^{f_\nu(d)} \right] && \text{(B.4)} \\
&= \sum_{1 \leq i < j \leq n} \sum_{d \in \mathcal{D}(w)} f_{[i,\xi,j]}(d) \left[\prod_{\nu \in d} v(\nu)^{f_\nu(d)} \right] && \text{(B.5)}
\end{aligned}$$

((B.3) vient de (B.2), (B.4) vient de ce que $\Delta(w)$ et $\mathcal{D}(w)$ sont en bijection et (B.5) est obtenu par inversion des sommes.)

C'est donc cette dernière expression que l'on cherche à calculer dans la suite.

B.2 Binarisation de la grammaire

Les algorithmes qui viennent nécessitent l'utilisation d'une grammaire hors-contexte dont les règles contiennent au maximum deux symboles de queue. Or, on veut pouvoir les utiliser avec des grammaires hors-contextes quelconques, ainsi qu'avec des grammaires à substitution d'arbre.

On va donc décrire dans cette section :

- la transformation d'une grammaire à substitution d'arbre en une grammaire hors-contexte ;

[3]Comme on se place dans un semi-anneau commutatif, l'ordre dans lequel ce produit est exécuté n'a pas d'importance.

- la transformation d'une grammaire hors-contexte en une grammaire hors-contexte binaire ;
- le passage des valeurs associées aux règles, d'un modèle à l'autre, de façon à ce que les valeurs des dérivations soient conservées.

B.2.1 $TSG \rightarrow CFG$

Soit $\mathcal{G}_{TS} = (\mathcal{T}, \mathcal{NT}, \mathcal{R}_{TS}, S)$ une grammaire à substitution d'arbres, et

$$v_{TS} \ : \begin{cases} \mathcal{R}_{TS} & \longrightarrow \mathcal{A} \\ r_{TS} & \longmapsto v_{TS}(r_{TS}) \end{cases}$$

une fonction des règles de $\mathcal{G}_{TS}$ dans un semi-anneau commutatif $\mathcal{A}$. La valeur d'une dérivation d est définie comme le produit des valeurs $v(r_{TS})$ des règles qui la composent.

On définit la grammaire hors-contexte équivalente $\mathcal{G}_{CF} = (\mathcal{T}, \mathcal{NT}, \mathcal{R}_{CF}, S)$ en conservant les symboles terminaux et non-terminaux, ainsi que le symbole initial S de $\mathcal{G}_{TS}$, et en construisant l'ensembles des règles $\mathcal{R}_{CF}$ comme l'image de $\mathcal{R}_{TS}$ par la fonction :

$$map_{TS \rightarrow CF} \ : \begin{cases} \mathcal{R}_{TS} & \longrightarrow \mathcal{R}_{TS} \times \mathcal{NT} \times ((\mathcal{T} \cup \mathcal{NT})^* \\ \alpha & \longmapsto (\alpha, racine(\alpha) \rightarrow feuilles(\alpha)) \end{cases}$$

L'image d'une règle α est donc une règle hors-contexte dont la partie gauche est la racine de α, et la partie droite est la suite des feuilles de α, parcourues de gauche à droite. Les règles obtenues sont de plus *indexées* sur les règles d'origine, de façon à garantir que $map_{TS \rightarrow CF}$ est une bijection de $\mathcal{R}_{TS}$ dans $\mathcal{R}_{CF}$.

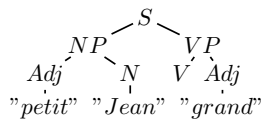
	TSG	CFG
règle	$\alpha =$ 	$(\alpha, S \rightarrow$ "petit" "jean" V "grand"$)$
valeur	$v(\alpha)$	$v(\alpha)$

FIG. B.2 – Transformation d'une grammaire TS en une grammaire CF.

Il apparaît clairement que $map_{TS \rightarrow CF}$ permet également de construire une bijection de l'ensemble des dérivations d'une phrase w dans $\mathcal{G}_{TS}$ dans l'ensemble des dérivations de w dans $\mathcal{G}_{CF}$, le passage d'une dérivation à son image (resp. antécédent) se faisant en remplaçant chaque règle y apparaissant par son image par $map_{TS \rightarrow CF}$ (resp. $map_{TS \rightarrow CF}|_{\mathcal{R}_{CF}}{}^{-1}$).

En posant :

$$v_{CF} \ : \ \begin{cases} \mathcal{R}_{CF} & \longrightarrow \mathcal{A} \\ r_{CF} = (\alpha, X \rightarrow Y_1 \dots Y_n) & \longmapsto v_{CF}(r_{CF}) = v_{TS}(\alpha) \end{cases}$$

on voit également que les dérivations en bijection auront les mêmes valeurs, car les éléments qu'elles contiennent ont les mêmes valeurs.

B.2.2 $CFG \rightarrow bCFG$

Soit $\mathcal{G}_{CF} = (\mathcal{T}, \mathcal{NT}, \mathcal{R}_{CF}, S)$ une grammaire hors-contexte, les règles de $\mathcal{R}_{CF}$ étant de la forme $(\alpha, X \rightarrow Y_1 \dots Y_n)$ vue précédemment, où α représente un identificateur qui peut être comme précédemment une référence à un fragment d'une grammaire à substitution d'arbre. Soit

$$v_{CF} \ : \ \begin{cases} \mathcal{R}_{CF} & \longrightarrow \mathcal{A} \\ r_{CF} & \longmapsto v_{CF}(r_{CF}) \end{cases}$$

une fonction des règles de $\mathcal{G}_{CF}$ dans un semi-anneau commutatif $\mathcal{A}$. La valeur d'une dérivation d est définie comme le produit des valeurs $v(r_{CF})$ des règles qui la composent.

On construit une grammaire hors-contexte binaire équivalente $\mathcal{G}_b = (\mathcal{T}, \mathcal{NT}_b, \mathcal{R}_b, S)$ en conservant les terminaux $\mathcal{T}$ et le symbole initial S de $\mathcal{G}_{CF}$. Les règles de $\mathcal{R}_{CF}$ sont traduites ainsi dans $\mathcal{R}_b$:

- les symboles de $\mathcal{NT}_{CF}$ sont gardés dans $\mathcal{NT}_b$;
- les règles unaires et binaires de $\mathcal{R}_{CF}$ sont gardées telles quelles dans $\mathcal{R}_b$;
- les règles d'arité supérieure sont décomposées en règles binaires en introduisant des symboles intermédiaires : une règle $(\alpha, X \rightarrow Y_1 \dots Y_n)$, avec $n > 2$ est traduite en insérant :
  - les symboles $\{Y_1Y_2\bullet, \ \dots \ , \underbrace{Y_1 \dots Y_{n-1}}_{n-1}\bullet \}$ dans $\mathcal{NT}_b$,
  - les règles $\left\{ \begin{array}{llll} (\emptyset, & Y_1Y_2\bullet & \rightarrow Y_1 & Y_2) \\ (\emptyset, & Y_1Y_2Y_3\bullet & \rightarrow Y_1Y_2\bullet & Y_3) \\ \vdots & \vdots & \vdots & \vdots \\ (\emptyset, & Y_1Y_2 \dots Y_{n-1}\bullet & \rightarrow Y_1Y_2 \dots Y_{n-2}\bullet & Y_{n-1}) \\ (\alpha, & X & \rightarrow Y_1Y_2 \dots Y_{n-1}\bullet & Y_n) \end{array} \right\}$ dans $\mathcal{R}_b$.

On peut se convaincre que l'ensemble des dérivations d'une phrase w dans $\mathcal{G}_{CF}$ est en bijection avec l'ensemble de ses dérivations dans $\mathcal{G}_b$. Pour passer d'une dérivation de $\mathcal{G}_{CF}$ à son image dans $\mathcal{G}_b$, on remplace chacun de ses éléments $(\alpha, X \rightarrow Y_1 \dots Y_n)$, par la composition $(\alpha, X \rightarrow Y_1Y_2 \dots Y_{n-1} \bullet Y_n) \ \circ \dots \circ (\emptyset, Y_1Y_2\bullet \ \rightarrow Y_1 \ Y_2)$.

CF	v_{CF}
$(\alpha, S \rightarrow$ "petit" "jean" V "grand")	$v(\alpha)$

bCF	v_{bCF}
$(\alpha, S \rightarrow$ petit_jean_V • "grand")	$v(\alpha)$
$(\emptyset,$ petit_jean_V• $\rightarrow$ petit_jean • $V)$	1
$(\emptyset,$ petit_jean• $\rightarrow$ "petit" "jean")	1

FIG. B.3 – Transformation d'une grammaire CF en une grammaire CF binaire.

On fait l'opération inverse pour trouver l'antécédent d'une dérivation dans $\mathcal{G}_b$. Cette opération est possible du fait qu'une règle $(\alpha, X \rightarrow Y_1 Y_2 \dots Y_{n-1} \bullet \quad Y_n)$ ne peut apparaître dans une dérivation de $\mathcal{G}_b$ que si elle est le premier élément de la composition des règles : $(\alpha, X \rightarrow Y_1 Y_2 \dots Y_{n-1} \bullet \quad Y_n) \quad \circ \cdots \circ (\emptyset, Y_1 Y_2 \bullet \quad \rightarrow Y_1\ Y_2)$.

En posant :

$$v_b \ : \ \begin{cases} \mathcal{R}_b & \longrightarrow \mathcal{A} \\ r_b = (\alpha, \_) & \longmapsto v_b(r_b) = (v_{CF}(\alpha) \\ r_b = (\emptyset, \_) & \longmapsto v_b(r_b) = 1 \end{cases}$$

on voit également que les dérivations en bijection auront les mêmes valeurs : à chaque règle de $\mathcal{G}_{CF}$ correspond une règle de $\mathcal{G}_b$ de même valeur dans la dérivation équivalente, et les autres règles de $\mathcal{G}_b$ y apparaissant ont pour valeur l'élément neutre pour le produit dans $\mathcal{A}$.

B.3 Notations et définitions

Posons :

- $V(d) = \prod_{\nu \in d} v(\nu)^{f_\nu(d)}$: $V(d)$ est la valeur associée à d, qui apparaît comme le produit des valeurs de ses nœuds ν.
- On appelle sous-dérivation un sous-arbre d'une dérivation. Une sous-dérivation de d dominée par un nœud ξ est le sous-arbre de d constitué de ξ et de ses descendants. Une sous-dérivation de d dominant ξ est l'arbre de d obtenu en retirant à d tous les descendants de ξ. Ces deux objets n'existent que si ξ est un nœud de d.
- L'ensemble des sous-dérivations des dérivations de $w_1 \dots w_n$ dominées par $[i, \xi, j]$ est noté $\mathcal{I}_\mathcal{D}([i, \xi, j])$.
- L'ensemble des sous-dérivations des dérivations de $w_1 \dots w_n$ dominant $[i, \xi, j]$ est noté $\mathcal{E}_\mathcal{D}([i, \xi, j])$.

–

$$\text{Interne}([i,\xi,j]) = \sum_{d \in \mathcal{I}_{\mathcal{D}}([i,\xi,j])} V(d)$$

C'est la valeur « inside » de $[i,\xi,j]$, i.e. la somme des valeurs des sous-dérivations dominées par $[i,\xi,j]$.

–

$$\text{Externe}([i,\xi,j]) = \sum_{d \in \mathcal{E}_{\mathcal{D}}([i,\xi,j])} V(d) \tag{B.6}$$

C'est la valeur « outside » de $[i,\xi,j]$, i.e. la somme des valeurs des sous-dérivations dominant $[i,\xi,j]$.

Le calcul de (B.1) se déroule en quatre temps :

1. Calcul factorisé des valeurs $\text{Interne}(\nu)$ pour tout nœud ν des dérivations de w ;
2. Calcul factorisé des valeurs $\text{Externe}(\nu)$ pour tout nœud ν des dérivations de w ;
3. Calcul pour tous les nœuds ν de :

$$\sum_{d \in \mathcal{D}} f_\nu(d) V(d) = \text{Interne}(\nu) * \text{Externe}(\nu) \tag{B.7}$$

4. Calcul pour tous ξ de :

$$\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)} \right] = \sum_{1 \le i < j \le n} \sum_{d \in \mathcal{D}(w)} f_{[i,\xi,j]}(d) \left[\prod_{\nu \in d} v(\nu)^{f_\nu(d)} \right] \tag{B.8}$$

B.3.1 Calcul des valeurs internes

On veut calculer :

$$\text{Interne}(\nu) = \sum_{d \in \mathcal{I}_{\mathcal{D}}(\nu)} V(d) \tag{B.9}$$

Dans une dérivation d de $\mathcal{I}_{\mathcal{D}}(\nu)$, les fils de ν forment un ensemble de 0, 1 ou 2 nœuds notés : $s_d(\nu) = \{\nu_1; \ldots; \nu_n\}$. Le sous-arbre de d dominé par le fils ν_i de ν est noté $d(\nu_i)$. Comme d est obtenu en adjoignant $\{d_1 \ldots d_n\}$ à ν, et la valeur associée à d étant le produit des valeurs de ses nœuds, on a :

$$V(d) = v(\nu) * \prod_{\nu' \in s_d(\nu)} V(d(\nu'))$$

Soit $\mathcal{I}_{\mathcal{D}}(\nu, \{\nu_1; \ldots; \nu_n\})$ l'ensemble des arbres de $\mathcal{I}_{\mathcal{D}}(\nu)$ dans lesquels ν a pour fils $\{\nu_1 \ldots \nu_n\}$. On peut remarquer que quels que soient les sous-arbres

dominés par $\nu_1 \dots \nu_n$ dans $\mathcal{I}_\mathcal{D}(\nu_1) \dots \mathcal{I}_\mathcal{D}(\nu_n)$, leur adjonction à ν donne un arbre de $\mathcal{I}_\mathcal{D}(\nu, \{\nu_1; \dots; \nu_n\})$.

$$\mathcal{I}_\mathcal{D}(\nu, \{\nu_1; \dots; \nu_n\}) = \bigcup_{\{d_1,\dots,d_n\} \in \mathcal{I}_\mathcal{D}(\nu_1)\times\dots\times\mathcal{I}_\mathcal{D}(\nu_n)} < \nu : d_1, \dots, d_n >$$

en notant : $< \nu : d_1, \dots, d_n >$ l'arbre obtenu par adjonction de $d_1, \dots, d_n$ à ν.

Trivialement, deux arbres de $\mathcal{I}_\mathcal{D}(\nu, \{\nu_1; \dots; \nu_n\})$ diffèrent si l'un au moins de leurs fils domine des sous-arbres différents. Donc :

$$\begin{aligned}
&\sum_{d \in \mathcal{I}_\mathcal{D}(\nu, \{\nu_1;\dots;\nu_n\})} V(d) \\
&= \sum_{d \in \bigcup_{\{d_1,\dots,d_n\} \in \mathcal{I}_\mathcal{D}(\nu_1)\times\dots\times\mathcal{I}_\mathcal{D}(\nu_n)} \{< \nu : d_1, \dots, d_n >\}} V(d) \\
&= \sum_{\{d_1,\dots,d_n\} \in \mathcal{I}_\mathcal{D}(\nu_1)\times\dots\times\mathcal{I}_\mathcal{D}(\nu_n)} v(\nu) * \prod_{i=1}^{n} V(d_i)
\end{aligned}$$

Enfin, soit $S_\nu = \bigcup_{d \in \mathcal{I}_\mathcal{D}(\nu)} \{s_d(\nu)\}$ l'ensemble des fils de ν dans les arbres de $\mathcal{D}$. On remarque que :

$$\mathcal{I}_\mathcal{D}(\nu) = \bigcup_{\{\nu_1 \dots \nu_n\} \in S_\nu} \quad \bigcup_{d_1 \in \mathcal{I}_\mathcal{D}(\nu_1), \dots, d_n \in \mathcal{I}_\mathcal{D}(\nu_n)} \{< \nu : d_1, \dots, d_n >\}$$

et que deux arbres ayant des fils de ν différents sont évidemment différents. Donc :

$$\begin{aligned}
&\sum_{d \in \mathcal{I}_\mathcal{D}(\nu)} V(d) \\
&= \sum_{(d_1,\dots,d_n) \in \bigcup_{\{\nu_1 \dots \nu_n\} \in S_\nu} \mathcal{I}_\mathcal{D}(\nu_1)\times\dots\times\mathcal{I}_\mathcal{D}(\nu_n)} v(\nu) * \prod_{i=1}^{n} V(d_i) \\
&= \sum_{\{\nu_1 \dots \nu_n\} \in S_\nu} \quad \sum_{(d_1,\dots,d_n) \in \mathcal{I}_\mathcal{D}(\nu_1)\times\dots\times\mathcal{I}_\mathcal{D}(\nu_n)} v(\nu) * \prod_{i=1}^{n} V(d_i) \\
&= v(\nu) \sum_{\{\nu_1 \dots \nu_n\} \in S_\nu} \prod_{i=1}^{n} \sum_{d_i \in \mathcal{I}_\mathcal{D}(\nu_i)} V(d_i) \\
\Rightarrow \quad &\text{Interne}(\nu) = v(\nu) \sum_{\{\nu_1 \dots \nu_n\} \in S_\nu} \prod_{i=1}^{n} \text{Interne}(\nu_i) \qquad \text{(B.10)}
\end{aligned}$$

On obtient ainsi une formule de récurrence pour calculer Interne(ν) en fonction des valeurs Interne(ν_i) de ses fils.

B.3.2 Calcul des valeurs externes

On a besoin de quelques notations supplémentaires pour expliciter ce calcul :

- Père$_d(\nu)$ désigne le père de ν dans la dérivation d ;
- Frère$_d(\nu)$ désigne le frère de ν dans la dérivation d, s'il en possède un.

On remarque que :

1. en choisissant n'importe quel arbre d_p dans $\mathcal{E}_{\mathcal{D}}(\text{Père}_d(\nu))$,
2. en y ajoutant le nœud Père$_d(\nu))$ à d_p à la position qu'il occupe dans d,
3. en y adjoignant en tant que fils de Père$_d(\nu)$ n'importe quel arbre d_f de $\mathcal{I}_{\mathcal{D}}(\text{Frère}_d(\nu))$,
4. en y adjoignant en tant que fils de Père$_d(\nu)$ n'importe quel arbre de $\mathcal{I}_{\mathcal{D}}(\nu)$,

on obtient un arbre de $\mathcal{D}$. En effet, la structure ainsi créée correspond à un arbre de la grammaire $\mathcal{G}$, dont les feuilles sont par construction les mots $w_1 \dots w_n$.

Un arbre composé des trois premiers éléments de la liste précédente est donc un arbre de $\mathcal{E}_{\mathcal{D}}(\nu)$ puisque c'est un arbre de $\mathcal{D}$ duquel on a retiré la partie dominée par ν. On le note $< d_p, pere_d(\nu), d_f >$.

Donc :

$$\bigcup_{d \in \mathcal{D}} \bigcup_{(d_p, d_f) \in \mathcal{E}_{\mathcal{D}}(\text{Père}_d(\nu)) \times \mathcal{I}_{\mathcal{D}}(\text{Frère}_d(\nu))} \{< d_p, pere_d(\nu), d_f >\} = \mathcal{E}_{\mathcal{D}}(\nu)$$

De plus, deux arbres ainsi composés diffèrent pour tous quadruplets différents $(pere_d(\nu), frere_d(\nu), d_p, d_f)$

Donc, en notant $P\grave{e}res_{\mathcal{D}}(\nu)$ l'ensemble des pères de ν dans $\mathcal{D}$,
et $Fr\grave{e}res_{\mathcal{D}}(\nu, p)$ l'ensemble des frères de ν dans $\mathcal{D}$ ayant p pour père :

$$
\begin{aligned}
& \sum_{d' \in \bigcup_{d \in \mathcal{D}} \quad \bigcup_{(d_p, d_f) \in \mathcal{E}_{\mathcal{D}}(\mathrm{Père}_d(\nu)) \times \mathcal{I}_{\mathcal{D}}(\mathrm{Frère}_d(\nu))} \{<d_p, pere_d(\nu), d_f>\}} V(d') \\
= & \sum_{d' \in \bigcup_{p \in Pères_{\mathcal{D}}(\nu)} \quad \bigcup_{f \in Frères_{\mathcal{D}}(\nu,p)} \quad \bigcup_{(d_p, d_f) \in \mathcal{E}_{\mathcal{D}}(p) \times \mathcal{I}_{\mathcal{D}}(f)} \{<d_p, p, d_f>\}} V(d') \\
= & \sum_{p \in Pères_{\mathcal{D}}(\nu)} \sum_{f \in Frères_{\mathcal{D}}(\nu,p)} \sum_{(d_p, d_f) \in \mathcal{E}_{\mathcal{D}}(p) \times \mathcal{I}_{\mathcal{D}}(f)} V(d_p) * v(p) * V(d_f) \\
= & \sum_{p \in Pères_{\mathcal{D}}(\nu)} v(p) \sum_{f \in Frères_{\mathcal{D}}(\nu,p)} \sum_{d_p \in \mathcal{E}_{\mathcal{D}}(p)} V(d_p) \sum_{d_f \in \mathcal{I}_{\mathcal{D}}(f)} V(d_f) \\
= & \sum_{p \in Pères_{\mathcal{D}}(\nu)} v(p) \sum_{f \in Frères_{\mathcal{D}}(\nu,p)} \mathrm{Externe}(p)\, \mathrm{Interne}(f)
\end{aligned}
$$

(on utilise les conventions suivantes :
- si ν apparaît comme un fils unique de p dans une dérivation, alors $Frères_{\mathcal{D}}(\nu, p)$ contient un élément nul ;
- $\mathcal{I}_{\mathcal{D}}(nul)$ contient un arbre $vide$;
- $\mathrm{Interne}(vide) = V(vide) = 1$, élément neutre pour la multiplication dans le semi-anneau considéré.

Ainsi, la valeur externe de ν est déterminée par une relation de récurrence :

$$\mathrm{Externe}(\nu) = \sum_{p \in Pères_{\mathcal{D}}(\nu)} v(p) \sum_{f \in Frères_{\mathcal{D}}(\nu,p)} \mathrm{Externe}(p)\, \mathrm{Interne}(f) \qquad \text{(B.11)}$$

où :
- $Pères_{\mathcal{D}}(\nu)$ est l'ensemble des pères de ν dans les dérivations de $\mathcal{D}$,
- $Frères_{\mathcal{D}}(\nu, p)$ est l'ensemble des frères de ν lorsque ν a pour père p dans les dérivations de $\mathcal{D}$, avec éventuellement un élément nul lorsque ν est fils unique de p, avec la convention $\mathrm{Interne}(nul) = 1$.

B.3.3 Calcul de $\sum_{d \in \mathcal{D}} f_\nu(d) V(d)$

La valeur de $f_\nu(d)$ peut être 0 ou 1, selon que ν est ou non présent dans d. Donc :

$$\sum_{d \in \mathcal{D}} f_\nu(d) V(d) = \sum_{d \in \mathcal{D} / \nu \in d} V(d)$$

Soit $\mathcal{D}(\nu) = \{d \in \mathcal{D} / \nu \in d\}$. On définit l'application c_ν de $\mathcal{D}(\nu)$ dans $\mathcal{I}_{\mathcal{D}}(\nu) \times \mathcal{E}_{\mathcal{D}}(\nu)$ qui à d fait correspondre $(d_i(\nu), d_e(\nu))$, $d_i(\nu)$ étant le sous-arbre de d dominé par ν, ν compris, et $d_e(\nu)$ étant le complément de $d_i(\nu)$ dans d.

Il apparaît que pour tout couple $(d_1, d_2) \in \mathcal{I}_\mathcal{D}(\nu) \times \mathcal{E}_\mathcal{D}(\nu)$, l'arbre d obtenu en adjoignant d_1 à d_2 à la position fils « manquante » de d_2[4] est une dérivation de w, donc appartient à $\mathcal{D}$. L'image de d par c_ν est naturellement $(d_1, d_2) : c_\nu$ est donc surjective. De plus, pour deux couples distincts $(d_1, d_2), (d'_1, d'_2)$ de $\mathcal{I}_\mathcal{D}(\nu) \times \mathcal{E}_\mathcal{D}(\nu)$, les arbres d et d' obtenus par cette opération sont différents : c_ν est injective. Finalement, c_ν est une bijection de $\mathcal{D}(\nu)$ dans $\mathcal{I}_\mathcal{D}(\nu) \times \mathcal{E}_\mathcal{D}(\nu)$.

Quant aux valeurs, on peut écrire :

$$\begin{aligned} \forall d \in \mathcal{D}(\nu)/(d_1, d_2) = c_\nu(d),\ V(d) &= \prod_{\nu' \in d} v(\nu') \\ &= \prod_{\nu' \in d_1} v(\nu') \prod_{\nu' \in d_2} v(\nu') \\ &= V(d_1)V(d_2) \end{aligned}$$

Ainsi :

$$\begin{aligned} \sum_{d \in \mathcal{D}} f_\nu(d)V(d) &= \sum_{d \in \mathcal{D}/\nu \in d} V(d) \\ &= \sum_{(d_1,d_2) \in \mathcal{I}_\mathcal{D}(\nu) \times \mathcal{E}_\mathcal{D}(\nu)} V(d) \\ &= \sum_{(d_1,d_2) \in \mathcal{I}_\mathcal{D}(\nu) \times \mathcal{E}_\mathcal{D}(\nu)} V(d_1)V(d_2) \\ &= \sum_{d_1 \in \mathcal{I}_\mathcal{D}(\nu)} V(d_1) \sum_{d_2 \in \mathcal{E}_\mathcal{D}(\nu)} V(d_2) \\ &= \text{Interne}(\nu)\,\text{Externe}(\nu) \end{aligned} \tag{B.12}$$

On calcule donc facilement cette valeur lorsque Interne(ν) et Externe(ν) sont connus.

B.3.4 Calcul de $\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)}\right]$

D'après l'équation (B.5), on a :

$$\begin{aligned} \sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)}\right] &= \sum_{1 \le i < j \le n} \sum_{d \in \mathcal{D}(w)} f_{[i,\xi,j]}(d) \left[\prod_{\nu \in d} v(\nu)^{f_\nu(d)}\right] \\ &= \sum_{1 \le i < j \le n} \sum_{d \in \mathcal{D}(w)} f_{[i,\xi,j]}(d)V(d) \end{aligned} \tag{B.13}$$

[4]Comme d_2 est obtenu en retirant un sous-arbre dominé par $\nu = [i, \xi, j]$ d'un arbre d de $\mathcal{D}$, le père de ν a des indices (i', j') englobant (i, j), et ses fils dans d_2 ont des indices extérieurs à (i, j). C'est ainsi qu'on trouve la position « manquante » dans d_2.

Donc, d'après ce qui précède :

$$\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)} \right] = \sum_{1 \leq i < j \leq n} \text{Interne}([i, \xi, j]) * \text{Externe}([i, \xi, j])$$

B.4 Algorithme

On a montré que les valeurs Interne et Externe sont faciles à calculer si les nœuds sont ordonnés, c'est-à-dire si l'on est capable d'assurer que les valeurs Interne associées aux fils sont connues avant d'effectuer le calcul sur le père, ou que la valeur Externe de tous les pères d'un nœud est connue avant de la calculer sur ce nœud.

Dans la suite, on commence donc par ordonner les nœuds dans une liste, de façon à ce que l'indice d'un nœud dans cette liste soit supérieur aux indices de ses descendants. Les calculs des valeurs Interne se fera alors en parcourant la liste par indices croissants, et celui des valeurs Externe par indices décroissants.

B.4.1 Structures de données

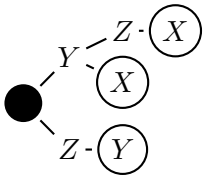
- Une **cellule** $cells[i, j]$ contient une *liste de* **nœuds-têtes**. Ce sont les nœuds des dérivations labellisés par des symboles et dominant les feuilles $w_i \ldots w_{j-1}$.
- Un **nœud-tête** est une structure $\{X, liste\_regles, \text{Interne}, \text{Externe}\}$ comprenant :
  - X : un symbole de la grammaire, terminal ou non,
  - $liste\_regles$: une *liste de* **nœuds-règles**. Ce sont les nœuds des dérivations labellisés par des règles de tête X et fils dans au moins une dérivation du nœud-tête qui le contient.
  - (Interne, Externe) : les valeurs Interne et Externe du nœud.
- Un **nœud-règle** est une structure $\{R, liste\_paires\_tetes, \text{Interne}, \text{Externe}\}$ comprenant :
  - R : une règle binaire $X \rightarrow Y\ Z$, ou une règle unaire $X \rightarrow Y$.
  - $liste\_paires\_tetes$: une *liste de paires de pointeurs*, chaque pointeur désignant un **nœud-tête**. Une paire de pointeurs (Y, Z) correspond aux deux fils de la règle dans une dérivation. L'un, ou les deux pointeurs de la paire, peuvent être nuls, selon l'arité de la règle.
  - (Interne, Externe) : les valeurs Interne et Externe du nœud.
- Le **liste** des nœuds est un tableau de pointeurs sur des *nœuds-têtes*, ordonné de façon à ce que les *descendants* d'un nœud apparaissent avec des indices *inférieurs* à celui du nœud.

B.4.2 Création de la table d'analyse

L'algorithme utilisé pour créer la table d'analyse *cells* est l'algorithme CYK. Il consiste à remplir la table en partant des cellules qui couvrent le moins de mots de la phrase, donc les cellules $(i, i+k)$ en faisant croître k. On est ainsi assuré pour toutes les règles binaires que toutes les parties droites sont déterminées avant l'examen d'une cellule, car les éléments de la partie droite d'une règle binaire couvrent moins de mots que sa tête. Un traitement spécial est nécessaire pour les règles unaires, ou filaires, pour lesquels les parties droite et gauche couvrent les mêmes mots. Il faut donc regarder à chaque ajout d'un *nœud-tête* si des règles unaires peuvent s'appliquer dans la même cellule. On est cependant sûr de la terminaison de l'algorithme du fait de la finitude de l'ensemble des symboles de tête.

Remarques :

- Il est important, dans le traitement des règles unaires, de parcourir les nœuds-têtes dans leur ordre de création, de façon à ce que les symboles ainsi ajoutés à la liste comme têtes de règles soient examinés **ultérieurement** pour déterminer s'ils sont eux-mêmes partie droite d'une règle unaire.
- Optimisation : la grammaire est stockée sous forme d'arbre de symboles, les têtes des règles étant les feuilles. Par exemple, si les règles $X \rightarrow Y\ Z$, $X \rightarrow Y$ et $Y \rightarrow Z$ sont les règles, l'arbre de codage est :



  La recherche d'une partie droite de règle se fait par un parcours de l'arbre. C'est pourquoi, dans le cas des règles binaires, on commence par chercher le nœud de l'arbre correspondant au premier symbole de partie-droite, recherche qui se fait en temps linéaire sur le nombre de symboles, puis le second, qui se fait encore en temps linéaire sur le nombre de symboles. Cela permet du reste d'éliminer d'emblée les couples de nœuds-têtes dont le premier symbole n'apparaît pas en premier symbole de partie droite de règle.

B.4.3 Création de la liste ordonnée des nœuds-têtes

L'ordonnancement des nœuds se fait par un parcours récursif de la table, en profondeur d'abord. On ajoute un nœud à la liste une fois que tous ses descendants ont été ajoutés, de façon à garantir que son indice soit supérieur à ceux de ses descendants.

Remarque : On utilise une grammaire sans boucle, i.e. pour laquelle une succession de règles unaires ne peut pas permettre de récrire un symbole

/* Initialisation */

$cells[i, i+k]$ est vide pour tout $1 \leq i \leq n$ et tout $2 \leq k \leq n-i+1$
pour tout $1 \leq i \leq n$: $cells[i, i+1] \leftarrow \{w_i, \emptyset, 0, 0\}$

/* Remplissage */

pour chaque longueur de couverture **k** allant de 2 à n
  pour chaque indice de début **i** allant de 1 à $n-k+1$
    pour chaque longueur de coupure **t** allant de 1 à $k-1$
      pour chaque nœud-tête $\mathbf{N_1} = \{\mathbf{Y}, \_, \_, \_\}$ de $cells[i, i+t]$:
        recherche des règles de $\mathcal{G}_{bin}$ de forme $(\_ \rightarrow \mathbf{Y}\ \_)$.
        s'il y en a :
          pour chaque nœud-tête $\mathbf{N_2} = \{\mathbf{Z}, \_, \_, \_\}$
          de $cells[i+t, i+k]$:
            pour chaque règle **R** de $\mathcal{G}_{bin}$ de forme $(\mathbf{X} \rightarrow Y\mathbf{Z})$:
              Recherche d'un nœud-tête $\{\mathbf{X}, liste\_regles, \_, \_\}$
              dans $cells[i, i+k]$
                s'il n'existe pas, ajout de $\{X, \emptyset, 0, 0\}$.
              Ajout de $(\&N_1, \&N_2)$ (pointeurs sur N_1 et N_2)
              à $liste\_règles$.
    parcours des nœuds-têtes $N = \{Y, \_, \_, \_\}$ de $cells[i, i+k]$
    dans l'ordre de leur création :
      pour chaque règle $X \rightarrow Y$ de $\mathcal{G}_{bin}$
        Recherche d'un nœud-tête $\{X, liste\_regles, \_, \_\}$
        dans $cells[i, i+k]$
          s'il n'existe pas, ajout de $\{X, \emptyset, 0, 0\}$.
        Ajout de $(\&N, 0)$ (pointeur sur N) à $liste\_règles$.

FIG. B.4 – Algorithme CYK de création de la table des dérivations

```
/* Initialisation */

liste_nœuds ← ∅


/* Boucle principale */

Pour chaque nœud-tête ν_t = {S, liste_règles, _, _} de cells[1, n]
    Appel de la procédure Parcours(&ν_t)


/* Parcours en profondeur de la table */

procédure Parcours(&ν_t) :
  | Si &ν_t ∉ liste_nœuds :
  |   | Soient X et liste_nœuds /ν_t = {X, liste_nœuds, _, _}
  |   | Pour chaque {R, liste_paires, _, _} de liste_nœuds :
  |   |   | Pour chaque (&ν_t1, &ν_t2) ∈ liste_paires :
  |   |   |   | Appel de la procédure Parcours(&ν_t1)
  |   |   |   | Si &ν_t2 ≠ 0 :
  |   |   |   |   | Appel de la procédure Parcours(&ν_t2)
  |   | Ajout de &ν_t à la fin de liste_nœuds.
```

FIG. B.5 – Création de la liste ordonnée des nœuds

en lui-même. Cela garantit que l'algorithme se termine : dans le parcours descendant, les règles unaires ne peuvent pas conduire à l'exécution d'une boucle infinie, car le nombre de symboles disponibles pour une cellule donnée $cells[i,j]$ est fini ; les règles binaires conduisent quant à elles à des cellules strictement plus petites que la cellule qui les contient, la « taille » d'une cellule $cells[i,j]$ étant $(j-i)$ (nombre de mots dominés par les éléments de la cellule). Chaque nœud de la table d'analyses est parcouru une seule fois dans cet algorithme.

B.4.4 Calcul des valeurs Interne(ν)

La suite est triviale. On utilise la formule (B.10) qui calcule Interne(ν) en fonction de la valeur Interne de ses fils. Dans l'algorithme, explicité par la figure B.6, $in(\nu)$ représente la troisième valeur de ν :
$\nu = \{X, \textit{liste\_règles}, in(\nu), ex(\nu)\}$.

```
/* Initialisation */

Pour chaque &ν de liste_nœuds :
    Si ν = {X, ∅, 0, 0}    (feuille dans les dérivations) :
        in(ν) ← 1

/* Parcours de la liste dans le sens enfants → parents */

Pour chaque &ν de liste_nœuds, par indice croissant :
    Soit ν = {X, liste_règles, in(ν), 0}
    Pour chaque ν_r de liste_règles
        Soit ν_r = {R, liste_paires, in(ν_r), 0}
        Pour chaque (&ν_1, &ν_2) de liste_paires :
            Si &ν_2 = 0 :
                in(ν_r) ← in(ν_r) + v(R) * in(ν_1)
            Sinon :
                in(ν_r) ← in(ν_r) + v(R) * in(ν_1) * in(ν_2)
        in(ν) ← in(ν) + in(ν_r)
```

FIG. B.6 – Calcul des valeurs internes

B.4.5 Calcul des valeurs Externe(ν)

Le calcul des valeurs Externe n'est guère plus difficile. On utilise la formule (B.11) qui calcule Externe(ν) en fonction de la valeur Externe de ses

pères et Interne de son éventuel frère. L'algorithme est donné par la figure B.7.

/* Initialisation */

On initialise à 1 la valeur externe du nœud-tête de $cells[1,n]$ correspondant au symbole initial de la grammaire :
si $\nu = [S, \textit{liste\_règles}, in(\nu), ex(\nu)] \in cells[1,n]$, faire $ex(\nu) \leftarrow 1$.

/* Parcours de la liste dans le sens *parents* $\rightarrow$ *enfants* */

Pour chaque $\&\nu$ de *liste_nœuds*, **par indice décroissant** :
- Soit $\nu = \{X, \textit{liste\_règles}, in(\nu), ex(\nu)\}$
- **Pour chaque** ν_r de *liste_règles*
  - Soit $\nu_r = \{R, \textit{liste\_paires}, in(\nu_r), ex(\nu_r)\}$
  - $ex(\nu_r) \leftarrow ex(\nu)$
  - **Pour chaque** $(\&\nu_1, \&\nu_2)$ de *liste_paires* :
    - **Si** $\&\nu_2 = 0$:
      - $ex(\nu_1) \leftarrow ex(\nu_1) + v(R) * ex(\nu)$
    - **Sinon** :
      - $ex(\nu_1) \leftarrow ex(\nu_1) + v(R) * ex(\nu) * in(\nu_2)$
      - $ex(\nu_2) \leftarrow ex(\nu_2) + v(R) * ex(\nu) * in(\nu_1)$

FIG. B.7 – Calcul des valeurs externes

B.4.6 Calcul des valeurs $\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)}\right]$

Il ne reste qu'à effectuer une somme sur les éléments de la table ayant ξ comme label en utilisant la formule (B.13). On peut calculer ces valeurs pour tous les ξ de la table avec l'algorithme de la figure B.8, dans lequel $V_w(\xi)$ finit avec la valeur : $\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)}\right]$:

/* Initialisation */

V_w : un tableau de valeurs, initialisées à 0,
indexé sur les labels des nœuds des dérivations.
/* Parcours de la liste des nœuds */

Pour chaque $\&\nu$ de *liste_nœuds* :
  Soit $\nu = \{X, \textit{liste\_règles}, in(\nu), ex(\nu)\}$
  $V_w(X) \leftarrow V_w(X) + in(\nu) * ex(\nu)$
  Pour chaque ν_r de *liste_règles*
    Soit $\nu_r = \{R, \textit{liste\_paires}, in(\nu_r), ex(\nu_r)\}$
    $V_w(R) \leftarrow V_w(R) + in(\nu_r) * ex(\nu_r)$

FIG. B.8 – Calcul des valeurs $\sum_{\delta \in \Delta(w)} f_\xi(\delta) \left[\prod_{\alpha \in \delta} v(\alpha)^{f_\alpha(d)}\right]$

Annexe C

Algorithme de décodage itératif

Nous présentons dans cette annexe la version étendue d'un article publié en 2001 [Rozenknop 01]. Elle traite de problèmes liés à l'utilisation de modèles génératifs dans des systèmes de reconnaissance de la parole. Elle apporte une réponse différente de celle présentée dans cette thèse : plutôt que de chercher à modifier les modèles, on modifie leur utilisation. L'algorithme défini permet d'extraire une interprétation d'un ensemble d'hypothèses selon le critère de la probabilité moyenne maximale ; la probabilité moyenne est la moyenne géométrique de la probabilité d'une interprétation sur le nombre de mots la constituant, ou encore sur le nombre d'étapes nécessaires pour la produire dans un processus stochastique. Cette solution algorithmique a pour but de remplacer les paramètres empiriques β destinés à favoriser l'insertion de mots dans l'utilisation combinée d'un modèle stochastique de langage et d'un modèle acoustique.

C.1 Modèles de langage valués pour la reconnaissance de la parole

C.1.1 Principe mathématique

La plupart des systèmes de reconnaissance de la parole reposent sur le principe suivant [Rabiner 93] :

- Le signal à décoder est représenté par une série de vecteurs acoustiques $O = (o_i)_{1 \leq i \leq |O|}$, les observations, contenant des coefficients calculés sur des fenêtres temporelles successives.
- Un modèle acoustique de la langue permet l'évaluation de la probabilité $P_a(O|M)$ d'obtenir une observation O sachant la séquence des mots prononcés $M = (m_i)_{1 \leq i \leq |M|}$.
- Le décodage est effectué en trouvant $M^0 = \text{Argmax}_M \, P(M|O)$, la séquence de mots la plus probablement prononcée connaissant le signal. L'évaluation directe de cette quantité étant malaisée, on utilise la for-

mule de Bayes, pour obtenir : $M^0 = \text{Argmax}_M P_a(O|M)P_l(M)$, ce qui permet l'utilisation directe du modèle acoustique, mais nécessite de définir une probabilisation *a priori* sur l'ensemble des séquences de mots M, en d'autres termes un modèle de langage probabiliste $P_l(M)$.

C.1.2 Problème de la dépendance du coût par rapport à la taille des solutions

Les modèles de langage probabilistes les plus utilisés sont des modèles génératifs reposant sur des N-grammes, dont les paramètres sont donc les probabilités $P(m_i|m_{i-N}...m_{i-1})$ de produire un mot m_i connaissant les N mots précédents. D'autres, comme les grammaires stochastiques hors-contextes, permettent de modéliser des dépendances à plus long terme entre les mots. L'intérêt de tels modèles est double : d'une part, leurs paramètres sont facilement obtensibles à partir de bases d'exemples ; d'autre part, l'existence d'algorithmes efficaces autorisent leur exploitation effective pour le décodage de signaux de parole [Meteer 93].

Ces modèles présentent hélas un inconvénient majeur : la probabilité de produire une phrase dépend fortement du nombre de mots qui la composent, et, pour être plus précis, décroît très rapidement lorsque ce nombre augmente. Or, en reconnaissance de la parole, le nombre de mots prononcés n'étant pas connu *a priori*, ces modèles ont une forte tendance à extraire comme solution les hypothèses les plus courtes !

Pour corriger ce comportement, la plupart des systèmes utilisent une version modifiée du critère à maximiser : $M^0 = \text{Argmax}_M P_a(O|M)P_l(M)/\gamma^{|M|}$, où γ est un paramètre empiriquement calculé et $|M|$ le nombre de mots de M (voir [Ogawa 98] sur ce sujet, qui propose en alternative l'utilisation de probabilités de Bernouilli). En passant à l'opposé du logarithme, on obtient : [1]

$$M^0 = \underset{M}{\text{Argmin}} - \log P_a(O|M) - \log P_l(M) + |M| \log \gamma$$

et, en posant $C(M) = -\log P_a(O|M) - \log P_l(M)$ et $\beta = -\log \gamma$, le critère s'écrit :[2]

$$M^0 = \underset{M}{\text{Argmin}}\, C(M) - \beta|M|$$

L'intérêt de cette approche, pour les modèles évoqués plus haut, est que les mêmes algorithmes efficaces peuvent extraire la phrase qui minimise ce

[1] En réalité, cette adaptation n'est pas suffisante : on utilise en plus un facteur empirique α de mise à l'échelle du modèle de langage, le critère devenant alors :

$$M^0 = \underset{M}{\text{Argmin}} -\alpha^{-1} \log P_a(O|M) - \log P_l(M) + |M| \log \gamma$$

[2] En réalité, $C(M) = -\alpha^{-1} \log P_a(O|M) - \log P_l(M)$

nouveau critère. Mais deux problèmes subsistent : (1) le paramètre β doit être déterminé empiriquement de façon à maximiser le score de reconnaissance sur un ensemble de test ; (2) le choix d'un paramètre β global n'est pas forcément optimal[3].

C.2 Critère du coût moyen

C.2.1 Idée générale

On dispose d'un modèle acoustique couplé à un modèle de langage, qui, étant donné un signal, associent à chaque phrase M du langage un coût $C(M)$, représentant l'adéquation de la phrase au signal. Le problème est que ce coût croît (à peu près linéairement) avec le nombre de mots de la phrase, ce qu'on cherche à éviter pour ne pas « avantager » les phrases les plus courtes. Une idée naturelle est alors d'utiliser comme critère de sélection la moyenne du coût sur le nombre de mots $|M|$ de la phrase, donc d'extraire la solution :

$$M^0 = \underset{M}{\text{Argmin}} \frac{C(M)}{|M|}$$

(on notera dans la suite : $\overline{C}(M) = C(M)/|M|$).

Cette idée, quoique simple dans sa formulation, n'avait pas jusqu'à présent de solution algorithmique performante : l'adaptation brutale des algorithmes habituels[4] pour minimiser ce critère multiplie par $\max|M|$ à la fois leur complexité et l'espace mémoire nécessaire à leur déroulement[5] (voir [Wilpon 89]).

La méthode proposée permet de calculer $\text{Argmin}_M \overline{C}(M)$ en utilisant itérativement l'algorithme qui calcule $\text{Argmin}_M C(M) - \beta|M|$. L'espace mémoire requis n'est donc nullement augmenté ; le nombre d'itérations nécessaires à l'obtention du résultat est le seul coût algorithmique supplémentaire, et peut être majoré par $\max|M| - |M^0|$. Sur l'expérience réalisée, le facteur de surcoût reste en pratique très faible par rapport à ce majorant.

C.2.2 Algorithme de décodage itératif

C.2.2.1 Ingrédients

On dispose des éléments suivants :

- un ensemble E de phrases appartenant à un langage L, chaque phrase M étant constituée de $|M|$ mots,

[3] Nous avons en effet constaté expérimentalement, avec un modèle de N-grammes, qu'en faisant varier β certaines phrases sont mieux reconnues, d'autres moins bien.

[4] Viterbi pour un modèle de N-grammes, parseur CYK ou Earley pour une grammaire stochastique

[5] $\max|M|$ étant la taille maximale des phrases qui peuvent être à l'origine du signal observé

- une fonction de coût C qui à chaque élément M de L associe un réel $C(M)$,
- un algorithme $\mathcal{A}(E, C, \beta)$ qui permet d'extraire $\mathrm{Argmin}_{M \in E} C(M) - \beta|M|$ pour n'importe quel réel β.

C.2.2.2 Réalisation

L'algorithme $\mathcal{I}(\mathcal{A}, E, C)$ suivant permet de trouver une solution M^0 de $\mathrm{Argmin}_{M \in E} \overline{C}(M)$, où $\overline{C} = C(M)/|M|$:

1. Initialisation : on pose[6] $\overline{C}(M_{-1}) = 0$.
2. Itérations : on calcule $M_i = \mathrm{Argmin}_{M \in E} C(M) - \overline{C}(M_{i-1}) \cdot |M|$ en utilisant l'algorithme $\mathcal{A}(E, C, \overline{C}(M_{i-1}))$[7].
3. Critère d'arrêt : on cesse les itérations lorsque $|M_i| = |M_{i-1}|$. M_i est alors une solution du problème.

C.2.2.3 Preuve

Théorème 1. L'algorithme $\mathcal{I}(\mathcal{A}, E, C)$ converge vers une solution $M^0 = \mathrm{Argmin}_{M \in E} \overline{C}(M)$ en un nombre d'itérations inférieur à $|M_1| - |M^0|$.
Lemme 1. Le coût moyen $\overline{C}(M_i)$ décroît strictement avec i, pour i supérieur à 0 et tant que $\overline{C}(M^0)$ n'a pas été atteint :

$$\forall i \geq 0 \ \left[\overline{C}(M_i) > \overline{C}(M^0) \Rightarrow \overline{C}(M_i) > \overline{C}(M_{i+1})\right]$$

Démonstration du lemme 1 :
Notons $C'_i(M) = C(M) - \overline{C}(M_i)|M|$.
Par définition de $\overline{C}$, on remarque immédiatement que :
$C'_i(M) = |M|(\overline{C}(M) - \overline{C}(M_i))$.
L'algorithme $\mathcal{A}(E, C, \overline{C}(M_i))$ trouve une solution M_{i+1} qui minimise $C'_i(M)$, d'où :

$$\begin{aligned}
& M_{i+1} = \mathcal{A}(E, C, \overline{C}(M_i)) \\
\Rightarrow \quad & M_{i+1} = \operatorname*{Argmin}_{M \in E} C'_i(M) \\
\Rightarrow \quad & C'_i(M_{i+1}) \leq C'_i(M^0) \\
\Rightarrow \quad & |M_{i+1}|(\overline{C}(M_{i+1}) - \overline{C}(M_i)) \leq |M^0|(\overline{C}(M^0) - \overline{C}(M_i)) \\
\Rightarrow \quad & \overline{C}(M_{i+1}) \leq \overline{C}(M_i) + \frac{|M^0|}{|M_{i+1}|}(\overline{C}(M^0) - \overline{C}(M_i)) \\
\Rightarrow \quad & \overline{C}(M_{i+1}) < \overline{C}(M_i)
\end{aligned}$$

[6] M_{-1} n'existant pas, ceci n'est qu'une convention d'écriture. La valeur initiale de $\overline{C}(M_{-1})$ est sans importance pour la correction de l'algorithme. Le choix du 0 est arbitraire, et de fait, si l'on a une estimation *a priori* de la valeur de l'optimum $\overline{C}(M^0)$, le choix de cette estimation pour $\overline{C}(M_{-1})$ accélérera la convergence de l'algorithme par rapport au choix de la valeur 0.

[7] i est l'indice de l'itération en cours, et vaut 0 pour la première itération.

La dernière ligne de la démonstration vient de l'hypothèse $\overline{C}(M_i) > \overline{C}(M^0)$, et de la stricte positivité de $|M|$ pour tout M appartement à E.

Lemme 2. La taille $|M_i|$ des solutions successives décroît strictement pour i supérieur à 1 et tant que $\overline{C}(M^0)$ n'a pas été atteint :

$$\forall i \geq 1 \ \left[\overline{C}(M_i) > \overline{C}(M^0) \Rightarrow |M_{i+1}| < |M_i|\right]$$

Démonstration du lemme 2 :
Pour $i \geq 1$,

$$\begin{aligned} & M_i = \mathcal{A}(E, C, \overline{C}(M_{i-1})) \\ \Rightarrow \quad & M_i = \underset{M \in E}{\text{Argmin}}\, C'_{i-1}(M) \\ \Rightarrow \quad & C'_{i-1}(M_{i+1}) \geq C'_{i-1}(M_i) \\ \Rightarrow \quad & |M_{i+1}|(\overline{C}(M_{i+1}) - \overline{C}(M_{i-1})) \geq |M_i|(\overline{C}(M_i) - \overline{C}(M_{i-1})) \end{aligned}$$

Or d'après le lemme 1, $\overline{C}(M_i) > \overline{C}(M_{i+1})$, ce qui permet de minorer le second membre de l'inégalité précédente, et d'obtenir par transitivité :

$$|M_{i+1}|(\overline{C}(M_{i+1}) - \overline{C}(M_{i-1})) \geq |M_i|(\overline{C}(M_{i+1}) - \overline{C}(M_{i-1}))$$

Toujours d'après le lemme 1, $\overline{C}(M_{i-1}) > \overline{C}(M_i)$ (car $i - 1 \geq 0$), donc $\overline{C}(M_{i-1}) > \overline{C}(M_{i+1})$. On peut alors simplifier les deux membres de l'inégalité précédente, en la renversant, ce qui donne finalement :

$$|M_{i+1}| < |M_i|$$

Démonstration du théorème 1 :
Le lemme 2 montre que la taille des solutions M_i décroît strictement pour $i \geq 1$ tant que $\overline{C}(M_i) > \overline{C}(M^0)$. Comme cette taille est un entier strictement positif, elle cesse forcément de décroître pour un certain $i = i_f$, ce qui implique que $\overline{C}(M_{i_f}) = \overline{C}(M^0)$. L'algorithme atteint donc la solution du problème en un nombre fini d'itérations, et comme $|M_i|$ décroît strictement de $i = 1$ à $i = i_f - 1$, le nombre d'itérations i_f vérifie : $i_f \leq |M_1| - |M_{i_f}| = |M_1| - |M^0|$.

C.3 Exemples d'applications

Il ne s'agit pas ici de décrire en détail des modèles de langage, ni leur mise en pratique dans le domaine de la reconnaissance de la parole, mais plutôt d'expliquer brièvement pourquoi l'algorithme de décodage itératif leur est particulièrement adapté.

Nous nous intéressons maintenant à deux modèles génératifs qui définissent une phrase comme une réalisation d'un processus stochastique. Un

tel processus consiste en une succession de tirages aléatoires d'éléments $(x)_i$ parmi un ensemble X, selon une probabilité qui ne dépend que du passé $h_i = (x_j)_{j<i}$. Les probabilités $(P(x_i|h_i))$ constituent les paramètres du modèle de langage. La probabilité d'une réalisation est donnée par $P((x)_i) = \prod_{i=1}^{N} P(x_i|h_i)$, et le «coût linguistique» d'une réalisation par l'opposé du logarithme de cette probabilité.

Dans les deux modèles exposés ci-après, on suppose qu'un module de traitement du signal a produit un treillis dont chaque arc est associé à un mot du langage avec son coût acoustique. Les différents parcours dans ce treillis représentent ainsi l'ensemble des phrases possibles parmi lesquelles le modèle de langage doit sélectionner celle de coût minimal.

C.3.1 Modèle de N-grammes de mots

Dans ce modèle, la probabilité $P(x_i|h_i)$ du i^e mot x_i dépend uniquement des N mots qui le précèdent. Les paramètres du modèle s'écrivent alors $-\log P(x_i|x_{i-N}...x_{i-1})$, et le coût d'une phrase est :
$C_l(M) = -\sum_{i\leq|M|} \log P(x_i|x_{i-N}...x_{i-1})$.

Si tant est que les paramètres du modèle sont tous du même ordre de grandeur, on voit facilement que le coût croît «à peu près linéairement» avec le nombre de mots de la phrase, car ce coût est une somme de termes positifs, d'ordres de grandeur comparables, et dont le nombre égale la taille de la phrase. On trouve donc là une première motivation pour utiliser l'algorithme de décodage itératif.

Le décodage d'un treillis de mots avec un modèle de langage de type N-grammes est efficacement réalisé à l'aide de l'algorithme de Viterbi. Pour décoder avec le critère $\min C(M) - \beta|M|$, il suffit d'utiliser les paramètres $-\log P(x_i|x_{i-N+1}...x_{i-1}) - \beta$. Cela ne modifie en rien la complexité du décodage. La connaissance d'un algorithme efficace pour résoudre ce problème est la seconde motivation pour utiliser l'algorithme itératif.

C.3.2 Grammaire stochastique hors-contexte

Ce modèle est constitué (1) d'un ensemble $\mathcal{NT}$ de symboles non-terminaux, (2) d'un ensemble $\mathcal{T}$ de symboles terminaux (les mots), (3) d'un ensemble $\mathcal{R}$ de règles de réécriture, chaque règle étant associée à une probabilité, et (4) d'un symbole de plus haut niveau $X_0 \in \mathcal{NT}$. Les règles de la grammaire sont de deux types distincts : les unes, notons $X \rightarrow Y_1...Y_n$, réécrivent un élément X de $\mathcal{NT}$ en une suite de symboles Y_i de $\mathcal{NT}$; les autres, notons-les $X \rightarrow t$ et appelons-les «règles terminales», ré-écrivent un symbole X de $\mathcal{NT}$ en un mot t de $\mathcal{T}$. En génération, une phrase est obtenue par une succession de réécritures d'une chaîne de symboles, en partant de la chaîne initiale (X_0). À chaque étape du processus, le symbole non-terminal

le plus à gauche (G) est réécrit en choisissant une des règles dont il est la tête, selon la probabilité $P(X \rightarrow \alpha | X = G)$. Si $(R)_i$ est la suite des règles utilisées pour produire une phrase à partir de (X_0), le coût de cette suite est $C_l((R)_i) = -\sum_i \log P(R_i)$, où $P(R_i)$ est la probabilité de réécrire le symbole de tête de R_i à l'aide de la règle R_i. Comme plusieurs réalisations de ce processus peuvent produire la même phrase, et pour rendre possible l'utilisation du modèle en reconnaissance de la parole, le coût d'une phrase sera le coût minimal des réalisations qui peuvent la produire.

La dépendance du coût d'une phrase sur sa taille est moins évidente avec ce modèle qu'avec le modèle de N-grammes, car le nombre d'étapes du processus qui l'engendre (le nombre de termes sous la somme) est différent du nombre de mots. Nous avons tout de même pu constater sur quelques expériences [Chappelier 99a] que le coût croît «à peu près linéairement» avec le nombre de mots de la phrase.

Le décodage d'un treillis de mots avec une SCFG peut être réalisé efficacement à l'aide d'un algorithme de type CYK. Pour décoder avec le critère $\min C(M) - \beta|M|$, il suffit d'utiliser les paramètres $\log P(X \rightarrow t) - \beta$ pour toutes les règles terminales. Cela ne modifie en rien la complexité du décodage et rend possible l'utilisation de l'algorithme itératif pour extraire le coût moyen minimal. Attention cependant : certaines alternatives à CYK ne peuvent pas être utilisées ; il s'agit des algorithmes (comme A* [Chelba 00]) qui utilisent la croissance du coût au cours du processus stochastique pour accélérer l'extraction de la solution : remplacer les paramètres $-\log P(X \rightarrow t)$ par $-\log P(X \rightarrow t) - \beta$ supprime cette décroissance et rend donc ces algorithmes inopérants.

C.3.3 Un exemple de déroulement

Afin d'illustrer l'algorithme itératif, nous présentons dans cette partie un petit exemple de décodage d'un treillis de mots à l'aide d'une grammaire stochastique. Le treillis à décoder est présenté dans la figure C.1, et les règles de la grammaire dans la figure C.2. Le treillis peut avoir trois interprétations qui apparaissent sous forme d'arbres syntaxiques dans la figure C.3, avec leurs coûts associés. On rappelle que le coût d'un arbre est la somme des coûts des règles qui le constituent, et que le coût moyen est cette même somme divisée par le nombre de feuilles de l'arbre.

Il apparaît clairement que la première interprétation, A_1, a le coût le plus bas, ce qui est dû à sa petite taille. On vérifie ici que l'algorithme itératif mène à l'extraction de la deuxième interprétation, qui est celle ayant le coût moyen le plus bas.

Les étapes de l'algorithme sont détaillées dans la figure C.4, chaque ligne y représentant une itération, avec : (1) l'indice de l'itération, (2) le coût moyen de l'interprétation extraite lors de l'itération précédente, (3) le critère à minimiser, (4,5,6) les valeurs du critère pour les différentes interprétations

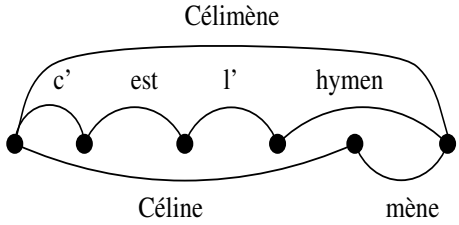


FIG. C.1 – Treillis d'hypothèses issu d'un module de reconnaissance de la parole. Les trois chemins possibles ayant des réalisations phonétiques très proches, on considère leurs probabilités acoustiques comme égales, et on n'en tient pas compte dans l'évaluation de leurs coûts.

Règle R		$P(R)$	$C(R)$	Règle R		$P(R)$	$C(R)$
R_{11}	S → NP	1/3	0,477	R_{23}	V → mène	1/2	0,301
R_{21}	S → NP V	1/3	0,477	R_{33}	V → est	1/2	0,301
R_{31}	S → P V A N	1/3	0,477	R_{35}	N → hymen	1/15	1,176
R_{12}	NP → Célimène	1/2	0,301	R_5	N → bateau	14/15	0,030
R_{22}	NP → Céline	1/2	0,301	R_4	A → un	1/2	0,301
R_{32}	P → c'	1	0	R_{34}	A → l'	1/2	0,301

FIG. C.2 – Règles de la grammaire avec leurs probabilités et leurs coûts. Le coût d'une règle vaut $-\log P(R)$.

M	A_1	A_2	A_3
Représentation graphique	S NP Célimène	P NP V Céline mène	P P V A N c' est l' hymen
Règles	R_{11}, R_{12}	R_{21}, R_{22}, R_{23}	R_{31}, R_{32}, R_{33}, R_{34}, R_{35}
Probabilité	1/6	1/12	1/180
Coût	0,778	1,079	2,255
Coût moyen par mot	0,778	0,540	0,564

FIG. C.3 – Arbres correspondant aux différents chemins du treillis de la figure C.1.

possibles, (7) l'interprétation qui minimise le critère et (8) son coût moyen.

Comme évoqué précédemment, l'algorithme d'analyse syntaxique peut extraire la solution qui minimise le critère $C'_i(M)$ en utilisant les coûts modifiés $C(R_\alpha) - \overline{C}(M_{i-1})$ à la place des coûts $C(R_\alpha)$ pour les règles terminales R_α ($\alpha \in \{12, 22, 23, 32, 33, 34, 35, 4, 5\}$).

On peut remarquer que, comme prévu par les lemmes 1 et 2, le coût moyen des solutions successives diminue à partir de $i = 0$, et que leur taille diminue à partir de $i = 1$.

Itération	$\overline{C}(M_{i-1})$	$C'_i(M)$	Valeurs de $C'_i(M)$			M_i	$\overline{C}(M_i)$
			A_1	A_2	A_3		
$i = 0$	0	$C(M)$	0,778	1,08	2,26	A_1	0,778
$i = 1$	0,778	$C(M) - 0,778\|M\|$	0	−0,477	−0,852	A_3	0,565
$i = 2$	0,565	$C(M) - 0,565\|M\|$	0,213	−0,051	0	A_2	0,540
$i = 3$	0,540	$C(M) - 0,540\|M\|$	0,238	0	0,1	A_2	0,540

FIG. C.4 – Déroulement de l'algorithme itératif

C.4 Résultat expérimental

L'algorithme présenté a été testé pour une tâche d'extraction de mots-clefs à partir d'un signal, tâche dans laquelle il s'est révélé particulièrement efficace.

Le modèle de langage considéré dans cette expérience est une chaîne de Markov cachée représentant un mot-clef. À chaque état est associée une répartition de probabilité d'émission de vecteurs acoustiques. Une «phrase» du langage est un chemin dans la chaîne, depuis son état initial jusqu'à son état final, chaque état représentant un «mot». L'alignement d'un chemin sur les vecteurs acoustiques du signal permet de définir le coût de ce chemin comme l'opposé du logarithme du produit des probabilités que chaque état émette le vecteur acoustique qui lui est associé. L'extraction du chemin de coût minimal est classiquement réalisée à l'aide d'un algorithme de Viterbi, et l'algorithme de décodage itératif est utilisé pour calculer le minimum du coût moyen par état. Si ce coût moyen passe sous un certain seuil, fixé à l'avance, le mot-clef est considéré comme présent dans le signal. Autrement, il n'est pas détecté.

Les tests ont été effectués sur le corpus BREF [Lamel 91], constitué de phrases lues au microphone. 3736 phrases ont été utilisées pour entraîner le modèle acoustique. 242 phrases, reposant sur un lexique de 2300 mots, et 100 mots-clefs choisis aléatoirement ont été utilisés pour les tests.

La courbe de détection, obtenue en faisant varier le seuil de détection, est présentée dans [Silaghi 00] ; elle traduit de bonnes performances du système[8],

[8]par exemple, on obtient 80% de détection pour moins de 4 fausses alarmes/mot-

en comparaison d'expériences similaires
[Bourlard 94], sans qu'il ait été nécessaire d'optimiser les résultats à l'aide de paramètres empiriques.

Le nombre d'itérations nécessaire pour déterminer le chemin de coût moyen minimal varie entre 3 et 5, selon les mots-clefs et les échantillons. Sachant que le nombre de vecteurs acoustiques par échantillon est en moyenne de 200 (ce qui constitue donc la borne supérieure théorique pour le nombre d'itérations nécessaire), il apparaît que l'algorithme a en pratique un comportement très efficace.

C.5 Conclusion

Nous avons décrit une classe de modèles de langage valués, utilisés pour la reconnaissance de la parole et permettant de décoder un signal en extrayant la phrase de coût minimal. Ces modèles ayant souvent la propriété de faire dépendre le coût d'une phrase de sa longueur, on en a dérivé un «meta-algorithme», aussi général que possible, qui extrait du signal une phrase de coût moyen minimal, et qui repose sur l'itération d'un algorithme spécifique au modèle de langage considéré. Sur une première expérience, on a constaté que le nombre d'itérations nécessaire avant la convergence reste très faible, même par rapport à sa valeur maximale théorique, ce qui rend ce «meta-algorithme» intéressant du point de vue de son efficacité. En revanche, la pertinence de l'utilisation du coût moyen comme critère d'extraction doit encore être évaluée pour d'autres modèles de langage stochastiques.

clef/heure de signal ; ou encore : 90 % de détection avec environ 9 fausses alarmes/mot-clef/heure

Annexe D

Extraits des bases et corpus utilisés

A titre illustratif, cette annexe contient quelques extraits des bases et des grammaires utilisées pour les expériences.

D.1 Symboles

D.1.1 Grammaires à substitution d'arbres

Le format des arbres élémentaires des grammaires d'arbres est le suivant :

$$(p(\alpha))\alpha$$

- pour les grammaires standards (STSG), $p(\alpha)$ est la probabilité de l'arbre élémentaire α ;
- pour les grammaires gibbsiennes, $p(\alpha) = \exp(\lambda_\alpha)$ est l'exponentielle du potentiel de l'arbre élémentaire α ;
- les arbres élémentaires sont donnés sous forme crochetée. Les symboles suivis d'une étoiles sont des *points de substitution* : ce sont des non-terminaux, feuilles de l'arbre élémentaire. Dans un processus de génération d'arbres, d'autres arbres élémentaires viennent se substituer à ces symboles.

D.1.2 Grammaires hors-contextes

Elles se décomposent en deux parties :

- **la partie syntaxique** : les règles de grammaires produisent des symboles non-terminaux et préterminaux, les préterminaux correspondant à des catégories morpho-syntaxiques. Un préterminal est représenté par un index entier précédé du signe « : ».

Le format d'une règle de grammaire est :

$$r(p(r))$$

où
- $p(r)$ est la probabilité de la règle r dans une SCFG,
- $p(r) = \exp(\lambda_r)$ est l'exponentielle du potentiel λ_r de la règle r dans une GCFG.

– **la partie lexicale** : les règles terminales produisent des symboles terminaux à partir de préterminaux.
Le format des règles terminales est :

$$\text{terminal}|\text{préterminal}|p(\text{préterminal} \rightarrow \text{terminal})$$

où
- $p(\text{préterminal} \rightarrow \text{terminal})$ représente la probabilité de la règle préterminal $\rightarrow$ terminal dans une SCFG,
- $p(\text{préterminal} \rightarrow \text{terminal})$ représente l'exponentielle du potentiel de la règle préterminal $\rightarrow$ terminal dans une GCFG,

D.2 Expériences sur le corpus ATIS-750

D.2.1 Base

La base ATIS-750 utilisée est une base d'arbres délexicalisés : les feuilles sont des préterminaux de la grammaire et non des mots du lexique.

Elle est donnée sous forme crochetée.

```
[ s [ :1 np_ ] [ vp [ :2 vb_ ] [ np [ :3 pp_ ] ] [ np [ :4 dt_
  ] [ :5 nn_ ] [ pp [ pp [ :6 in_ ] [ np [ :4 dt_ ] [ :7 np1_ ]
  [ :5 nn_ ] ] ] [ pp [ :8 to_ ] [ np [ :5 nn_ ] ] ] ] ] ] ]

[ s [ :1 np_ ] [ vp [ :2 vb_ ] [ np [ :3 pp_ ] ] [ np [ :4 dt_
  ] [ :5
  nn_ ] [ pp [ pp [ :6 in_ ] [ np [ :7 np1_ ] [ :7 np1_ ] [ :5
  nn_ ] ] ] [ pp [ :8 to_ ] [ np [ :5 nn_ ] ] ] ] ] ] ]

[ s [ :1 np_ ] [ vp [ :2 vb_ ] [ np [ :3 pp_ ] ] [ np [ :4 dt_
  ] [ :9 jj_ ] [ :10 nns_ ] [ pp [ pp [ :6 in_ ] [ np [ :7 np1_
  ] [ :7 np1_ ] ] ] [ pp [ :8 to_ ] [ np [ :7 np1_ ] ] ] ] [ pp
  [ :6 in_ ] [ np [ :5 nn_ ] ] ] ] ] ]

[ s [ :1 np_ ] [ vp [ :2 vb_ ] [ np [ :3 pp_ ] ] [ np [ :4 dt_
  ] [ :10 nns_ ] [ pp [ :6 in_ ] [ np [ :10 nns_ ] [ pp [ pp [
```

```
  :6 in_ ] [ np [ :7 np1_ ] ] ] [ pp [ :8 to_ ]

[ np [ :7 np1_ ] ] ] ] [ pp [ :6 in_ ] [ np [ :11 cd_ ] [ :12
  rb_ ] ] ] ] ] ] ] ]
```

D.2.2 Grammaire STSG « par têtes »

```
(0.00567654)  [ sq [ :1 * ] [ vp [ :21 vbz_ ] [ adjp * ] ] ]
(0.00141763)  [ s [ :1 * ] [ vp [ :21 vbz_ ] [ pp * ] ] ]
(0.000938086) [ s [ :1 * ] [ :8 * ] [ vp * ] [ :12 * ] [ pp *
              ] ]
(0.0234522)   [ s [ :1 * ] [ :8 * ] [ vp * ] ]
(0.00327869)  [ sq [ :1 * ] [ :21 * ] [ vp * ] ]
```

D.2.3 Grammaire GTSG « par têtes »

```
(3.3476)       [ sq [ :1 * ] [ vp [ :21 vbz_ ] [ adjp * ] ] ]
(0)            [ s [ :1 * ] [ vp [ :21 vbz_ ] [ pp * ] ] ]
(3.15064)      [ s [ :1 * ] [ :8 * ] [ vp * ] [ :12 * ] [ pp *
               ] ]
(0.987445)     [ s [ :1 * ] [ :8 * ] [ vp * ] ]
(3.26215e-36)  [ sq [ :1 * ] [ :21 * ] [ vp * ] ]
```

D.2.4 Grammaire STSG « Min-Max »

```
(0.000651324)  [ np [ :5 * ] [ :7 * ] [ :16 * ] [ :7 * ] ]
(0.000217108)  [ np [ :5 * ] [ :16 * ] [ :5 * ] ]
(0.0516717)    [ np [ :5 nn_ ] ]
(0.0256188)    [ np [ :5 nn_ ] [ :11 cd_ ] ]
(0.000217237)  [ np [ :5 nn_ ] [ :11 cd_ ] [ pp [ :6 in_ ] [ np
               [ :7 np1_ ] [ adjp [ :9 jj_ ] ] ] ] [ pp [ pp [
               :6 in_ ] [ np [ :7 np1_ ] ] ] [ pp [ :8 to_ ] [
               np [ :7 np1_ ] ] ] ] ]
```

D.2.5 Grammaire GTSG « Min-Max »

```
(0.348686)     [ np [ :5 * ] [ :7 * ] [ :16 * ] [ :7 * ] ]
(3.89543e-06)  [ np [ :5 * ] [ :16 * ] [ :5 * ] ]
(0.132497)     [ np [ :5 nn_ ] ]
(0.414084)     [ np [ :5 nn_ ] [ :11 cd_ ] ]
(0.215281)     [ np [ :5 nn_ ] [ :11 cd_ ] [ pp [ :6 in_ ] [ np
               [ :7 np1_ ] [ adjp [ :9 jj_ ] ] ] ] [ pp [ pp [
               :6 in_ ] [ np [ :7 np1_ ] ] ] [ pp [ :8 to_ ] [
               np [ :7 np1_ ] ] ] ] ]
```

D.2.6 Grammaire SCFG

D.2.6.1 Grammaire

```
s  -> sq           (0.00207469)
s  -> :1 :21 np vp (0.00207469)
sq -> :1 :21 np vp (0.00732601)
sq -> :1 :21 np    (0.010989)
sq -> :1 :21 np pp (0.003663)
```

D.2.6.2 Lexique

```
rp_  | 30 | 1
rb_  | 12 | 1
rbs_ | 29 | 1
vbz_ | 21 | 1
vbn_ | 26 | 1
```

D.2.7 Grammaire GCFG

D.2.7.1 Grammaire

```
s  -> sq           (0.00837215)
s  -> :1 :21 np vp (56.4197)
sq -> :1 :21 np vp (4.7257)
sq -> :1 :21 np    (0.00987039)
sq -> :1 :21 np pp (0.0112806)
```

D.2.7.2 Lexique

```
rp_  | 30 | 1
rb_  | 12 | 1
rbs_ | 29 | 1
vbz_ | 21 | 1
vbn_ | 26 | 1
```

D.3 Expériences sur le corpus SUZANNE

D.3.1 Base

```
[ X [ L [ Ns [ :1 A ] [ :4 good ] [ :5 man ] ] [ :20 , ] [ Nns
  [ :2 Emmett ] ] ] ]

[ X [ L [ P [ :15 In ] [ Ns [ :1 a ] [ :5 word ] ] ] [ :20 , ]
  [ Ns [ :5 plenty ] ] ] ]
```

```
[ X [ S [ :66 There ] [ Vab [ :40 are ] ] [ Np [ :36 several ]
  [ :4 possible ] [ :5 sources ] ] ] ]

[ X [ S [ :66 There ] [ Vsb [ :26 was ] ] [ Ns [ :1 a ] [ :4
  ragged ] [ :5 volley ] ] ] ]

[ X [ S [ :66 There ] [ Vsb [ :26 was ] ] [ Ns [ :56 no_one ]
  [ P [ :37 but ] [ :108 me ] ] ] ] ]
```

D.3.2 Grammaire SCFG

D.3.2.1 Grammaire

```
X -> L?+                  (0.00075358)
X -> Tb                   (0.000502386)
X -> Tg                   (0.00251193)
S -> Tg :20 Nns Vd Ns J   (0.000271739)
S -> Tg :20 Nns Vd Ns S+  (0.000271739)
```

D.3.2.2 Lexique

```
earlier | 49 | 0.038961
earlier | 14 | 0.0576923
easy    |  4 | 0.00106496
easy    | 33 | 0.00153374
eaten   | 27 | 0.00067981
```

D.3.3 Grammaire GCFG

D.3.3.1 Grammaire

```
X -> S@                   (0.0735665)
X -> L?+                  (1)
X -> Tb                   (0.00122321)
X -> Tg                   (0.131904)
S -> Tg :20 Nns Vd Ns J   (1)
S -> Tg :20 Nns Vd Ns S+  (1.08162)
```

D.3.3.2 Lexique

```
earlier | 49 | 0.296446
earlier | 14 | 2.92427
easy    |  4 | 99.4655
easy    | 33 | 0.00767591
eaten   | 27 | 1
```

Index

Index des auteurs

Bibliographie

[Abney 97a] Steven Paul Abney.
The Balancing Act., chapitre Statistical methods and linguistics.
The MIT Press, Cambridge, Massachussetts, 1997.

[Abney 97b] Steven Paul Abney.
Stochastic Attribute-Value Grammars.
Computational Linguistics, vol. 23, no. 4, pages 597–618, 1997.

[Barton 87] G.E. Barton, R. Berwick et E.S. Ristad.
Computationnal complexity and Natural Language.
A Bradford Book, The MIT press, 1987.

[Baum 72] L. E. Baum.
An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process.
Inequalities, vol. 3, pages 1–8, 1972.

[Berger 96] Adam L. Berger, Vincent J. Della Pietra et Stephen A. Della Pietra.
A Maximum Entropy Approach to Natural Language Processing.
Dans : Computational Linguistics, volume 1 de *22*, pages 39–71. The MIT Press, March 1996.

[Berger 97] Adam L. Berger.
Convexity, Maximum Likelihood and All That, 1997.

[Besançon 02] Romaric Besançon.
Intégration de connaissances syntaxiques et sémantiques dans les représentations vectorielles de textes.
PhD thesis, EPFL, 2002.

[Black 92] E. Black, F. Jelinek, John D. Lafferty et D. M. Magerman.
Towards history-based grammars : Using richer models for probabilistic parsing.
Dans : Proceedings of the DARPA Speech and Natural Language Workshop, pages 31–37, 1992.

[Bod 92] Rens Bod.
Applying Monte Carlo Techniques to Data Oriented Parsing.
Dans : Proceedings Computational Linguistics in the Netherlands, Tilburg (The Netherlands), 1992.

[Bod 93] Rens Bod.
Using an annotated corpus as a stochastic grammar.
Dans : Proceedings EACL'93, Utrecht, 1993.

[Bod 95] Rens Bod.
Enriching linguistics with Statistics : Performance Models of Natural Language.
PhD thesis, University of Amsterdam, 1995.

[Bod 96] Rens Bod et Remko Scha.
Data-Oriented Language Processing : An Overview.
Rapport technique LP-96-13, Departement of Computational Linguistics, University of Amsterdam, 1996.
cmp-lg/9611003.

[Bod 97] Rens Bod et Remko Scha.
Data-Oriented Language Processing.
Dans : S. Young et G. Bloothooft, editeurs, Corpus-based methods in language and speech processing, volume 2 de *Text, Speech and Language Technology*, pages 137–173. Kluwer Academic Publishers, Dordrecht : The Netherlands, 1997.

[Bod 98] Rens Bod.
Beyond Grammar, An Experience-Based Theory of Language.
Numeéro 88 in CSLI Lecture Notes. CSLI Publications, Standford (CA), 1998.

[Boite 00] René Boite, Hervé Bourlard, Thierry Dutoit, Joël Hancqand et Henri Leich.
Traitement de la parole.
Presses polytechniques et universitaires romandes, 2000.

[Bonnema 99] Remko Bonnema, Paul Buying et Remko Scha.
A New Probability Model for Data Oriented Parsing.
Dans : P.Dekker et G.Kerdiles, editeurs, Proceedings of the 12th Amsterdam Colloquium, Amsterdam, 1999. Institute for Logic, Language and Computation.

[Booth 73] T. L. Booth et R. A. Thompson.
Applying Probability Measures to Abstract Languages.
IEEE Transactions on Computers, vol. C-22, no. 5, pages 442–450, may 1973.

[Bourlard 94] Hervé Bourlard, B. D'Hoore et J.-M. Boite.
Optimizing recognition and rejection performance in wordspotting systems.
Dans : IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, volume I, pages 373–376, Adelaide, Australia, 1994.

[Briscoe 93] T. Briscoe et J. Carroll.
Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars.
Computational Linguisics, vol. 19, no. 1, pages 25–59, 1993.

[Brown 59] D. Brown.
A Note on Approximations to Discrete Probability Distributions.
Information and Control, vol. 2, pages 386–392, 1959.

[Carroll 92] J. Carroll et T. Briscoe.
Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars.
Dans : Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language, pages 33–38, Cambridge, MA, 1992.

[Chappelier 98] Jean-Cédric Chappelier et Martin Rajman.
A Generalized CYK Algorithm for Parsing Stochastic CFG.
Dans : TAPD'98 Workshop, pages 133–137, Paris (France), 1998.

[Chappelier 99a] Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop.
Lattice Parsing for Speech Recognition.
Dans : Proc. of 6ème conférence sur le Traitement Automatique du Langage Naturel (TALN99), pages 95–104, Cargèse (France), jul 1999.

[Chappelier 99b] Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop.
Lattice Parsing for Speech Recognition.
Dans : Proc. of 6ème conférence sur le Traitement Automatique du Langage Naturel (TALN'99), pages 95–104, July 1999.

[Chappelier 99c] Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop.
Lattice Parsing for Speech Recognition.
Dans : Proc. of 6ème conférence sur le Traitement Auto-

matique du Langage Naturel (TALN'99), pages 95–104, 12-17 juillet 1999.

[Chappelier 00] Jean-Cédric Chappelier et Martin Rajman.
Monte-Carlo Sampling for NP-Hard Maximization Problems in the Framework of Weighted Parsing.
Dans : D.N. Christodoulakis, editeur, Natural Language Processing – NLP 2000, numéro 1835 in Lecture Notes in Artificial Intelligence, pages 106–117. Springer, 2000.

[Chappelier 01a] Jean-Cédric Chappelier et Martin Rajman.
Grammaire à substitution d'arbres de complexité polynomiale : un cadre efficace pour DOP.
Dans : TALN'2001, volume 1, pages 133–142, 2001.

[Chappelier 01b] Jean-Cédric Chappelier et Martin Rajman.
Polynomial Tree Substitution Grammars : an efficient framework for Data-Oriented Parsing.
Technical report no. 01-370, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), 2001.

[Chappelier 02a] J.-C. Chappelier, M. Rajman et A. Rozenknop.
Polynomial Tree Substitution Grammars : Characterization and New Examples.
Dans : Proc. of 7th conference on Formal Grammar, 2002.

[Chappelier 02b] Jean-Cédric Chappelier, Martin Rajman et Antoine Rozenknop.
Grammaires à substitution d'arbres polynomiales : caractérisation et nouveaux exemples.
Dans : TALN'02, 9, pages 365–370, Nancy, juin 2002. ATALA.

[Charniak 93] Eugene Charniak.
Statistical Language Learning.
The MIT Press, Cambridge, Massachusetts, 1993.

[Charniak 00] Eugene Charniak.
A Maximum-Entropy-Inspired Parser.
Dans : Proceedings of the 2000 Conference of the North American Chapter of the Association for Computational Linguistics, New Brunswick NJ, 2000.

[Chelba 00] C. Chelba.
Exploiting Syntactic Structure for Natural Language Modeling.
PhD thesis, John Hopkins University, Baltimore, Maryland, 2000.

[Chomsky 57] Noam Chomsky.
Syntactic structures.
La Haye : Mouton, 1957.

[Chomsky 59] Noam Chomsky.
On certain formal properties of grammars.
Dans : Information and control, volume 2, pages 137–167, 1959.

[Collins 99] M. Collins.
Head-Driven Statistical Models for Natural Language Parsing.
PhD thesis, University of Pennsylvania, 1999.

[Collins 02] Michael Collins et Nigel Duffy.
New Ranking Algorithms for Parsing and Tagging : Kernels over Discrete Structures, and the Voted Perceptron.
Dans : ACL2002, pages 263–270, July 2002.

[Darroch 72] J.N. Darroch et D. Ratcliff.
Generalized Iterative Scaling for Log-linear Models.
Dans : Annals of Mathematical Statistics, pages 1470–1480, 1972.

[Della Pietra 97] Stephen A. Della Pietra, Vincent J. Della Pietra et John D. Lafferty.
Inducing Features of Random Fields.
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, pages 380–393, 1997.

[Dempster 77] M. M. Dempster, N. M. Laird et D. B. Jain.
Maximum likelihood from incomplete data via the EM algorithm.
Journal of the Royal Statistics Society, vol. 39, pages 1–38, 1977.

[Dörre 93] J. Dörre et M. Dorna.
Computational Aspects of Constraint-Based Language Description I, chapitre CUF, a formalism for linguistic knowledge representation, pages 3–22.
DYANA-2 Deliverable R1.2.B, 1993.

[Dörre 94] J. Dörre, M. Dorna et J. Junger.
The CUF User's Manual.
Institut für Maschinelle Schprachverarbeitung (IMS), Universität Stuttgart, Germany, Stuttgart, Germany, 1.6 edition, July 1994.

[Earley 86] J. Earley.
Readings in Natural Language Processing, chapitre "An

efficient context-free parsing algorithm", pages 25–33. Morgan Kaufmann Publishers, 1986.

[Eisele 94] A. Eisele.
Computational Aspects of Constraint-Based Language Description II, chapitre Towards probabilistic extensions of constraint-based grammars.
DYANA-2 Deliverable R1.2.B, 1994.

[Fu 75] K.-S. Fu et T. L. Booth.
Grammatical inference : Introduction and survey.
IEEE Transactions on Systems, Man, and Cybernetics, vol. 5, no. 4, pages 409–423, 1975.

[Ginsburg 66] S. Ginsburg.
The mathematical theory of Context-Free Languages.
McGraww-Hill, 1966.

[Goodman 96] Joshua T. Goodman.
Efficient Algorithms for parsing the DOP model.
Dans : Proc. of the Conf. on Empirical Methods in Natural Language Processing, pages 143–152, May 1996.

[Goodman 97] Joshua T. Goodman.
Probabilistic feature grammars.
Dans : Proceedings of the fifth International Workshop of Parsing Technologies, pages 89–100, 1997.

[Goodman 98] Joshua T. Goodman.
Parsing Inside-Out.
PhD thesis, Harvard University, Cambridge, Massachusetts, May 1998.

[Grenander 67] Ulf Grenander.
Syntax-controlled probabilities.
Rapport technique, Division of Applied Mathematics, Brown University, Providence, Rhode Island, 1967.

[Johnson 98] Mark Johnson.
PCFG Models of Linguistic Tree Representations.
Computational Linguistics, vol. 24, no. 4, pages 613–632, December 1998.

[Joshi 87] A. K. Joshi.
Mathematics of Language, chapitre An introduction to tree adjoining grammars.
John Benjamins, 1987.

[Joshi 92] A. K. Joshi et Yves Schabes.
Tree Automata and Languages, chapitre Tree-adjoining grammars and lexicalised grammar, pages 409–431.
Elsevier, New York, 1992.

[Lafferty 96] John D. Lafferty.
Gibbs-Markov models.
Dans : Computing Science and Statistics, volume 27, pages 370–377, 1996.

[Lamel 91] L.F. Lamel, J.-L. Gauvain et M. Eskénazi.
BREF, a large vocabulary spoken corpus for French.
Dans : Eurospeech'91, pages 505–508, 1991.

[Lari 90] K. Lari et S. Young.
The estimation of stochastic context-free grammars using the Inside-Outside algorithm.
Computer Speech and Language, vol. 4, pages 35–56, 1990.

[Lee 69] E. Lee et L. Zadeh.
Note on fuzzy languages.
Dans : Information Sciences, volume I, pages 421–434. 1969.

[Magerman 94] D. M. Magerman.
Natural language parsing as statistical pattern recognition.
PhD thesis, Dept. of Computer Science, Stanford University, 1994.

[Manning 97] C. D. Manning et B. Carpenter.
Probabilistic parsing using left-corner language models.
Dans : Proceedings of the Fifth International Workshop on Parsing Technologies, pages 147–158, 1997.

[Mark 91] Kevin Mark, Michael Miller, Ulf Grenander et Steve Abney.
Parameter Estimation for Constrained Context-Free Language Models.
Dans : Proceedings of the DARPA Speech and Natural Language Workshop, pages 146–149, Proceedings of the DARPA Speech and Natural Language Workshop, 1991. Morgan Kaufmann.

[Meteer 93] M. Meteer et J.R. Rohlicek.
Statistical language modeling combining N-gram and context-free grammars.
Dans : Proc.of ICASSP'93, volume 2, pages 37–40, 1993.

[Miller 90] Philip Miller et Thérèse Torris.
Formalismes syntaxiques pour le traitement automatique du langage naturel.
Hermes, 1990.

[Nederhof 94] M. Nederhof.
Linguistic parsing and program transformations.
PhD thesis, Katholieke Universiteit Nijmegen, Nijmegen, The Netherlands, 1994.

[Niemann 97] H. Niemann.
Prosodic processing and its use in Verbmobil.
Dans : Proc.ICASSP-97, Munich, Germany, Apr 21-24 1997.

[Ogawa 98] A. Ogawa, K. Takeda et F. Itakura.
Balancing acoustic and linguistic probabilities.
Dans : ICASSP98, 1998.

[op den Akker 88] R. op den Akker.
Parsing attribute grammars.
PhD thesis, Department of Computer Science, University of Twente, Enschede, The Netherlands, 1988.

[op den Akker 94] R. op den Akker et H. ter Doest.
Weakly restricted stochastic grammars.
Dans : Proceedings of the 15th International Conference on Computational Linguistics, pages 927–934, 1994.

[Pereira 80] F. C. N. Pereira et D. H. D. Warren.
Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks.
Dans : Artificial Intelligence, volume 13, pages 231–278. 1980.

[Rabiner 93] L. R. Rabiner et B. Juang.
Fundamentals of Speech Recognition.
Prentice-Hall, 1993.

[Resnik 92] P. Resnik.
Probabilistic Tree-Adjoining Grammar as a framework for statistical natural language processing.
Dans : Proceedings of the 14th International Conference on Computational Linguistics, pages 418–424, 1992.

[Riezler 96] S. Riezler.
Quantitative constraint logic programming for weighted grammar applications.
Dans : C. Retor, editeur, Lecture Notes in Computer Science, numéro 1328 in Logical Aspects of Computational Linguistics (LACL'96), pages 345–365. 1996.

[Riezler 98a] S. Riezler.
Statistical inference and probabilistic modeling for

constraint-based nlp.
Dans : W. Hess B. Schröder W. Lenders et T. Portele, editeurs, Computers, Linguistics, and Phonetics between Language and Speech, Proceedings of the 4th Conference on Natural Language Processing (KONVENS98), pages 111–124, 1998.

[Riezler 98b] S. Riezler.
Statistical inference for probabilistic constraint logic programming.
Dans : H. Wiklicky et A. di Pierro, editeurs, Probabilistic logic and randomised computation, pages 1–10. ESSLLI98 workshop, 1998.

[Rozenknop 99] Antoine Rozenknop.
Grammaire Hors-Contexte Thermodymamique pour la reconnaissance de la parole.
Rapport technique, EPFL, 1999.

[Rozenknop 00] Antoine Rozenknop et Andrzej Drygajlo.
A Prosodic/Syntactic Model for Integrating Prosody in a Continuous Speech Recognition System.
Dans : Workshop "Prosody 2000 : speech recognition and synthesis", Krakov, Pologne, Octobre 2000.

[Rozenknop 01] Antoine Rozenknop et Marius-Calin Silaghi.
Algorithme de décodage de treillis selon le critère du coût moyen pour la reconnaissance de la parole.
Dans : Actes de la 8$^{\text{ème}}$ conférence sur le Traitement Automatique des Langues Naturelles (TALN'2001), pages 391–396, Tours, juillet 2001. Association pour le Traitement Automatique des Langues.

[Rozenknop 02a] Antoine Rozenknop.
Gibbsian Context-Free Grammar for Parsing.
Dans : P. Sojka, I. Kopecek et K. Pala, editeurs, Text, Speech and Dialogue (5th International Conference, TSD 2002), pages 49–56, Brno, Czech Republic, September 2002. Springer.

[Rozenknop 02b] Antoine Rozenknop.
Une grammaire hors-contexte valuée pour l'analyse syntaxique.
Dans : 9ème Conférence Annuelle sur le Traitement Automatique des Langues Naturelles, volume 1, pages 95–104, Nancy, juin 2002. Association pour le Traitement Automatique des Langues (ATALA).

[Sampson 94] G. Sampson.
The Susanne Corpus, Release 3.
Dans : School of Cognitive & Computing Sciences, Brighton (England), 1994. University of Sussex, Falmer.

[Scha 90] Remko Scha.
Language theory and Language Technology : competence and performance.
Dans : de Kort et Leerdam, editeurs, Computertoepassingen in de Neerlandistiek. LVVN-jaarboek, Almere (The Netherlands), 1990.
in Dutch.

[Schabes 92] Yves Schabes.
Stochastic Lexicalized Tree-Adjoining Grammars.
Dans : Proc. 14th Int. Conf. of Computationnal Linguistics (COLING), pages 426–432, Nantes (France), August 1992.

[Schabes 93] Yves Schabes et R. C. Waters.
Stochastic lexicalised tree-insertion grammar.
Dans : Proceedings of the Third International Workshop on Parsing Technologies, pages 257–265, 1993.

[Schabes 94] Yves Schabes et R. C. Waters.
Tree-insertion grammar : a cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced.
Rapport technique 94-13, Mitsubishi Electric Research Labs, Cambridge MA, 1994.

[Silaghi 00] Marius-Calin Silaghi et Hervé Bourlard.
A new keyword spotting approach based on iterative dynamic programming.
Dans : ICASSP2000, Istanbul, 2000.

[Sima'an 96a] Khalil Sima'an.
Computational Complexity of Probabilistic Disambiguation by means of Tree Grammars.
Dans : Proceedings of COLING'96, volume 2, pages 1175–1180, Copenhagen (Denmark), 1996.
cmp-lg/9606019.

[Sima'an 96b] Khalil Sima'an.
Efficient disambiguation by means of stochastic tree substitution grammars.
Dans : R. Mitkov et N. Nicolov, editeurs, Recent Advances in NLP, volume 136 de *Current Issues in Linguistic Theory.* Benjamins, Amsterdam (The Netherlands), 1996.

[Sima'an 99] Khalil Sima'an.
Learning Efficient Disambiguation.
Master's thesis, Université d'Amsterdam, 1999.

[Suppes 72] P. Suppes.
Semantics for Natural Languages, chapitre Probabilistic Grammars for Natural Languages, pages 741–762.
D. Reidel Publishing Company, Dordracht-Holland, 1972.

[ter Doest 94] H. ter Doest.
Stochastic grammars : Consistency and inferences.
Master's thesis, Departement of Computer Science, University of Twente, Enschede, The Netherlands, 1994.

[Wetherell 80] C. Wetherell.
Probabilistic languages : A review and some open questions.
ACM Computing Surveys, vol. 12, no. 4, pages 361–379, 1980.

[Wilpon 89] J. G. Wilpon, L. R. Rabiner, C.-H. Lee et E. R. Goodman.
Application of Hidden Markov Models of keywords in unconstrained speech.
Dans : ICASSP'89, pages 254–257, 1989.

[Winkler 95] G. Winkler.
Image analysis, random fields and dynamic Monte-Carlo methods, a mathematical introduction.
Dans : Applications of Mathematics, volume 27. Springer, Berlin, Heidelberg, New York, 1995.

[Wright 91] J. Wright et J. Wrigley.
Generalised LR Parsing, chapitre GLR parsing with probabilities, pages 113–128.
Kluwer Academic Publishers, Dordrecht, The Nederlands, 1991.

[Zadeh 65] L. Zadeh.
Fuzzy sets.
Dans : Information and Control, volume 8, pages 338–353. 1965.

ROZENKNOP Antoine
route Du Bois, 10
1024 Ecublens
tél : 021 691 89 22
email : Antoine.Rozenknop@epfl.ch
29 ans. Français. Célibataire.

Curriculum Vitae

FORMATION

1998-2002 :	Études de doctorat à l'**Ecole Polytechnique Fédérale de Lausanne**, DI-LIA.
1996-1998 :	**Ecole Nationale Supérieure des Télécommunications**. Spécialisation en Intelligence artificielle. **DEA IARFA** simultané (Intelligence Artificielle, Reconnaissance des Formes et Applications) : • Modèles Neuromimétiques, • Systèmes multi-agents, • Traitement du langage naturel,
1994-1996 :	**Ecole Polytechnique** (France).
1993-1994 :	Service Militaire en tant qu'officier formateur à Montlhéry.
1991-1993 :	**Classes Préparatoires** au Lycée Berthollet (Annecy).
1991 :	**Baccalaureat C** (Mention TB), obtenu au Lycée Berthollet.

EXPERIENCE

1995, 1 mois	**Stage ouvrier** dans le bâtiment à St-Paul de la Réunion.
1996, 3 mois	**Stage ingénieur** au Cemagref et à l'IIMI, développement. d'un logiciel de base de données au Pakistan (VisualFoxPro).
1997, 3 mois	**Stage de recherche** au LIA de l'EPFL, sur la reconnaissance de la parole.
1998, 6 mois	**Stage de recherche** à l'Institut National des Télécommunications, sur la reconnaissance de l'écriture par des méthodes fractales.
1998-2002	**Assistant** au LIA de l'EPFL : 1999-2000 : cours de programmation (C++) ; 2000-2002 : cours de Théorie de l'Information.

COMPETENCES

langues

Anglais : 590 au TOEFL. Plusieurs séjours en pays anglophones (6 mois).
Allemand : niveau scolaire. Plusieurs séjours en Allemagne (1 mois).
Arabe classique : trois ans d'étude. Séjour linguistique au Caire.

informatique

Maîtrise des environnements Mac, Dos, Windows, Unix, Linux.
Programmation en C, C++, Pascal, Basic, Prolog, Camel, VHDL, SQL.

électronique

Réalisation (Logicielle et Matérielle) d'une carte pour la configuration de microprocesseurs.

LOISIRS

Sport	: Escrime, Ski.
Musique	: Violon, Guitare.

PUBLICATIONS

Conférences

Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop. *Lattice Parsing for Speech Recognition.* Dans : Proc. of 6ème conférence sur le Traitement Automatique du Langage Naturel (TALN99), pages 95–104, Cargèse (France), jul 1999.

Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop. *Lattice Parsing for Speech Recognition.* Dans : Proc. of 6ème conférence sur le Traitement Automatique du Langage Naturel (TALN'99), pages 95–104, July 1999.

Jean-Cédric Chappelier, Martin Rajman, Ramon Aragüés et Antoine Rozenknop. *Lattice Parsing for Speech Recognition.* Dans : Proc. of 6ème conférence sur le Traitement Automatique du Langage Naturel (TALN'99), pages 95–104, 12-17 juillet 1999.

J.-C. Chappelier, M. Rajman et A. Rozenknop. *Polynomial Tree Substitution Grammars : Characterization and New Examples.* Dans : Proc. of 7th conference on Formal Grammar, 2002.

Jean-Cédric Chappelier, Martin Rajman et Antoine Rozenknop. *Grammaires à substitution d'arbres polynomiales : caractérisation et nouveaux exemples.* Dans : TALN'02, 9, pages 365–370, Nancy, juin 2002. ATALA.

Antoine Rozenknop et Andrzej Drygajlo. *A Prosodic/Syntactic Model for Integrating Prosody in a Continuous Speech Recognition System.* Dans : Workshop "Prosody 2000 : speech recognition and synthesis", Krakov, Pologne, Octobre 2000.

Antoine Rozenknop et Marius-Calin Silaghi. *Algorithme de décodage de treillis selon le critère du coût moyen pour la reconnaissance de la parole.* Dans : Actes de la $8^{\text{ème}}$ conférence sur le Traitement Automatique des Langues Naturelles (TALN'2001), pages 391–396, Tours, juillet 2001. Association pour le Traitement Automatique des Langues.

Antoine Rozenknop. *Gibbsian Context-Free Grammar for Parsing.* Dans : P. Sojka, I. Kopecek et K. Pala, editeurs, Text, Speech and Dialogue (5th International Conference, TSD 2002), pages 49–56, Brno, Czech Republic, September 2002. Springer.

Antoine Rozenknop. *Une grammaire hors-contexte valuée pour l'analyse syntaxique.* Dans : 9ème Conférence Annuelle sur le Traitement Automatique des Langues Naturelles, volume 1, pages 95–104, Nancy, juin 2002. Association pour le Traitement Automatique des Langues (ATALA).