N° d'ordre: 2496

THÈSE

Présentée devant

Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Jacques CHODOROWSKI

Équipe d'accueil : CORDIAL
École Doctorale : Sciences Pour l'Ingénieur
Composante universitaire : ENSSAT-IRISA

Titre de la thèse :

Inférence grammaticale
pour l'apprentissage de la syntaxe
en reconnaissance de la parole et dialogue oral

soutenue le 7 juin 2001 devant la commission d'examen

M. :	Colin	DE LA HIGUERA	Président
MM. :	Marc	EL-BÈZE	Rapporteurs
	Enrique	VIDAL	
MM. :	Geraldine	DAMNATI	Examinateurs
	Christian	JACQUEMIN	
	Jacques	NICOLAS	
	Laurent	MICLET	

Toutes les pièces doivent s'assembler sans forcer.
Souvenez vous que ce que vous assemblez est ce que vous venez de démonter.
Donc si vous ne parvenez pas à le remonter il doit y avoir une raison.
En aucun cas n'utilisez de marteau.

Manuel de maintenance IBM, 1925

Remerciements

Il est étonnant de constater “au bout du chemin” le nombre de personnes impliqués dans un travail aussi personnel qu'une thèse. Sans intervention des membres de plusieurs communautés scientifiques, des responsables administratifs et ceux des infrastructures diverses, de la famille et des amis, ce travail n'aurait sûrement pas abouti. Les quelques phrases de remerciements ci-dessous sont alors bien peu pour exprimer ma reconnaissance à tous ceux qui m'ont aidé dans ce periple — ma gratitude n'en est pas moindre.

Je remercie en premier lieu mon directeur, Laurent Miclet. Son soutien infaillible notamment dans les moments critiques, sa faculté à savoir “remettre sur les rails” un thésard, sa disponibilité et son écoute ont été exeptionnels. Je ne peux que souhaiter à chaque doctorant de bénéficier d'un encadrement de qualité semblable. Les conversations scientifiques mais également celles qui sortaient du strict cadre du travail sont un bagage de compétences et de souvenirs que j'emporte avec une grande reconnaissance.

Je remercie Colin de la Higuera, qui m'a fait l'honneur de présider le jury.

Je remercie mes relecteurs, Marc El-Bèze et Enrique Vidal d'avoir contribué de façon siginificative à l'amélioration de ce travail et au final de l'avoir jugé.

Je remercie les membres du jury, Geraldine Damnati, Christian Jacquemin et Jacques Nicolas d'avoir apporté un ensemble de remarques qui ont aidé à améliorer ce document.

Je remercie les membres du Laboratoire Lannionais d'Informatique de m'avoir acceuilli au sein de leur équipe.

Je remercie le personnel de l'Ecole Nationale Supérieure de Sciences Appliquées et des Techniques de m'avoir apporté l'aide administrative et logistique au cours de ce travail.

Je remercie les membres de l'équipe DIH/IPS/RVA de France Télécom R&D Lannion d'avoir suivi ce travail et d'avoir fourni le materiel nécessaire au bon déroulement de la thèse.

Je remercie Thierry Moudenc de l'équipe DIH/IPS/VMI de France Télécom R&D Lannion pour sa compréhension : mener un travail sur deux fronts aurait été impossible sans les conditions qu'il m'a accordées.

Je remercie Franck et François pour les (longues) discussions sur les aspects logiciels relatifs à ce travail.

Je remercie Séverine pour sa patience, sa compréhension et pour sa façon de m'en-

courager avec la gaîté qui lui est innée.

Je remercie Hèlène, Laurent et Bruno pour leur bonne humeur et leur aide dans l'organisation (totalement autonome et remarquable) du "pot" de thèse.

Merci à tous ceux qui m'ont soutenu dans ce travail et qui n'ont pas été cités ici, en espérant de pouvoir leur rendre la pareille un jour...

Je réserve des remerciements particuliers à mes parents. Leur courage et l'effort qu'ils ont fourni pour me permettre de faire ce parcours sont sans commune mesure. Ce travail leur est dedié.

Table des matières

Introduction **3**

I État de l'art **5**

1 Modèles de langage **7**
1.1 Objectif des modèles de langage 7
1.1.1 Bref historique 7
1.1.2 Place des modèles de langage dans le processus de reconnaissance de la parole continue 8
1.2 Panorama succinct des modèles de langage 10
1.2.1 Modèles symboliques 10
1.2.2 Modèles statistiques 17
1.2.3 Approches mixtes 21
1.2.4 Autres approches 22
1.3 Construction d'un modèle de langage 23
1.4 Évaluation d'un modèle de langage 25
1.4.1 Évaluation des modèles probabilistes 25
1.4.2 Évaluation des modèles symboliques 26
1.5 Conclusion 27

2 Inférence Grammaticale **29**
2.1 Introduction 29
2.1.1 Bref historique 29
2.1.2 Objectifs 29
2.1.3 Applications 30
2.2 Aspects formels de l'Inférence Grammaticale 31
2.2.1 Objet de l'Inférence Grammaticale 31
2.2.2 L'espace de recherche de l'Inférence Régulière 32
2.2.3 Les critères d'identification 36
2.2.4 Complexité calculatoire du processus d'inférence 38
2.3 Évaluation de la qualité du modèle appris 39
2.3.1 Cas non-stochastique 39
2.3.2 Cas stochastique 40

2.4 Les algorithmes d'inférence grammaticale 40
2.5 Conclusion 42

3 Méthodes de correction d'erreurs pour les Séquences et Langages Formels 43
3.1 Introduction 43
3.2 Définitions et notations 46
3.2.1 L'espace des séquences 46
3.2.2 Analyse des différences entre les séquences 47
3.3 Distances entre séquences 49
3.3.1 La distance d'édition 49
3.3.2 Quelques améliorations 51
3.3.3 Comparaison d'une séquence et d'un Graphe Orienté Acyclique . 54
3.3.4 Comparaison de deux DAG 57
3.4 Distance entre une séquence et un Langage Régulier 59
3.4.1 Algorithme de Wagner 60
3.4.2 Distance pondérée entre une séquence et un Automate Fini . . . 62
3.5 Conclusion 65

II Grammaires Formelles : construction et application à la reconnaissance vocale 67

4 Algorithme ECGI 69
4.1 Motivation du choix de l'algorithme 69
4.2 ECGIA non-probabiliste 70
4.2.1 Présentation informelle 70
4.2.2 Algorithme 71
4.3 ECGIA stochastique 74
4.4 Paramétrage de l'algorithme 75
4.4.1 Évaluation expérimentale de la méthode ECGI originale 75
4.4.2 Évaluation des coûts des opérations d'édition à partir de classes grammaticales de la langue française 76
4.4.3 Calcul automatique des poids des opérations d'édition 78
4.5 Conclusion 84

5 Lissages des grammaires régulières stochastiques 85
5.1 Lissages des modèles N-grams 85
5.2 Lissage des grammaires régulières non-stochastiques 86
5.2.1 Distance entre un corpus et un modèle syntaxique. 86
5.2.2 Application 87
5.3 Lissage des grammaires régulières stochastiques 89
5.3.1 Lissage par discounting 90
5.3.2 Lissage par interpolation 91

5.4 Lissage par estimation des probabilités des opérations d'édition 91
5.4.1 Estimation des paramètres du lissage 91
5.4.2 Problème du mot inconnu 96
5.5 Évaluation expérimentale des modèles lissés 100
5.5.1 Lissage à lexique fermé 101
5.5.2 Lissage avec modélisation du mot inconnu 103
5.5.3 Grammaires Stochastiques *versus* Bigrams 107
5.6 Conclusion 109

6 Modèles de langage à base de classes 111
6.1 Utilisation des classes dans les modèles de langage 111
6.1.1 Les N-grams 112
6.1.2 Les grammaires régulières 112
6.2 Détermination des classes 117
6.2.1 Classes de la langue naturelle 117
6.2.2 Calcul automatique des classes 117
6.3 Résultats 119
6.4 Conclusion 119

7 Réduction de la taille du modèle 121
7.1 Motivation 121
7.2 Classement des séquences 123
7.2.1 Construction du tableau de distances 123
7.2.2 Classement et création des groupes 123
7.2.3 Apprentissage des "Grammaires Noyaux" 125
7.2.4 Application sur les corpus rééls 126
7.3 Conclusion 129

Conclusion 133

A Présentation des corpus d'apprentissage 139
A.1 Langage L_0 139
A.2 Corpus AGS 140
A.3 Corpus ATIS 140

B Classes grammaticales de la langue française du corpus AGS 143

Table des figures

1.1 Treillis de mots 9
1.2 Grammaire Régulière 13
1.3 Automate Fini Non-déterministe 15
1.4 Automate Fini Déterministe 15
1.5 Automate Universel 16
1.6 Grammaire Régulière Stochastique 20
1.7 Automate Fini Stochastique Non-déterministe 21

2.1 Automate Canonique Maximal 33
2.2 Arbre Accepteur des Préfixes 33
2.3 Automate dérivé du PTA 34
2.4 Espace de recherche $Lat(PTA)$ 35

3.1 Distance de Hamming entre les mots “préposition” et “prérogative” . . 44
3.2 Transformation du mot “arrivage” en mot “mirages” 45
3.3 Traces possibles entre les mots “voiture” et “véhicule” 48
3.4 Listing entre les mots “voiture” et “véhicule” 48
3.5 Trace avec des compressions et des expansions 53
3.6 Graphe Acyclique Orienté 55
3.7 Graphe Acyclique Orienté valué sur les arcs 57
3.8 Graphe d’hypothèses 58
3.9 Graphe d’analyse 63

4.1 Apprentissage par ECGIA 73
4.2 Apprentissage par ECGIA : changement d’ordre de présentation d’exemples 74
4.3 AGS : coûts à partir des classes grammaticales françaises 78
4.4 L_0, nombre d’états et de transitions créés par ECGI itératif 80
4.5 L_0, perplexité du modèle inféré par ECGI itératif 81
4.6 AGS, nombre d’états et de transitions créés par ECGI itératif 81
4.7 AGS, perplexité du modèle inféré par ECGI itératif 82
4.8 ATIS, nombre d’états et de transitions créés par ECGI itératif 82
4.9 ATIS, perplexité du modèle inféré par ECGI itératif 83

5.1 AGS — distance entre une grammaire et un corpus 88

5.2 AGS_{bis} — distance entre une grammaire et un corpus 89
5.3 Grammaire étendue $G^{'}$ 93
5.4 Grammaire lissée 98
5.5 L_0, lissage, courbes "β" et "λ" 101
5.6 AGS, lissage, courbes "β" 102
5.7 AGS, lissage, courbes "λ" 102
5.8 AGS_{bis}, lissage, courbes "β" et "λ" 103
5.9 ATIS, lissage, courbes "β" 104
5.10 AGS, lissage avec mot inconnu, courbes "β" 105
5.11 AGS, lissage avec mot inconnu, courbes "λ" 106
5.12 ATIS, lissage avec mot inconnu, courbes "β" 107
5.13 ATIS, lissage avec mot inconnu, courbes "λ" 107
5.14 L_0, perplexité du bigram *versus* perplexité de la GSR 108
5.15 AGS, perplexité du bigram *versus* perplexité de la GSR 109
5.16 ATIS, perplexité du bigram *versus* perplexité de la GSR 110

6.1 Perplexité du modèle à base de classes 119
6.2 Perplexité des modèles à base de classes 120

7.1 Dendrogramme de classification des séquences 124
7.2 AGS – distance entre les "grammaires noyaux" et un corpus 127
7.3 ATIS – distance entre les "grammaires noyaux" et un corpus 128

A.1 Grammaire du langage L_0 142

Présentation du mémoire

Introduction

L'étude de la "compréhension" du langage humain par une machine a été entamée dès l'apparition des outils (des machines éléctroniques mais également des théories et des algorithmes associés) capables de réaliser cette tâche de façon automatique — en particulier ceux conçus par Turing ([Turing, 1950]). Elle s'appuie sur des études linguistiques, antérieures à l'avènement des ordinateurs, et tente de résoudre le problème posé par les différences entre une langue naturelle, naturellement ambiguë, et un langage artificiel dont l'interprétation doit être fonctionnellement déterministe. Un nombre important de travaux, liés à la diversité de la communauté scientifique propre à ce domaine, montre l'envergure et la complexité de ce problème. Il doit être abordé sur différents niveaux, car la seule connaissance du sens des mots n'est pas suffisante pour interpréter un énoncé : il est indispensable de disposer en plus des connaissances structurelles (règles morphologiques de construction des mots, règles syntaxiques de composition des phrases), éventuellement des connaissances extra-langagières (le domaine traité, l'intention d'un locuteur, ...).

Le problème pratique posé en *reconnaissance de la parole* consiste à associer à une suite de sons (représentés par un signal acoustique) un texte composé de mots d'une langue, agencés suivant des lois (les règles de grammaire). La résolution de cette tâche passe par une succession de traitements *acoustiques* et *syntaxiques*, chacun s'appuyant sur un ensemble de connaissances représentées par des *modèles*. L'analyse acoustique seule ne suffit pas à reconnaître exactement la phrase prononcée, entre autres à cause de la variabilité de la voix humaine. Ainsi, dans un système de reconnaissance traditionnel, elle est associée à *l'analyse syntaxique*, dont le but est de déterminer la séquence acoustique la plus adéquate par rapport à un *modèle de langage*.

Dans ce document, nous nous penchons sur la construction automatique et l'évaluation de modèles syntaxiques utilisés dans le contexte de reconnaissance de la parole. Ce domaine est riche en résultats théoriques et en méthodologies. Il existe cependant une monotonie quant à leur emploi dans les applications réelles : on ne rencontre en pratique que les modèles probabilistes, à dépendance syntaxique fixe faible, et en ceci très éloignés de la syntaxe de la langue naturelle. L'objectif de notre étude est de proposer une approche alternative, reposant sur les modèles hybrides : les grammaires stochastiques régulières, qui intègrent l'information structurelle et probabiliste.

Les techniques employées au cours des travaux exposés ici font en majorité partie du domaine de l'apprentissage automatique. Certaines parties de ce manuscrit font appel à des connaissances *a priori* sur la langue française.

Le corps du document est divisé en deux grandes parties. La première, composée de trois chapitres, introduit les concepts manipulés dans la deuxième partie. Ainsi le premier chapitre parcourt l'ensemble des approches dont l'objet est la modélisation des langages formels et naturels, présente les principes de leur construction et de leur évaluation. Le deuxième chapitre est focalisé sur l'apprentissage automatique des grammaires formelles. Il présente le cadre formel et méthodologique du domaine. Finalement, le troisième chapitre décrit en détail une technique particulière — *la correction d'erreurs pour les séquences et langages formels* — qui sont à la base des propositions figurant dans la suite du document.

La deuxième partie est composée de quatre chapitres et regroupe l'ensemble des travaux réalisés par l'auteur. Le premier chapitre décrit un algorithme d'apprentissage des grammaires régulières — ECGI ([Rulot & Vidal, 1988]) — et en propose une amélioration. Le deuxième chapitre soulève le problème de l'évaluation de la qualité des grammaires stochastiques et non-stochastiques, et, par prolongement de ce thème, propose une nouvelle technique de *lissage*. Le chapitre suivant montre que l'utilisation des techniques de *clustering*[1] appliquées aux cas des grammaires régulières probabilistes s'avère efficace. Enfin, le dernier chapitre de cette partie présente une méthode originale pour apprendre les grammaires à partir du corpus des séquences préalablement classées.

Les annexes situées en fin du rapport contiennent la description des jeux de données (corpus, dictionnaire) utilisées au cours de nos travaux.

Ce travail a été effectué dans le cadre de la convention CTI 97-1B-004, signée avec France Télécom.

[1] terme anglais habituellement utilisé pour désigner les techniques de classification automatique non-supervisée.

Première partie

État de l'art

Chapitre 1

Modèles de langage

L'étude du langage naturel est usuellement divisée en quatre niveaux : lexical, syntaxique, sémantique et pragmatique [Jurafsky & Martin, 2000]. Chacun d'eux est générateur d'ambiguïtés : un même mot peut avoir plusieurs significations, par exemple le mot *ferme* peut désigner un objet, une qualité ou une action ; une phrase peut avoir plusieurs interprétations grammaticales et sémantiques, comme : "Jean voit la porte et la ferme" ; enfin des suites de phrases peuvent avoir plusieurs interprétations, comme dans : "Mon oncle construit un bateau. Il faudra que je m'en occupe." (le *en* peut s'appliquer à l'*oncle*, à l'action de *construire* ou au *bateau*). De ce fait, pour tout système de traitement automatique de la langue naturelle, la résolution satisfaisante des ambiguïtés est un des objectifs prioritaires. Pour être efficace, elle doit atteindre deux buts : produire l'analyse la plus naturelle c'est-à-dire l'interprétation préférée d'une phrase donnée, tout en limitant les ressources computationelles.

Ce chapitre met un accent particulier sur la modélisation syntaxique d'un langage : nous mentionnons les modèles couramment utilisés (en particulier ceux qui ont leur place dans une chaîne de reconnaissance de la parole), ainsi que les diverses approches pour les construire et pour les évaluer.

1.1 Objectif des modèles de langage

1.1.1 Bref historique

Durant les trente dernières années, beaucoup d'efforts ont été consacrés pour permettre aux ordinateurs de "comprendre" le langage humain, au moins à un degré limité. Deux communautés relativement indépendantes sont concernées par ces travaux : d'une part celle du Traitement Automatique de la Langue Naturelle (TALN) qui étudie essentiellement la langue écrite. Elle a surtout produit des méthodes *symboliques* qui permettent de faire de l'analyse syntaxique des textes, l'assignation du sens à une phrase, l'extraction des connaissances à partir des corpus, etc. D'autre part, la communauté de Traitement Automatique de la Langue Parlée (TALP) s'est trouvée face au constat qu'il est pratiquement impossible d'identifier correctement les mots prononcés *uniquement* à partir du signal sonore. On a donc conçu des modèles syntaxiques *locaux* : leur rôle

consiste à prédire le mot suivant en prenant en compte quelques mots déjà reconnus. Ces méthodes ont recours essentiellement aux outils *probabilistes*.

Depuis peu, on assiste à un rapprochement des deux familles. Les outils de traitement automatique des langues s'enrichissent de l'utilisation des techniques statistiques (par exemple l'association de probabilités aux règles d'une grammaire). De l'autre coté, on tend à injecter des connaissances linguistiques dans les modèles purement probabilistes (comme les classes grammaticales des mots, des connaissances *a priori* sur les règles des grammaires pour guider le décodage d'un treillis acoustique, etc.).

En pratique, les techniques et les connaissances actuelles étant insuffisantes pour modéliser la totalité d'une langue, les applications se limitent en général à des domaines où le vocabulaire, la syntaxe, la sémantique et la pragmatique (ou une combinaison des quatre) sont restreints.

1.1.2 Place des modèles de langage dans le processus de reconnaissance de la parole continue

Les systèmes de TALP travaillent sur un signal de parole non décodé, en opposition avec les systèmes de TALN qui s'occupent des énoncés linguistiques (habituellement du texte électronique). Une étape de reconnaissance[1] est donc nécessaire ; elle a longtemps constitué un point d'achoppement majeur du fait de l'extrême variabilité acoustique (intralocuteur et interlocuteur) du signal de parole. Les systèmes actuels de reconnaissance de parole reposent sur l'emploi généralisé des HMM[2]. Il s'agit de modèles stochastiques, dans lesquels les réalisations acoustiques des phonèmes et les séquences de phonèmes constituant l'énoncé sont modélisées sous forme probabiliste ([Rabiner, 1989]). Un modèle de langage est utilisé pour contraindre la reconnaissance et réduire ainsi l'espace de recherche du décodage acoustico-phonétique. Cette modélisation (également statistique dans la majorité des applications) repose généralement sur une estimation, à partir de grands corpus, des probabilités d'occurrence d'un mot connaissant ses $N-1$ prédécesseurs ; on appelle ceci le modèle "N-gram" (cf. section 1.2.2). Connaissant l'observation acoustique O correspondant au signal de parole, le processus de reconnaissance consiste alors à rechercher la séquence de mots $\widehat{w}$ (dans l'ensemble W de toutes les phrases possibles) qui maximise la probabilité *a posteriori* $p(w \mid O)$:

$$\widehat{w} = \underset{\{w \in W\}}{argmax}\ p(w|O) = \underset{\{w \in W\}}{argmax} \frac{p(w)p(O|w)}{p(O)}$$

avec :

$p(w)$ **:** probabilité d'occurrence de la séquence w via un modèle de langage N-gram ;

$p(O|w)$ **:** probabilité d'observation de O connaissant W, calculée par un modèle acoustique HMM ;

[1] le passage de l'énoncé sous forme de signal sonore vers sa forme textuelle.

[2] de l'anglais : *Hidden Markov Model* — chaînes de Markov cachées.

$p(O)$: probabilité du signal acoustique O.

La difficulté de déterminer avec exactitude les mots prononcés en se servant uniquement de la relation directe entre le signal et le modèle des mots vient notamment des homophonies ou des mots à sonorité proche (par exemple *vent* et *vend*, *boue* et *bout*, etc.). Pour désambiguïser parmi les nombreuses propositions de suites de mots que produit la partie acoustique du logiciel de reconnaissance, il est indispensable de recourir à l'information contextuelle. Ce rôle est rempli par le modèle de langage.

Usuellement, l'ensemble des hypothèses pour une phrase prononcée est (implicitement) organisé en un *treillis de mots* (cf. FIG. 1.1). Nous ne donnons ici qu'une description très schématique de sa construction : étant donné un vocabulaire associé à la tâche, chaque mot du vocabulaire se voit attribuer un score sonore sur un certain intervalle de temps. Afin de restreindre l'espace de recherche[3], les mots avec un score[4] supérieur à une valeur fixe sont considérés comme invalides et rejetés. Les mots restants sont organisés en un graphe acyclique, orienté selon l'axe du temps. La suite de mots solution (idéalement : la phrase prononcée) est un des chemins qui mènent à travers ce graphe. Le parcours est fait dans le sens croissant des unités de temps (ici : de la gauche vers la droite), le début et la fin étant le mot initial et le mot final de l'énoncé. Le modèle de langage sert à guider la recherche de ce chemin. Ce processus consiste à déterminer la suite la plus probable relativement au modèle.

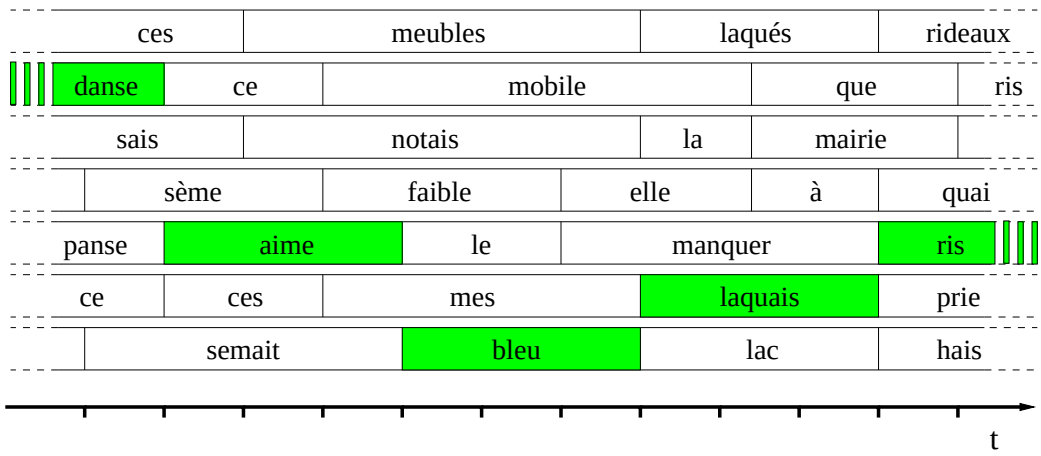


FIG. 1.1: Fragment d'un treillis de mots. Les tronçons correspondent chacun à un mot doté d'un score acoustique (omis sur la figure). Les mots sur le fond grisé, mis bout à bout, constituent une solution retournée par l'algorithme d'alignement avec un modèle de langage. Sur cet exemple artificiel, le texte suivant ([Cros, 1873]) a été "modélisé" :
Dans ces meubles laqués, rideaux et dais moroses,
Danse, aime, bleu laquais. Ris d'oser des mots roses.

[3] action nécessaire si l'on veut obtenir un temps de calcul raisonnable : le treillis augmente plus qu'exponentiellement en fonction de la taille du vocabulaire.

[4] donné en général par : $-\log p(O|W)$.

La technique la plus courante est celle de *décodage par l'algorithme de Viterbi* ([Viterbi, 1967]) qui est une variante de la programmation dynamique : le treillis est parcouru de la gauche vers la droite et à chaque intervalle temporel t_i le meilleur chemin est calculé en fonction des valeurs obtenues pour t_{i-1}, cumulé avec des valeurs estimées par le modèle à t_i.

1.2 Panorama succinct des modèles de langage

Deux tendances principales se dégagent actuellement en linguistique informatique, l'une basée sur l'utilisation de grammaires formelles étendues, l'autre sur des modèles statistiques. Cette section présente successivement les deux approches, leur formalisme et leur emploi pratique. Quelques approches hybrides, issues pour la plupart du domaine de l'apprentissage automatique, sont présentées en troisième partie.

1.2.1 Modèles symboliques

1.2.1.1 Approches algébriques

Notions de base

Les notions présentées ici font partie de la *Théorie de Langages* et sont essentielles à la définition des grammaires formelles.

La plus petite unité de la théorie de langages est un *symbole* (ou encore une *lettre*, un *élément* ou un *caractère*). La façon de le représenter peut être quelconque : une lettre de l'alphabet romain, un nombre, un dessin, etc. L'ensemble des symboles utilisés dans un problème donné forme un *alphabet*.

Définition 1 *Un* alphabet *est un ensemble* fini *de symboles*[5]. □

Un alphabet est souvent noté par Σ et sa taille (le nombre de symboles qu'il contient) par $|\Sigma|$. Les symboles d'un alphabet sont utilisés pour composer les *phrases* :

Définition 2 *Une* phrase *définie sur un alphabet est une suite* dénombrable *d'éléments de cet alphabet.* □

À la place du mot *phrase* on utilise aussi largement les termes *séquence* ou *chaîne*[6]. Ce sont des éléments d'un *langage* :

[5] remarquons que les modèles HMM attribuent une probablilité à une séquence écrite sur un alphabet qui peut être infini : les symboles sont des vecteurs dans $\mathbb{R}^p$.

[6] en *Théorie de Langages* on utilise plutôt le terme : *mot*. Nous preférons cependant l'emploi du terme *phrase*, plus adapté pour décrire la syntaxe de la langue naturelle dont le traitement automatique occupe une place centrale dans ce document. Par ailleurs, en traitement de la langue naturelle, l'alphabet Σ est le vocabulaire de l'application (et non l'alphabet de la langue écrite ou un alphabet phonétique).

Définition 3 *Un* langage *est un ensemble de phrases définies sur le même alphabet.* □

Pour décrire fonctionnellement un langage (ou pour générer des phrases qui le composent), il est nécessaire de disposer de son *modèle*. En effet une simple énumération de toutes les phrases n'est pas toujours possible (dans le cas des langages *infinis*) ni même satisfaisante d'un point de vue pratique. La description la plus employée est une *grammaire*.

Grammaires formelles

Définition 4 *Une grammaire est un quadruplet $G = (N, \Sigma, R, S)$ où :*

- *N est l'ensemble d'éléments non-terminaux.*
- *Σ est l'ensemble des symboles terminaux qui composent l'alphabet. On dit que le langage généré par la grammaire est défini sur Σ.*
- *R est un ensemble fini de règles, également appelées* productions *ou* règles de réécriture.
- *S est le symbole de départ ou "axiome" ($S \in N$). C'est à partir de ce symbole que l'on commencera la génération de mots au moyen de la grammaire.*

□

Une règle s'écrit sous la forme de deux parties séparées par une flèche. Sa signification intuitive est que la partie gauche peut être remplacée (ou "réécrite") par la partie droite de cette règle. Par exemple :

$$A \rightarrow xB$$

signifie : "A peut être réécrit en : xB" (cf. Exemple 1, page 13).

Les grammaires sont habituellement classées en quatre types, suivant les restrictions que l'on place sur la forme de leurs règles. À chaque classe de grammaires correspond un ensemble de propriétés formelles et un type de machine théorique (un *automate*) qui analyse (ou produit) le langage correspondant. Plus la grammaire est contrainte, plus l'automate est simple, plus son fonctionnement est efficace (c'est-à-dire qu'il a besoin de moins de ressources en temps et en espace mémoire pour analyser une phrase d'une longueur donnée). Les types de grammaires proposées par [Chomsky, 1957] sont les suivants :

Type 0 : Pas de restriction sur les règles. Les grammaires de ce type, appelées les *grammaires à structure de phrase*, sont équivalentes à la Machine de Turing.

Type 1 : Grammaires *sensibles au contexte*. Les règles s'écrivent

$$\alpha A \gamma \rightarrow \alpha \beta \gamma$$

avec :

$$A \in N, \alpha, \beta, \gamma \in (\Sigma \cup N)^*$$

Ici, un non-terminal A est remplacé par une suite de symboles β, s'il se trouve dans un contexte spécifique : entre α et γ.

Type 2 : Grammaires *hors-contexte*. Dans ces grammaires, toutes les productions doivent avoir la forme :

$$A \to \beta$$

où $A \in N$ et où il n'y a pas de restriction sur β. Une grammaire est donc hors-contexte si le membre gauche de chacune de ses productions est constitué d'un seul symbole non-terminal.

Type 3 : Grammaires *régulières*. Une grammaire est régulière si toutes ses productions ont l'une des deux formes :

$$A \to aB$$
$$A \to a$$

où $A, B \in N$ et $a \in \Sigma^*$. Toute production d'une grammaire régulière a donc un membre gauche constitué d'un seul non-terminal et un membre droite constitué d'un symbole terminal, éventuellement suivi d'un seul non-terminal.

Les types de grammaires énumérés plus haut sont liés par une relation d'inclusion stricte :

$$\text{Type 3} \subset \text{Type 2} \subset \text{Type 1} \subset \text{Type 0}$$

Par exemple, toute grammaire régulière est aussi une grammaire hors-contexte ; par conséquent les familles de langages correspondants sont également inclus les uns dans les autres : un langage engendré par une grammaire de type 3 peut l'être également par une grammaire de type 2, mais l'inverse n'est pas toujours vrai.

Le problème de l'*analyse syntaxique d'une phrase* (celui de savoir si une séquence quelconque appartient ou non au langage de la grammaire) n'a pas le même degré de complexité selon la classe de grammaire considérée. Pour les grammaires de type 3, les ressources en temps nécessaires sont proportionnelles à la longueur de la séquence (complexité linéaire). Pour les grammaires de type 2, les ressources en temps nécessaires sont, dans le pire des cas, proportionnelles à la longueur de la séquence élevée à une certaine puissance p (complexité polynômiale — en pratique $p = 3$ ou $p = 2$: voir [Aho & Ullman, 1972]). Pour les grammaires de type 1 la complexité est une fonction exponentielle de la longueur de la séquence, ce qui range le problème de l'analyse de ces grammaires dans la catégorie des problèmes dits NP-difficiles.

La Fig. 1.2 de l'exemple qui suit montre une grammaire de type 3 (grammaire régulière) qui génère tous les nombres impairs, supérieurs ou égaux à 5 et représentés en base 2 :

Exemple 1 :

$G = (N, \Sigma, R, S)$ avec :

$N = \{S, A, B\}$

$\Sigma = \{0, 1\}$

$$R = \left\{ \begin{array}{l} S \to 1A \\ A \to 0A \\ A \to 1A \\ A \to 0B \\ A \to 1B \\ B \to 1 \end{array} \right\}$$

FIG. 1.2: Une Grammaire Régulière.

Selon cette grammaire la chaîne 1011 est générée par :

$$S \to 1A \to 10A \to 101B \to 1011$$

□

Chaque type de grammaire décrit un type correspondant de langage. En particulier les *langages réguliers*, décrits par les grammaires de type 3, possèdent un riche cadre algébrique. La classe des langages réguliers est fermée par les opérations d'*union*, de *concaténation* et de *fermeture itérative*[7] et de la définition d'un *complément*. L'autre avantage de cette classe vient de la variété des descriptions de ses éléments : une grammaire régulière peut aussi être définie par une expression régulière ou par un automate fini.

Automates finis

Intuitivement, un automate est une machine qui, étant donnée une phrase, la reconnaît (ou : *l'accepte*) comme appartenant ou non au langage qu'il décrit. Nos travaux étant focalisés sur l'apprentissage des automates finis, nous donnons leur définition ci-dessous :

Définition 5 *Un Automate Fini Déterministe (AFD) est défini par un quintuplet* $M = (Q, \Sigma, \delta, q_0, F)$ *où :*

- Q *est un ensemble fini d'états ;*
- Σ *est un alphabet ;*

[7] appelée également *Fermeture de Kleene* et notée L^* : c'est l'ensemble des séquences formées par la concaténation de p séquences de L, avec $p \in \mathbb{N}$.

- $\delta : Q \times \Sigma \to Q$ *est une fonction de transition qui indique pour chaque état et symbole lu quel est l'état suivant (au plus un) de l'automate ;*
- $q_0 \in Q$ *est l'état initial ;*
- $F \subseteq Q$ *est l'ensemble d'états accepteurs.*

□

Remarque 1 *Si* $\forall q \in Q, \forall a \in \Sigma, \delta(q,a)$ *contient exactement un élément, l'automate est dit* complet *ou* complètement spécifié. □

Définition 6 *Un Automate Fini Non-déterministe (AFN) est défini par un quintuplet* $M = (Q, \Sigma, \Delta, q_0, F)$ *où :*

- Q, Σ, q_0, F *ont la même signification que pour un AFD*
- $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$ *($\mathcal{P}(Q)$ dénote l'ensemble des parties de Q) est la relation de transition qui indique pour chaque état et symbole lu quels sont les états suivants (visibles) de l'automate.*

□

On montre qu'un langage représenté par un AFN peut l'être également par un AFD :

Théorème 1 ([Aho & Ullman, 1972]) *Pour tout automate fini non-déterministe, il est possible de construire un automate fini déterministe équivalent (i.e. qui reconnaît le même langage) .* □

La notion d'*acceptation* d'une phrase par un automate est définie comme suit :

Définition 7 *Une* configuration *d'un automate déterministe pour une chaîne* $a_1 \ldots a_n$ *finie est un couple* $(q^i, w_i) \in Q \times \Sigma^*$ *où* q^i *est l'état atteint après l'acceptation du préfixe*[8] $a_1 \ldots a_i$ *et* w_i *est le suffixe* $a_{i+1} \ldots a_n$. □

Définition 8 *L'*exécution *(en anglais :* parsing*) d'un automate déterministe sur une chaîne* $a_1 \ldots a_n$ *est la suite de configurations :*

$$(q^0, a_1 \ldots a_n) \vdash (q^1, a_2 \ldots a_n) \vdash \ldots \vdash (q^{n-1}, a_n) \vdash (q^n, \epsilon).$$

On note en raccourci : $(q^0, a_1 \ldots a_n) \vdash^* (q^n, \epsilon)$. □

Définition 9 *On dit qu'un automate* accepte *une chaîne* $a_1 \ldots a_n$ *si* $(q^0, a_1 \ldots a_n) \vdash^* (q, \epsilon)$ *et* $q \in F$*, autrement dit s'il existe une suite d'états* $(q^0, \ldots, q^n)$ *(éventuellement non unique dans le cas d'un automate non-déterministe) telle que* $q^0 = q_0$, $q \in F$ *et* $q^{i+1} = \delta(q^i, a_{i+1})$ *dans le cas d'un AFD, ou* $q^{j+1} \in \Delta(q^i, a_{i+1})$ *dans le cas d'un AFN.* □

[8] cette notion, ainsi que celle de *suffixe*, intuitive ici, est définie formellement dans la Définition 27, page 46.

Le langage accepté par un automate fini A est donc défini par :

$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : (q_0, w) \vdash^* (q, \epsilon)\}$$

Définition 10 *On dit qu'un automate A* accepte exactement *un langage L si A accepte L et n'accepte aucune chaîne qui n'appartient pas à L.* □

Exemple 2 :

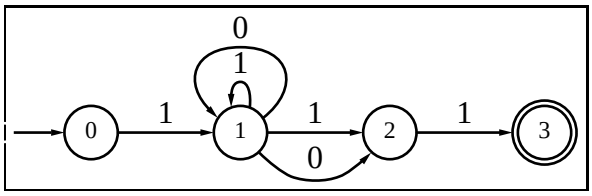


FIG. 1.3: Un Automate Fini Non-déterministe. L'état initial est représenté par un cercle pointé par une flèche n'ayant pour origine aucun état, l'état final est représenté par un double cercle.

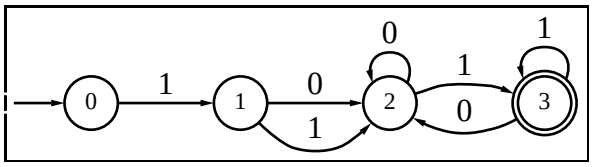


FIG. 1.4: Un Automate Fini Déterministe.

L'automate de la FIG. 1.3 est non-déterministe. Il correspond à la grammaire de la FIG. 1.2. L'exécution de cet automate pour accepter la chaîne 1011 est :

$$(q_{0I}, 1011) \vdash (q_1, 011) \vdash (q_1, 11) \vdash (q_2, 1) \vdash (q_{3F}, \epsilon)$$

L'AFD équivalent (obtenu par les opérations de *minimisation* et *détérminisation*, voir par exemple [Wolper, 1991]) est représenté sur la FIG. 1.4. La même séquence est alors acceptée par :

$$(q_{0I}, 1011) \vdash (q_1, 011) \vdash (q_2, 11) \vdash (q_{3F}, 1) \vdash (q_{3F}, \epsilon)$$

(ici on voit que c'est en complexité linéaire). □

Un automate est dit *universel* lorsqu'il accepte le *langage universel* Σ^* (cf. FIG. 1.5).

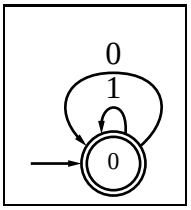


FIG. 1.5: L'Automate Universel (AU) sur l'alphabet {0,1}.

Théorème 2 ([Aho & Ullman, 1972]) *L'automate déterministe possédant le moins d'états et acceptant le langage L est appelé* automate canonique *(ou encore* automate déterministe minimal*) du langage L et noté $A(L)$. Cet automate est unique (à la numérotation des états près) pour un langage donné.* □

Dans la suite de ce mémoire, nous aborderons souvent le problème qui consiste à *reconnaître*[9] des phrases d'un langage, c'est-à-dire distinguer parmi tous les mots construits sur un alphabet donné ceux qui font partie du langage et ceux qui n'en font pas partie. Les premiers seront appellés des *exemples positifs*, les seconds des *exemples négatifs* (ou encore : des *instances positives* et des *instances négatives*).

1.2.1.2 Approches linguistiques

Dans l'informatique linguistique, la tendance dominante est celle d'une lexicalisation de plus en plus marquée de la grammaire, grâce en particulier à la généralisation du formalisme des grammaires d'unification de traits. Initiée par l'apparition des grammaires lexicales-fonctionnelles (LFG[10], voir [Kaplan & Bresnan, 1982]) puis par celle des grammaires syntagmatiques (GPSG[11] [Gazdar et al., 1985], puis HPSG[12] [Pollard & Sag, 1987]), cette évolution est désormais portée par les grammaires d'arbres adjoints (TAG[13] [Abeillé, 1988; Shabes, 1990]). Aucun des modèles cités plus haut n'est équivalent à une grammaire régulière. Les modèles GPSG et HPSG sont équivalents à une grammaire hors contexte, les modèles LFG et TAG sont équivalents à une grammaire contextuelle [Abeillé, 1993]. La complexité calculatoire de l'analyse que nécessitent ces modèles (cf. paragraphe 1.2.1.1) fait qu'ils sont rarement utilisés dans les applications de traitement de la parole.

[9] à ne pas confondre avec le terme *identifier*, qui dans la terminologie de l'Apprentissage Automatique possède un sens distinct : *apprendre exactement*.

[10] de l'anglais : *Lexical Functional Grammar*.

[11] de l'anglais : *Generalized Phrase Structure Grammar* — grammaire syntagmatique généralisée.

[12] de l'anglais : *Head-driven Phrase Structure Grammar* — grammaire syntagmatique guidée par les têtes.

[13] de l'anglais : *Tree Adjoining Grammar*.

1.2.2 Modèles statistiques

Le rôle d'un modèle statistique est d'attribuer une probabilité d'occurrence à une phrase donnée. Cette probabilité est évaluée par une *valeur de vraisemblance*, calculée grâce au modèle. L'estimation de ses paramètres (probabilités) est en général obtenue à partir des fréquences d'apparition des mots dans un corpus de référence.

1.2.2.1 Le modèle courant : les N-grams

Le modèle statistique le plus répandu est le modèle N-gram [Jelinek, 1990]. Il repose sur l'hypothèse que la probabilité d'un symbole w_i dans une phrase $W = w_1 w_2 \ldots w_M$ dépend uniquement des $N-1$ mots[14] qui le précèdent : $p(w_i) = p(w_i|w_{i-N+1}w_{i-N+2} \ldots w_{i-1})$. Ainsi la probabilité de W (de longueur M) est donnée par :

$$p(W) \simeq p(w_1)p(w_2|w_1)\ldots p(w_{N-1}|w_1 w_2 \ldots w_{N-2}) \prod_{i=N}^{M} p(w_i|w_{i-N+1}, \ldots, w_{i-1})$$

La probabilité d'occurrence du symbole w_i connaissant son contexte $w_{i-N+1}\ldots w_{i-1}$ est estimée par :

$$p_{N-gram}(w_i|w_{i-N+1}\ldots w_{i-1}) = \frac{c(w_{i-N+1}\ldots w_{i-1}w_i)}{\sum_{v\in\Sigma} c(w_{i-N+1}\ldots w_{i-1}v)}$$

où $c(w_{i-N+1}\ldots w_{i-1}v)$ est le nombre d'occurrences du N-gram $w_{i-N+1}\ldots w_{i-1}v$ dans le corpus.

Théoriquement, e nombre de paramètres à estimer dans un modèle N-gram conçu sur un alphabet Σ est $|\Sigma|^N$. Afin de limiter ce nombre, on fixe en général N à 2 (modèle dit *bigram*) ou à 3 (modèle dit *trigram*). Cette limitation s'avère cependant insuffisante dans le cas des corpus réels ; Jelinek ([Jelinek, 1985]) montre qu'un modèle trigram estimé sur un corpus de la langue anglaise de 1 500 000 mots couvre (sous-entendu : *attribue une probabilité non nulle à*) seulement 25% des trigrams[15] dans un corpus indépendant de 300 000 mots. Ceci est inacceptable, aussi bien de point de vue théorique (les phrases syntaxiquement correctes sont rejetées) que pratique (impossible d'évaluer un tel modèle par les techniques courantes : cf. section 1.4). Une solution à ce problème est le *lissage* des probabilités du modèle N-gram.

Cette technique consiste à attribuer une certaine probabilité à des événements qui n'ont pas été vus dans le corpus de référence. Parmi les méthodes de lissage qui ont été proposées, citons l'*interpolation* ([Jelinek, 1990]) où l'on utilise des probabilités pondérées des N-grams d'ordre inférieur ; par exemple, pour les trigrams la probabilité d'un mot devient :

$$p^{'}(w_i|w_{i-2}w_{i-1}) = \lambda_1 p(w_i) + \lambda_2 p(w_i|w_{i-1}) + \lambda_3 p(w_i|w_{i-2}w_{i-1})$$

[14] compte tenu du contexte linguistique de ces travaux, le terme "mot" est employé ici au même titre que le terme "symbole".

[15] par abus de langage, "trigram" signifie ici également un triplet particulier, composé des mots du lexique.

où λ_1, λ_2, λ_3 sont des poids attribués respectivement à unigram (N=1), bigram et trigram tels que : $\lambda_1 + \lambda_2 + \lambda_3 = 1$ et $\lambda_{i\in[1...3]} \geq 0$.

D'autres techniques de lissage consistent à redistribuer une certaine "masse" de probabilités sur les N-grams non-vus (*Backing-Off* [Katz, 1997; Jelinek, 1985], *Absolute Discounting* [Ney & Essen, 1993], etc.). L'étude de ces techniques est approfondie au Chapitre 5.

1.2.2.2 Modèles N-multigram

Dans le modèle N-gram, le mécanisme de redondance qui organise l'agencement des mots en phrases est supposé instaurer des dépendances qui s'exercent systématiquement sur tout N-uplet de mots consécutifs. En réalité, les dépendances entre les mots peuvent être de longueur variable, inférieures ou supérieures à N. Par conséquent, les hypothèses N-gram conduisent parfois à fonder la prédiction d'un mot sur des dépendances non pertinentes, parfois à ne pas exploiter toute la redondance disponible. La modélisation du langage par les processus N-multigram ([Bimbot et al., 1994; Bimbot et al., 1995]) consiste à considérer que les phrases sont formées de syntagmes de mots, dont la longueur est comprise entre 1 et N. Contrairement donc aux hypothèses N-grams, la longueur du contexte conditionnant chaque mot n'est pas figée : elle peut varier entre 0 et $N-1$. Par exemple la probabilité de la suite des quatre symboles $a\ b\ c\ d$ évaluée avec un modèle 3-multigram est :

$$p_{3-\mu gram}(abcd) = \sum \begin{cases} p(a)p(bcd) \\ p(abc)p(d) \\ p(ab)p(cd) \\ p(ab)p(c)p(d) \\ p(a)p(bc)p(d) \\ p(a)p(b)p(cd) \\ p(a)p(b)p(c)p(d) \end{cases}$$

tandis que celle évaluée avec le modèle 3-gram est :

$$p_{3-gram}(abcd) = p(a)p(b|a)p(c|ab)p(d|bc)$$

Une étude de l'application des multigrams dans le traitement du langage écrit et parlé peut être consultée dans [Deligne, 1996].

1.2.2.3 Grammaires probabilistes

C'est une approche hybride à la modélisation syntaxique, mélangeant les aspects symboliques (règles de réécriture) et stochastiques (probabilités de déclenchement associées à ces règles). Plus généralement, chaque transition de la grammaire (grammaire lexicalisée, automate à états finis, ...) se voit associer une probabilité conditionnelle d'application compte tenu du contexte d'analyse. Les grammaires probabilistes fournissent en sortie un ou plusieurs arbres de dérivation classés par probabilités décroissantes. Ces probabilités globales correspondent au produit des probabilités

de transitions correspondant à chaque dérivation, suivant l'hypothèse d'indépendance conditionnelle inhérente aux modélisations markoviennes.

Les grammaires stochastiques, qui seront un des points de nos études expérimentales dans le Chapitre 4 et les suivants, sont un bon compromis entre la richesse de représentation et le surcoût calculatoire occasionné par l'adjonction des probabilités. Nous nous intéressons plus particulièrement aux *grammaires stochastiques* régulières que nous définissons comme suit ([Fu, 1970]) :

Définition 11 *Une Grammaire Stochastique Régulière (SRG*[16]*) G est un quadruplet (N, Σ, R_p, S), où N est un ensemble fini de symboles non-terminaux ; Σ est un ensemble fini de symboles terminaux ; R_p est un ensemble de règles de la forme $A \rightarrow aB$ ou $A \rightarrow a$, pour tout $A, B \in N$ et $a \in \Sigma \cup \{\epsilon\}$, munie chacune d'une valeur réelle positive $p(r \in R_p)$ telle que :*

$$\sum_{a \in \Sigma, B \in N} p(A \rightarrow aB) + \sum_{a \in \Sigma} p(A \rightarrow a) = 1 \text{ pour tout } A \in N$$

et où S est l'axiome. □

Définition 12 *La probabilité $p(x|G)$ pour qu'une SRG G génère une séquence $x \in \Sigma^*$, $x = x_1 x_2 \ldots x_n$ est :*

$$p(x|G) = \sum_{\mathbf{d}(x) \in \{\mathbf{d}\}} p(\mathbf{d}(x))$$

où $\mathbf{d}(x)$ est une dérivation *de x par G de la forme :*

$$S \rightarrow x_1 A_1 \rightarrow x_1 x_2 A_2 \rightarrow \ldots \rightarrow x_1 x_2 \ldots x_{n-1} A_{n-1} \rightarrow x_1 x_2 \ldots x_{n-1} x_n = x$$

$\{\mathbf{d}\}$ est l'ensemble de dérivations possibles et :

$$p(\mathbf{d}(x)) = p(S \rightarrow x_1 A_1) p(A_1 \rightarrow x_2 A_2) \ldots p(A_{n-1} \rightarrow x_n)$$

□

Définition 13 *Une SRG définit une distribution de probabilités sur Σ^* si :*

$$\sum_{x \in \Sigma^*} p(x|G) = 1$$

On dit alors que G est consistante. □

[16] de l'anglais : *Stochastic Regular Grammar.*

La définition de R_p donnée dans la Définition 11 est suffisante pour que G soit consistante [Higuera (de la), 1996a].

L'exemple ci-dessous reprend la grammaire G de la FIG. 1.2 dont les règles ont été probabilisées :

Exemple 3 :

$G_p = (N, \Sigma, R_p, S)$ avec :

$N = \{S, A, B\}$

$\Sigma = \{0, 1\}$

$$R_p = \left\{ \begin{array}{lcl} S \to 1A & , & p = 1.0 \\ A \to 0A & , & p = 0.3 \\ A \to 1A & , & p = 0.1 \\ A \to 0B & , & p = 0.2 \\ A \to 1B & , & p = 0.4 \\ B \to 1 & , & p = 1.0 \end{array} \right\}$$

FIG. 1.6: Une Grammaire Régulière Stochastique.

La probabilité pour que G_p (FIG. 1.6) génère la chaîne 1011 est :

$$p(1011) = p(S \to 1A)p(A \to 0A)p(A \to 1B)p(B \to 1) = 1.0 * 0.3 * 0.4 * 1.0 = 0.12$$

(unique façon de générer la chaîne 1011).

La grammaire est consistante, conséquence de ce que la somme des probabilités des règles qui ont le même symbole non-terminal à gauche (une règle pour S, quatre règles pour A et une règle pour B) est égale à 1. □

Comme pour les grammaires régulières non-stochastiques, il est possible de représenter une SRG par un *automate stochastique*. L'équivalent de la grammaire G_p de l'exemple précédent est montré sur la FIG. 1.7.

Exemple 4 :

Chaque transition est affectée de la probabilité de la règle qui lui correspond dans G_p et chaque état possède un certaine probabilité d'être un état d'acceptation. Ici, seul l'état 3 possède une probabilité non-nulle d'être terminal et qui vaut 1. La probabilité d'un chemin dans l'automate est alors le produit des probabilités de toutes les transitons empruntées, multiplié par la probabilité de l'état courant d'être un état d'acceptation. La probabilité de l'exécution :

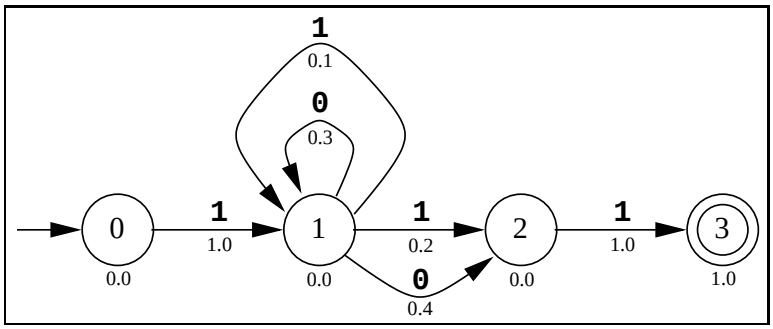


FIG. 1.7: Automate Fini Stochastique Non-déterministe.

$$\begin{array}{ccccccccc} (q_0, 1011) & \vdash & (q_1, 011) & \vdash & (q_1, 11) & \vdash & (q_2, 1) & \vdash & (q_3, \epsilon) \\ 1.0 & \times & 0.3 & \times & 0.4 & \times & 1.0 & \times & 1.0 \times 1.0 \end{array}$$

vaut : □

Remarque 2 *On ne sait pas déterminiser un automate stochastique non-détérministe : le calcul qui permettrait d'obtenir la même distribution de probabilités sur Σ^* qu'avec l'automate résultant n'est pas connu.* □

1.2.3 Approches mixtes

L'apparition des modèles mixtes, mélangeant l'information statistique et linguistique est relativement récente et reste encore au stade de l'investigation. Des travaux ont cependant été entrepris en constatant les limites des approches purement symboliques et purement statistiques. D'une part, les grammaires symboliques sont plus adaptées à la modélisation des régularités de la langue qu'à ses exceptions : considérées collectivement, ces dernières présentent une fréquence d'occurrence non négligeable. Leur prise en compte conduit généralement vers des problèmes de maintenance, notamment à cause des ambiguïtés ou contradictions créées par l'apport des nouvelles connaissances. De plus, comme nous l'avons déjà fait remarquer dans le paragraphe 1.2.1.2, le temps de traitement associé à ces approches est souvent trop élevé. D'autre part, les modèles stochastiques actuels, basés sur la seule observation de séquences restreintes d'entités linguistiques (mots, catégories syntaxiques, etc.) ne peuvent pas rendre compte de la structure profonde des énoncés. La seule modélisation de l'ordonnancement linéaire du langage constitue un compromis technique sous-optimal pour de nombreuses classes d'applications. Un exemple d'illustration de ce problème est la gestion très approximative des accords effectués par les systèmes de dictée vocale, résultant de l'impossibilité de ces modèles à rendre compte des dépendances à longue distance : les erreurs d'accord dans ces applications relèvent de régularités modélisables par un nombre limité de règles ([Smaïli et al., 1992]).

Trois stratégies concernant la conception des systèmes mixtes peuvent être envisagés : la première consiste à coupler en série les modules statistique et symbolique : en règle générale, un des modules est chargé de compenser ou de corriger les insuffisances de son partenaire (voir [Spriet & El-Bèze, 1997] ou [Chow & Schwartz, 1989] pour des applications basées sur ce type d'architecture). Les deux autres stratégies sont une intégration dans un seul processus des approches symbolique et statistique. L'une consiste à *probabiliser* un modèle symbolique (voir par exemple [Shabes, 1992] pour les grammaires d'arbres adjoints ou [Lafferty et al., 1992] pour les grammaires de liens), l'autre à *intégrer des contraintes symboliques* dans un processus stochastique (voir [Hetherington et al., 1996; Kenny et al., 1994] pour les travaux sur les treillis de mots en reconnaissance de la parole). Une synthèse sur les techniques mixtes appliquées à la reconnaissance de la parole est contenue dans [TALN, 1999].

1.2.4 Autres approches

La taille croissante des bases de données linguistiques rend les techniques d'Apprentissage Automatique très intéressantes pour la conception des modèles de langage. Les approches présentées ci-dessous font appel à des théories de ce domaine, adaptées aux applications de TALN et de TALP.

Modélisation de langage à l'aide de la logique des prédicats. Les prédicats de la logique du premier ordre peuvent être utilisés pour représenter des grammaires hors-contexte et sont en conséquence bien adaptés pour définir les aspects structurels d'un langage. Les programmes logiques sont formés des clauses, usuellement exprimées dans un formalisme PROLOG ou un de ses derivés. Par exemple :

$$s(A, B) \rightarrow gn(A, C), gv(C, D), gn(D, B);$$

signifie qu'une phrase est constituée d'un groupe nominal, suivi d'un groupe verbal, lui-même suivi d'un autre groupe nominal. Cette approche nécessite un apport de connaissances linguistiques ; toutes les connaissances *a priori* (catégories lexicales, règles syntaxiques, etc...) sont exprimées sous forme de prédicats. Plus récemment, une nouvelle branche de l'intelligence artificielle : l'ILP[17] ([Muggleton & Raedt, 1994]) s'est intéressée à l'apprentissage automatique de clauses à partir d'exemples (donc ici : à l'apprentissage des syntaxes à partir des séquences).

Modèles appris par cœur. C'est un ensemble d'hypothèses (dans le cas d'un langage : de phrases) qui est stocké en mémoire. Le principe d'utilisation d'un tel "modèle" repose sur le calcul des analogies entre le modèle en mémoire et un nouvel exemple. Pour classer les exemples (par exemple pour valider leur appartenance à un langage donné) il faut alors définir une *similarité* qui prédit son appartenance à une catégorie particulière. Cette approche a été appliquée dans des différentes tâches du TALN : désambiguïsation syntaxique et sémantique ([Cardie, 1995; Cardie, 1996]), analyse morphologique

[17] de l'anglais : *Inductive Logic Programming.*

([Daelemans et al., 1996; van den Bosch et al., 1996]), traduction automatique ([Jones, 1996]) ou prosodie ([Blin & Miclet, 2000]).

À la frontière de ce type de représentation on peut également citer la méthode DOP[18] ([Kaplan et al., 1996]) qui permet d'effectuer une analyse et une désambiguïsation statistique des phrases. La représentation des structures syntaxiques consiste ici à extraire d'un corpus d'apprentissage annoté toutes les plus petites sous-analyses possibles (par exemple des arbres lexicalisés dans le cas du formalisme Tree-DOP) et à les stocker en mémoire. La généricité de la méthode réside dans la possiblilité de recombiner ces sous-analyses de façon à obtenir des analyses absentes du corpus d'apprentissage.

Arbres de décision. Les arbres de décision ont été employés avec succès dans des diverses applications de TALN/TALP, par exemple dans un tâche de désambiguïsation du sens des mots [Kuhn & Mori, 1995; Loupy et al., 2000] ou dans la stylisation de la fréquence fondamentale du signal de la parole [Traber, 1995].

En ce qui concerne l'analyse *syntaxique* des textes, leur emploi est encore assez marginal. Une telle application peut-être consultée dans [Black et al., 1992] ou encore dans [Magerman, 1995] visant à étiqueter des corpus. Dans le principe, l'analyse la plus probable d'une phrase est obtenue à partir de la séquence des décisions prises en fonction d'un arbre construit sur un corpus d'apprentissage.

Réseaux connexionnistes. Au cours de la dernière décennie, l'étude des réseaux connexionnistes (RC) a amené leur emploi dans des applications du traitement de la langue naturelle. Un RC entraîné[19] possède des propriétés d'abstraction et de généralisation de l'association entre les données d'entrée et celles de sortie. D'autres applications possibles des RC pour la modélisation de la syntaxe ou de la sémantique sont décrites entre autres dans [Reilley & Sharkey, 1992].

1.3 Construction d'un modèle de langage

L'élaboration manuelle d'une grammaire (à base de connaissances) par un expert représente une opération complexe et fastidieuse. Même si ce constat reste sujet de polémiques (voir par exemple [Chanod & Tapanainen, 1995]), on associe souvent à cette approche des problèmes de maintenance et d'extension de la grammaire à partir d'une certaine taille critique. Les grammaires symboliques ont ainsi du mal à tenir le pari contradictoire d'une large couverture associée à une grande précision : tout enrichissement de la grammaire se traduit en effet généralement par une perte de robustesse due à l'apparition de contradictions entre la grammaire initiale et les nouvelles connaissances introduites. De ce fait, la conception des modèles de langage pour des systèmes de traitement de la langue naturelle s'oriente souvent vers des méthodes automatiques

[18] de l'anglais : *Data Oriented Parsing*.

[19] en général par l'algorithme classique de rétropropagation du gradient de l'erreur, voir par exemple [Rumelhart et al., 1986].

(voir le paragraphe suivant) ou semi-automatiques. Ces dernières consistent, soit à enrichir un modèle symbolique (éventuellement conçu par un expert) par l'adjonction des probabilités, soit symétriquement, à ajouter des connaissances linguistiques dans l'analyse stochastique (approches mentionnées dans le paragraphe 1.2.3).

Apprentissage automatique

Comme il a été mentionné dans la section précédente, les techniques d'Apprentissage Automatique sont actuellement de plus en plus appliquées aux différents aspects du Traitement du Langage Naturel. Il y a en effet une convergence entre ces deux domaines qui s'explique en particulier par les raisons suivantes :

- Le besoin de traitement automatique du langage écrit et oral a considérablement augmenté : recherche automatique sur le Web à partir du contenu des pages, reconnaissance de la parole, traduction automatique, etc. Pour ces applications et beaucoup d'autres, les corpus disponibles sont désormais de taille suffisante pour que des algorithmes d'apprentissage puissent en extraire des modèles réalistes. Notons que ces corpus sont le plus souvent composés uniquement d'exemples positifs, ce qui pose un problème particulier aux algorithmes d'apprentissage (comment limiter la généralisation ?).
- Les algorithmes d'Apprentissage Automatique se sont développés selon d'une part un axe principalement symbolique (ILP, Inférence Grammaticale. . .), d'autre part selon un axe numérique (décision statistique bayésienne, réseaux connexionnistes, etc.). Actuellement, la synthèse se fait grâce à des méthodes mixtes symboliques-numériques : par exemple, l'apprentissage par arbres de décision, ou l'apprentissage de grammaires stochastiques. Ces méthodes permettent en particulier d'envisager l'extraction de concepts structurés à partir de gros ensembles de données bruitées. C'est exactement le problème posé en traitement des langues naturelles.

Compte tenu de la variété et de la complexité des problèmes posés par le traitement de la langue naturelle, un bon nombre de méthodologies différentes d'apprentissage automatique ont déjà été testées. Nous avons mentionné certaines d'entre elles dans les sections précédentes (ILP, arbres de décision, réseaux neuronaux, . . .). On peut également citer :

- Les méthodes bayésiennes (par exemple [Nigam et al., 1998]),
- L'inférence grammaticale algébrique et stochastique (présentée dans le chapitre suivant, voir aussi [ICGI, 1998]),
- etc.

Divers colloques et congrès ont déjà été consacrés à l'apprentissage automatique appliqué au traitement de la langue naturelle. On pourra en particulier consulter les actes de [ICML, 1997; AAAI, 1998].

1.4 Évaluation d'un modèle de langage

Afin de mesurer l'adéquation du modèle construit à la tâche pour laquelle il a été conçu, il est nécessaire d'utiliser une *méthode d'évaluation*. En règle générale, la qualité d'un modèle est évaluée d'après sa capacité à prédire un corpus de test. Pour cela, les modèles probabilistes disposent d'un riche cadre théorique fourni par la *Théorie de l'Information*. Ses éléments et son application en pratique font le sujet du paragraphe 1.4.1. Le paragraphe 1.4.2 traite du problème d'évaluation des modèles symboliques.

1.4.1 Évaluation des modèles probabilistes

1.4.1.1 Entropie

Définition 14 *Soit une suite d'observations $o_1 o_2 \ldots o_T$ émise par une source O. On appelle* entropie *et on note $H(O)$ la valeur moyenne de la quantité d'information apportée par o_i Cette valeur est proportionnelle à la mesure d'incertitude que la connaissance de o_i retire au destinataire.* □

L'entropie permet d'évaluer la *prédicitibilité* de la source. Elle est maximale lorsque toutes les observations sont équiprobables. Elle est minimale lorsque l'observation à venir est certaine.

Définition 15 *Soit P un processus particulier qui régit les émissions de O et qui est identifié par un ensemble de paramètres Θ. L'*imprédicitibilité*, ou encore la* quantité d'information *contenue dans une suite d'observations $o_1 o_2 \ldots o_T$ étant donné (P, Θ) vaut :*

$$-log\ \mathcal{L}(o_1 o_2 \ldots o_T | P, \Theta)$$

où $\mathcal{L}$ est la fonction de vraisemblance mesurant les valeurs de probabilités. □

Définition 16 *Soit $O^{'}$ une* source ergodique[20] *dont les émissions sont régies par (P, Θ). L'entropie de $O^{'}$ est approximée par la quantité moyenne d'information apportée par une seule observation $o_1 o_2 \ldots o_T$:*

$$H(O^{'}|P, \Theta) \simeq -\frac{1}{T}\ \log\ \mathcal{L}(o_1 o_2 \ldots o_T | P, \Theta)$$

□

Les notations proposées plus haut peuvent être transposées au cas des langages probabilistes : (P, Θ) est assimilé à un modèle M d'un langage stochastique $L(M)$

[20] une source ergodique est telle que toute suite suffisamment longue de ses émissions est représentative de son comportement et permet donc d'en étudier la structure statistique.

qu'il génère ; $o_1 o_2 \ldots o_T$ correspond à un corpus représentatif de $L(M)$ qu'on note : $S = \{x_1 x_2 \ldots x_T\}$. L'entropie de S estimée avec M est alors :

$$H(S|M) \simeq -\frac{1}{T} \log\, p(x_1 x_2 \ldots x_T|M)$$

En pratique M et donc $L(M)$, est presque toujours inconnu. Le travail de conception d'un modèle de langage consiste à construire un modèle $M^{'}$ qui approxime le mieux $L(M)$.

Dans le cas des modèles probabilistes, ce travail peut être vu comme la recherche d'une certaine distribution de probabilités. Pour évaluer $M^{'}$ on fait appel à la mesure d'*entropie croisée* qui est proportionnelle à la distance entre les distributions de probabilités de $L(M)$ et de $L(M^{'})$. Elle est définie par :

$$H(L(M), L(M^{'})) \simeq H(S, L(M^{'})) = -\frac{\sum_{x_1}^{x_T} \; log\; p(x|M)}{\sum_{x_1}^{x_T} |x|}$$

où $|x|$ est la longueur d'une séquence x.

1.4.1.2 Perplexité

La *perplexité* est un des critères qui permettent d'évaluer la qualité d'un modèle syntaxique pourvu d'une distribution de probabilités [Bahl et al., 1983; Dupont, 1996]. Elle est déduite directement de l'entropie et peut être comprise comme le *pouvoir de prédiction* de ce modèle. Ce pouvoir est d'autant plus grand que le nombre de symboles proposés par le modèle comme le symbole suivant d'une séquence est petit. Si le modèle ne peut proposer aucun symbole, alors son pouvoir prédicitif est nul et sa perplexité est maximale. Si l'on utilise le logarithme en base 2 dans la formule précedente alors la perplexité est définie par :

$$Q(S, L(M)) = 2^{H(S, L(M))}$$

Ce calcul nécéssite d'une part la consistance du modèle (voir Définition 11, page 19), d'autre part l'attribution d'une probabilité non nulle à toute séquence composée des symboles de Σ.

1.4.2 Évaluation des modèles symboliques

Un modèle symbolique ne pouvant par définition fournir une estimation probabiliste d'appartenance d'une séquence au langage qu'il représente, les techniques citées plus haut ne peuvent s'appliquer directement. Dans [El-Bèze et al., 1997] les auteurs montrent qu'il est possible de calculer la perplexité d'un modèle non-stochastique par le biais du *Jeu de Shannon* ([Shannon, 1951]). Il s'agit, pour une phrase tronquée, de classer par rang de préférence les éléments du vocabulaire. L'élément classé premier a le plus de chances de compléter la phrase, puis le deuxième, etc. La propriété remarquable de cette méthode est son objectivité dans le cadre d'une campagne d'évaluation

comparative : les valeurs de perplexité sont estimées sur le site de l'organisateur, sans nécéssiter la réimplantation de chaque modèle testé.

Parmi d'autres critères (prise en compte des dépendances syntaxiques à long terme, possibilité d'adaptation progressive du modèle, etc.), la mesure la plus fréquemment utilisée en informatique linguistique est la *précision* et le *rappel*. Appliqués au problème de reconnaissance d'une séquence par un modèle de langage, nous adaptons leurs définitions comme suit :

Définition 17 *Soit M le modèle d'un langage $L(M)$. Soit $M^{'}$ le modèle d'un langage $L(M^{'})$ qui approxime le langage $L(M)$.*

La précision *est le taux de séquences classées correctement par $M^{'}$ parmi un ensemble de séquences générées par M.*

Le rappel *est le taux de séquences classées correctement par M parmi un ensemble de séquences générées par $M^{'}$.*

□

En pratique, la mesure de rappel (qui peut-être vue comme une mesure de *sur-généralisation*[21] et de *sous-couverture*[22] si l'on dispose d'un ensemble de test constitué d'exemples positifs et d'exemples négatifs) n'est pas applicable car on ne dispose pas en règle générale du modèle M. Elle permet cependant d'évaluer une méthode de construction du modèle (cf. paragraphe 2.3.1) en utilisant un modèle artificiel.

1.5 Conclusion

Dans ce chapitre, nous avons fourni un aperçu global des diverses approches visant à représenter un langage de façon structurée. Tous les points abordés ont pour but de présenter le cadre global dans lequel se situe l'ensemble de nos travaux : choix d'un type de modèle, sa construction et son évaluation. En particulier, dans le domaine du traitement de la langue parlée, où les contraintes temporelles sont importantes, le choix des grammaires régulières, y compris probabilisées, nous semble être un bon compromis entre la puissance du formalisme et le temps d'analyse. Les méthodes de conception de ce type de modèle sont abordées dans le chapitre suivant. Les travaux empiriques concernant l'utilisation d'un modèle particulier dans une chaîne de reconnaissance de la parole seront exposés dans le Chapitre 4 et les suivants.

[21] dans ce cas : taux de reconnaissance des séquences qui n'appartiennent pas à $L(M)$.
[22] dans ce cas : taux de rejet des séquences qui appartiennent à $L(M)$.

Chapitre 2

Inférence Grammaticale

L'Inférence Grammaticale est une des disciplines de l'Apprentissage Automatique : elle est constituée d'un ensemble de théories et de méthodes pour *apprendre des grammaires à partir d'exemples*. L'objet de ce chapitre est de présenter ses notions fondamentales : les critères pour l'inférence correcte, la nature de l'espace de recherche de solutions et la complexité de la tâche. Nous le terminons par une classification de méthodes et nous citons les algorithmes les plus représentatifs de ce domaine.

2.1 Introduction

2.1.1 Bref historique

Déjà dans les années cinquante ([Chomsky & Miller, 1957]) l'inférence grammaticale a été proposée comme une technique particulière pour apprendre automatiquement des grammaires à partir de phrases exemples. L'article fondamental de Gold [Gold, 1967] a fourni des bases théoriques à la construction des modèles d'apprentissage de ce domaine. Depuis, des nombreux travaux effectués au sein des diverses communautés (apprentissage, reconnaissance des formes syntaxiques et structurelles, traitement de la langue, réseaux neuronaux, théories des langages formels, théorie de l'information), ont contribué à l'établissement de la discipline de l'inférence grammaticale et à la conception des méthodes algorithmiques efficaces. La grande variété des domaines participant à ces travaux fait que les publications qui s'y rapportent ont été dispersées dans des actes de conférences et des revues hétérogènes, essentiellement liés aux thèmes d'apprentissage, de reconnaissance de la parole ou de la reconnaissance des formes. Depuis 1992, l'inférence grammaticale possède un colloque qui lui est spécifiquement dédié : *ICGI* (International Colloquium on Grammatical Inference).

2.1.2 Objectifs

Lorsqu'on applique un processus d'inférence grammaticale (une *méthode d'inférence*) sur un ensemble d'exemples (données d'entrée), le résultat attendu est d'obtenir un

modèle de cet ensemble (données de sortie : *une grammaire, un automate, une expression régulière, ...*). Dans cette démarche, il est préalablement nécessaire de préciser un certain nombre de paramètres :

- *la condition d'apprentissage* : il s'agit d'énoncer le critère de convergence selon lequel l'inférence sera correcte. Les critères classiques seront présentés dans le paragraphe 2.2.3 ;
- *la complexité du processus* : il dépend de la *classe* de la grammaire inférée. Dans le cas général, il a été montré que l'apprentissage des grammaires sans restriction à partir des exemples positifs (i.e. appartenant au langage ciblé) et des exemples négatifs (ou des contre-exemples) est un problème NP-difficile. Si l'on cherche une complexité polynômiale et une inférence exacte, on doit se restreindre à des classes des grammaires particulières. Ce point sera étudié plus en détail dans le paragraphe 2.2.4 ;
- *la classe de la méthode d'inférence* : elle est liée au point précédent. Pour les classes de grammaires dont on sait que l'identification est NP-difficile, il conviendra d'utiliser des méthodes *heuristiques*. Les méthodes *caractérisables* sont utilisées lorsqu'on vise une classe de grammaires (prouvée être) identifiable en temps polynômial. Une séparation de méthodes d'inférence selon les classes est présentée dans la section 2.4 ;
- *l'ensemble d'apprentissage* : les connaissances sur cet ensemble influent sur le choix de la méthode d'inférence. Il est fait notamment en fonction de la présence ou de l'absence d'un échantillon négatif (le manque des données négatives est souvent observé dans le cas réel). De même il est déterminant de savoir si un *échantillon caractéristique* (cf. paragraphe 2.2.2) est présent dans l'ensemble des exemples.

Les objets obtenus par le processus d'inférence varient bien sûr en fonction de l'utilisation à laquelle ils sont destinés. Des méthodes d'apprentissage ont été conçues pour les types de modèles présentés dans le Chapitre 1 : les grammaires hors-contexte générales, les grammaires linéaires, les grammaires régulières, les transducteurs. De plus, il existe des méthodes permettant d'apprendre une distribution de probabilités sur ces structures (on parle alors d'*inférence grammaticale stochastique*).

2.1.3 Applications

Dans la pratique, l'inférence grammaticale est encore sous-exploitée comparée à des méthodes plus "séduisantes"[1] d'autres sous-domaines de l'apprentissage automatique (les méthodes connexionnistes, les méthodes de reconnaissance statistique de formes, l'apprentissage des modèles de Markov...). Il existe néanmoins un nombre important de publications dédiées aux applications pour lesquelles l'apprentissage des grammaires fournit des modèles efficaçes ([Higuera (de la), 2000]). Un des premiers domaines où les grammaires induites ont été appliquées est la reconnaissance des formes : les formes

[1] sous-entendu : bénéficiant des apports d'une communauté scientifique plus vaste.

géométriques peuvent être encodées en un ensemble équivalent de séquences (habituellement en décrivant par un alphabet les propriétés — direction, longueur — des segments qui les composent). La caractérisation d'une forme consiste alors dans la construction d'une structure syntaxique qui régit la composition du langage ainsi obtenu (voir par exemple [Vidal et al., 1993]). La même technique peut s'appliquer lors d'une tâche de reconnaissance des lettres manuscrites ou imprimées ([Lucas et al., 1994]) ou plus étonnant, dans l'apprentissage des styles musicaux (le solfège étant facilement assimilable à un ensemble fini des symboles), proposé dans [Cruz & Vidal, 1998]. Un autre champ d'application est le domaine du Traitement Automatique de la Parole et de la Langue Naturelle. Bien que les grammaires formelles soient un modèle insuffisant pour traiter l'ensemble des règles syntaxiques auxquelles est soumis le langage humain, il a été montré que, sur un domaine restreint (modélisation des requêtes orales, traduction automatique [Castellanos et al., 1994], étude morphologique des mots [Itai, 1994], étude des structures des documents [Ahonen & Mannila, 1994], catégorisation des séquences [Fred, 1996]), elles s'avèrent être un outil bien adapté. Le travail présent s'inscrit dans ce cadre. D'autres communautés scientifiques s'intéressent aux techniques de l'inférence grammaticale, en particulier celles qui étudient les régles qui dirigent la composition des bioséquences (ADN ou ARN) ([Brazma et al., 1998]). Finalement, un emploi intéressant de l'inférence grammaticale est le test de violabilité des codes de cryptage : un algorithme polynômial apprend un modèle de codes, si ce modèle est capable de prédire un code correct alors celui-ci est rejeté ([Takada, 1996]).

2.2 Aspects formels de l'Inférence Grammaticale

2.2.1 Objet de l'Inférence Grammaticale

Les objets cibles des méthodes proposées en inférence grammaticale (IG) sont des grammaires formelles dont nous avons donné la définition et la hiérarchie dans le Chapitre 1. En règle générale, on cherche à réduire la complexité du problème par la recherche d'une classe particulière, comme par exemple les grammaires linéaires, les grammaires hors-contexte faibles, les grammaires hors-contexte pures (voir par exemple [Higuera (de la), 1996b] pour un article de synthèse) et (surtout) les grammaires régulières (voir Chapitre 1). Quelques méthodes ont été proposées pour apprendre des des grammaires hors-contexte (grammaires de type 2) sans restrictions (par exemple Giordano dans [Giordano, 1994]). Finalement, peu de travaux ont été consacrés à l'apprentissage des modèles décrivant les langages "context-sensitive" (grammaires de type 1), comme les Expressions Régulières Augmentées ([Alquézar et al., 1996]) ou les ATN[2] ([Chou & Fu, 1976]).

Les grammaires régulières (grammaires de type 3), possèdent plusieurs représentations équivalentes : expressions régulières, automates à états finis déterministes, automates à états finis non-déterministes et une extension naturelle : les transducteurs rationnels. Ces objets ont été largement étudiés et utilisés en traitement des séquences,

[2] de l'anglais : *Augmented Transition Networks.*

leur apprentissage porte le nom d'*Inférence Régulière*. Cette dernière possède une cadre formel bien défini (cf. paragraphe 2.2.2) et bénéficie des nombreux résultats théoriques ([Angluin, 1978; Gold, 1978; Higuera (de la), 1997]), expérimentaux et applicatifs (voir les actes de [ICGI, 1994; ICGI, 1996; ICGI, 1998; ICGI, 2000]). La principale faiblesse des langages réguliers, par rapport aux grammaires hors-contexte, est la perte de la capacité de modélisation de l'*enchâssement*. Ceci consiste, dans le langage naturel, à imbriquer des propositions comme par exemple dans :

Le chat, dont *je t'ai parlé hier*, boit du lait

Cependant, ce handicap des grammaires régulières n'est pas si grave, sachant que dans l'utilisation courante de la langue naturelle le niveau d'imbrication dépasse rarement l'unité. Le traitement de ce problème nécessite de répéter les "sous-grammaires" dans la grammaire du niveau supérieur. L'illustration de ce cas est donnée dans la FIG. A.1 (page 142), où le modèle régulier du sous-langage représentant des formes géométriques est placé à plusieurs reprises dans le modèle régulier global.

Utilisation de probabilités. Dans le cas pratique, l'ensemble d'apprentissage est souvent dépourvu de données négatives : il est plus courant d'observer les instances d'une classe d'évènements que leur contre-exemples — un corpus de la langue naturelle notamment est composé des phrases du langage ciblé et rarement (pour ainsi dire jamais) des phrases qui n'en sont pas. De plus, l'information complémentaire constituée par la fréquence d'apparition des mots ou des phrases n'est pas prise en compte dans les approches citées plus haut. La solution pour affiner les modèles à apprendre est alors l'introduction des probabilités qu'on associe aux règles de production. On parle dans ce cas de l'*Inférence Stochastique* et les objets cibles sont des grammaires ou transducteurs stochastiques, ainsi que des modèles N-grams (qui sont un cas particulier des grammaires régulières — voir [Ron et al., 1994]).

2.2.2 L'espace de recherche de l'Inférence Régulière

Le processus d'inférence régulière peut être vu comme la recherche d'un automate inconnu A (éventuellement stochastique) à partir duquel l'échantillon positif I_+ est supposé être généré. Dans [Dupont & Miclet, 1998] les auteurs définissent ce qu'est l'espace dans lequel s'effectue la recherche sous un biais d'apprentissage particulier. Cette section s'en inspire dans sa majeure partie.

Définition 18 *Un échantillon I_+ est* structurellement complet *relativement à un automate A, s'il existe une acceptation de I_+ par A telle que :*
- *toute transition de A soit exercée ;*
- *tout état final de A soit utilisé comme état d'acceptation.*

□

Ce biais d'apprentissage est important. En effet, sous l'hypothèse que l'on ne recherche que les automates pour lesquels I_+ est structurellement complet, on dispose de résultats

forts permettant de limiter l'espace de recherche. Avant de citer les théorèmes correspondants, nous allons définir les notions d'*automate canonique maximal* et d'*arbre accepteur* des préfixes.

Définition 19 *À partir d'un échantillon I_+, on peut construire* l'automate canonique maximal*, noté $MCA(I_+)$* [3] *qui est l'automate comportant le plus grand nombre d'états tel que $L(MCA) = I_+$ et I_+ est structurellement complet pour $MCA(I_+)$ (*FIG. *2.1).* □

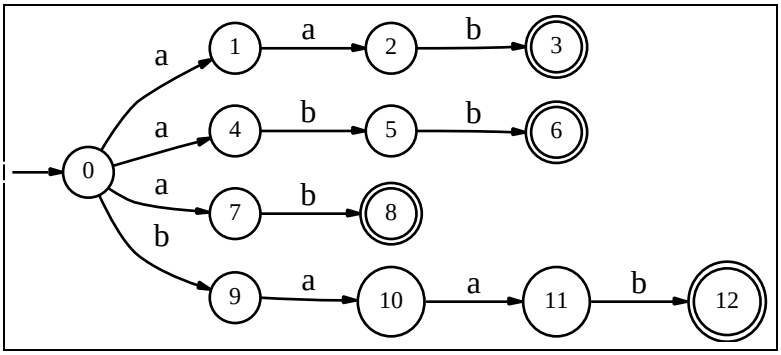


FIG. 2.1: Automate $MCA(I_+)$ avec $I_+ = \{aab, ab, abb, baab\}$.

Définition 20 L'arbre accepteur des préfixes *[Miclet, 1979], noté $PTA(I_+)$,* [4] *est l'automate obtenu en fusionnant les états partageant les mêmes préfixes*[5] *du $MCA(I_+)$ (*FIG. *2.2).* □

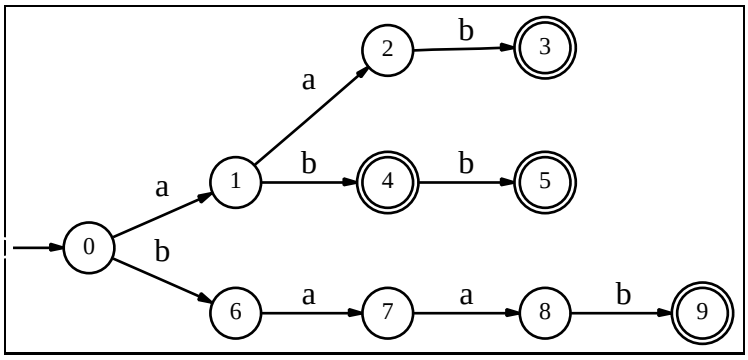


FIG. 2.2: Automate $PTA(I_+)$ avec $I_+ = \{aab, ab, abb, baab\}$.

[3] de l'anglais : *Maximal Canonical Automaton.*
[4] de l'anglais : *Prefix Tree Acceptor.*
[5] u est un préfixe de l'état q si $q \in \delta(q_0, u)$.

Théorème 3 *[Miclet, 1979; Dupont et al., 1994] Soit I_+ un échantillon positif d'un langage régulier L et soit $A(L)$ n'importe quel automate acceptant exactement L. Si I_+ est structurellement complet relativement à $A(L)$ alors $A(L)$ peut être obtenu par dérivation du $MCA(I_+)$.* □

La dérivation d'un automate s'effectue en fusionnant deux de ses états. Par exemple, le PTA de la Fig. 2.2 peut être obtenu par les dérivations successives du MCA de la Fig. 2.1 (l'état 1 avec l'état 4 puis avec l'état 7 et l'état 5 avec l'état 8). La Fig. 2.3 montre un automate résultat de la fusion des états 4 et 7 du PTA ainsi construit. Le langage accepté par cet automate est le langage accepté par le PTA augmenté des séquences *bab* et *abab*. La fusion peut être considérée alors comme un *opérateur de généralisation* : le langage de l'automate dont on fusionne les états est inclus dans les langages des automates dérivés.

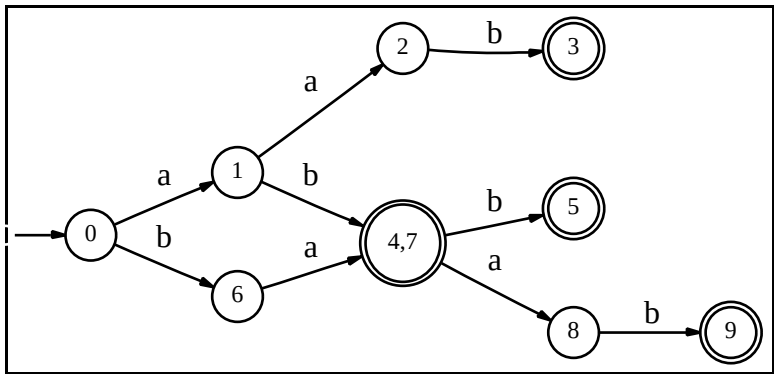


Fig. 2.3: Automate A dérivé du PTA de la Fig. 2.2 par la fusion des états 4 et 7. On a : $L(PTA) \subseteq L(A)$.

On peut facilement voir que la fusion successive de tous les états conduit vers l'obtention de l'automate universel AU, en générant un nombre exponentiel d'automates intermédiaires. L'ensemble des automates dérivés du MCA (resp. PTA), qui est partiellement ordonné par la relation de dérivation (cohérente avec l'inclusion des langages que reconnaissent ces automates), est un *treillis*[6]. Les automates MCA (resp. PTA) et AU en sont respectivement les éléments nul et universel. On note ce treillis $Lat(MCA)$ (resp. $Lat(PTA)$). Ceci permet alors de spécifier l'espace de recherche :

Théorème 4 *Soit α l'ensemble des automates pour lesquels un échantillon positif I_+ est structurellement complet. L'ensemble α est exactement $Lat(MCA(I_+))$.* □

[6]un *treillis* est un ensemble ordonnée E dans lequel tout couple d'éléments admet un plus petit majorant et un plus grand minorant.

Si l'on ne recherche que des automates canoniques, le théorème suivant, d'après [Dupont et al., 1994], permet de réduire encore l'espace de recherche.

Théorème 5 *Soit I_+ un échantillon positif et soit Λ l'ensemble des langages réguliers tels que $\forall L \in \Lambda$, I_+ est structurellement complet pour $A(L)$, où $A(L)$ désigne l'automate canonique de L. On a alors :*

$$\forall L \in \Lambda, A(L) \in Lat(PTA(I_+))$$

□

Ainsi formalisé, l'espace de recherche possède les propriétés suivantes :

Propriété 1 $Lat(PTA(I_+)) \subseteq Lat(MCA(I_+))$ □

Propriété 2 *Il existe des échantillons positifs I_+ pour lesquels certains langages ne sont représentés que par des AFN dans $Lat(MCA(I_+))$.* □

Propriété 3 *Il existe des échantillons positifs I_+ pour lesquels l'ensemble des langages qui peuvent être identifiés dans $Lat(PTA(I_+))$ est strictement inclus dans l'ensemble des langages qui peuvent être identifiés dans $Lat(MCA(I_+))$* □

L'ensemble de ces notions est à la base de plusieurs méthodes d'inférence par cheminement dans le treillis des solutions (voir Fig. 2.2), le plus souvent sous contrôle d'un échantillon négatif (cf. section 2.4).

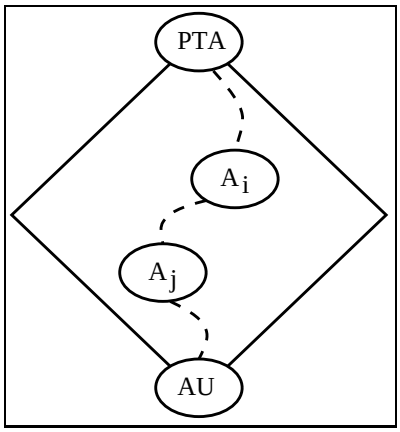


Fig. 2.4: Cheminement dans l'espace de recherche $Lat(PTA)$. On a : $L(PTA) \subseteq L(A_i) \subseteq L(A_j) \subseteq L(AU)$.

2.2.3 Les critères d'identification

Pour définir ce qu'est une inférence correcte, il faut énoncer les conditions pour lesquelles la grammaire apprise est considérée comme valide. Les critères les plus étudiés sont présentés dans les paragraphes qui suivent.

2.2.3.1 Identification à la limite

C'est le critère le plus courant. Il a été proposé par Gold [Gold, 1967] et constitue un protocole formel pour apprendre une grammaire correspondant à un ensemble d'apprentissage :

Définition 21 *Soit Σ^* l'ensemble de toutes les séquences de longueur finie sur un alphabet Σ, incluant le mot vide ϵ. Soit un ensemble $I_+ \subseteq L \subseteq \Sigma^*$ de séquences connues pour appartenir à un langage L et $I_- \subseteq (\Sigma^* - L)$ un ensemble de séquences qui n'appartiennent pas à L. Notons $S = I_+ \cup I_-$ et S_k les k premiers éléménts de S. Une classe de langage C est identifiable par présentation séquentielle des données s'il existe un algorithme d'inférence grammaticale A tel que pour tout langage L de C et pour tout S : $\exists n \in \mathbb{N} : \forall k > n, L(A(S_k)) = L(A(S_n)) = L$.* □

Informellement : une méthode identifie à la limite une grammaire G, si à partir d'un certain pas du processus d'apprentissage et pour tous les pas suivants G engendre le même langage.

Gold propose deux résultats fondamentaux qui découlent de ce critère :

1. *Toute classe effectivement énumérable de langages peut être inférée à partir d'une présentation complète de données (positives et négatives).*
   L'importance de ce résultat (les classes *context-free* et *context-sensitive* sont énumérables) est cependant diminuée par le fait que les données négatives ne sont pas toujours disponibles.
2. *Aucune classe superfinie de langages*[7] *ne peut être identifiée à la limite uniquement avec la présentation de données positives.*
   Puisque les langages réguliers sont une classe superfinie de langages, on ne peut pas identifier à la limite cette classe uniquement à partir d'un échantillon positif. On sait que sous condition d'existence d'un *ensemble caractéristique d'exemples* (ou *échantillon caractéristique* — cf. Définition 23 page 38), tel que sa présence dans les données positives implique que la méthode d'apprentissage retourne une solution correcte, il est possible d'apprendre exactement et en temps polynômial des sous-classes des langages réguliers.

[7] une *classe superfinie de langages* est une classe qui contient tous les langages finis et au moins un langage infini. Les langages réguliers constituent une classe superfinie de langages ([Hopcroft & Ullman, 1979]).

2.2.3.2 Identification Approximative et Identification PAC

Les résultats de l'identification à la limite démontrent la grande importance de l'ensemble de données négatives. Cependant, un certain nombre de travaux (notamment [Wharton, 1974] et [Valiant, 1984]) ont été menés pour trouver des critères satisfaisants avec uniquement un échantillon positif.

La convergence PAC. Soit Σ un alphabet. Soit L un langage défini sur Σ^*. Soit C un concept (une grammaire) appris sur L. On dira que C est correct pour L si, pour tout mot m de Σ^*, on peut décider si m appartient ou non à L. On dira que C est *approximativement correct* si la probabilité qu'il classe mal un mot est "petite".

Un algorithme d'apprentissage est correct si pour tout langage L défini sur Σ^*, le concept qu'il apprend est correct pour L. On dira qu'un algorithme d'apprentissage est *probablement correct* si la probabilité qu'il apprenne une grammaire approximativement correct est "grande".

Valiant a formalisé cette notion intuitive en créant le critère PAC :

Définition 22 ([Valiant, 1984]) *On dit que l'algorithme A PAC-identifie une classe de grammaires formelles C_G si et seulement si pour tout réel ϵ et δ ($\in$ [0,1]), pour toute grammaire G de C_G, A avec les exemples de $L(G)$ (étiquetés sous une distribution D sur Σ^*) retourne une hypothèse G' telle que : $P_r(d_D(L(G), L(G')) > \epsilon) < \delta$ où l'on note $d_D(L(G), L(G'))$ la somme des probabilités des chaînes de la différence symétrique entre $L(G)$ et $L(G')$.* □

Kearns et Valiant [Kearns & Valiant, 1989] ont prouvé que si l'on pouvait *PAC*-identifier la classe des automates finis déterministes, alors les clés *RSA* pourraient être décryptées. Il s'agit donc d'un problème NP-difficile.

L'apprentissage PAC possède des versions affaiblies (moins contraignantes) : l'apprentissage PAC-faible et PAC-simple que nous n'exposerons pas ici.

Variations sur la convergence PAC Même si la *PAC*-apprenabilité des AFD reste un problème ouvert, la communauté conjecture que le résultat sera négatif. De plus, les AFD sont les objets non triviaux les plus simples de la théorie des langages. Devant les résultats négatifs, on peut avoir plusieurs attitudes : revoir le contexte d'apprentissage (*i.e.* on permet à l'algorithme d'apprentissage d'obtenir des informations supplémentaires sur son objectif : par exemple, Natarajan [Natarajan, 1991] décrit un algorithme permettant une identification PAC en temps polynômial à l'aide de "requêtes d'appartenance" vers un oracle) ou poser des restrictions sur la classe de grammaires à apprendre[8].

[8] on peut restreindre la classe elle-même ou restreindre la classe des distributions de probabilités que l'on autorise au langage cible.

2.2.3.3 Autres protocoles

On peut aussi voir l'apprentissage comme une tâche d'enseignement : un *oracle* remplit la fonction du "professeur" fournissant à son "élève" : l'algorithme d'apprentissage, les meilleurs exemples, sélectionnés en fonction de ce dernier. Goldman et Mathias définissent l'*enseignabilité* d'un concept comme l'existence d'un ensemble caractéristique de données pour un élève particulier. En d'autres termes, s'il existe un ensemble caractéristique de données permettant à l'algorithme d'inférer correctement une grammaire G, alors la classe de G : $C(G)$ est *enseignable*. De la Higuera ([Higuera (de la), 1997]) exploite ce résultat pour le lier avec la notion d'identification à la limite de Gold. Remarquons que dans le cas pratique ce critère est rarement applicable : il implique l'existence de l'objet recherché (d'une grammaire cible dans le cas d'inférence grammaticale).

Il existe plusieurs autres travaux sur le relâchement de la contrainte forte qu'est l'identification exacte. Citons ici les "requêtes d'équivalence" [Angluin, 1987] qui impliquent également un oracle, ou la EX^k-identification [Barzdin, 1974] où l'on prend en compte le nombre maximal d'anomalies entre le langage inféré et le langage correct. Pour des informations plus complètes sur la notion d'apprenabilité en inférence grammaticale, le lecteur pourra se reporter à [Higuera (de la) et al., 1996].

2.2.4 Complexité calculatoire du processus d'inférence

Gold a démontré un résultat fondamental sur la complexité de l'inférence régulière à partir d'échantillons finis :

Théorème 6 ([Gold, 1978]) *Le problème de l'identification à la limite du plus petit AFD compatible avec I_+ et I_- est NP-difficile*[9]. □

Suite à ce résultat négatif, diverses approches ont tenté de diminuer la complexité de recherche de l'automate cible. Une façon d'aborder le problème consiste à limiter la recherche à une sous-classe de représentation de langage, prouvée être identifiable en temps polynômial : ces méthodes sont dites *caractérisables*.

De la Higuera donne un résultat générique qui permet de fixer le cadre de l'apprenablilité des grammaires en prenant en compte la complexité spatiale et temporelle relativement à la taille du concept à inférer :

Définition 23 ([Higuera (de la), 1997]) *Une classe de représentation $\mathcal{R}$ est polynômialement identifiable à partir de données fixées si et seulement si il existe deux polynômes p et q et un algorithme A tel que :*

[9] ce théorème n'est pas en contradiction avec le résultat énoncé dans la Définition 21 : un AFD est identifiable polynômialement si l'*échantillon caractéristique* est présent dans I_+.

1. *Étant donné un échantillon* (S_+, S_-) *de taille* m*, A renvoie une représentation* R *dans* $\mathcal{R}$ *compatible avec* (S_+, S_-) *en temps inférieur à* $p(m)$*.*
2. *Pour toute représentation* R *de taille* n*, il existe un échantillon caractéristique* (CS_+, CS_-) *de taille au plus* $q(n)$*, pour lequel, sur les données* (S_+, S_-) *avec* $S_+ \supseteq CS_+$ *et* $S_- \supseteq CS_-$ *A retourne une représentation* $R^{'}$ *équivalente à* R*.*

□

À notre connaissance, seuls la classe des automates à états finis déterminstes a été prouvée être polynômialement identifiable.

Ce résultat négatif fait que la plupart des approches pratiques sont fondées sur une information *a priori* qui guide la recherche de la solution : ce sont des méthodes *heuristiques*. Elles relâchent la contrainte d'identification exacte au profit d'une complexité temporelle raisonnable.

2.3 Évaluation de la qualité du modèle appris

2.3.1 Cas non-stochastique

Dans le cas où le modèle inféré n'est pas muni d'une distribution de probabilités, celui-ci peut uniquement statuer sur l'*appartenance* ou la *non-appartenance* d'une phrase donnée au langage qu'il décrit. Une évaluation triviale du modèle est alors le calcul du taux d'acceptation des données positives (adéquation du modèle au langage ciblé) et du taux de rejet des données négatives (mesure de la sur-généralisation). Comme nous l'avons déjà remarqué auparavant, cette dernière n'est pas toujours possible dans le cas réél. Lorsque l'ensemble d'apprentissage ne contient que des exemples positifs, une façon possible d'évaluer le résultat d'inférence est le calcul de la distance D entre l'ensemble de test I_+ et la grammaire apprise G : pour D petit, la généralisation du concept est bonne. Si l'on dispose du calcul de la distance entre une phrase et une grammaire (par exemple celui proposé dans le Chapitre 3), on peut définir D comme la somme des distances entre G et chaque élément de I_+. Cette méthode est l'affinement du calcul du taux de reconnaissance, malheureusement elle ne permet pas de contrôler la sur-généralisation. Le protocole d'évaluation des grammaires non-stochastiques fondé sur ce critère est décrit et utilisé dans la section 5.2.

Évaluation des méthodes d'apprentissage. Pour comparer des *méthodes d'inférence*, on a usuellement recours aux *bancs d'essai*. Il s'agit d'un ensemble de grammaires (de taille réduite) qui, pour une classe donnée de langage, contiennent chacune un problème particulier de modélisation. Le protocole d'évaluation consiste à inférer les grammaires sur les ensembles d'apprentissage (corpus de phrases) correspondants aux grammaires du benchmark, puis de calculer leur capacité à classer les exemples positifs et négatifs (si ces derniers ont été générés). Il n'y a pas de banc d'essai "standard" pour l'ensemble de méthodes existantes ; en inférence régulière, [Miclet, 1979], Tomita

[Tomita, 1982], suivis de Miclet et Gentile [Miclet & Gentile (de), 1994] puis de Dupont [Dupont, 1994] ont proposé un ensemble de langages réguliers "typiques", qui ont servi à comparer l'efficacité des plusieurs algorithmes [Dupont, 1994; Giordano, 1996; Higuera (de la) et al., 1996; Coste & Nicolas, 1998][10]. Le danger de la généralisation de l'emploi de tels benchmarks réside dans la tendance à trop les cibler par les nouveaux algorithmes, ignorant de fait leur comportement dans les cas pratiques (peu de données d'apprentissage, nombre de symboles terminaux important).

2.3.2 Cas stochastique

Pour évaluer la qualité d'une grammaire stochastique apprise on cherche idéalement à comparer la distribution de probabilités du langage qu'elle génére avec celle du langage ciblé. En pratique, on ne connaît pas la distribution visée : on l'approxime avec l'ensemble de test. La technique d'estimation consiste alors à calculer l'entropie croisée (cf. section 1.4) entre la distribution d'un ensemble généré par le modèle inféré et l'ensemble de validation. Deux points doivent alors pris en considération :

- Engendrer un ensemble de mots sur un alphabet donné en respectant une distribution de probabilités doit suivre un processus parfaitement aléatoire (au risque de fausser le résultat par la prise en compte de la distribution de la source aléatoire).

- La taille des ensembles doit être suffisamment grande pour que les informations soient statistiquement représentatives et suffisamment petite pour être manipulable en pratique. Récemment, Carrasco a proposé [Carrasco, 1997] une technique de calcul de l'approximation de l'entropie relative directement à partir des distributions. Cette méthode a l'avantage d'éviter la génération d'ensembles d'exemples. En contrepartie, elle ne s'applique qu'à des grammaires stochastiques régulières.

Le résultat habituellement fourni est la valeur de perplexité (pouvoir prédictionnel du modèle) que l'on calcule à partir de l'entropie croisée.

2.4 Les algorithmes d'inférence grammaticale

L'inférence des grammaires a fait objet des nombreux travaux depuis le résultat de Gold mentionné en introduction. En conséquence, ce domaine est pourvu d'un nombre important d'algorithmes d'apprentissage, qui ont recours à des techniques aussi diverses que les reséaux neuronaux, les algorithmes génétiques, etc. Lorsqu'on est amené à choisir une de ces méthodes, plusieurs critères de choix peuvent entrer en considération, comme par exemple la nature des données disponibles, la nature du langage ciblé ou

[10] Il convient de mentionner ici les compétitions d'apprentissage *Abbadingo* ([Lang & Pearlmutter, 1997]), suivi de *Gowachin* ([Lang et al., 1998]), dont le but consiste à identifier des grammaires régulières qui ont servi à produire des données d'apprentissage. La complexité des constructions syntaxiques varie d'une grammaire à l'autre.

la tâche à accomplir par le modèle. Une classification possible est présentée ci-dessous (les méthodes citées se limitent au domaine de l'inférence régulière).

Inférence avec et sans échantillon négatif. Les contre-exemples sont utilisés pour limiter l'espace de recherche et éviter la sur-généralisation. Cependant en TALN on dispose rarement de données négatives, car la notion du "contre-exemple" dans la langue naturelle nécessite une définition : on sait que ne sont vraiment utiles que des séquences (contre-exemples) "proches" du langage ciblé (les *near miss*).

Parmi les méthodes qui utilisent un échantillon négatif citons la plus représentative : RPNI[11], développée par Oncina et Garcia [Oncina & Garcia, 1992] et indépendamment par Lang [Lang, 1992] et qui répond au critère de l'identification à la limite. Cet algorithme, qui procède par fusions successives d'états à partir du PTA (cf. section 2.2.2), a été le sujet de nombreuses études du fait des propriétés fortes des modèles qu'il infére. Il possède plusieurs versions et extensions (incrémentale : [Dupont, 1996], probabiliste : [Carrasco & Oncina, 1994]). Un autre algorithme RIG [Miclet & Gentile (de), 1994] est fondé sur le parcours de l'espace de recherche présenté dans la section 2.2.2, mais il n'est pas raisonnable de l'utiliser en pratique. Citons également l'algorithme GIG [Dupont, 1994], qui voit le problème de l'inférence grammaticale comme un problème d'optimisation et y applique un algorithme génétique.

Les méthodes qui infèrent sans échantillon négatif sont assez nombreuses, peut-être en raison de la plus grande disponibilité des données. Pour la plupart, elles utilisent des connaissances *a priori* et de ce fait il n'existe pas d'algorithme représentatif dans cette famille. Un algorithme : ECGI[12], basé sur les techniques de correction d'erreurs sera décrit en détail dans le Chapitre 4.

Les approches caractérisables *versus* les approches empiriques. Rappelons qu'une méthode est *caractérisable* si on peut démontrer qu'elle identifie à coup sûr un automate donné, à condition que dans les données d'apprentissage se trouve un certain ensemble suffisant de phrases. Cet ensemble "caractéristique" dépend de l'algorithme et de l'automate cible. Une méthode caractérisable qui n'utilise que des exemples positifs ne peut que converger sur des automates définissant une sous-classe stricte des langages réguliers. En opposition, les approches empiriques, bien que souvent basées sur des éléments théoriques, proposent des solutions qui ne sont pas généralement caractérisables théoriquement. D'un point de vue pratique, une méthode caractérisable ne présente pas d'avantage particulier sur une méthode empirique, puisque l'automate cible est par définition inconnu. Mais il est important pour un concepteur d'algorithme de pouvoir le qualifier par une telle propriété de convergence. L'algorithme RPNI, cité plus haut, appartient à la famille des méthodes caractérisables, l'algorithme ECGI est une approche empirique.

[11] de l'anglais : *Regular Positive Negative Inference.*
[12] de l'anglais : *Error Correcting Grammar Inference.*

Les méthodes probabilistes *versus* les méthodes non-probabilistes. Ici il s'agit de décider de l'emploi du modèle appris : lorsqu'il est nécessaire de disposer d'une distribution de probabilités sur le langage modélisé, il convient de se tourner vers une méthode qui infère des grammaires dont les règles sont munies de probabilités. On peut alors distinguer deux approches : celle dont la construction du modèle est guidée par une certaine information statistique contenue dans le corpus (comme par exemple dans l'algorithme ALERGIA [Carrasco & Oncina, 1994]) et celle qui construit le modèle suivant un autre critère, puis utilise les fréquences d'occurrences des règles apprises pour fournir une distribution de probabilités (c'est le cas de la version stochastique de l'algorithme ECGI). Remarquons qu'il est également possible de probabiliser une grammaire non-stochastique à partir du corpus d'apprentissage : il faut dans ce cas préciser le traitement des règles non utilisées lors du décodage des séquences de ce corpus (cas qui peut se présenter par principe de généralisation, notamment dans les grammaires non-déterministes).

2.5 Conclusion

Nous avons vu que l'inférence grammaticale et en particulier l'inférence régulière, est pourvue d'un cadre formel important et possède plusieurs résultats théoriques qui, pour la plupart, prouvent la complexité calculatoire du domaine. Dans la majorité des cas, les validations des méthodes d'apprentissage sont faites sur des langages comportant un petit nombre de symboles, avec des ensembles d'apprentissage supposés suffisamment grands et (pour les algorithmes qui les nécéssitent) la présence de l'échantillon négatif. Dans une tâche réelle, en particulier en TALN, les conditions d'apprentissage sont souvent inverses : la taille de l'alphabet Σ est grande, les ensembles d'apprentissage sont petits par rapport à $|\Sigma|$ et les exemples négatifs sont indisponibles. Dans ce genre de situation, les méthodes stochastiques sont généralement privilégiées, d'autant qu'elles peuvent bénéficier des apports des techniques d'affinement utilisées pour des modèles de langage plus simples, comme la réestimation des probabilités (algorithme "EM" [Dempster et al., 1977], en particulier sous la forme de l'algorithme de Baum-Welsh [Baum, 1972] utilisé pour les HMM) ou le *lissage* que nous aborderons plus loin dans ce document — en particulier dans le Chapitre 5.

Chapitre 3

Méthodes de correction d'erreurs pour les Séquences et Langages Formels

Dans ce chapitre nous présentons le cadre global des *méthodes de comparaison* et de *correction d'erreurs* pour des séquences. La première partie du chapitre introduit les notions fondamentales ainsi que les principales applications. La section qui suit expose son cadre formel et ses mécanismes essentiels. La troisième et dernière partie est consacrée plus particulièrement aux algorithmes utilisés pour comparer des séquences entre elles, pour comparer une séquence et un ensemble de séquences modélisé par un DAG[1] ou par un Automate à États Finis, ou encore pour comparer deux DAG. Les notions présentées ici sont une introduction au chapitre suivant consacré à l'apprentissage des modèles de langage par les techniques de correction d'erreurs.

3.1 Introduction

Il est souvent nécessaire de comparer deux ou plusieures séquences, vecteurs, fonction continues etc. pour mesurer l'importance de leur différence. Les méthodes de comparaison des séquences sont utilisées pour la reconnaissance des formes, la correction d'erreurs et pour l'identification des relations. Ces méthodes sont divisées en deux catégories majeures : discrètes et continues. Les premières traitent des séquences composées à partir d'un alphabet fini, par exemple les symboles des acides aminés, les caractères d'une table ASCII ou les phonèmes d'une langue. Dans le second cas (continu), les objets étudiés sont des signaux (par exemple les résultats de la chromatographie des gaz ou le signal de la parole), éventuellement multidimensionels. Au cours du traitement informatique, le temps est naturellement discrétisé (*digitalisé*), le signal est donc transformé en une séquence dont l'alphabet est composé d'un nombre *a priori* infini d'éléments (en réalité : seulement un nombre fini, souvent très grand, après quantifica-

[1] de l'anglais : *Directed Acyclic Graph* — un graphe orienté acyclique.

tion).

En pratique, les méthodes de comparaison des séquences trouvent leur emploi dans un ensemble de domaines très hétérogènes. La biologie moléculaire par exemple, étudie des longues séquences de protéines et d'acides nucléiques. La comparaison des séquences codant l'information génétique (ADN ou ARN) permet, entre autres, d'aiguiller les réponses à des questions concernant l'*homologie* des macromolécules (recherche d'une molécule ancêtre commune, recherche de leur degré de ressemblance). Dans le traitement de la parole, un signal analogique après échantillonage peut être vu comme une séquence et comparé ainsi à d'autres échantillons afin de trouver le mot prononcé. La comparaison des séquences permet également de détecter les endroits où les mots sont compressés ou tronqués par une mauvaise prononciation. Les sciences informatiques sont naturellement consommatrices des techniques de correction d'erreurs, notamment lors de l'analyse des données entrées par un utilisateur humain (commandes clavier, code des programmes, comparaison des fichiers etc.), ou encore dans les techniques liées aux transmissions des données à travers les réseaux (élimination du bruit, vérification des codes). Parmi d'autres applications, citons le cas du traitement de texte qui utilise ces méthodes lors d'une vérification orthographique élémentaire, où les mots écrits sont successivement comparés à un ensemble de mots contenus dans la base lexicale du programme.

On constate plusieurs degrés de complexité dans la comparaison. Il existe parfois une correspondance naturelle entre les éléments, composants, coordonnées, points du temps etc. dans l'une et l'autre séquence (en particulier l'ordre et la position) et dans ce cas il est facile d'en effectuer la comparaison, en ne considérant que des éléments concernés. Ainsi, les méthodes les plus simples pour comparer les séquences ne s'appliquent qu'entre les séquences de même longueur. Un exemple en est la *distance de Hamming* ; son calcul consiste à compter le nombre d'éléments différents, comparés successivement deux à deux selon leur rang dans la séquence (cf. FIG. 3.1).

rang	1	2	3	4	5	6	7	8	9	10	11
	p	r	é	p	o	s	i	t	i	o	n
	p	r	é	r	o	g	a	t	i	v	e
nombre cumulé de différences	0	0	0	1	1	2	3	3	3	4	5

$$d_{Hamming}(\texttt{préposition},\texttt{prérogative}) = 5$$

FIG. 3.1: Distance de Hamming entre les mots "préposition" et "prérogative".

Des comparaisons plus complexes sont requises quand la correspondance n'est pas connue d'avance, par exemple à cause d'une perturbation survenue par le gain ou par la perte d'éléments dans l'une ou dans les deux séquences. L'approche générale utilisée dans ce cas consiste à rechercher la meilleure correspondance en optimisant toutes les correspondances possibles qui respectent des conditions imposées, comme par exemple

la préservation de l'ordre des éléments dans la séquence.

Il convient donc de définir exactement ce que l'on entend par le mot "correspondance" appliqué aux éléments des séquences. Usuellement, il s'agit de définir des opérations qui peuvent s'appliquer sur ces éléments, influant sur leur présence ou modifiant leur ordre dans la séquence. Dans le cadre de la *théorie de la comparaison des séquences* [Kruskal & Sankoff, 1983] on se limite à l'utilisation des opérations suivantes :

- *substitution*
- *suppression* et *insertion*
- *compression* et *expansion*
- *permutation* de deux éléments

Elles peuvent être considérées comme universelles, même si leur utilisation varie énormement d'une situation à une autre. Lors de la comparaison de deux séquences, le type de différence le plus évident et le plus courant est la *substitution* (ou le *remplacement*) d'un élément par un autre à la même place dans la séquence. Les *suppressions* et les *insertions* sont indispensables lorsqu'on compare des séquences de longueur différente. Les *compressions* et les *expansions* sont essentiellement liées à des séquences obtenues par l'échantillonnage du temps, à partir d'une fonction continue. Finalement, les *permutations* (en anglais : *swaps*) ont été introduites dans [Lowrance & Wagner, 1975] pour traiter en particulier les erreurs de saisie. L'exemple de la Fig. 3.2 fournit une illustration d'utilisation successive de chacune des opérations.

mot *source* : **arrivage**, mot *cible* : **mirages**

1. la substitution de la lettre **a** par la lettre **m** donne : **m***rrivage*
2. la suppression de **v** donne : *mrriage*
3. l'insertion de **s** donne : *mrriage***s**
4. la compression de **rr** en **r** donne : *m***r***iages*
5. la permutation de **r** et **i** donne : *m***ir***ages*

Fig. 3.2: Transformation du mot "arrivage" en mot "mirages".

Il est maintenant possible de définir la notion de ressemblance entre les séquences comme un cumul des opérations citées plus haut : étant données deux séquences il existe toujours un ensemble d'opérations qui, appliquées successivement à partir de la première, permettent de la transformer et d'obtenir la deuxième. Le nombre minimum d'opérations, ou la somme des valeurs qu'on peut leur attribuer, constitue alors la *mesure de ressemblance* ou encore la *distance* entre les deux séquences. Les méthodes algorithmiques présentées dans la suite de ce chapitre ont pour but de déterminer la nature et l'ordre des opérations afin d'obtenir une suite optimale, en termes de nombre ou en termes de coût.

3.2 Définitions et notations

3.2.1 L'espace des séquences

Certaines des notions présentées dans cette section figurent dans le Chapitre 1 et sont rappelées ici, par souci de cohérence.

Définition 24 *Un* alphabet Σ *est un ensemble fini de symboles :* $\Sigma = \{a, b, \ldots, n\}$. *Sa taille (le nombre de ses éléments) est notée* $|\Sigma|$. □

Notons que les développements ultérieurs de ce chapitre ne nécessitent pas explicitement que l'on travaille avec des alphabets finis.

Définition 25 *Une* phrase, *ou* séquence[2] *sur* Σ *est une suite dénombrable d'éléments de* Σ, *notée par la* concaténation *(la juxtaposition de gauche à droite) de ses éléments. La* longueur n *d'une phrase* x *est le nombre de symboles qui la composent ; on la note* $|x|$. *La notation* x *peut se développer en :* $x(1:n)$ *ou* $x(1:|x|)$, *ou encore, en précisant le nom des symboles qui la composent, en* $x = (x_1, \ldots, x_n)$. □

Exemple 1 : Soient deux phrases sur l'alphabet $\Sigma = \{0, 1\}$: $x = 00011$ et $y = 001100$. On a : $|x| = 5$ et $|y| = 6$. □

Définition 26 *L'ensemble des phrases sur* Σ *se note* Σ^*. *Il comporte un élément particulier, celui dont la longueur est nulle, noté* ϵ *: la* phrase vide. Σ^* *possède une opération interne associative, l'extension de la* concaténation, *dont* ϵ *est l'élément neutre.* □

Remarque 3 *Le symbole* ϵ *sera également employé dans ce document pour désigner une* lettre vide. □

Exemple 2 : La concaténation de x et de y, notée simplement xy, produit la phrase : $xy = 00011001100$. □

On voit que $xy \neq yx$, ce qui traduit le fait que cette opération n'est pas commutative. Notons de plus que : $\forall x \in \Sigma^*,\ x\epsilon = \epsilon x = x$.

Définition 27 z *est une* sous-phrase *de la phrase* x *s'il existe* u *et* t *dans* Σ^* *avec :* $x = uzt$. *On peut noter :*

$$z = x(|u| + 1 : |u| + |z|)$$

[2] on dit aussi un *mot*, une *chaîne*, ...

en particulier, on a :

$$x(i:i) = x_i$$

Quand $u = \epsilon$, z *est un* préfixe *de* x. *Il peut donc s'écrire :*

$$z = x(1:|z|)$$

□

Exemple 3 : La phrase $z = 0001 = x(1:4)$ est un préfixe de $x = 00011$. □

3.2.2 Analyse des différences entre les séquences

L'objectif de cette section est de présenter les analyses qui mènent à une définition de différence (ou de similitude...) entre deux séquences par la construction d'une liste de différences *élémentaires* (que nous avons également appelées les *opérations* en introduction) entre les éléments de ces séquences. Dans cette approche, nous nous limitons aux opérations de *substitution*, de *suppression* et d'*insertion* que nous appellerons désormais les *opérations élémentaires*. Ces trois types d'opérations sont suffisants pour transformer, pas à pas, une séquence *source* en une séquence *cible*. Une suite d'opérations ainsi construite sera nommée un *listing*.

Notations

Pour représenter les trois opérations d'édition, nous utiliserons le signe $\rightarrow$ qui aura une signification de *remplacement*. Ainsi :

- la *substitution* d'un symbole a par un symbole b sera notée : $(a \rightarrow b)$;
- la *suppression* d'un symbole a sera notée : $(a \rightarrow \epsilon)$ (remplacement de a par la lettre vide ϵ) ;
- l'*insertion* d'un symbole a sera notée : $(\epsilon \rightarrow a)$ (remplacement de ϵ par a).

Trace

La *trace* d'une séquence source a vers une séquence cible b, comme l'illustre la Fig. 3.3 (avec a = "voiture" et b = "véhicule"), est une représentation graphique des opérations à appliquer afin de transformer a en b. Les présences et les absences des traits entre les éléments ont des significations précises :

1. deux éléments identiques reliés par un trait symbolisent une *identité* ;
2. deux éléments différents reliés par un trait symbolisent une *substitution* ;
3. l'élément de la séquence a qui n'est relié à aucun trait symbolise une *suppression* ;
4. l'élément de la séquence b qui n'est relié à aucun trait symbolise une *insertion*.

Si on se limite aux trois opérations citées plus haut, les traits sont soumis aux deux contraintes suivantes :

1. un élément ne peut être relié qu'à un seul trait ;
2. les traits ne peuvent se croiser.

Dans les cas où l'on accepte d'autres opérations élémentaires, des traits (qui gardent les mêmes significations) peuvent se croiser localement dans le cas d'une permutation, ou être deux ou plus à "partir" ou à "arriver" sur un élément dans le cas d'une compression ou d'une expansion. Des exemples en seront donnés dans le paragraphe 3.3.2.

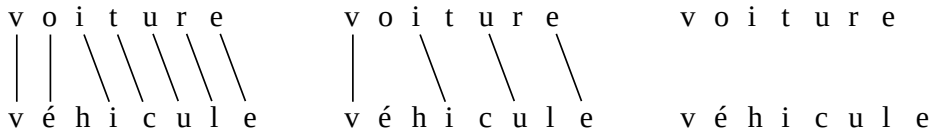


FIG. 3.3: Traces possibles entre les mots "voiture" et "véhicule".

Dans l'exemple présenté sur la FIG. 3.3 la première trace (à gauche) transforme le mot "voiture" en mot "véhicule" avec un nombre d'opérations élémentaires minimal. La trace de droite signifie que l'on supprime toutes les lettres du premier mot et on insère toutes les lettres du second.

Listing

Le *listing* entre une séquence source a et une séquence cible b est une suite de séquences et d'opérations élémentaires. Dans cette suite, deux séquences consécutives ne diffèrent que d'une opération élémentaire. Le listing de la FIG. 3.4 correspond à la première trace de la FIG. 3.3.

v o i t u r e

$(o \to$ é$)$: substitution de 'o' par 'é'

v é i t u r e

$(\epsilon \to h)$: insertion de 'h'

v é h i t u r e

$(t \to c)$: substitution de 't' par 'c'

v é h i c u r e

$(r \to l)$: substitution de 'r' par 'l'

v é h i c u l e

FIG. 3.4: Listing entre les mots "voiture" et "véhicule".

Remarquons qu'une trace peut correspondre à plusieurs listings. En effet l'ordre des opérations n'a pas d'importance dans une trace.

Comme nous l'avons vu, il peut y avoir plusieurs analyses différentes entre deux séquences. Elles varient dans l'ordre et en nombre d'opérations élémentaires à appliquer pour effectuer la transformation. Il est naturel de chercher à réduire ce nombre afin d'avoir une meilleure vision de la différence globale entre deux séquences. Ce constat nous mène à la définition suivante :

Définition 28 *La* différence *entre deux séquences a et b est la caractéristique du listing contenant le nombre minimal d'opérations élémentaires à appliquer pour transformer a en b.* □

Nous verrons dans la section suivante comment munir cette notion d'une valeur numérique.

3.3 Distances entre séquences

Nous avons approché intuitivement la notion de *distance* entre les séquences : la Définition 28 permet directement de définir cette notion comme le nombre d'opérations élémentaires nécessaires pour transformer la séquence source en séquence cible. Nous verrons qu'il est possible d'affiner cette approche, en affectant des *poids* (ou des *coûts*) à des opérations de transformation. De plus, lors de la comparaison d'une séquence et d'un ensemble de séquences il est souvent utile, voir imposé, d'organiser cet ensemble en un graphe orienté acyclique. Dans certains cas il est même utile de comparer deux ensembles représentés sous forme de graphes. Les algorithmes qui permettent d'effectuer ces comparaisons diffèrent peu et partagent le même principe fondé sur une relation de programmation dynamique.

3.3.1 La distance d'édition

La méthode que nous présentons dans ce paragraphe a été découverte plus ou moins indépendamment par de nombreux chercheurs autour du début des années 70, intéressés par la correction orthographique, la reconnaissance de la parole, la comparaison de codes génétiques, etc... Elle a profité ultérieurement d'un grand nombre d'améliorations ; nous la présentons comme une technique "canonique", simple, mais riche de toutes les notions nécessaires. On connaît ce calcul sous divers noms (correspondant éventuellement à des variantes) : distance de *Levenshtein*, distance de *Wagner et Fisher*, distance d'*édition*.

Notions de base

Il faut d'abord définir une fonction réelle positive ζ sur $\Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\}$, qui peut s'interpréter comme le "coût" de remplacer une lettre par une autre, en généralisant ce remplacement à la suppression et à l'insertion.

Exemple 4 : Le remplacement dans la chaîne $x = 00011$ du caractère $x_3 = 0$ par le

caractère 1 donnera la chaîne 00111, avec un coût de $\zeta(0 \to 1)$. La suppression dans x de x_4 produira la chaîne 0001, pour un coût de $\zeta(1 \to \epsilon)$. L'insertion juste après le caractère x_2 du caractère 0 produira : 000011, pour un coût de $\zeta(\epsilon \to 0)$. □

Une *opération élémentaire* de modification d'une phrase consiste donc à appliquer une fois la fonction ζ. Ceci peut être répété ; il est facile de voir qu'il suffit d'appliquer une suite finie d'opérations élémentaires à une phrase x pour la transformer en toute autre phrase y désirée. Il est également facile de voir qu'il existe une infinité de façons de transformer de la sorte x en y. Ces deux remarques conduisent à la définition suivante :

Définition 29 *Soit une suite d'opérations élémentaires permettant de transformer la phrase x en la phrase y. Le* coût *de cette transformation est défini comme la somme des coûts des opérations élémentaires de la suite.*

La distance d'édition $D(x, y)$ *est le coût de la transformation la moins coûteuse permettant de passer de x à y par une suite d'opérations élémentaires.* □

Calcul

Nous allons donner sans démonstration les formules permettant de calculer cette valeur (un algorithme simple s'en déduit immédiatement) et nous justifierons (mais toujours sans démonstration) son appellation de "distance".

Soient deux phrases $x(1:n)$ et $y(1:m)$. Le calcul de $D(x, y)$ s'opère en remplissant une matrice $M(0:n, 0:m)$ de taille $(n+1) \times (m+1)$, dont l'élément courant est calculé de manière à ce que :

$$M(i, j) = D(x(1:i), y(1:j))$$

La puissance de l'algorithme vient du fait que le calcul de $M(n, m)$, qui est la valeur $D(x, y)$ cherchée, se fait par une récurrence de *programmation dynamique* : le nombre de calculs à faire est alors simplement de l'ordre de $n \times m$.

Les formules utilisées sont les suivantes :

$$M(0, 0) = 0$$

$$M(i, 0) = \Sigma_{k=1}^{k=i} \zeta(x_k \to \epsilon) \; pour \; 1 \leq i \leq m$$

$$M(0, j) = \Sigma_{k=1}^{k=j} \zeta(\epsilon \to y_k) \; pour \; 1 \leq j \leq n$$

$$M(i, j) = min \begin{cases} M(i-1, j) + \zeta(x_i \to \epsilon) \\ M(i, j-1) + \zeta(\epsilon \to y_j) \\ M(i-1, j-1) + \zeta(x_i \to y_j) \end{cases}$$

Remarque 4 *Au sens mathématique du terme une "distance"* Δ *sur un espace* $E \times E$ *est une application de* $E \times E$ *dans* $\mathbb{R}^+$ *qui vérifie les conditions :*

1. $\Delta(x, y) = 0 \Longleftrightarrow x = y$
2. $\forall x, y \in \Sigma, \Delta(x, y) = \Delta(y, x)$ *(symétrie)*
3. $\forall x, y, z \in \Sigma, \Delta(x, y) \leq \Delta(x, z) + \Delta(z, y)$ *(inégalité triangulaire)*

Au cours des travaux exposés dans ce mémoire, nous ne nous sommes pas toujours restreints à cette stricte définition, en particulier en ce qui concerne les points 2. et 3. Les causes en sont les études empiriques décrites dans le Chapitre 4 et les suivants.

Une caractéristique forte de cette méthode est la propriété suivante :

Théorème 7 ([Wagner & Ficher, 1974]) *Si* ζ *est une distance sur* $\Sigma \cup \{\epsilon\} \times \Sigma \cup \{\epsilon\}$*, alors* D *est une distance sur* $\Sigma^* \times \Sigma^*$*.* □

Remarque 5 *Lorsque la fonction* ζ *est munie des* coûts unitaires *c'est-à-dire est de forme :*

$$\forall a, b \in \Sigma, a \neq \epsilon, b \neq \epsilon \quad \begin{array}{ll} \zeta(a \to b) & = \begin{cases} 0 & si\ a = b \\ 1 & si\ a \neq b \end{cases} \\ \zeta(\epsilon \to a) & = 1. \\ \zeta(a \to \epsilon) & = 1. \end{array}$$

il est facile de voir que la distance calculée est également le nombre *minimal d'opérations élémentaires nécessaires pour transformer une séquence en une autre.* □

3.3.2 Quelques améliorations

Si l'on veut introduire d'autres possibilités de transformations élémentaires, en plus de la substitution d'un caractère à un autre, de la destruction et de l'insertion d'un caractère, il faut les choisir avec le souci de conserver les aspects théoriques et algorithmiques de la méthode de base. Une extension correcte autorise la permutation de deux caractères voisins : la phrase $x = (x_1, \ldots, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \ldots, x_n)$ devient ainsi : $(x_1, \ldots, x_{i-1}, x_{i+1}, x_i, x_{i+2}, \ldots, x_n)$, pour un certain coût. Un algorithme de programmation dynamique permet alors de calculer la transformation la moins coûteuse de toute phrase x en toute phrase y. Notons qu'ici ζ doit être la distance "triviale" (à coûts unitaires) décrite ci-dessus et que le coût de la permutation doit lui-même être indépendant des caractères sur lesquels il porte.

Comparaison Dynamique

La distance d'édition utilise le procédé de la Programmation Dynamique, lui-même dérivé du *Principe d'Optimalité* de Bellman, dont un énoncé informel est le suivant :

Toute sous-politique d'une politique optimale est elle-même optimale.

Dans l'algorithme précédent, ceci se concrétise par le calcul des valeurs $M(i,j) = D(x(1:i), y(1:j))$ comme les points de passage vers $D(x,y)$ qui ne seront jamais remis en question. On peut imaginer de chercher des contraintes de transformation différentes de celles de la distance d'édition, permettant néanmoins d'appliquer une technique de Programmation Dynamique, dont l'intérêt majeur est la rapidité de calcul. C'est à partir de cette idée que les méthodes de comparaison entre séquences dites de *comparaison dynamique* ont été imaginées. Il en existe un grand grand nombre de variantes, conçues pour la plupart de manière *ad hoc* pour résoudre un problème spécifique. Nous décrivons ici une variante d'un algorithme qui a beaucoup été utilisé en reconnaissance de la parole avant d'être détrôné par les techniques statistiques.

L'idée de base est donc de trouver le meilleur "couplage"[3] entre deux séquences qui représentent l'élocution de deux mots ; si les mots sont phonétiquement identiques, il n'en existe pas moins des différences acoustiques dues aux caractéristiques du locuteur, à la vitesse d'élocution, etc... Telle voyelle peut être allongée pour un effet d'insistance, telle fin de mot modifiée ou même une erreur d'articulation peut se produire. La distance entre deux signaux, qui sont les traces sonores de deux mots, doit donc tenir compte le moins possible des effets d'allongements et de raccourcissement locaux.

Exemple 5 : Pour illustrer ce problème, notons (très symboliquement) `a` le son correspondant à cinq centièmes de seconde de la voyelle `/a/`, en supposant que seule sa longueur varie (mais qu'il reste acoustiquement identique ; notons de même `s` le son correspondant à cinq centièmes de seconde de la voyelle `/s/`. Deux élocutions du mot "sas" seront codées par exemple comme : $x =$`ssaas` et $y =$`sassss`. Idéalement, puisqu'il s'agit de deux exemples de la même classe, leur distance devrait être nulle ; ce ne serait pas le résultat fourni par la distance d'édition. □

On utilise donc ici un autre jeu de transformations entre lettres. On interdit les suppressions et les omissions, mais on autorise une lettre à se transformer en plusieurs autres, situées en séquence (par le biais des opérations de *compression* et d'*expansion*) ; le coût d'une telle transformation composée est naturellement la somme de ceux des transformations élémentaires qui la composent. Pour calculer la distance entre deux phrases x et y, on va donc construire une suite de couples d'indices, le premier parcourant sans omission la phrase x en croissant au sens large, le second la phrase y.

Exemple 6 : Sur l'exemple précédent : $x =$`ssaas`, $y =$ `sassss`, la suite de couples la plus évidente (et la meilleure) est la suivante :

$$\{(1,1),(2,1),(3,2),(4,2),(5,3),(5,4),(5,5),(5,6)\}$$

Elle correspond à une transformation naturelle, puisque dans ce cas particulier chaque couple d'indices associe deux lettres identiques : par exemple, le couple $(5,3)$ correspond au couplage de x_5=`s` avec $y_3 =$ `s`. Au total, la distance entre ces deux phrases est donc nulle. Notons que la suite des indices dans x, ici $\{1,2,3,4,5,5,5\}$ et celle des

[3] en anglais : *matching*.

indices dans y : $\{1, 1, 2, 2, 3, 4, 5, 6\}$ répondent bien à la contrainte énoncée ci-dessus : elles sont croissantes. La trace qui correspond à cette transformation est représentée sur la FIG. 3.5. On remarquera que dans ce cas, plusieurs traits partent d'une même lettre. □

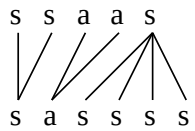


FIG. 3.5: Une trace avec des compressions et des expansions.

Il reste à définir le coût d'une transformation élémentaire, représentée par un couple d'indices. Dans le cas le plus simple, on peut se contenter de la distance ζ définie entre les lettres de l'alphabet. Mais en général on la pondère par un coefficient α calculé en considérant le couple d'indices où l'on se trouve au moment du calcul en fonction de celui ou ceux qui précèdent. Ceci apparaîtra ci-dessous dans la formule récurrente du calcul.

Il est aisé de constater que le principe d'optimalité s'applique encore ; dans le cas le plus simple, l'algorithme de comparaison dynamique se ramènera à la construction d'une matrice M, ayant la même définition que celle de l'algorithme de la distance d'édition : $M(i, j) = D(x(1 : i), y(1 : j))$; en général, la dimension $n \times m$ sera suffisante.

Dans ce cas, on aura les formules :

$$M(1, 1) = 0$$

$$M(i, 1) = \Sigma_{k=1}^{k=i} \zeta(x_k \to y_1) \; pour \; 1 \leq i \leq m$$

$$M(1, j) = \Sigma_{k=1}^{k=j} \zeta(x_1 \to y_k) \; pour \; 1 \leq j \leq n$$

$$M(i, j) = min \begin{cases} M(i-1, j) + \zeta(x_i \to y_j) \\ M(i, j-1) + \zeta(x_i \to y_j) \\ M(i-1, j-1) + \alpha \times \zeta(x_i \to y_j) \end{cases}$$

avec $\alpha > 1$, en général $\alpha = 2$.

Mais rien n'interdit de considérer des "contraintes de cheminement"[4] plus complexes, comme par exemple :

[4] ce terme provient de la représentation graphique classique des algorithmes de comparaison dynamique : on trace x en abscisse, y en ordonnée et leur association est un chemin continu entre les coordonnées $(1, 1)$ et (n, m), dont les limites sur la pente illustrent les équations de récurrence du calcul.

$$M(1,1) = 0$$

$$M(i,1) = \Sigma_{k=1}^{k=i} \zeta(x_k \to y_1) \; pour \; 1 \leq i \leq m$$

$$M(1,j) = \Sigma_{k=1}^{k=j} \zeta(x_1 \to y_k) \; pour \; 1 \leq i \leq n$$

$$M(i,j) = min \begin{cases} M(i-2,j-1) + \frac{1}{2}[\zeta(x_{i-1} \to y_j) + \zeta(x_i \to y_j)] \\ M(i-1,j-1) + \zeta(x_i \to y_j) \\ M(i-1,j-2) + \frac{1}{2}[\zeta(x_i \to y_{j-1}) + \zeta(x_i \to y_j)] \end{cases}$$

On obtient comme précédemment la valeur $D(x,y)$ en prenant la valeur de $M(n,m)$. Notons que D n'est pas une distance au sens rigoureux : dans l'exemple précédent, on avait par exemple $D(x,y) = 0$ pour $x \neq y$, ce qui violait la première condition donnée. L'inégalité triangulaire n'est évidemment pas assurée non plus et dans certains jeux de contraintes, le résultat du calcul n'est même pas symétrique. Il vaut mieux, plutôt que de distance, parler de *mesure de dissemblance* entre éléments de Σ^*.

3.3.3 Comparaison d'une séquence et d'un Graphe Orienté Acyclique

Lorsqu'il s'avère nécessaire de comparer une séquence et un ensemble de séquences, on peut bien évidemment utiliser l'algorithme précédent en le répétant sur tous les éléments de l'ensemble. Cependant, si cet ensemble de séquences doit servir de modèle (par exemple comme modèle d'un *langage fini*), il est commode et efficace de le réorganiser en un *graphe acyclique*. Les méthodes pour effectuer cette réorganisation peuvent être diverses : l'algorithme le plus simple consiste à factoriser dans un graphe les préfixes et les suffixes des séquences ; on peut également avoir recours aux techniques d'apprentissage (le Chapitre 4 en propose une) ou construire le modèle de l'ensemble manuellement, etc. L'algorithme qui permet alors de calculer la distance entre la séquence et l'ensemble est directement dérivé de l'algorithme précédent.

Avant d'écrire la relation de récurrence, nous avons besoin de définir un certain nombre de notions utiles à la description de l'algorithme :

Définition 30 *Un* graphe acyclique orienté, *construit sur un alphabet* Σ, *est un graphe orienté quelconque, ne contenant aucun cycle et dont les nœuds sont étiquetés par les lettres de* Σ. □

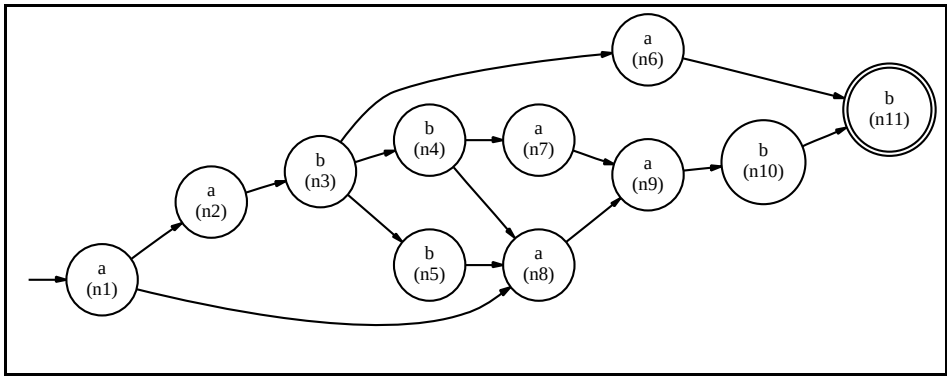


FIG. 3.6: Un graphe acyclique orienté. Les nœuds de ce graphe sont numérotés de 1 à 11 (notation (nx)) et étiquetés par les lettres a et b.

Définition 31

Un *successeur* *d'un nœud n est chaque autre nœud qui peut être atteint de n en empruntant un seul arc.*

Un *prédécesseur* *d'un nœud n est chaque autre nœud à partir duquel on peut atteindre n en empruntant un seul arc.*

Une *source* *est un nœud sans prédécesseurs.*

Un *puits* *est un nœud sans successeurs.*

□

Définition 32

Un *chemin* *dans un graphe est une suite $n^1 \ldots n^T$ de nœuds, telle que chaque nœud $n^i, i > 1$ est le successeur du nœud n^{i-1}.*

□

Définition 33

Une *séquence générée par un graphe* *est la suite des lettres qui étiquettent les nœuds d'un chemin entre une* source *et un* puits.

Une *séquence initiale* *est n'importe quel chemin partant d'une* source.

□

Définition 34

La densité maximale *d'un graphe G est le nombre maximal de prédécesseurs qu'un nœud de G peut posséder.*

□

Calcul

Soit une séquence $a(1:T)$ et un graphe acyclique orienté G dont les nœuds sont : $n_1, n_2, \ldots, n_K$. La distance entre un nœud n_j et un élément a_i est l'opération élémentaire qui permet de transformer l'étiquette (une lettre de l'alphabet Σ) de n_j en a_i. Nous notons par n_{j^*} tous les prédécesseurs du nœud n_j et par j^* tous les indices de ces nœuds. Pour initialiser l'algorithme, il faut adjoindre à a une lettre vide $a_0 = \epsilon$ et à G un noeud n_0, etiqueté par ϵ qui sera une source unique, prédécesseur de toute source de G. La relation de récurrence qui permet de remplir la matrice de programmation dynamique $M(0:T, 0:K)$ est déduite de la relation du paragraphe 3.3.1 :

$$M(i,j) = \min \begin{cases} M(i-1,j) + \zeta(a_i \to \epsilon) \\ \min_{j^*} M(i-1,j^*) + \zeta(a_i \to n_j) \\ \min_{j^*} M(i,j^*) + \zeta(\epsilon \to n_j) \end{cases}$$

Le calcul est initialisé avec :

$$M(0,0) = 0$$

$$M(i,0) = \Sigma_{k=1}^{k=i} \zeta(a_k \to \epsilon) \; pour \; 1 \leq i \leq T$$

$$M(0,j) = \Sigma_{k=1}^{k=j} \min_{k^*} \zeta(\epsilon \to n_{k^*}) \; pour \; 1 \leq j \leq K$$

Un pas particulier de cette récurrence peut être vu comme la recherche du chemin de coût minimal pour arriver du nœud n_0 au nœud n_j avec le préfixe $a(0:i)$. Le coût optimal est trouvé en parcourant la matrice M dans la colonne qui correspond à l'indice de la dernière lettre de a et pour tous les puits de G :

$$D(a,G) = \min_j M(K, j | n_j \in \{puits \; de \; G\})$$

La complexité de cet algorithme dépend de la taille de a, de celle de G (exprimée en nombre de nœuds) et de la densité maximale de G (notée ici : τ). Elle est de l'ordre : $\mathcal{O}(\tau T K)$.

L'exemple qui suit montre le calcul entre une séquence et le graphe de la FIG. 3.6.

Exemple 7 :
La séquence considérée est : $a = ababb$. On utilise les coûts unitaires pour effectuer le *calcul d'alignement* de a sur G .

$a(1:i)$	G (fig 3.6)										
	n1	**n2**	**n3**	**n4**	**n5**	**n6**	**n7**	**n8**	**n9**	**n10**	**n11**
$a(1:1) =$ **a**	0	1	2	3	3	3	4	1	2	3	4
$a(1:2) =$ **ab**	1	1	1	2	2	2	3	1	2	2	3
$a(1:3) =$ **aba**	2	1	2	3	3	1	2	1	1	3	2
$a(1:4) =$ **abab**	3	2	1	2	2	2	3	2	2	1	1
$a(1:5) =$ **ababb**	4	3	2	1	1	2	3	2	3	2	1

Ce tableau se lit : "il y a au moins $x(i,j)$ opérations d'édition à effectuer pour transformer le préfixe $a(1:j)$ en un préfixe du graphe G menant au nœud n_i".

L'alignement optimal du graphe G sur la séquence a est le chemin :

$$n1 \rightsquigarrow n8 \rightsquigarrow n9 \rightsquigarrow n10 \rightsquigarrow n11 \ (= aaabb)$$

On a alors :

$$D(a,G) = d(ababb, aaabb) = 1$$

□

Remarque 6 *On peut calculer de la même manière la distance entre une séquence et un automate fini sans cycle : il existe en effet une équivalence entre un automate valué sur les transitions uniquement et un automate valué sur les états uniquement. L'algorithme qui effectue la transformation entre les deux est connu (voir par exemple [Hopcroft & Ullman, 1979]).*

Exemple 8 : Les graphes des figures 3.6 et 3.7 sont équivalents. La transformation dans l'autre sens est moins triviale : pour "pousser" les étiquettes sur les nœuds il faut, dans le cas général, démultiplier ceux-ci.
NB : tous les arcs qui mènent au même nœud portent la même étiquette. □

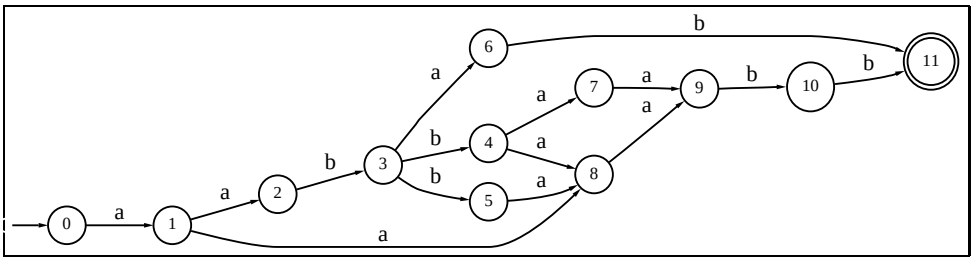


FIG. 3.7: Un graphe acyclique orienté valué sur les arcs.

3.3.4 Comparaison de deux DAG

Les approches présentées dans les sections précédentes ont superposé un modèle de langage et une séquence. Il peut cependant s'avérer nécessaire de traiter une suite de séquences formant un ensemble d'hypothèses et d'en déterminer une qui soit la plus proche du modèle. Ici encore, il est possible d'appliquer l'algorithme du paragraphe qui précède en traitant les séquences une par une. Il est cependant souvent commode de les organiser en un graphe d'hypothèses, par exemple par souci de place de mémoire,

d'efficacité de traitement, ou suite à une approche méthodologique (l'élimination des hypothèses faibles peut être faite en effectuant un élagage des sous-chemins du graphe). La FIG. 3.8 montre un graphe dont l'ensemble de chemins permet de générer quatre séquences : *abbaab, abbabb, abaab, ababb.* La méthode vue dans 3.3.3 peut être étendue à la comparaison de deux DAG ; le but de l'algorithme est alors de trouver un chemin dans chacun des deux graphes, de façon à ce que la distance entre les deux séquences correspondant à ces chemins soit minimale — selon la définition de la distance énoncée plus haut.

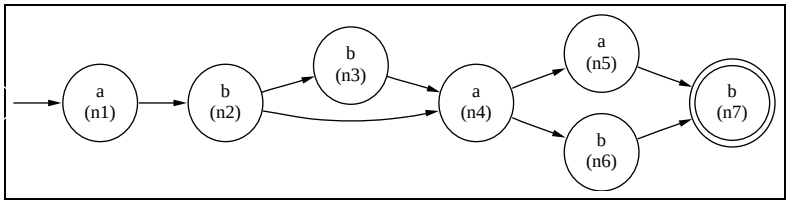


FIG. 3.8: Un graphe d'hypothèses.

Calcul

Soient deux graphes acycliques orientés G_1 et G_2 dont les nœuds sont notés respectivement : $n_1^{G_1}, n_2^{G_1}, \ldots, n_T^{G_1}$ et $n_1^{G_2}, n_2^{G_2}, \ldots, n_K^{G_2}$. Comme pour l'algorithme de comparaison d'une séquence et d'un DAG, il est nécessaire d'initialiser le traitement en ajoutant une source unique à chacun des deux graphes que l'on souhaite comparer. Soient $n_0^{G_1}$ et $n_0^{G_2}$ ces sources, respectivement pour G_1 et G_2. Alors la matrice de programmation dynamique $M(0:T, 1:K)$ est initialisée par :

$$M(0,0) = 0$$

et ses valeurs sont calculées par la relation :

$$M(i,j) = min \begin{cases} M(i^*, j) + \zeta(n_i^{G_1} \to \epsilon) \\ \min_{i^*} \min_{j^*} M(i^*, j^*) + \zeta(n_i^{G_1} \to n_j^{G_1}) \\ \min_{j^*} M(i, j^*) + \zeta(\epsilon \to n_j^{G_2}) \end{cases}$$

(ici, la notation confond le nœud et l'étiquette qu'il porte).

La distance entre les deux graphes est déduite de la matrice M. C'est la transformation du coût minimal calculée pour un chemin dans l'un des deux graphes :

$$D(G_1, G_2) = \min_i \min_j M(i|n_i^{G_1} \in \{puits\ de\ G_1\}, j|n_j^{G_2} \in \{puits\ de\ G_2\})$$

L'alignement entre les graphes peut être facilement retrouvée par le parcours à l'envers de la matrice M, en partant de la case qui contient la valeur de $D(G_1, G_2)$. Nous illustrons plus loin, dans l'Exemple 9, l'application du calcul de la distance entre le graphe

de la FIG. 3.6 et celui de la FIG. 3.8.

La complexité de l'algorithme dépend du nombre des nœuds dans chacun des graphes, ainsi que de la densité de ceux-ci. Notons par τ la densité maximale du graphe G_1 et par μ celle du graphe G_2. Alors la complexité est dominée par : $\mathcal{O}(\tau\mu TK)$.

Exemple 9 : Application de l'algorithme entre le graphe de la FIG. 3.8 et celui de la FIG. 3.6. Les *coûts unitaires* (cf. Remarque 5, page 51) ont été utilisés pour la fonction ζ :

G_2 (fig 3.8)	G_1 (fig 3.6)										
	n1	**n2**	**n3**	**n4**	**n5**	**n6**	**n7**	**n8**	**n9**	**n10**	**n11**
n1	0	1	2	3	3	3	4	1	2	3	4
n2	1	2	1	2	2	2	3	1	2	2	3
n3	2	2	1	1	1	2	2	2	2	2	2
n4	2	1	2	2	2	1	1	1	1	2	2
n5	3	2	2	3	3	2	2	2	1	2	2
n6	3	2	1	2	2	2	3	3	2	1	1
n7	4	3	2	1	1	3	3	3	2	1	1

Ce tableau se lit : "il y a au moins $M(i,j)$ opérations d'édition à effectuer pour transformer un préfixe du graphe G_1 menant au nœud n_i en un préfixe du graphe G_2 menant au nœud n_j".

Les chemins (séquences) les plus proches dans les graphes G_1 et G_2 sont respectivement :

$$n1 \rightsquigarrow n8 \rightsquigarrow n9 \rightsquigarrow n10 \rightsquigarrow n11 \ (= aaabb)$$

dans G_1 et

$$n1 \rightsquigarrow n2 \rightsquigarrow n4 \rightsquigarrow n6 \rightsquigarrow n7 \ (= ababb)$$

dans G_2. On a alors :

$$D(G_1, G_2) = d(aaabb, ababb) = 1$$

□

3.4 Distance entre une séquence et un Langage Régulier

Nous avons vu dans le Chapitre 2 que, étant donnés une séquence a et un modèle de langage formel (non-stochastique) M, tous les deux définis sur un alphabet Σ, il est possible d'affirmer ou d'infirmer l'appartenance de a au langage L modélisé par M.

Pour aller au-delà de cette analyse triviale, nous présentons ici des travaux dont le but est de déterminer la suite d'opérations élémentaires (éventuellement nulle si $a \in L$) qui permettent de transformer toute séquence composée sur Σ en la séquence la plus proche de a et reconnue par M. La complexité des algorithmes qui permettent de mener ce type d'analyse dépend de la complexité du langage selon la hiérarchie des langages établie par Chomsky. Dans [Kruskal & Sankoff, 1983] Wagner énonce le résultat suivant :

1. Si L ne comporte qu'une seule séquence b, le problème revient à comparer deux séquences. La complexité théorique de l'algorithme est alors : $\mathcal{O}(|a||b|)$.
2. Si L est un langage régulier, modélisé par un automate à états finis A, alors l'analyse corrective peut être menée en temps $\mathcal{O}(|a|P)$, où P est le nombre d'arcs de A (ou encore : P est le nombre de règles de production de la grammaire régulière $G(L)$).
3. Si L est un langage "context-free" et si la grammaire "context-free" $G(L)$ possède P règles de productions (modifiées de façon à ce qu'aucune d'entre elles n'ait, en côté droit, plus de deux symboles, par exemple par transformation en "forme normale de Chomsky" [Hopcroft & Ullman, 1969]) alors la complexité de l'analyse corrective est en $\mathcal{O}(|a|^3 P)$.

3.4.1 Algorithme de Wagner

Cette méthode, proposée dans [Wagner, 1974], permet de calculer la distance, exprimée en nombre d'opérations élémentaires, entre un langage régulier L modélisé par la grammaire G et une séquence a quelconque (définie sur le même alphabet que L). Il s'agit alors de déterminer le nombre minimal de modifications (*substitutions, insertions et suppressions*) à appliquer sur a afin de produire une séquence b qui soit acceptée par G (ou par son représentation équivalente, l'automate A).

On suppose donc être en présence d'un automate à états finis A, dont les états seront notés ici : q_i, q_j et numérotés de 0 à K, l'état de rang 0 étant initial et d'une séquence $a(1:T)$, tous les deux définis sur un alphabet Σ.

Un raisonnement de programmation dynamique assure alors que :

$$\Gamma(k, q_i) = \min_{q_j} \Gamma(k-1, q_j) + V(q_j, q_i, a(k))$$

et

$$\Gamma(0, q_i) = \begin{cases} 0 & \text{si } q_i \text{ est l'état initial de } A \\ \infty & \text{sinon} \end{cases}$$

avec :

$\Gamma(k, q_i)$ **:** nombre minimal d'opérations d'édition nécessaires pour transformer les k premiers caractères de a en une séquence b', acceptée partiellement par A (et qui fait de q_i l'état courant d'acceptation).

$V(q_j, q_i, a(j))$ **:** nombre minimal d'opérations d'édition nécessaires pour transformer le symbole $a(j)$ en une séquence $w(q_j, q_i)$ faisant de q_i l'état courant d'acceptation, atteint à partir de q_j, lui-même atteint au pas précedent de la récurrence.

Si on appelle $P(q_j, q_i)$ la longueur d'un des plus courts chemins (en nombre de transitions) entre q_j et q_i et $L(q_j, q_i, \alpha)$ un indice valant 1 si le symbole α appartient à un de ces chemins, 0 sinon, on a alors :

$$V(q_j, q_i, \alpha) = \begin{cases} 1 & \text{si } q_j = q_i \\ P(q_j, q_i) - L(q_j, q_i, \alpha) & \text{sinon} \end{cases}$$

La longueur du plus court chemin entre q_j et q_i peut se calculer par l'algorithme de Floyd, grâce à la récurrence :

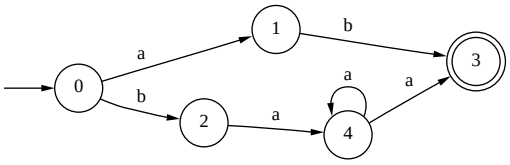
$$P^{l+1}(q_j, q_i) = \min\{P^l(q_j, q_i), P^l(q_j, l+1) + P^l(l+1, q_i)\}$$

où $P^l(q_j, q_i)$ est la longueur d'un plus court chemin entre q_j et q_i ne passant que par les états de rang inférieur ou égal à l (voir par exemple [Deo, 1974]).

Au cours du calcul de $P(q_j, q_i)$ on peut également déterminer la valeur de $L(q_j, q_i, \alpha)$. À la fin du calcul, si F est l'ensemble des états finals de l'automate, la distance cherchée entre a et cet automate vaut : $\min_{q_i \in F} \Gamma(T, q_i)$

Exemple 10 :

Automate A :



séquence $a = aaa$

Sous-phrase	État				
	0	**1**	**2**	**3**	**4**
$a(1:1) = a$	1	0	1	1	1
$a(1:2) = aa$	2	1	2	1	1
$a(1:3) = aaa$	3	2	3	1	1

$D(a, A) = 1$

□

Remarque 7 *Dans le même ordre d'idées, on peut, par un calcul sur l'expression régulière qui le représente, étendre un langage régulier pour lui permettre de comprendre des phrases entachées d'un certain type d'erreur. Les règles algébriques de ce calcul ont*

été proposés dans [Thomason, 1974]. Elles sont à la base des travaux présentés dans le Chapitre 4. □

Cette méthode s'applique de façon efficace sur les automates avec des cycles. Sa faiblesse vient du fait qu'il est impossible d'utiliser d'autres valeurs que l'unité pour la fonction ζ. Dans les cas où il convient de pondérer les opérations d'édition, il faut se tourner vers des méthodes soumises à certaines contraintes, ou dont la complexité calculatoire est plus importante.

3.4.2 Distance pondérée entre une séquence et un Automate Fini

Des travaux récents ([Bouloutas et al., 1991; Amengual & Vidal, 1998]) ont donné naissance à des méthodes permettant de calculer la distance entre une séquence et un automate à états finis quelconque (et notamment un automate contenant des boucles ou des cycles), avec les poids des opérations élémentaires affectées des valeurs réelles positives. Nous allons donner une brève description de ces méthodes dans ce paragraphe, en insistant plus particulièrement sur leur complexité temporelle respective. En effet il est intéressant de comparer la complexité d'un algorithme classique de décodage de séquences (dérivé de l'*algorithme de Viterbi*) avec celle des approches qui permettent en plus détecter les différences entre le modèle et la séquence qu'on lui présente.

Le problème de l'analyse d'une séquence a, étant donné un modèle de langage régulier (un automate à états finis A), peut être vu comme la recherche du chemin du coût minimal à travers le graphe acyclique G produit par l'algorithme de Viterbi. Les nœuds q_k^i de ce graphe correspondent aux états q^i atteints à l'étape k, associée au k-ième symbole de a. Chacun des arcs (q_k^i, q_{k+1}^j) est créé (et valué) par la transition empruntée entre l'état q^i, atteint avec a_k et l'état q^j (éventuellement identique au q^i dans A), atteint avec a_{k+1} (cf. Fig. 3.9a). Ce graphe peut être également créé lors de l'analyse corrective : les opérations de substitution et d'insertion seules permettent au graphe de conserver sa propriété acyclique (figures 3.9a et 3.9b). En utilisant en plus des opérations de suppression, on peut y voir apparaître des cycles, comme le montre la Fig. 3.9c.

Le problème se ramène alors à la recherche du chemin le moins coûteux dans un graphe *cyclique*. La solution proposée par [Boulotas Hart Schwartz 91] est résumée dans la relation de récurrence suivante :

$$C(q_{k+1}^j) = \min_{\substack{\forall i \in \delta^{-1} \text{ à l'étape } k \\ \forall l \text{ à l'étape } k+1}} \{C(q_k^i) + W(q_k^i, q_{k+1}^l) + \mathcal{C}(q_{k+1}^l, q_{k+1}^j)\}$$

avec :

- δ^{-1} : la fonction *inverse* de la fonction de transition δ de l'automate A.
- $C(q_k^i)$: le coût du *chemin du coût minimal* entre un état initial de A et l'état q^i à l'étape k.

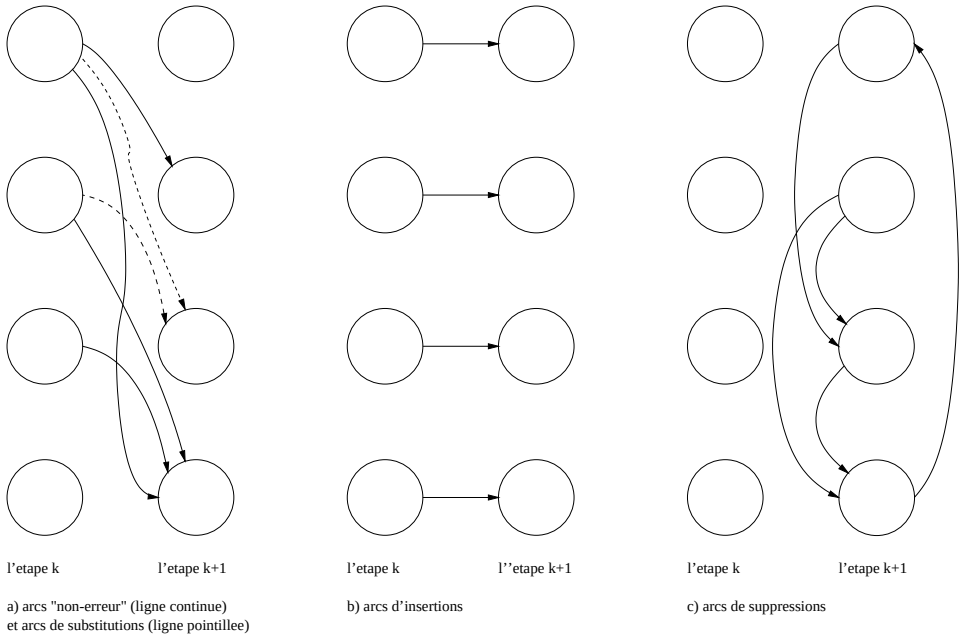


FIG. 3.9: Un graphe d'analyse.

- $W(q_k^i, q_{k+1}^l)$: le coût de la *transition de coût minimal* entre l'état q^i à l'étape k et l'état q^l à l'étape $k+1$.
- $\mathcal{C}(q_{k+1}^l, q_{k+1}^j)$: le coût du chemin de coût minimal de l'état q^l vers l'état q^j, tous les deux à l'étape $k+1$.

Ainsi, le coût $C(q_{k+1}^j)$ du chemin de coût minimal dans le graphe d'analyse, menant d'un état initial de A vers l'état q^j à l'étape $k+1$, est le minimum des coûts de tous les chemins passant par q^i (*prédécesseur* de q^l) à l'étape k et par q^l à l'étape $k+1$. La composante essentielle de cette relation, la fonction $\mathcal{C}$, peut être calculée en appliquant directement l'*algorithme de Dijkstra* ([Dijkstra, 1959]) qui permet d'obtenir le chemin le moins coûteux entre deux nœuds d'un graphe. Elle joue un rôle prédominant dans la complexité globale de l'algorithme. Une solution algorithmique ([Amengual & Vidal, 1998]) propose une complexité temporelle en $\mathcal{O}(|Q|(|Q|+\tau)|a|)$ (avec τ : densité maximale de l'automate). Si τ est négligeable par rapport à $|Q|$ alors elle peut être réécrite plus simplement avec : $\mathcal{O}(|Q|^2|a|)$. Partant du constat que cet algorithme est trop coûteux en temps dans les cas pratiques (comparé à l'algorithme de décodage de Viterbi, lui-même en $\mathcal{O}(|Q||a|)$, les auteurs ont proposé une solution derivée de la relation précédente, fondée sur l'ordonnancement des états selon l'*ordre topologique* adapté aux graphes comportant des cycles[5], (le *depth-first topological sort*[6]). Cette

[5] originalement, la notion d'*ordre topologique* n'a de sens que pour les graphes sans cycles.

[6] les nœuds sont classés selon les tailles croissantes des préfixes de longueur minimale avec lesquels ils peuvent être atteints.

nouvelle relation de récurrence s'écrit :

$$C(q_{k+1}^j) = \min_{\substack{\forall i \in \delta^{-1}(q_l) \text{ à l'étape } k \\ \forall l \in \delta_T^{-1}(q_j) \text{ à l'étape } k+1}} \{C(q_k^i) + W(q_k^i, q_{k+1}^l) + W_T(q_{k+1}^l, q_{k+1}^j)\}$$

avec :

- $C(q_k^i)$, $W(q_k^i, q_{k+1}^l)$, δ^{-1} : même signification que dans la relation précédente.
- δ_T^{-1} : la généralisation de la fonction δ^{-1} ; elle retourne, pour un état donné q, un ensemble d'états qui sont ses *prédécesseurs topologiques* dans l'automate A.
- $W_T(q_{k+1}^l, q_{k+1}^j)$: le chemin de coût minimal menant de l'état q^l vers l'état q^j, les deux à l'étape $k+1$ et qui passe *uniquement* par les états qui sont les prédécesseurs topologiques de q^j dans A.

Ici, similairement à la première méthode, le coût $C(q_{k+1}^j)$ du chemin de coût minimal pour arriver d'un état initial de A à l'état q^j à l'étape $k+1$ est le coût minimal de tous les chemins passant par l'état q^i à l'étape k, par l'état q^l (le successeur immédiat de q^i, ou encore : $q^l \in \delta(q^i)$) à l'étape $k+1$ et par un ou plusieurs arc créés à l'étape $k+1$ par les opérations de suppression. L'algorithme associé à cette relation effectue en prétraitement un tri topologique des états de A. Ce calcul est fait en $\mathcal{O}(|Q||a|)$. La complexité temporelle de l'analyse corrective elle-même est alors en $\mathcal{O}(|Q|\tau|a|)$. Si τ est petit par rapport à $|Q|$ (cas souvent rencontré dans la pratique) alors l'efficacité de cet algorithme est comparable à celle de l'algorithme de Viterbi.

Pour illustrer les méthodes décrites dans ce paragraphe, nous reprenons l'exemple du paragraphe 3.4.1 et nous ajoutons des coûts aux opérations d'édition :

Exemple 11 :
Il est possible de regrouper l'ensemble des coûts dans une *matrice de coûts* M_ζ :

$M_\zeta =$

	ϵ	a	b
ϵ	0	2	1.5
a	0.5	0.5	1
b	1.5	1	0.5

En colonne figurent les lettres remplaçantes, en ligne les lettres remplacées. Par exemple on lit : l'opération de remplacement de ϵ par a (*insertion* de a) coûte 2. En utilisant la notation introduite au début de ce chapitre on écrira : $\zeta(\epsilon \to a) = 2$.

Le calcul d'alignement est effectué avec l'automate A et la séquence a de l'Exemple 10 de la page 61.

Sous-phrase	État				
	0	**1**	**2**	**3**	**4**
ϵ	0	0.5	1.5	2	2
$a(1:1) = a$	2	0.5	1	1.5	1.5
$a(1:2) = aa$	4	2.5	3	1.5	1.5
$a(1:3) = aaa$	6	4.5	5	2.5	3

$$D(a, A) = 3$$

Notons que le calcul de la distance est valide également dans le cas où le coût de la substitution d'une lettre par elle même n'est pas nul. □

Remarque 8 *L'ordre et le type d'opérations obtenues par le* backtracking *diffère de ceux calculés dans l'Exemple 10. Par exemple, pour arriver avec le préfixe a en état* **3**, *on applique d'abord la* suppression *de a (coût = 0.5), puis la* substitution *de b par a (coût = 1), pour un coût total de 1.5. En utilisant les coûts unitaires seule la* suppression *de b s'applique (ici : emprunter la transition entre l'état* **0** *et l'état* **1** *coûte 0.5, la suppression de b entre l'état* **1** *et l'état* **3** *coûte 1.5, ce qui donne un coût total de 2). Le raisonnement en terme d'optimisation du* nombre d'opérations d'édition *ne s'applique pas dans le cas des opérations d'éditions pondérées.* □

3.5 Conclusion

La comparaison des séquences permet d'obtenir deux types d'information : les distances et les analyses correctives optimales (par exemple les *listings*). L'utilisation de ces résultats varie bien sûr en fonction de l'application : les distances sont utiles lorsqu'on aborde les problèmes d'identification (choisir la population la plus proche étant donné un individu, défini par un ensemble de caractéristiques et une mesure de ressemblance basée sur ces caractéristiques). Les distances *correctives* (à opposer aux distances d'emploi plus général, comme la distance euclidienne ou la distance de Hamming) sont particulièrement adaptées aux situations où l'on peut obtenir une séquence a à partir d'une autre séquence b, par un jeu d'opérations bien défini. Les traces des transformations s'avèrent utiles lorsqu'il est nécessaire de comprendre l'évolution d'une ou d'un ensemble de séquences (nous avons cité déjà l'exemple de l'étude des macromolécules), ou, comme nous le verrons dans le chapitre suivant, lorsqu'elles sont utilisées dans un processus de généralisation, où l'information sur les erreurs détectées sert à enrichir le modèle d'un ensemble de phrases.

Deuxième partie

Grammaires Formelles : construction et application à la reconnaissance vocale

Chapitre 4

Algorithme ECGI

Le choix parmi les méthodes d'inférence des grammaires formelles, mentionné dans le Chapitre 2, est plutôt vaste. Notre ambition n'étant pas de concevoir une nouvelle technique d'apprentissage, mais de proposer l'utilisation de l'inférence grammaticale dans une tâche de traitement de la langue naturelle, nous avons eu à choisir un algorithme particulier. Ce chapitre expose les raisons de ce choix et détaille le principe du fonctionnement de l'algorithme. Nous y ajoutons une étude approfondie de ses propriétés qui est à l'origine des méthodologies proposées plus loin dans ce document.

4.1 Motivation du choix de l'algorithme

Dans les tâches réelles de traitement de données (reconnaissance de la parole, reconnaissance optique des caractères etc.) il est rare de disposer de contre-exemples. En apprentissage automatique et singulièrement en inférence grammaticale, l'absence des exemples négatifs engendre surtout le risque de sur-généralisation ; pour pallier ce problème, il y a deux approches possibles : la première consiste à limiter la recherche de la solution à une sous-classe particulière des langages réguliers, la deuxième utilise une heuristique pour guider la généralisation. Nous nous sommes intéressé à un des algorithmes qui adopte cette seconde approche. En effet, les sous-classes caractérisables des langages réguliers sont définies par leurs propriétés algébriques et non pas par leur adéquation à un problème particulier.

Parmi les méthodes heuristiques, notre choix s'est porté sur l'algorithme ECGIA (Error Correcting Grammatical Inference Algorithm [Rulot & Vidal, 1988; Rulot et al., 1989]) pour plusieurs raisons :

- Il infère des grammaires non récursives. Cette caractéristique affaiblit considérablement les possibilités théoriques d'apprentissage, mais en pratique la récursivité est peu présente en langue naturelle et singulièrement dans les corpus dont nous disposons (voir Annexe A). La généralisation réalisée par ECGI ne cherche pas à découvrir des boucles (peu vraisemblables dans nos données) mais à factoriser les sous-phrases communes.

- Il est fondé sur une métrique entre lettres de l'alphabet. Dans le cas de la langue naturelle, on peut raisonnablement penser utiliser des connaissances lexicales et/ou syntaxiques pour aider à l'établissement d'une métrique réaliste, par expertise (deux noms de villes sont plus "proches" l'un de l'autre qu'un verbe d'un adjectif). Cette métrique peut aussi être établie automatiquement par une analyse du corpus : par exemple deux mots qui se trouvent souvent dans le même contexte ont vraisemblablement des fonctions linguistiques proches.
- Il a déjà été utilisé avec succès sur des gros corpus de données réelles, en particulier en traitement d'images et en reconnaissance de mots isolés.

L'algorithme ne respecte pas le critère d'inférence à la limite (voir paragraphe 2.2.3.1) : Gold a prouvé que ce critère est impossible à appliquer dans le cas d'apprentissage uniquement à partir des exemples positifs. Son fonctionnement est inductif : le modèle appris génère, dans la plupart de cas, un ensemble de séquences plus vaste que l'ensemble d'apprentissage.

4.2 ECGIA non-probabiliste

4.2.1 Présentation informelle

Le but de l'algorithme est de modéliser la concaténation des sous-chaînes des éléments de l'échantillon positif, ainsi que la longueur de ces sous-chaînes en construisant un AFN. Il procède de façon *incrémentale* : le modèle initial est construit à partir du premier mot présenté et il est amélioré par l'ajout de nouvelles règles résultant de l'*analyse corrective* des mots suivants. À chaque présentation d'un nouvel exemple, l'automate courant tente de reconnaître celui-ci. Si la tentative échoue, l'automate est mis à jour par l'ajout des nouveaux états et transitions afin de reconnaître le mot présenté. Le nombre d'états et de transitions à ajouter est minimisé en appliquant un algorithme de programmation dynamique. L'automate résultat est un automate acyclique non-déterministe qui accepte toutes les séquences de I_+ ainsi qu'un ensemble d'autres séquences qui sont des concaténations des sous-chaînes des mots de I_+. Le nombre d'états ajoutés pour reconnaître une séquence peut être vu comme une mesure de distance entre cette séquence et l'automate courant, et il est possible de déterminer la séquence la plus "proche" faisant partie du langage représenté par l'automate (cf. Chapitre 3). L'algorithme de programmation dynamique calcule une distance au sens de la distance corrective de Levenshtein, c'est à dire l'ensemble de règles de suppression, d'insertion et de substitution de symboles nécessaires pour adapter la grammaire afin que celle-ci reconnaisse le mot. Il est cependant possible d'utiliser d'autres critères (par exemple la plus longue sous-séquence commune) pour calculer cette distance, l'adaptant ainsi à la tâche et à la nature des données d'apprentissage.

4.2.2 Algorithme

Les grammaires apprises par ECGIA sont régulières, non-déterministes et non-récursives. Elles sont définies par un quadruplet[1] (N, Σ, R, S). Les règles construites possèdent la propriété suivante :

Propriété 4 *$\forall A, B, C \in N$, $\forall b, a \in \Sigma$, si $(B \rightarrow aA) \in R$ et $(C \rightarrow bA) \in R$ alors $b = a$.* □

Ceci veut dire que le même terminal est associé avec toutes les productions ayant le même non-terminal en partie droite. Par conséquent, toutes les transitions pointant sur un état donné de l'automate équivalent sont nécessairement étiquetées par le même terminal.

Informellement, l'automate équivalent est une *machine de Moore* (les états sont valués par les symboles de Σ : [Kohavi, 1970]). Dans la représentation habituelle d'automates (graphes valués sur les arcs uniquement) cela implique que toutes les transitions pointant sur le même état possèdent la même étiquette (l'équivalence entre les deux représentations des automates finis est mentionnée dans le paragraphe 3.3.3).

Définition 35 *Chaque règle de R possède des "règles d'erreur" associées :*

- *Insertion (de a) :*
  $A \rightarrow aA$, $\forall (A \rightarrow bB) \in R$, $\forall (A \rightarrow b) \in R$;
- *Substitution (de a par b) :*
  $A \rightarrow aB$, $\forall (A \rightarrow bB) \in R$,
  $A \rightarrow a$, $\forall (A \rightarrow b) \in R$;
- *Suppression (de b) :*
  $A \rightarrow \epsilon B$, $\forall (A \rightarrow bB) \in R$,
  $A \rightarrow \epsilon$, $\forall (A \rightarrow b) \in R$.

□

Définition 36 *La* grammaire étendue $G' = (N, \Sigma, R', S)$ *de G est la grammaire obtenue en ajoutant les règles d'erreur à R.* □

Définition 37 *La* dérivation corrective optimale $D_{G'}(x)$ *de la chaîne $x \in \Sigma^*$ est la dérivation de x, qui utilise le nombre minimal de règles d'erreur de $\{R' \setminus R\}$.* □

L'algorithme se présente alors comme suit :

[1] pour rappel : N est un ensemble de symboles non-terminaux, Σ est l'ensemble des symboles terminaux, R est l'ensemble de règles construites sur N et S est l'axiome.

Algorithme ECGI

entrée I_+
sortie Une grammaire G compatible avec I_+ et vérifiant la Propriété 4

début

$a \leftarrow I_+^1 (= a_1 \ldots a_n)$;
$N \leftarrow \{A_0, \ldots, A_{n-1}\}$;
$\Sigma \leftarrow \{a_1, \ldots, a_n\}$;
$R \leftarrow \{A_{i-1} \rightarrow a_i A_i, i = 1, \ldots, i = n\} \cup \{A_{n-1} \rightarrow a_n\}$;
$S \leftarrow A_0$;
$G \leftarrow (N, \Sigma, R, S)$; // grammaire courante
pour $i \leftarrow 2$ **jusqu'à** $|I_+|$ // boucle sur les éléments de I_+
    $b \leftarrow I_+^i$;
    $D \leftarrow D_{G'}(b)$;
    $G \leftarrow G \cup \{D\}$; // ajout de règles correctives à la grammaire courante
fin pour
retourner G ;

fin ECGI

La dérivation corrective optimale $D_{G'}(b)$ renvoie des règles de correction $r_1, \ldots, r_k$.

On appellera la *règle non-erreur* r une règle de correction telle que :

1. r est une règle de substitution
2. r remplace une lettre par elle-même

L'ajout de règles respecte la procédure suivante :

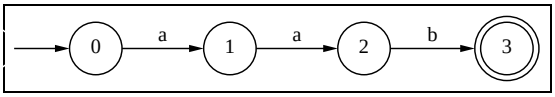
Pour chaque paire de règles non-erreur (r_i, r_j) de $D_G(b)$ telle que $r_i = (C_{i-1} \rightarrow c_i C_i)$, $r_j = (C_{j-1} \rightarrow c_j C_j)$, $i < j$, entre lesquelles il n'y a que des règles d'erreur, on ajoute à la grammaire courante les règles selon le principe suivant :

- si $b = b_1 \ldots c_i \alpha c_j \ldots b_l$ ($\alpha = x_1 \ldots x_m$) alors ajouter à la grammaire courante les $m+1$ règles suivantes (avec les terminaux et les non-terminaux correspondants) : $C_i \rightarrow x_1 X_1; \ldots; X_{m-1} \rightarrow x_m X_m; X_m \rightarrow c_j C_j$
- si $\alpha = \epsilon$ alors uniquement la règle $C_i \rightarrow c_j C_j$ est ajoutée.

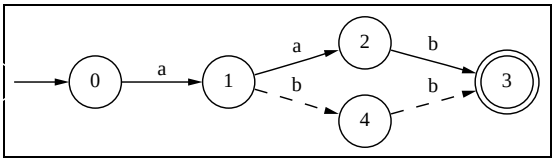
La complexité globale de l'algorithme, dominée par la fonction d'analyse corrective est : $\mathcal{O}(|\Sigma||I_+|^2)$.

L'exemple qui suit illustre l'apprentissage d'un automate à partir des échantillons $\{aab, abb, ab, baab\}$.

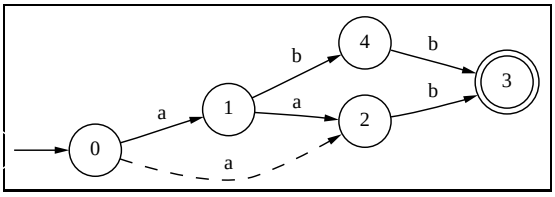
Exemple 1 :



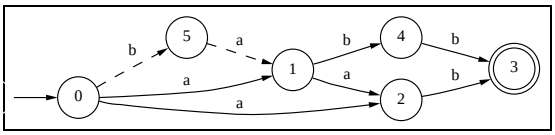
1. Initialisation de la grammaire avec la séquence 'aab'



2. Analyse corrective de 'abb' provoque la substitution de a par b



3. Analyse corrective de 'ab' provoque la suppression de a



4. Analyse corrective de 'baab' provoque l'insertion de b

FIG. 4.1: Apprentissage par l'algorithme ECGI.

Sur cet exemple simple, on voit que la généralisation effectuée amène l'automate à accepter une phrase n'appartenant pas à l'ensemble des exemples présentés : *babb*. Comme nous l'avons mentionné au début de cette section, l'algorithme est sensible à l'ordre de présentation des séquences. Avec le même ensemble d'apprentissage présenté dans l'ordre : *ab*, *baab*, *aab*, *abb*, ECGI produit le modèle :

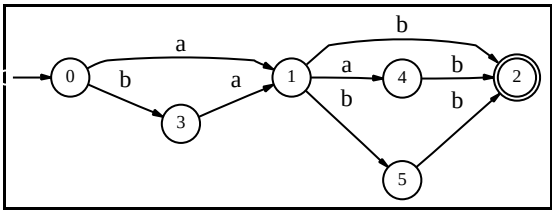


FIG. 4.2: Apprentissage par l'algorithme ECGI avec un ordre différent de présentation d'exemples.

Cette propriété est étudiée expérimentalement dans le Chapitre 7. □

4.3 ECGIA stochastique

L'algorithme ECGI possède une extension stochastique, proposée par les mêmes auteurs dans [Rulot & Vidal, 1988].

Rappelons brièvement (cf. Définition 11, page 19) qu'on peut affecter des valeurs réelles positives à des transitions d'un automate fini, sous la condition que la somme des valeurs sur les transitions partant d'un état soit égale à 1 (voir Définition 11 page 19). L'automate est alors dit *probabilisé*, ou *stochastique*. On peut alors calculer pour toute phrase x la probabilité $p(x|G)$ d'avoir été engendrée par cet automate. On démontre que ce modèle est *consistant*, c'est à dire qu'il munit Σ^* d'une distribution de probabilités. En effet ([Fu, 1970; Higuera (de la), 1996a]) :

$$\sum_{x \in \Sigma^*} p(x|G) = 1$$

Dans l'algorithme ECGI stochastique, l'estimation de probabilités de chaque règle (ou transition) est faite pendant le processus d'apprentissage de la manière suivante :
Pour une règle quelconque $r = A \rightarrow x_i B \in R$:

- $n_t(r) = n_{t-1}(r) + N_r(x)$ et $n_{t_0}(r) = 1$
- $m_t(A) = m_{t-1}(A) + M_A(x)$ et $m_{t'_0}(A) = 1 \quad \forall A \in N$
- $p_t(r) = n_t(r)/m_t(A)$

avec :

t : numéro d'ordre de la séquence x ;

t_0 , t'_0 : numéros des étapes de création de r et de A ;

$N_r(x)$: nombre de fois (0 ou 1) que r a été utilisée pour reconnaître x ;

$M_A(x)$: nombre de fois (0 ou 1) que A a été utilisé pour reconnaître x.

Nous obtenons alors une grammaire structurellement identique à celle apprise par ECGI non-stochastique, mais munie en plus de probabilités sur les règles et équivalente donc à un automate stochastique.

4.4 Paramétrage de l'algorithme

L'algorithme ECGI étant fondé sur les techniques de correction d'erreurs, il hérite des propriétés des algorithmes issus de ce domaine. Dans sa version la plus simple, les opérations d'édition qui servent au calcul de la distance entre le modèle et l'exemple courant sont munies de poids unitaires. Il est pertinent cependant de privilégier certaines opérations par rapport aux autres, notamment dans le cas de modélisation d'une (partie de la) langue naturelle. Par exemple, un mot de valeur sémantique faible ("euh", "zut", etc.) est probablement moins représentatif qu'un autre, en conséquence l'opération de suppression de ce mot devrait être privilégiée (i.e. lors du calcul d'une distance d'édition, la suppression de ce mot sera effectuée en priorité). De même, il serait moins "coûteux" (i.e. la valeur numérique du coût sera plus petite) de remplacer le mot "Paris" par "Lannion" que par un verbe, etc. Ces observations sont à la base des propositions de la section 4.4.2. Une autre approche, purement statistique, est proposée par la suite.

Il faut répéter que cet algorithme est sensible à l'ordre de présentation des séquences au cours de l'apprentissage : deux exécutions de l'algorithme sur un même ensemble de séquences présentées dans un ordre différent, produiront deux modèles *a priori* distincts, chacun approximant différemment le langage représenté par le corpus d'apprentissage. Nous étudions cette propriété dans le Chapitre 7 afin de construire des automates plus compacts en termes de nombre d'états et de transitions, tout en restant le plus représentatif possible du langage ciblé.

4.4.1 Évaluation expérimentale de la méthode ECGI originale

Nous donnons ici une brève description des corpus utilisés pour les expériences réalisées au cours de nos travaux. La description détailée de ces corpus figure dans l'Annexe A.

Trois corpus ont été utilisés :

- un corpus généré à partir d'un grammaire artificielle G_{L_0} (voir Annexe A.1), proche du langage naturel, composé de 10 000 phrases pour le corpus d'apprentissage et de 1 000 phrases pour le corpus de test ;
- le corpus AGS ([Sadek et al., 1996], voir Annexe A.2), mis à disposition par France Télécom[2], composé des requêtes transcrites, prononcées dans le cadre d'un dialogue entre un humain et un système automatique. Il contient 9 850 phrases pour le corpus d'apprentissage et 724 phrases pour le corpus de test ;

[2] dans le cadre de la convention CTI 97-1B-004.

– le corpus ATIS[3] (voir Annexe A.3) contenant des transcriptions de dialogues homme-machine portant sur les réservations automatiques des vols. Il est composé de 15 019 phrases pour le corpus d'apprentissage et de 1 001 phrases pour le corpus de test.

Les modèles appris par l'algorithme ECGI original (non-stochastique et stochastique) ont été évalués par le calcul du taux d'acceptation des phrases de test et par le calcul de la perplexité (elle est donnée ici à titre indicatif, le détail du calcul figure dans le Chapitre 5) :

Corpus	Taux de phrases reconnues (grammaire non-stochastique)	Perplexité (grammaire stochastique)
artificiel	98%	3.4
AGS	34.2%	47.6
ATIS	63.6%	30

La tâche la plus facile est celle posée par le corpus artificiel, les résultats sont moins bons pour le corpus ATIS et médiocres pour le corpus AGS. Concernant ce dernier en particulier, le corpus de test est très différent du corpus d'apprentissage. Cette constatation nous a amené à mener des expériences où le contenu des deux corpus est redistribué (voir Remarque 15 page 140, ainsi que la section 5.2.2, page 87).

4.4.2 Évaluation des coûts des opérations d'édition à partir de classes grammaticales de la langue française

4.4.2.1 Principe

Comme nous l'avons mentionné plus haut, certaines opérations impliquant les mots de la langue naturelle et en particulier la substitution d'un mot par un autre appartenant à la même classe grammaticale (ou sémantique), devraient être privilégiées lors du processus d'apprentissage. Une façon simple de réaliser en pratique cette constatation est de remplacer les coûts unitaires des substitutions entre les éléments de ces classes par des valeurs plus faibles. La matrice des coûts ainsi modifiée constitue alors un paramètre d'entrée de l'algorithme ECGI.

Disposant d'une classification des mots du corpus AGS (voir Annexe B) nous avons illustré cette approche par l'expérience décrite dans le paragraphe suivant. Auparavant, nous devions résoudre le problème venant du fait que les mots d'une langue naturelle sont souvent ambigus : la catégorie grammaticale de certains dépend de leur place dans la phrase. Pour cette expérience, nous avons choisi de n'utiliser qu'*une classe par mot*, même si cette approche est peu satisfaisante du point de vue d'un linguiste (voir également les arguments de la section 6.1, page 111). Des améliorations sont envisagées dans la conclusion de cette section.

[3] de l'anglais : Air Travel Information Service.

4.4.2.2 Protocole expérimental et résultats

On se propose de construire une matrice de coûts à partir des classes grammaticales de la langue française. Nous disposons pour cela d'un dictionnaire (dont la déscription figure dans l'Annexe B, à la page 143) qui attribue à un mot une ou plusieurs catégories grammaticales, affinées par des traits. Par exemple :

accessible	ADJEMS	ADJEFS

est un adjectif qui peut être utilisé au masculin ou au féminin et au singulier uniquement. Nous n'avons retenu que l'information portant sur la catégorie grammaticale du mot. Également, seule la première catégorie figurant dans la liste est retenue[4]. La matrice des côuts M_ζ est calculée en attribuant un coût arbitraire α (avec : $0 \leq \alpha \leq 1$) pour l'opération de substitution entre deux mots qui appartiennent à la même catégorie grammaticale et un coût unitaire pour toute autre opération d'édition :

$$M_\zeta^\alpha(x_i, x_j) = \begin{cases} 0 & si \quad x_i = x_j \\ \alpha & si \quad CG(x_i) = CG(x_j) \\ 1 & si \quad CG(x_i) \neq CG(x_j) \end{cases}$$

avec :

x_i, x_j : entrées lexicales du dictionnaire;

$CG(x_i)$: catégorie grammaticale de x_i.

Nous avons fait varier la valeur de α dans l'intervalle [0..2] par pas de 0.1 et nous avons calculé la perplexité du modèle appris (en utilisant la technique de lissage proposée dans le Chapitre 5) pour chaque M_ζ^α.

La Fig. 4.3 restitue les résultats de l'expérience. La perplexité minimale est obtenue avec $\alpha = 0.6$: $PP_{min} = 44.5$. Elle diminue d'environ 10% par rapport à la perplexité du modèle appris avec les coûts unitaires ($PP = 47.6$).

4.4.2.3 Discussion des résultats

Il s'avère, au vu des résultats, que l'injection des connaissances linguistiques améliore legèrement la qualité du modèle. Il est raisonnable de penser que l'intégration plus complète d'information apportée par les catégories grammaticales issues de la théorie linguistique pourrait rendre cette méthode plus efficace. Par exemple, l'attribution de la catégorie en fonction de sa fréquence a été mentionnée dans la note 4. L'approche plus satisfaisante consisterait cependant à attribuer l'étiquette grammaticale correcte à des mots au cours de la construction du modèle (ceci peut être réalisé par exemple par l'étiquetage préalable du corpus). La mise à jour du modèle devrait alors prendre en

[4] ce choix est fait de façon arbitraire (par un parcours séquentiel de données dont nous disposons). Il serait sans doute pertinent de classer les étiquettes grammaticales d'un mot en fonction des fréquences de leur emploi.

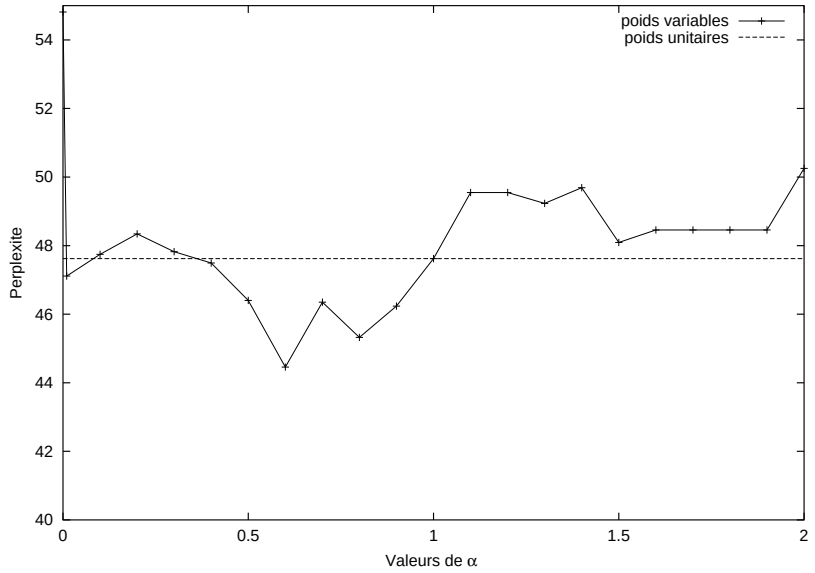


FIG. 4.3: Corpus AGS : perplexité obtenue avec les coûts calculés à partir des classes grammaticales de la langue française.

compte l'étiquette grammaticale du mot de la phrase courante[5] *et* l'étiquette grammaticale du mot dans le modèle. Ce problème cependant n'est pas trivial car il implique la construction et le *parsing* en parallèle d'un *modèle de langage à base de classes des mots* (abordé dans le Chapitre 6).

4.4.3 Calcul automatique des poids des opérations d'édition

4.4.3.1 Principe

Il est également possible d'estimer les valeurs de la matrice de coûts à partir du corpus d'apprentissage, en comptant les fréquences d'opérations d'édition utilisées lors de l'inférence de la grammaire. Nous utilisons pour cela une matrice M_{ECGI} de la forme suivante :

$M_{ECGI} =$

	ϵ	x_1	x_2	...	x_n
ϵ	0	$\Phi(\epsilon \to x_1)$	$\Phi(\epsilon \to x_2)$	...	$\Phi(\epsilon \to x_n)$
x_1	$\Phi(x_1 \to \epsilon)$	$\Phi(x_1 \to x_1)$	$\Phi(x_1 \to x_2)$	...	$\Phi(x_1 \to x_n)$
x_2	$\Phi(x_2 \to \epsilon)$	$\Phi(x_1 \to x_1)$	$\Phi(x_1 \to x_2)$	...	$\Phi(x_1 \to x_n)$
...	...	...	...	...	...
x_n	$\Phi(x_n \to \epsilon)$	$\Phi(x_n \to x_1)$	$\Phi(x_n \to x_2)$	...	$\Phi(x_n \to x_n)$

où $\Phi(x_i \to x_j)$ est le nombre de fois que l'opération $(x_i \to x_j)$ est utilisée pendant

[5]la phrase courante est celle qui sert à améliorer le modèle au cours d'un pas de l'algorithme.

l'apprentissage et $n = |\Sigma|$. Remarquons que l'opération $(\epsilon \to \epsilon)$ n'arrive jamais et n'est pas prise en compte.

Les *insertions* sont visibles sur la première ligne de la matrice, les *suppressions* sur la première colonne et les *substitutions* constituent les autres valeurs. En raisonnant en termes de coûts, il est logique de considérer qu'une opération qui arrive peu fréquemment doit être considérée comme plus coûteuse qu'une autre dont le nombre d'occurrences est élevé. Cette remarque nous permet donc de transformer M_{ECGI} en une matrice de coûts. Le choix de la méthode de calcul des coûts à partir des fréquences est posée arbitrairement et sa validité est évaluée de façon empirique. Nous avons testé deux approches :

1. Recalcul linéaire. Cette méthode consiste à normaliser séparement les insertions, les suppressions et les substitutions selon les relations :

$$\zeta(\epsilon \to x_i) = 1 - \frac{\Phi(\epsilon \to x_i)}{\max\{\Phi(\epsilon \to x_i)\}} \ \forall x_i \neq \epsilon$$
$$\zeta(x_i \to \epsilon) = 1 - \frac{\Phi(x_i \to \epsilon)}{\max\{\Phi(x_i \to \epsilon)\}} \ \forall x_i \neq \epsilon$$
$$\zeta(x_i \to x_j) = 1 - \frac{\Phi(x_i \to x_j)}{\max\{\Phi(x_i \to x_j)\}} \ \forall x_i \neq \epsilon \ et \ \forall x_j \neq \epsilon$$

Ici respectivement la suppression, l'insertion et la substitution les plus fréquentes ont un coût nul, les opérations qui n'arrivent jamais pendant l'apprentissage ont un coût unitaire.

2. Recalcul logarithmique. Cette méthode (proposée indépendament pour une tâche similaire[6] est inspirée d'un lissage simple des N-grams. Il s'agit d'ajouter une valeur γ comprise entre 0 et 1 à toutes les valeurs de la matrice M_{ECGI}. La matrice est ensuite normalisée suivant le type de l'opération, les coûts sont déduits par le calcul de logarithme des valeurs normalisées :

$$\zeta(\epsilon \to x_i) = -\log(\frac{\Phi(\epsilon \to x_i)+\gamma}{\sum_{k=1}^{|\Sigma|} \Phi(\epsilon \to x_k)+\gamma}) \ \forall x_i \neq \epsilon$$
$$\zeta(x_i \to \epsilon) = -\log(\frac{\Phi(x_i \to \epsilon)+\gamma}{\sum_{k=1}^{|\Sigma|} \Phi(x_k \to \epsilon)+\gamma}) \ \forall x_i \neq \epsilon$$
$$\zeta(x_i \to x_j) = -\log(\frac{\Phi(x_i \to x_j)+\gamma}{\sum_{k=1}^{|\Sigma|} \Phi(x_k \to x_j)+\gamma}) \ \forall x_i \neq \epsilon \ \forall x_j \neq \epsilon$$

Les événements fréquents ont une valeur de normalisation plus importante que ceux qui arrivent peu souvent, en conséquence leur coût, obtenu par la fonction $-\log$, est plus faible.

Note 1 : *Les tests comparatifs expérimentaux ont montré que le recalcul logarithmique donne des meilleurs résultats que le recalcul linéaire. La valeur de γ est estimée à environ* 0.005*. Nous n'avons pas de meilleure explication à ce résultat que le constat*

[6] la découverte de cette coincidence a été faite avec une surprise conjointe suite à des conversations informelles (mais cependant sérieuses) avec Pierre Dupont. Nous avons en réalité pensé à des approches très semblables à la même période.

expérimental. □

Nous pouvons alors réapprendre la grammaire G avec les nouvelles valeurs attribuées aux opérations correctives, en utilisant cette fois l'extension de l'algorithme ECGI proposée ci-dessus, puisque la matrice des coûts n'est plus unitaire. Ce processus d'apprentissage, d'estimation de la matrice des coûts, puis de réapprentissage peut être réiteré jusqu'à ce qu'un critère d'arrêt soit atteint (par exemple un seuil de différence entre deux matrices de coûts successives — ce qui sous-entend la convergence du processus, ou un nombre d'itérations fixé *a priori*).

4.4.3.2 Protocole experimental et résultats

L'expérience consiste à appliquer l'algorithme ECGI sur les données d'apprentissage (les trois corpus présentés dans l'Annexe A) une première fois avec les coûts unitaires, puis déduire des nouveaux coûts d'opérations d'édition par le calcul présenté dans le paragraphe précédent et les utiliser lors d'une nouvelle tâche d'apprentissage. Ce processus est réitéré plusieurs fois. Le nombre d'itérations est fixé arbitrairement de façon à être suffisant pour observer les performances des modèles selon :

- la *taille*, en termes de nombre d'états et de transitions ;
- la *perplexité*, calculée sur un corpus de test (en utilisant la méthode de lissage de grammaires proposée dans le Chapitre 5).

Deux paramètres sont estimés au cours de l'expérience :

- le nombre d'itérations nécessaires pour obtenir le meilleur modèle (évalué selon les critères de *taille* et de *perplexité*) ;
- le paramètre γ intervenant dans le recalcul des coûts des opérations d'édition.

Les résultats sont restitués pour les trois corpus étudiés.

Langage L_0

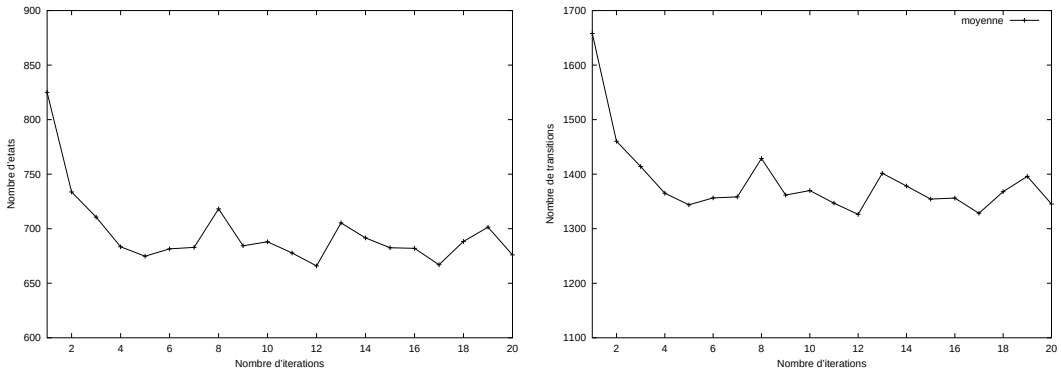


FIG. 4.4: Corpus L_0 : nombre d'états et de transitions créés par ECGI itératif.

En observant les courbes de la FIG. 4.4, on constate que les trois premières itérations font diminuer sensiblement la taille du modèle. On remarque que les variations du nombre d'états et du nombre de transitions sont corrélées.

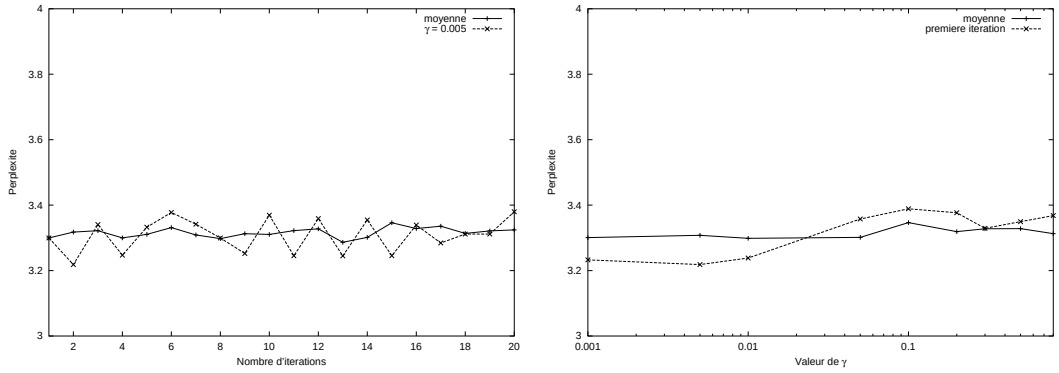


FIG. 4.5: Corpus L_0 : perplexité moyenne du modèle inféré par ECGI itératif. À gauche : les courbes correspondant à des valeurs de γ. À droite : les courbes correspondant à la perplexité moyenne et à la perplexité la plus faible obtenues pour la première itération.

La courbe de gauche de la FIG. 4.5 montre les variations de perplexité en fonction du nombre d'itérations. Elles sont très faibles, la perplexité la plus basse est atteinte dès la première itération. La courbe de droite permet de déterminer le valeur optimale de γ utilisée pour le recalcul logarithmique des coûts. Elle est estimée ici à 0.005.

Corpus AGS

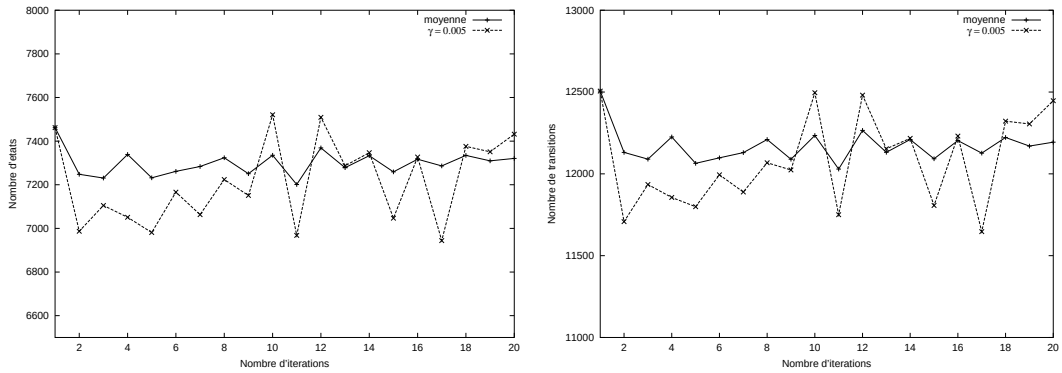


FIG. 4.6: Corpus AGS : nombre d'états et de transitions créés par ECGI itératif.

L'observation des courbes de la FIG. 4.6 permet de voir que la première itération donne un impact à la taille du modèle (environ 7 000 états et 11 700 transitions pour le plus petit modèle), les itérations suivantes étant des oscillations autour de cette valeur.

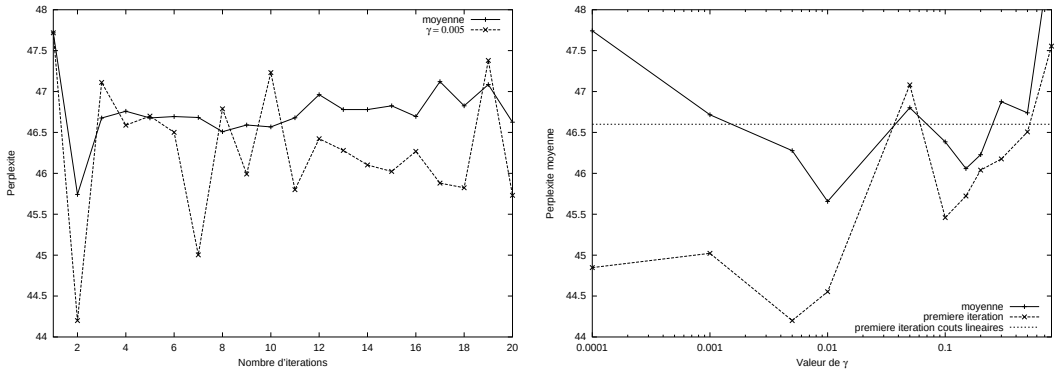


FIG. 4.7: Corpus AGS : perplexité moyenne du modèle inféré par ECGI itératif. À gauche : les courbes correspondant à des valeurs de γ. À droite : les courbes correspondant à la perplexité moyenne et à la perplexité la plus faible obtenues pour la première itération.

La valeur de γ optimale est estimée ici également à 0.005 (FIG. 4.7). La perplexité du modèle pour cette valeur est la plus faible après la première estimation des coûts.

Corpus ATIS

Le nombre d'itérations sur le corpus ATIS a été reduit à 5. En effet, les expériences sur les corpus précédents (un corpus artificiel et un corpus réél) montrent que seules les premières itérations ont une influence notable sur la qualité du modèle.

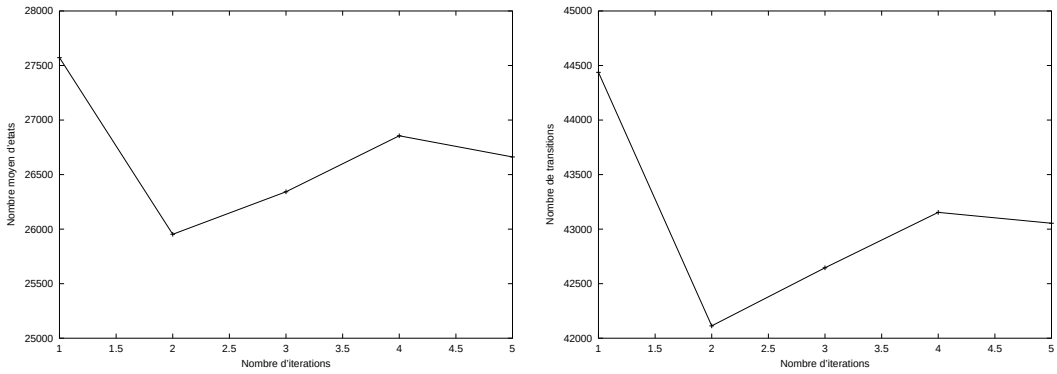


FIG. 4.8: Corpus ATIS : nombre d'états et de transitions créés par ECGI itératif.

L'évolution de la taille du modèle (FIG. 4.8) – sont représentées uniquement les valeurs moyennes calculées pour différentes valeurs de γ : voir FIG. 4.9) confirme l'importance de la première itération. Il serait intéressant alors d'analyser les variations des valeurs des matrices de coûts obtenues lors des itérations suivantes. Cette étude reste cependant à faire.

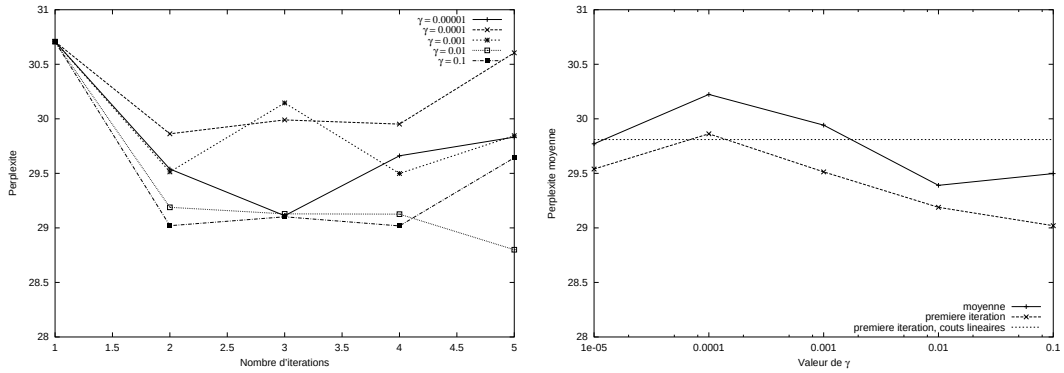


FIG. 4.9: Corpus ATIS : perplexité moyenne du modèle inféré par ECGI itératif. À gauche : les courbes correspondant à des valeurs de γ. À droite : les courbes correspondant à la perplexité moyenne et à la perplexité la plus faible obtenues pour la première itération.

La valeur optimale de γ (FIG. 4.9) est 0.1. Cependant les variations de la perplexité en fonction de γ sont très faibles (elles ne dépassent pas l'unité). L'étude plus complète de l'influence de ce paramètre reste à faire mais n'est pas essentielle pour cette approche.

4.4.3.3 Commentaires sur l'expérience

On constate expérimentalement sur un corpus non trivial, issu du langage L_0 de Feldman [Feldman et al., 1990] une convergence du processus au bout de quelques itérations. Elle s'accompagne de la réduction en taille de la grammaire apprise (nombre de règles et de non-terminaux en décroissance).

Pour des corpus issus des tâches réelles (AGS et ATIS), les résultats expérimentaux indiquent que la première itération réduit sensiblement la perplexité et la taille du modèle (en nombre d'états et de transitions), les suivantes n'ayant que peu d'impact.

La similitude de principe entre cette méthode et l'algorithme d'"Estimation-Maximisation" (EM : [Dempster et al., 1977]) nous a amené à l'appeler "ECGI EM-like". La preuve théorique de sa convergence est non-triviale. La démonstration de convergence de l'algorithme "EM", par exemple pour les HMM, s'appuie sur le fait que l'on réestime les probabilités sur une structure qui ne change pas lors des itérations, ce qui n'est pas le cas ici. Remarquons cependant que l'algorithme ECGI infère des modèles des langages finis. La longueur maximale de phrases est donnée par la profondeur du treillis (graphe orienté acyclique) appris. Il est possible, pour les besoins de la preuve, de considérer un treillis complet, formé des transitions apprises et celles non-vues au cours de l'apprentissage (voir section 5.3.1, page 90). Un autre moyen de s'appuyer sur une preuve de convergence déjà existante consiste à considérer chaque état de l'automate inféré comme émetteur d'une suite de symboles et en ceci l'assimiler à un modèle markovien. Cette preuve théorique reste cependant à faire.

4.5 Conclusion

Dans ce chapitre nous avons présenté la méthode choisie pour construire des modèles syntaxiques. Nous avons enrichi cet algorithme par le paramétrage des poids des opérations d'édition, approche qui nous a paru pertinente et théoriquement satisfaisante. Deux méthodes d'estimation des coûts de ces opérations ont alors été proposées : une fondée sur les connaissances linguistiques *a priori*, une autre purement statistique. Validées par les expérimentations, elles donnent des résultats meilleurs que l'algorithme dans sa version simple. Par ailleurs, la matrice de fréquences introduite ici est un objet statistique dont la manipulation est à la base des travaux présentés dans le chapitre suivant.

Chapitre 5

Lissages des grammaires régulières stochastiques

En Traitement de la Langue Naturelle, les données d'apprentissage sont logiquement constituées par des phrases prononcées ou écrites. Étant donnée la richesse de ce type de langage, il n'est pas raisonnable d'espérer retrouver dans les modèles appris toutes les constructions syntaxiques, même en utilisant les corpus de grande taille comme la Bible, les articles de journaux, les œuvres de Shakespeare etc., ou ceux représentatifs d'un domaine particulier (par exemple les corpus ATIS ou AGS). Pour remédier à l'insuffisance des données, on fait alors appel aux *techniques de lissage*. Ce chapitre en présente quelques unes, appliquées à des modèles N-grams. Plus loin, nous abordons le problème de lissage des grammaires régulières et nous introduisons une nouvelle technique de lissage par probabilisation des opérations d'édition.

5.1 Lissages des modèles N-grams

Lors de la construction des modèles N-grams à partir des corpus de la langue naturelle, on observe une part très importante des N-uplets ayant une fréquence nulle, ce qui entrave la modélisation de certaines phrases syntaxiquement correctes. Les techniques qui permettent de réévaluer les probabilités de ces événements non-vus et de leur attribuer une probabilité non-nulle portent le nom de "lissage". Elles sont regroupées essentiellement en deux familles :

- le *discounting*, qui consiste à redistribuer une certaine masse de probabilités entre les valeurs nulles de N-grams en l'enlevant de la masse totale de probabilités des événements rencontrés dans le corpus d'apprentissage.
- l'*interpolation*, qui fait appel à la hiérarchie des modèles N-grams : si un N-uplet $(w_{k-(N-1)}, \ldots, w_{k-1}, w_k)$ n'a pas été vu lors de la construction du N-gram on peut estimer la probabilité $p(w_k|w_{k-(N-1)}, \ldots, w_{k-1})$ à l'aide de la probabilité du modèle hiérarchiquement inférieur $(N-1)$-gram : $p(w_k|w_{k-(N-2)}, \ldots, w_{k-1})$.

Les techniques de lissage des modèles N-grams ont fait l'objet de nombreux travaux durant les dernières décennies (les travaux remontent jusqu'en 1953 avec l'algorithme de Good-Turing : [Good, 1953]). Les lissages proposés aujourd'hui sont très complexes, en contraste avec la relative pauvreté d'expression syntaxique des N-grams : il est discutable et discuté de ne représenter des dépendances syntaxiques qu'avec un nombre de mots fixé *a priori*.

5.2 Lissage des grammaires régulières non-stochastiques

Le lissage est une technique propre aux modèles probabilistes : le résultat souhaité est d'obtenir un modèle qui prédise de la manière la plus fiable possible les phrases d'un langage. En particulier, en fonction d'un contexte (et le plus couramment : des mots déjà prononcés, ou encore d'un *préfixe*) un modèle stochastique doit pouvoir prédire le mot suivant (i.e. fournir le mot le plus probable). Ce n'est bien sûr pas le cas des modèles non-probabilistes. Ceux-ci, sont au mieux capables de proposer un ensemble (équiprobable) de mots étant donné ce même contexte.

L'utilisation des techniques de correction d'erreurs amène à pondérer ce discours. Comme nous l'avons vu dans les chapitres précédents, le calcul de distance entre une grammaire et une phrase (séquence) permet d'obtenir une valeur interprétable comme une mesure d'"adéquation" de la phrase à la grammaire. Ceci nous amène à faire le parallèle entre ce calcul et une technique de lissage : les deux visent à fournir une valeur calculable pour toute séquence définie sur un alphabet donné et qui peut être transformée en un score d'adéquation au modèle. En termes de poids (ou de nombre) des opérations correctives, si parmi deux phrases une est plus "proche" que l'autre d'une grammaire donnée, on pourra dire qu'elle a "plus de chances" d'appartenir au langage modélisé. Il est possible de cette façon de prédire le mot suivant en fonction des valeurs pondérées des opérations d'édition, ou d'évaluer le degré de généralisation d'une grammaire. La première observation est à l'origine des techniques de lissage proposées dans la section 5.3 et les suivantes ; nous détaillons la deuxième ci-dessous.

5.2.1 Distance entre un corpus et un modèle syntaxique.

Définition 38 *Soient C un ensemble de séquences (un corpus) $\{x_1, \ldots, x_T\}$ construit sur un alphabet Σ et G une grammaire définie par le quadruplet (N, Σ, R, S). La distance entre le corpus C et G est la somme des distances entre G et chacune des séquences x_i :*

$$D(G, C) = \sum_{i=1}^{T} d(G, x_i)$$

□

Cette distance est calculable pour tout corpus constitué de séquences construites sur Σ et peut être utilisée avec les poids unitaires ou les poids pondérés, obtenus par

exemple par les méthodes vues dans le chapitre précedent.

Elle mesure uniquement le degré de généralisation du modèle et en ceci elle est plus faible que la mesure de perplexité pour les modèles stochastiques qui estime également le degré de sur-généralisation.

Elle peut être utilisée de plusieurs façons pour analyser un corpus :

- *Avec une valeur unique* : elle évalue la distance entre le corpus et le modèle, ce calcul pouvant être interpreté de la même façon que la perplexité d'un modèle stochastique (par comparaison des distances entre plusieurs modèles et un corpus de test particulier).
- *Par seuil* : des seuils prédéfinis de distances affinent l'analyse précédente : il est possible de dire le nombre de séquences reconnues à la distance d'un seuil donné. Par exemple, en utilisant les poids unitaires, dire qu'il y a 50 séquences reconnues à distance 1 équivaut à dire qu'il a 50 séquences qui ne diffèrent du modèle que par *une* opération d'édition. De même, dire que 100 séquences sont reconnues à la distance 0 est équivalent à dire qu'il y a 100 séquences reconnues par le modèle.
- *Par cumul* : cette analyse dérive de la précédente : pour un seuil fixé on donne le nombre de séquences qui ont été reconnues avec la distance inférieure ou égale au seuil. Ainsi un modèle qui reconnaît 100 séquences d'un corpus de test avec une distance de 3 ou moins est plus général qu'un modèle qui n'en reconnaît que 50.

Bien sûr, comme nous l'avons mentionné plus haut, il n'est pas possible d'évaluer le degré de sur-généralisation, ce qui est un désavantage considérable de cette méthode. Dans le cas où les exemples négatifs sont disponibles, la mesure du taux de rejet des séquences invalides peut appuyer cette méthode. Il est rare de disposer en pratique du modèle du langage cible. Cependant la combinaison des deux calculs — celui de la distance de la Définition 38 et du celui du taux de rejet — peut être utile si on compare plusieurs *méthodes* de construction de modèles, auquel cas on peut avoir recours aux grammaires artificielles.

L'exemple d'utilisation du critère de la distance entre un modèle et un corpus figure dans la section suivante.

5.2.2 Application

Au cours de l'application de la méthode proposée ci-dessus, nous avons utilisé deux corpus : le corpus AGS original (voir Annexe A) et un corpus créé suite à la Remarque 15 (page 140). Pour ce second corpus, nous avons réuni les corpus d'apprentissage et de test constituant le corpus AGS original, nous les avons brassés aléatoirement, puis cet ensemble a été redivisé en gardant les proportions d'origine. Ce corpus sera noté AGS_{bis}. Les résultats présentés pour ce corpus sont une moyenne des résultats obtenus sur 10 corpus ainsi construits (voir également [Chodorowski & Miclet, 1998]).

Lors de l'inférence, nous avons utilisé la version non-stochastique de l'algorithme ECGI en appliquant les coûts unitaires. Les modèles appris ont alors été évalués selon

quatre critères : les trois critères fondés sur les distances d'édition et la perplexité (calculée selon la méthode proposée dans la section et donnée ici à titre indicatif). De plus, nous avons ajouté l'évaluation du PTA (modèle appris "par cœur") construit sur le même corpus AGS, afin de mettre en évidence la capacité de généralisation de l'algorithme ECGI.

1. Corpus AGS

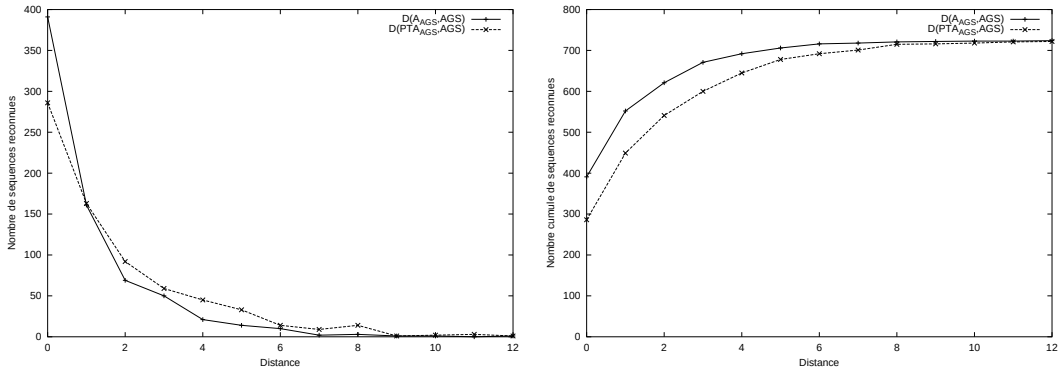


FIG. 5.1: Corpus AGS : distance entre le corpus de test AGS et une grammaire construite sur le corpus d'apprentissage AGS. À gauche, évaluation *par seuil*. À droite, évaluation *par cumul*.

Sur la FIG. 5.1, la courbe de gauche se lit (par exemple) : *"il y a (environ)* 400 *séquences (pour rappel : sur un total de* 724*) reconnues par l'automate A appris par l'algorithme ECGI sur le corpus AGS"*, ou encore : *"il y a (environ)* 25 *séquences dont la distance à l'automate vaut* 4*"*. Plus la pente de la courbe descendante est prononcée, plus la capacité de généralisation du modèle est grande. La courbe de droite se lit (par exemple) : *"il y a (environ)* 550 *séquences qui sont à la distance de* 1 *au plus de l'automate appris"*. Plus la pente de la courbe (forcément) montante est prononcée, plus la capacité de généralisation du modèle est grande. Cette courbe met également en évidence la généralisation du modèle appris par principe d'induction par rapport au modèle appris par cœur.

La distance entre le corpus de test et le modèle appris par ECGI vaut :

$$D(A(AGS), AGS_{test}) = 732$$

Puisque les coûts unitaires ont été utilisés pendant l'apprentissage, cela équivaut à dire qu'il y a 732 modifications élémentaires à appliquer sur les 724 séquences du corpus de test pour qu'elles soient acceptées par $A(AGS)$. Pour comparaison, la distance du PTA au même corpus vaut :

$$D(PTA(AGS), AGS_{test}) = 1\ 230$$

La perplexité de $A(AGS)$ vaut : $PP(A(AGS)) = 47.6$.

2. Corpus AGS_{bis}

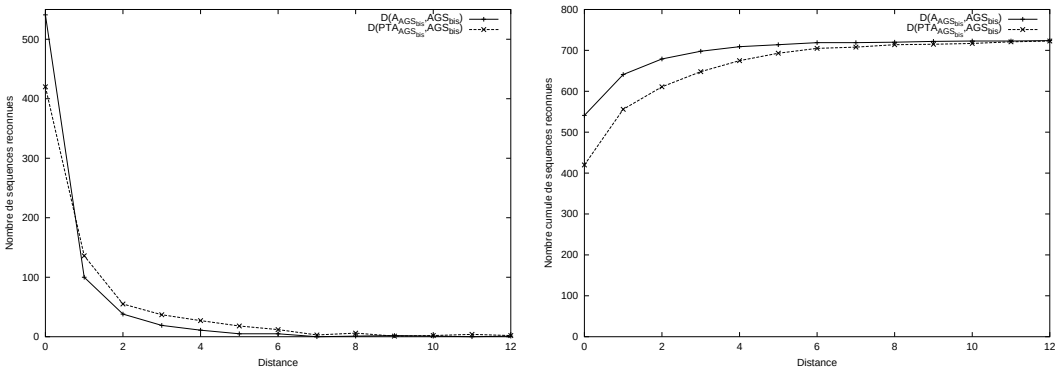


FIG. 5.2: Distance entre le corpus de test AGS_{bis} et une grammaire construite sur le corpus d'apprentissage AGS_{bis}. À gauche : évaluation *par seuil*, à droite : évaluation *par cumul*.

Les évaluations effectuées sur le corpus AGS_{bis} ont suivi le même protocole. On remarque alors (FIG. 5.2) que le modèle appris par cœur reconnaît (environ) 400 phrases contre (environ) 300 pour le corpus AGS original. Le modèle inféré reconnaît environ 550 phrases contre 400 pour le corpus précédent. Il est plus proche du corpus de test :

$$D(A(AGS_{bis}), AGS_{bis_{test}}) = 379$$

et :

$$D(PTA(AGS_{bis}), AGS_{bis_{test}}) = 810$$

Ces résultats montrent que les corpus AGS originaux (apprentissage et test) sont statistiquement invalides, il semble que les conditions dans lesquelles ils ont été conçus (comportement des locuteurs) ne soient pas identiques. La valeur de perplexité :

$$PP(A(AGS_{bis})) = 16.3$$

(cf. section 5.5 à la page 103) confirme cette constatation.

5.3 Lissage des grammaires régulières stochastiques

Rappelons brièvement qu'un automate fini peut voir ses transitions affectées de valeurs réelles positives, sous condition que la somme des valeurs sur les transitions partant d'un état soit égale à 1. L'automate stochastique ainsi construit est alors *consistant* (voir paragraphe 1.2.2.3).

Dans le cas général, étant données une grammaire stochastique G et une séquence x, il est possible de calculer la probabilité pour que x soit générée par G. Cette probabilité peut valoir zéro, seulement si la syntaxe de la séquence n'est pas reconnue par la grammaire (la probabilité d'aucune règle de G n'étant égale à zéro). Cependant, pour

pallier le manque de données d'apprentissage, qui conduit à une mauvaise estimation statistique du modèle, on doit lisser ses paramètres. Ce lissage permet en outre que toute chaîne puisse être engendrée par le modèle avec une probabilité non nulle.

Les lissages évoqués dans la section 5.1 sont naturellement une source d'inspiration pour les techniques de lissage des grammaires stochastiques. Nous nous proposons de faire une brève étude de deux approches — lissage par discounting et lissage par interpolation — dans les paragraphes qui suivent.

5.3.1 Lissage par discounting

Dans le cas des N-grams, le lissage par discounting consiste à estimer les probabilités des événements (N-uplets) non-rencontrés dans le corpus d'apprentissage. Par une démarche équivalente, nous observons qu'en apprentissage des grammaires régulières probabilistes les événements non-vus sont des composantes de ces modèles : les règles probabilisées. Ainsi, dans le cas de figure extrême, une grammaire lissée comporterait l'ensemble des règles de la forme :

$$X \to xY \ et \ X \to x, \ \begin{array}{l} \forall X, \forall Y \in N \\ \forall x \in \Sigma \end{array}$$

Informellement, ceci revient à ajouter aux règles déjà apprises[1] le reste de toutes les règles constructibles sur les ensembles N et Σ. Les probabilités de ces règles devraient être, comme dans le cas des N-grams, réestimées pour former un modèle consistant.

Remarque 9 *Une telle grammaire introduit des boucles (cas : $X = Y$) sur les règles, ce qui complique a priori l'alignement des séquences sur le modèle. Nous verrons à la page 96 que cela n'est pas en conflit avec l'approche que nous avons adoptée.* □

En représentation équivalente des grammaires régulières stochastiques par Automates Finis Stochastiques (AFS), cela consiste à construire et probabiliser des transitions allant d'un état vers chacun des états de l'AFS et cela pour chaque lettre de l'alphabet, ainsi qu'estimer la probabilité (*a priori* non-nulle) pour chaque état d'être un état d'acceptation. Il est facile de voir que cette approche impose un nombre de paramètres qui croît rapidement avec la taille du lexique et le nombre de non-terminaux construits pendant l'apprentissage : si $|N|$ est le nombre de non-terminaux et $|\Sigma|$ le nombre de symboles dans la grammaire apprise, alors le nombre de règles qu'il est possible de construire est : $|N||\Sigma|(|N| + 1)$. À titre d'exemple, le lexique du corpus ATIS comporte environ 1 000 mots et la grammaire apprise par algorithme ECGI à partir de ce corpus comporte environ 30 000 non-terminaux. Lisser cette grammaire en utilisant une telle approche impliquerait la construction de quelque 10^{12} règles. Il n'est pas, bien entendu, envisageable de manipuler un objet de cette taille dans le cas pratique. Il est cependant possible de réduire ce nombre sous certaines conditions. Dans cette

[1] munies de probabilités non-nulles, déduites du corpus d'apprentissage.

optique, la section 5.4 décrit une nouvelle approche de lissage de grammaires régulières stochastiques, fondée sur les probabilités des opérations d'édition.

5.3.2 Lissage par interpolation

Cette technique fait appel à une hiérarchie de modèles. Il convient donc en premier lieu de définir une relation d'ordre dans le cadre d'utilisation des grammaires régulières stochastiques. Les résultats théoriques présentés dans la section 2.2.2 fournissent naturellement une hiérarchie d'automates[2] et définissent l'élément le plus spécifique (le PTA) et le plus général (l'automate universel : AU). Ainsi, il est possible de faire appel à un modèle plus général lorsque le modèle courant est incapable de reconnaître une séquence. La difficulté majeure de cette technique, dont une étude prospective est donnée dans [Thollard & Higuera (de la), 1998], consiste dans le choix des modèles plus généraux : en effet dans le cas des N-grams il n'y a qu'un modèle d'ordre $N-1$ qui est plus général. Dans le treillis théorique d'automates, N peut être assimilé au nombre d'états, or le nombre d'automates correspondant à un nombre d'états donné y est *a priori* supérieur à un (sauf pour le PTA et le AU). Cette technique doit donc proposer le critère de choix du modèle plus général. De plus, d'un coté pratique, il n'est pas raisonnable d'envisager de construire physiquement le treillis complet d'automates, un moyen efficace de parcours de cette hiérarchie devrait être inclus dans ce type de lissage.

Cette technique faisant appel aux algorithmes basés sur la fusion d'états (comme par exemple RPNI, cité dans le paragraphe 2.4), son étude n'est pas abordée dans ce document.

5.4 Lissage par estimation des probabilités des opérations d'édition

Comme il a été mentionné plus haut, la technique de lissage par discounting consiste à créer les transitions manquantes et à leur redistribuer une partie des valeurs des probabilités des règles existantes. Nous proposons une telle technique dans les paragraphes qui suivent.

5.4.1 Estimation des paramètres du lissage

Dans la section 5.3.1, nous avons montré que le nombre de règles qu'il est nécessaire d'ajouter pour effectuer un lissage de type "discounting" de grammaires probabilistes peut être très grand. Nous proposons ici une méthode qui, tout en réduisant ce nombre, détermine la probabilité des règles manquantes sous contrainte essentielle de conservation de la consistance du modèle. Sa particularité réside dans le fait qu'elle ne construit pas explicitement l'intégralité du modèle, mais calcule une distribution de probabilités qui, avec un algorithme de programmation dynamique adapté (de type *Viterbi*), permet

[2] pour le rappel : le Théorème 4 définit un espace de recherche avec la relation d'ordre adjointe : $\subseteq$

d'attribuer une probabilité à toute séquence construite sur le même alphabet que ce modèle. Sa description est divisée en trois étapes étapes :

1. Définition d'une *grammaire augmentée* capable d'estimer toute séquence sur Σ.
2. Proposition d'une méthode d'estimation des probabilités des règles de cette grammaire.
3. Proposition du calcul des probabilités des opérations d'édition nécessaires à l'estimation mentionnée plus haut.

5.4.1.1 Grammaire augmentée $G^{'}$

Soit $G = (N, \Sigma, R, S)$ une grammaire apprise par l'algorithme ECGI étendu au cas stochastique. Soit $G^{'} = (N, \Sigma, R^{'}, S)$ la grammaire G augmentée par les *règles de correction d'erreurs* définies ainsi :

Insertion (de a) :	$A \rightarrow aA$,	$\forall (A \rightarrow bB) \in R$	$\forall a \in \Sigma$
Substitution (de b par a) :	$A \rightarrow aB$,	$\forall (A \rightarrow bB) \in R$	$\forall a \in \Sigma$
Suppression (de b) :	$A \rightarrow B$,	$\forall (A \rightarrow bB) \in R$	

La probabilité pour que $x \in \Sigma^*$ soit générée par $G^{'}$ est :

$$p(x|G^{'}) = \sum_{\forall \mathbf{D}^{'}_G(x)} (\prod_{\forall r_i \in \mathbf{D}^{'}_G(x)} p(r_i))$$

où $\mathbf{D}^{'}_G(x)$ est une dérivation correctionnelle (non-nécessairement optimale) de G pour engendrer x.

Une telle grammaire doit conserver la consistance du modèle. Autrement dit, elle doit munir Σ^* d'une distribution de probabilités en respectant la relation :

$$\sum_{x \in \Sigma^*} p(x|G^{'}) = 1 \tag{5.1}$$

La Fig. 5.3 montre l'AFS équivalent à une grammaire G (traits continus) à laquelle on a adjoint toutes les règles d'erreur (traits pointillés). Le mot vide ϵ (équivalent de la règle de suppression) est ici représenté par le caractère $\sim$. Remarquons qu'une séquence peut être acceptée de plusieurs façons.

À la vue des arguments présentés dans la section 5.3.1, l'avantage de cette approche est le nombre en général réduit de nouvelles règles introduites pour reconnaître la syntaxe des séquences de Σ^*. Ici, pour toute règle de forme $A \rightarrow xB$ on n'ajoute que des règles qui ont les mêmes non-terminaux du côté droit et du côté gauche de la règle, ainsi que des règles qui ont le même non-terminal des deux côtés.

Au total, le nombre de règles dans $G^{'}$ est donné par : $|\Sigma||N| + |\Sigma||R| + |R|$, avec :

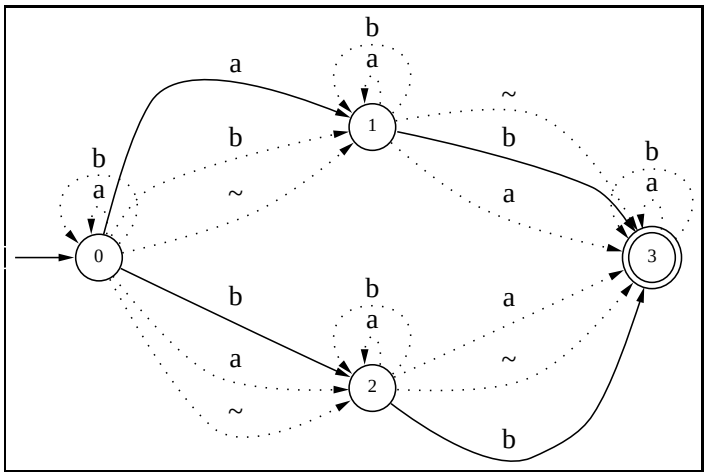


FIG. 5.3: Grammaire étendue $G^{'}$.

$|\Sigma||N|$ nombre de règles correspondant à l'opération d'insertion,
$|\Sigma||R|$ nombre de règles correspondant à l'opération de substitution,
$|R|$ nombre de règles correspondant à l'opération de suppression.
Le nombre de règles dans G (noté ici : $|R|$) est compris entre $|N|-1$ et $|N|(|N|-1)$ (c'est également le nombre de transitions dans l'automate équivalent).

On peut également raisonner en termes de densité de l'AFS équivalent. On définit τ_{moy} : la densité moyenne d'un automate comme le nombre moyen de transitions sortant de chaque état du AFS :

$$\tau_{moy} = \frac{|R|}{|N|}$$

Ainsi la taille de $G^{'}$ (en nombre de règles) est donnée par : $|\Sigma||N|(2\tau_{moy}+1)$.

Généralement τ_{moy} est plus petit que $|N|$ (si $\tau_{moy} = |N|$ alors l'automate est dit *complet* et reconnaît toute séquence sur Σ^*). À titre d'exemple, une grammaire apprise par ECGI sur le corpus ATIS comporte environ 30 000 états ($|N| = 30\ 000$) et 40 000 transitions ($|R| = 40\ 000$) ; ici τ_{moy} vaut 1.33 et peut donc être négligé devant $|N|$.

5.4.1.2 Calcul des probabilités des règles de $G^{'}$

En supposant que l'on connaît les probabilités des opérations correctionnelles $p(x_i \rightarrow x_j)$, nous pouvons écrire :

$$\forall x_i, x_j \in \{\Sigma \cup \epsilon\},\ \forall\ A \rightarrow x_j B \in R^{'}\ p(A \rightarrow x_j B) = p(A \rightarrow x_i B \in R) \times p(x_i \rightarrow x_j) \quad (5.2)$$

Pour que le modèle reste consistant (i.e. pour qu'il vérifie (5.1)), il est nécessaire que la somme des probabilités des règles dont la partie gauche est constituée d'un même

non-terminal soit égale à 1 :

$$\sum_{\forall x_i \in \{\Sigma \cup \epsilon\}, \forall B \in N | \exists A \to x_i B \in R'} p(A \to x_i B) = 1, \ \forall A \in N$$

Nous satisfaisons cette condition en ajoutant la contrainte :

$$\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} p(x_i \to x_j) = 1, \forall x_i \in \{\Sigma \cup \epsilon\} \tag{5.3}$$

Le problème revient alors à calculer les probabilités des opérations correctionnelles en respectant (5.3).

5.4.1.3 Estimation des probabilités des opérations correctionnelles

Nous pouvons déduire les probabilités des opérations d'édition à partir de la matrice M_{ECGI} introduite dans le paragraphe 4.4.3 en la normalisant ligne par ligne. En effet, après la normalisation d'une ligne correspondant à un symbole x_i, la somme des valeurs assignées aux opérations $(x_i \to x_j)$ vaut 1 pour tout $x_j \in \{\Sigma \cup \epsilon\}$, ce qui répond à la contrainte (5.3). Deux problèmes se posent alors :

1) Probabilités nulles des opérations $(x_i \to x_j)$**.** Nous obtenons les probabilités nulles pour les opérations d'édition qui n'ont pas été vues pendant l'apprentissage. Pour y remédier nous utilisons la technique de lissage d'*Absolute Discounting* [Ney & Essen, 1993] utilisée pour lisser des modèles de type N-grams que nous adaptons au lissage des probabilités d'édition de la manière suivante :

$$p(x_i \to x_j) = \begin{cases} \frac{M_{ECGI}(x_i,x_j)-\lambda}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(x_i,x_j)} & si\ M_{ECGI}(x_i,x_j) > 0 \\ \frac{\lambda*(|\Sigma|-n_0+1)}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(x_i,x_j)} * \Theta_{x_i}(x_j) & si\ M_{ECGI}(x_i \neq \epsilon, x_j) = 0 \\ \frac{\lambda*(|\Sigma|-n_0)}{\sum_{\forall x_j \in \{\Sigma \cup \epsilon\}} M_{ECGI}(\epsilon,x_j)} * \Theta_{\epsilon}(x_j) & si\ M_{ECGI}(\epsilon, x_j) = 0 \\ \Theta_{x_i}(x_j) & si \sum_{\forall x_j \in \Sigma} M_{ECGI}(x_i,x_j) = 0 \end{cases}$$

Avec :

n_0 **:** nombre de valeurs nulles sur une ligne de M_{ECGI}

$\Theta_{x_i}(x_j)$ **:** pourcentage du nombre d'opérations $(x_j \to x_i)$ correspondant aux valeurs nulles de $M_{ECGI}(x_i, x_j)$:

$$\sum_{x_j | M_{ECGI}(x_i,x_j)=0} \Theta_{x_i}(x_j) = 1$$

Le calcul de $\Theta_{x_i}(x_j)$ est fait séparément pour chaque ligne de M_{ECGI} : il remplace le facteur $\frac{1}{n_0}$ (spécifique pour chaque ligne) présent dans la technique *Absolute Discounting* classique. C'est une valeur heuristique, on l'interprète comme l'importance des insertions des x_j à la place d'autres lettres. On la calcule comme

suit :
Pour chaque valeur nulle $M_{ECGI}(x_i, x_j)$ sur une ligne correspondant à x_i on considère :

$$subst_{x_i}(x_j) = \sum_{x_k \in \Sigma, x_k \neq x_j} M_{ECGI}(x_k, x_j)$$
$$SUBST_{x_i} = \sum_{x_j | M_{ECGI}(x_i, x_j)=0} subst_{x_i}(x_j)$$

$subst_{x_i}(x_j)$ est ici la fréquence de substitution de tout symbole de $\Sigma \cup \{\epsilon\}$ par x_j, à l'exception de x_j lui-même, $SUBST_{x_i}$, est la somme de ces fréquences. Alors, $\Theta_{x_i}(x_j)$ est donnée par :

$$\Theta_{x_i}(x_j) = \frac{subst_{x_i}(x_j)}{SUBST_{x_i}}$$

si aucune valeur de $subst_{x_i}(x_j)$ est nulle, par :

$$\Theta_{x_i}(x_j) = \begin{cases} \frac{subst_{x_i}(x_j) - \lambda}{SUBST_{x_i}} \\ \frac{\lambda(n' - n'_0)}{SUBST_{x_i} n'_0} \end{cases}$$

s'il y a n'_0 valeurs nulles parmi n' valeurs de $subst_{x_i}(x_j)$, ou par :

$$\Theta_{x_i}(x_j) = \frac{1}{n'}$$

si toutes les n' valeurs de $subst_{x_i}(x_j)$ sont nulles.

Note 1 : *Nous utilisons la même valeur λ pour le lissage des valeurs $M_{ECGI}(x_i, x_j)$ et pour le lissage des valeurs $\Theta_{x_i}(x_j)$. Un paramétrage plus fin consisterait à utiliser une valeur distincte dans chaque partie de ce calcul.* □

Ce calcul (illustré sur l'Exemple 1, page 97) permet alors d'estimer, avec une valeur non nulle, la probabilité de chaque opération $(x_j \to x_i)$. Il garantit la satisfaction de la contrainte (5.3) : les probabilités des opérations $(x_j \to x_i)$ somment à 1 pour tout x_i.

2) Probabilités nulles des règles d'insertion. Dans la grammaire apprise G, il n'y a pas de règles correspondant aux opérations d'insertion $(A \to x_j A)$. Par conséquent, la relation (5.2) leur attribue une probabilité nulle. La solution consiste à enlever une fraction β de probabilités à chacune des règles $A \to x_j B \in R'$ et l'attribuer à $p(A \to x_j A \in R')$ par :

$$p(A \to x_j A \in R') = \beta * p(A \to x_i B \in R) * p(\epsilon \to x_j) \ \forall B \neq A, \ \forall x_i, x_j \in \Sigma$$

5.4.1.4 Calcul final des probabilités des règles de G'

Chaque règle appartenant à R' se voit attribuer alors une probablité calculée selon :

$$\begin{array}{l} p(A \to x_j B \in R') = (1-\beta) \times p(A \to x_i B \in R) \times p(x_i \to x_j) \\ p(A \to x_j A \in R') = \beta \times p(A \to x_i B \in R) \times p(\epsilon \to x_j) \end{array} \quad \forall x_i \in \Sigma, \forall x_j \in \{\Sigma \cup \epsilon\}$$

Le modèle reste consistant :
$\forall A, B \in N, A \neq B$:

$$\sum_{\forall x_i \in \Sigma} \sum_{\forall x_j \in \Sigma} p(A \to x_j B \in R^{'}) + p(A \to \epsilon B \in R^{'}) + p(A \to x_j A \in R^{'}) =$$

$$\begin{array}{rll} \sum_{\forall x_i \in \Sigma} \sum_{\forall x_j \in \Sigma} & (1-\beta) \times p(A \to x_i B \in R) \times p(x_i \to x_j) & + \\ & (1-\beta) \times p(A \to x_i B \in R) \times p(x_i \to \epsilon) & + \\ & \beta \times p(A \to x_i B \in R) \times p(\epsilon \to x_j) & = \end{array}$$

$$\begin{array}{rll} \sum_{\forall x_i \in \Sigma} & (1-\beta) \times p(A \to x_i B \in R) \times \sum_{\forall x_j \in \Sigma} (p(x_i \to x_j) + p(x_i \to \epsilon)) & + \\ \sum_{\forall x_i \in \Sigma} & \beta \times p(A \to x_i B \in R) \times \sum_{\forall x_j \in \Sigma} p(\epsilon \to x_j) & = \end{array}$$

$$(1 - \beta + \beta) \times \sum_{\forall x_i \in \Sigma} p(A \to x_i B \in R) = 1$$

car G est un modèle consistant. En conséquence $G^{'}$ est aussi un modèle consistant.

La complexité de l'algorithme utilisé pour le décodage correctionnel probabiliste des séquences n'est pas modifiée : elle correspond à la complexité des algorithmes présentés dans le Chapitre 3. Le surcoût est constant, il est occasionné par les opérations arithmétiques de calcul des probabilités des règles $R^{'}$. L'ajout de règles introduisant des boucles (ainsi que le calcul des probabilités associées) est effectué au cours de l'analyse corrective : uniquement des règles qui interviennent dans le décodage sont créées et estimées (cf. Remarque 9, page 90).

Nous obtenons ainsi un système complet d'inférence et d'évaluation des grammaires stochastiques par utilisation des techniques correctives. Remarquons que cette méthode s'applique également avec l'algorithme ECGI itératif, puisque le calcul des $p(x_i \to x_j)$ intervient uniquement après l'itération finale.

5.4.2 Problème du mot inconnu

Les travaux exposés plus haut supposent que l'on travaille avec un lexique fermé : tous les symboles présents dans le lexique du corpus d'apprentissage et dans le corpus de test sont représentés dans la matrice lissée. Le lissage des valeurs non-vues (les lignes de la matrice qui correspondent aux éléments du lexique présents dans le corpus de test et absents du corpus d'apprentissage) est assuré par la valeur $\Theta_{x_i}(x_j)$. Il est facile de passer de cette méthode vers la généralisation du traitement des mots nouveaux (ou les "mots inconnus"). Il suffit pour cela de n'utiliser que les symboles présents dans le corpus d'apprentissage pour construire la matrice de comptage de fréquences d'opérations d'édition, et d'y ajouter un symbole de mot inconnu. Si $|\Sigma|$ est la taille de l'alphabet du corpus d'apprentissage, alors la taille de la matrice construite devient : $(|\Sigma|+2)^2$ (ajout de la lettre vide et du *mot inconnu*). La matrice est lissée selon la même procédure. Lors du décodage, un mot hors-vocabulaire est étiqueté comme *mot inconnu* (codé par

UNK), les valeurs $p(UNK \to x_j \in \Sigma)$ et $p(x_j \in \Sigma \to UNK)$ sont déduites de la valeur $\Theta_{UNK}(x_j)$ par le calcul présenté dans le paragraphe précédent (voir Exemple 1).

Il est possible d'affiner la modélisation du mot inconnu par une approche semblable à la cross-validation qui consiste à diviser le corpus d'apprentissage en n paires de corpus (apprentissage et test). La différence entre le lexique du corpus d'apprentissage et le corpus de test constitue le lot des "mots inconnus". Les fréquences d'opérations d'édition impliquant ces mots au cours de l'apprentissage sont ensuite utilisées lors du lissage de la matrice (somme des n matrices obtenues lors des n procédures d'apprentissage). Le protocole expérimental et le comparatif des résultats ("lexique fermé" *versus* "modélisation du mot inconnu") figure dans la section 5.5.

L'exemple qui suit récapitule l'ensemble des techniques proposées dans dans cette section.

Exemple 1 :
Le corpus d'apprentissage utilisé dans cet exemple est : $\{ab, ab, ab, ab, bb, a\}$.
Lors de l'exécution de l'algorithme ECGI, le modèle est initialisé avec la première séquence, puis les fréquences des opérations d'édition (initialisées au départ à 0) sont déduites au cours de l'apprentissage :

$$\begin{array}{lll} ab: & \Phi(a \to a) = 1 & \Phi(b \to b) = 1 \\ ab: & \Phi(a \to a) = 2 & \Phi(b \to b) = 2 \\ ab: & \Phi(a \to a) = 3 & \Phi(b \to b) = 3 \\ bb: & \Phi(a \to b) = 1 & \Phi(b \to b) = 4 \\ a: & \Phi(a \to a) = 4 & \Phi(b \to \epsilon) = 1 \end{array}$$

La grammaire inférée G est représentée sous forme d'un automate sur la FIG. 5.4 en traits continus. Les règles ajoutés par le lissage sont en traits pointillés. Remarquons que pour les états *puits* il n'est pas nécessaire de représenter les règles d'insertion : la masse de probabilités enlevée des transitions sortantes est nulle.

L'alphabet $\{a, b\}$ est enrichi par le symbole UNK ($|\Sigma| = 3$). La matrice M_{ECGI} est déduite des valeurs présentés plus haut :

$M_{ECGI} =$

	Lettre remplaçante			
Lettre remplacée	ϵ	a	b	UNK
ϵ	-	0	0	0
a	0	4	1	0
b	1	0	4	0
UNK	0	0	0	-

Le paramètre de lissage λ est fixé à 0.6
1. Ligne 'ϵ'
Pour cette ligne, les fréquences des événements $(\epsilon \to a)$ et $(\epsilon \to b)$ et *a fortiori* de $(\epsilon \to UNK)$ sont nulles, la méthode impose le calcul de $\Theta_\epsilon(a)$, $\Theta_\epsilon(b)$ et de $\Theta_\epsilon(UNK)$:

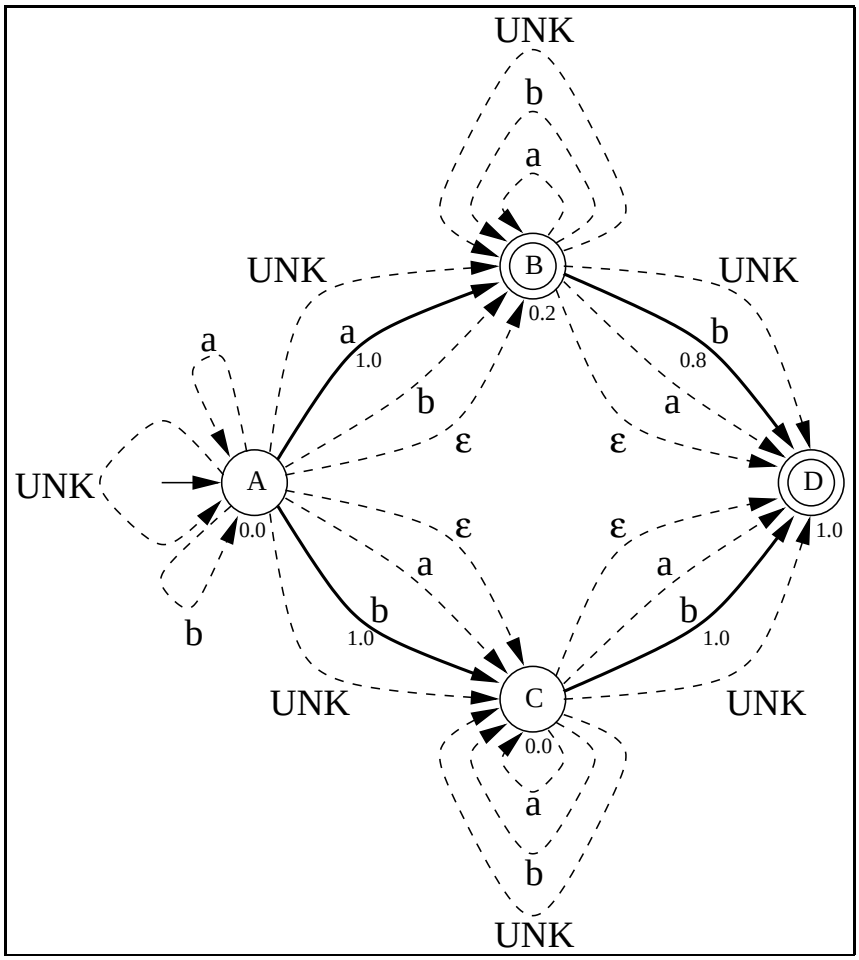


FIG. 5.4: Grammaire lissée.

$subst_\epsilon(a) = 0$ (le symbole a n'a jamais été utilisé pour remplacer un autre que lui même) ;

$subst_\epsilon(b) = 1$ (le symbole b a été utilisé 1 fois au cours de l'apprentissage pour en remplacer un autre) ;

$subst_\epsilon(UNK) = 0$ (le symbole UNK n'est pas utilisé pendant l'apprentissage) ;

D'où : $SUBST_\epsilon = 1$;

$$\Theta_\epsilon(a) = \frac{\lambda(3-2)}{1\times 2} = 0.3$$
$$\Theta_\epsilon(b) = \frac{1-\lambda}{1} = 0.4$$
$$\Theta_\epsilon(UNK) = \frac{\lambda(3-2)}{1\times 2} = 0.3$$

Toutes les valeurs de la ligne *epsilon* de la matrice étant nulles, les probabilités des

opérations correctives sont directement déduites des valeurs Θ_ϵ :

$$\begin{aligned}
&p(\epsilon \to a) = \Theta_\epsilon(a) = 0.3\\
&p(\epsilon \to b) = \Theta_\epsilon(b) = 0.4\\
&p(\epsilon \to UNK) = \Theta_\epsilon(UNK) = 0.3
\end{aligned}$$

2. Ligne 'a'
Les événements $(a \to \epsilon)$ et $(a \to UNK)$ ont des fréquences nulles, on calcule alors les valeurs de $\Theta_a(\epsilon)$ et de $\Theta_a(UNK)$:

$subst_a(\epsilon) = 1$ (le symbole ϵ a été utilisé 1 fois au cours de l'apprentissage pour en remplacer un autre) ;

$subst_\epsilon(UNK) = 0$ (le symbole UNK n'est pas utilisé pendant l'apprentissage) ;

$SUBST_a = 1$;

$$\begin{aligned}
&\Theta_a(\epsilon) = \tfrac{1-\lambda}{1} = 0.4\\
&\Theta_a(UNK) = \tfrac{\lambda(2-1)}{1\times 1} = 0.6
\end{aligned}$$

Les probablilités des opérations correctives sont calculées comme suit :

$$\begin{aligned}
&p(a \to \epsilon) = \tfrac{\lambda(3-2+1)}{5} \times \Theta_a(\epsilon) = 0.09\\
&p(a \to a) = \tfrac{4-\lambda}{5} = 0.68\\
&p(a \to b) = \tfrac{1-\lambda}{5} = 0.08\\
&p(a \to UNK) = \tfrac{\lambda(3-2+1)}{5} \times \Theta_a(UNK) = 0.14
\end{aligned}$$

3. Ligne 'b'
Les événements avec des fréquences nulles sont : $(b \to a)$ et $(b \to UNK)$. Les valeurs de $\Theta_b(a)$ et de $\Theta_b(UNK)$ sont calculées par :

$subst_b(a) = 0$;

$subst_b(UNK) = 0$;

On en déduit : $SUBST_b = 0$, donc $\Theta_b(a) = \frac{1}{2}$ et $\Theta_b(UNK) = \frac{1}{2}$. Les probabilités des opérations correctives sont calculées comme suit :

$$\begin{aligned}
&p(b \to \epsilon) = \tfrac{1-\lambda}{5} = 0.08\\
&p(b \to a) = \tfrac{\lambda(3-2+1)}{5} \times \Theta_b(a) = 0.12\\
&p(b \to b) = \tfrac{4-\lambda}{5} = 0.68\\
&p(b \to UNK) = \tfrac{\lambda(3-2+1)}{5} \times \Theta_b(UNK) = 0.12
\end{aligned}$$

4. Ligne 'UNK'
Son calcul est similaire à celui de la ligne 'ϵ' (ici l'opération $(UNK \to UNK)$ n'est pas prise en compte) : $subst_{UNK}(\epsilon) = 1$, $subst_{UNK}(a) = 0$, $subst_{UNK}(b) = 1$ et $SUBST_{UNK} = 2$;

$$\begin{aligned}
&\Theta_{UNK}(\epsilon) = \tfrac{1-\lambda}{2} = 0.2\\
&\Theta_{UNK}(a) = \tfrac{\lambda(3-1)}{2\times 1} = 0.6\\
&\Theta_{UNK}(b) = \tfrac{1-\lambda}{2} = 0.2
\end{aligned}$$

Les probabilités des opérations correctives sont déduites directement des valeurs Θ_{UNK} :

$$\begin{array}{l} p(UNK \to \epsilon) = \Theta_{UNK}(\epsilon) = 0.2 \\ p(UNK \to a) = \Theta_{UNK}(a) = 0.6 \\ p(UNK \to b) = \Theta_{UNK}(b) = 0.2 \end{array}$$

Nous nous proposons alors de trouver le meilleur décodage probabiliste de la séquence ac (qui ne fait pas partie du langage modélisé par l'automate appris). Le symbole c ne figure pas dans l'alphabet du modèle, il est remplacé par l'étiquette UNK. Les dérivations correctives possibles par rapport au modèle inféré sont :

$$\begin{array}{llllll} 1. & (a \to a) & (b \to UNK) & & \\ 2. & (\epsilon \to a) & (\epsilon \to UNK) & (a \to \epsilon) & (b \to \epsilon) \\ 3. & (a \to a) & (\epsilon \to UNK) & (b \to \epsilon) & \\ 4. & (b \to a) & (b \to UNK) & & \\ 5. & (\epsilon \to a) & (\epsilon \to UNK) & (b \to \epsilon) & (b \to \epsilon) \\ 6. & (b \to a) & (\epsilon \to UNK) & (b \to \epsilon) & \end{array}$$

La plus forte probabilité de la séquence aa est donnée par la première dérivation :

$$\begin{array}{rl} p(\mathrm{ac}|G) = & (1-\beta) \times p(A \to aB) \times p(a \to a) \\ & \times \\ & (1-\beta) \times p(B \to bC) \times p(b \to UNK) \\ & \times \\ & p(C \to \epsilon) \end{array}$$

En prenant $\beta = 0.05$:

$$\begin{array}{rcl} p(\mathrm{ac}|G) & = & 0.95 \times 1.0 \times 0.68 \\ & & \times \\ & & 0.95 \times 0.8 \times 0.12 \\ & & \times \\ & & 1.0 = \\ & = & 0.059 \end{array}$$

□

5.5 Évaluation expérimentale des modèles lissés

Nous regroupons dans cette section les résultats obtenus par l'approche proposée ci-dessus. L'évaluation a été menée sur les modèles appris sur les trois corpus figurant dans l'Annexe A. La première partie (5.5.1) décrit les expériences effectuées avec un lexique fermé, la deuxième (5.5.2) montre les résultats obtenus en prenant en compte la modélisation du mot inconnu. Enfin dans la troisième (5.5.3) nous comparons l'efficacité des modèles ainsi conçus avec les modèles bigrams .

La méthode de lissage des grammaires régulières par estimation des probabilités des opération d'édition fait intervenir deux paramètres :

λ : la masse de probabilités soustraite à une règle r et ajoutée aux règles non vues, possédant les mêmes non-terminaux que r.

β : la masse de probabilités soustraite à un ensemble de règles avec le même non-terminal à gauche et ajoutée aux règles d'*insertion*, possédant ce même symbole non-terminal à droite et à gauche de la règle.

Les courbes présentées dans cette section montrent la perplexité calculée pour chaque variation de ces deux paramètres. Les valeurs optimales de λ et de β sont obtenues pour les valeurs de perplexité les plus faibles.

5.5.1 Lissage à lexique fermé

La matrice de fréquences (notion introduite dans la section 4.4.3, page 78) est construite à partir du vocabulaire du corpus d'apprentissage, augmenté des symboles ϵ (mot vide) et UNK (mon iconnu). Sauf contre-indication, elle est obtenue à partir de la première itération du processus d'inférence initialisé avec les coûts unitaires (choix fait au vu des résultats montrés dans la section 4.4.3).

Langage L_0

Les corpus d'apprentissage et de test pour ce langage ont été construits à partir de la grammaire présentée dans l'Annexe A sur la Fig. A.1 (page 142). La taille des corpus d'apprentissage et de test est respectivement 10 000 et 1 000 séquences. Les résultats présentés sont une moyenne obtenue sur 10 corpus générés.

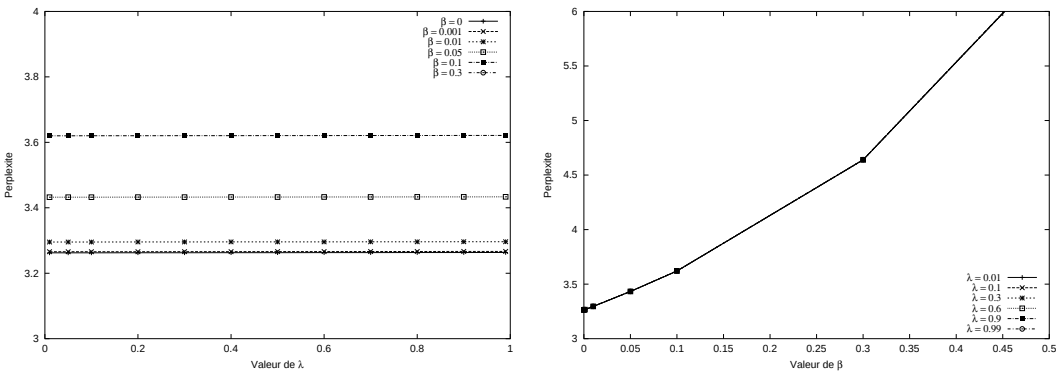


Fig. 5.5: Corpus L_0 : à gauche : courbes de perplexité en fonction du paramètre λ à β fixe, à droite : courbes de perplexité en fonction du paramètre β à λ fixe (les courbes sont superposées).

Ces courbes (Fig. 5.5) sont relativement peu significatives. Le lissage revêt ici peu d'importance car plus de 99% de séquences du corpus de test sont reconnues par des grammaires inférées. Ainsi la valeur optimale de β vaut 0 ce qui veut dire que les règles d'insertions ne sont quasiment pas utilisées lors du lissage. L'influence du paramètre

λ est également très faible (on le voit en particulier sur la courbe de gauche) car les règles non vues au cours de l'apprentissage sont rarement utilisées.

Corpus AGS

1. Corpus original. Pour lisser le modèle appris, nous avons utilisé deux matrices de fréquences obtenues après une itération :

M_{ECGI}^{unit} – en initialisant l'inférence avec les coûts unitaires ;

M_{ECGI}^{CGF} – en initialisant l'inférence avec les coûts déduits des catégories grammaticales des mots (cf. section 4.4.2).

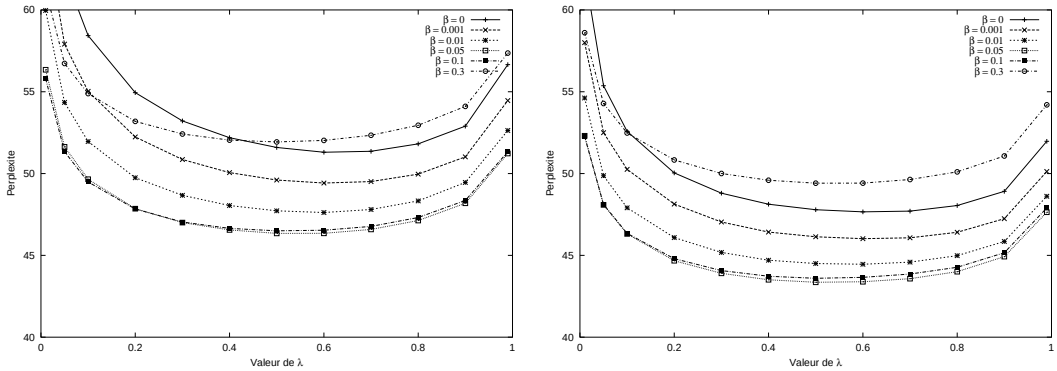


FIG. 5.6: Corpus AGS original : courbes de perplexité en fonction du paramètre λ à β fixe. À gauche : lissage avec M_{ECGI}^{unit}, à droite : lissage avec M_{ECGI}^{CGF}.

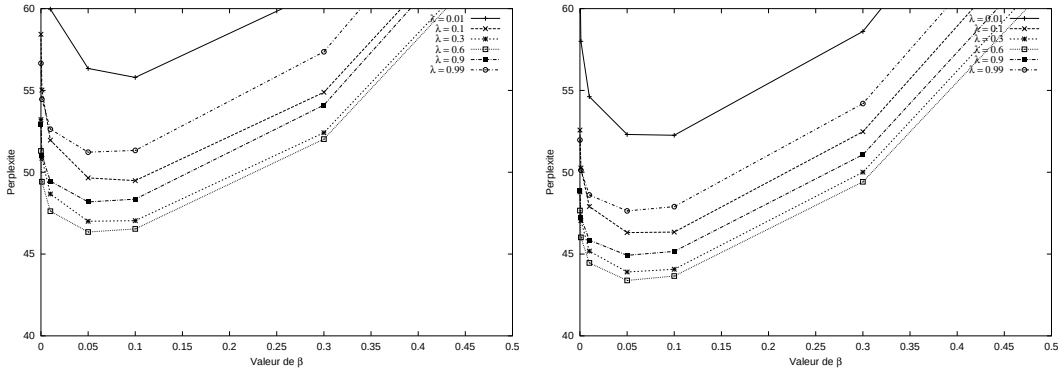


FIG. 5.7: Corpus AGS original : courbes de perplexité en fonction du paramètre β à λ fixe. À gauche : lissage avec M_{ECGI}^{unit}, à droite : lissage avec M_{ECGI}^{CGF}.

Les figures 5.6 et 5.7 permettent de déterminer les valeurs optimales de λ et β. Pour la valeur de perplexité la plus basse, dans les deux cas (coûts unitaires et coûts "linguistiques") $\lambda_{optimal} = 0.6$ (valeur caractéristique pour le lissage de type *discounting*

des modèles N-grams), $\beta_{optimal} = 0.05$. Cette dernière valeur indique qu'en moyenne l'opération d'insertion est utilisée avec un taux de 5% au cours du décodage.

Par ailleurs, la comparaison des deux paramétrages par coûts des opérations d'édition, montre la pertinence d'utilisation des côuts obtenus à partir des catégories grammaticales de la langue française. Ce résultat a été mentionné dans la section 4.4.3.2.

2. Corpus brassé. La Remarque 15 (page 140) et l'étude menée dans la section 5.2.2 sont à l'origine de résultats présentés ci-dessous. La création du corpus brassé AGS_{bis} a suivi le même protocole que celui décrit dans la section 5.2.2.

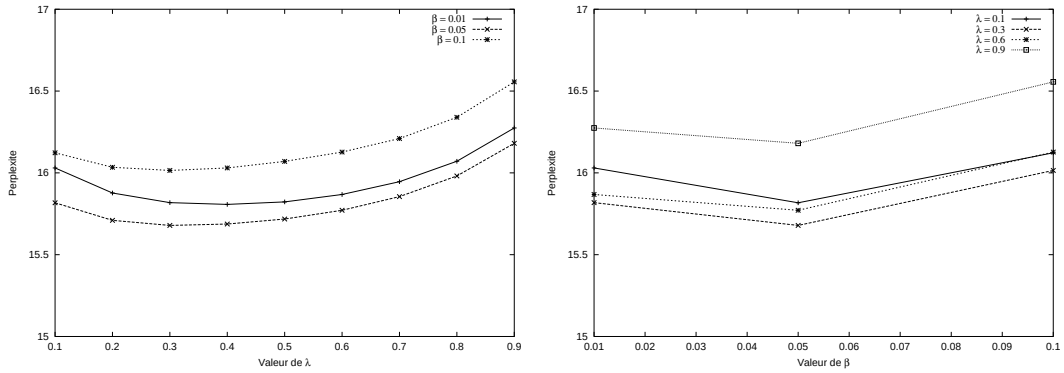


FIG. 5.8: Corpus AGS_{bis} : à gauche : courbes de perplexité en fonction du paramètre λ pour un β fixe, à droite : courbes de perplexité en fonction du paramètre β à λ fixe.

La FIG. 5.8 indique que la perplexité la plus faible ($PP(A(AGS_{bis})) = 16.3$) est obtenue pour $\lambda_{optimal} = 0.3$ et $\beta_{optimal} = 0.05$. Ce résultat a été mentionné dans la section 5.2.2.

Corpus ATIS

Les résultats obtenus avec le corpus ATIS sont représentés sur la FIG. 5.9. La perplexité la plus basse $PP(A(ATIS)) = 29.7$) correspond à des valeurs $\lambda_{optimal} = 0.3$ et $\beta_{optimal} = 0.1$.

Remarque 10 *Les paramètres $\lambda_{optimal}$ et $\beta_{optimal}$, déterminés ici par voie expérimentale, peuvent être également estimés automatiquement, par exemple par des techniques de type "Leaving-one-out" ([Ney & Essen, 1993]).* □

5.5.2 Lissage avec modélisation du mot inconnu

L'expérience décrite ici illustre la méthode de modélisation des mots inconnus proposée dans la section 5.4.2.

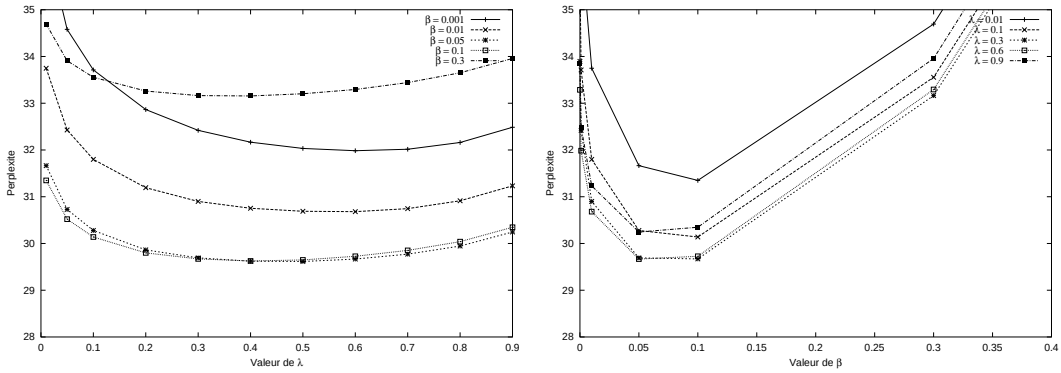


FIG. 5.9: Corpus ATIS : à gauche : courbes de perplexité en fonction du paramètre λ à β fixe, à droite : courbes de perplexité en fonction du paramètre β à λ fixe.

Protocole expérimental

Modéliser les mots inconnus consiste à attribuer des probabilités à des opérations $(x \to y)$ et $(y \to x)$ avec $x \in \Sigma \cup \{\epsilon\}$ et $y \notin \Sigma \cup \{\epsilon\}$. Pour estimer ces probabilités nous divisons un corpus d'apprentissage C en 11 parties égales $C_1, \ldots, C_{11}$. Chaque corpus C_i devient alors un corpus de test, les corpus restants étant réunis pour former les corpus d'apprentissage. Nous disposons alors de 11 couples de corpus : apprentissage : $C^i_{app} = \cup\{C_j\}\ \forall j \in [1..11],\ j \neq i$ et test : $C^i_{test} = C_i$, leur proportion en taille (en nombre de phrases) est de 10/1. La matrice M_{ECGI} est construite sur Σ_C (le lexique du corpus C) : $M_{ECGI} = |\Sigma_C \cup \{\epsilon, UNK\}| \times |\Sigma_C \cup \{\epsilon, UNK\}|$. Les valeurs de M_{ECGI} sont initialisées à 0. Les 11 grammaires G_i sont inférées sur les corpus C^i_{app} (avec les coûts unitaires), les fréquences des opérations d'éditions sont comptabilisées dans M_{ECGI} selon la procédure décrite dans la section 4.4.3. Ensuite, une analyse corrective est menée pour chaque corpus C^i_{test}, les mots du corpus de test qui ne figurent pas dans le corpus d'apprentissage sont utilisés pour mettre à jour les fréquences des opérations $(x \to UNK)$ et $(UNK \to x)$. La matrice résultante est alors utilisée pour lisser la grammaire construite sur C.

Langage L_0

Ce langage possède 25 éléments terminaux, tous figurant dans les corpus d'apprentissage et de test (possédant respectivement 10 000 et 1 000 séquences). Il n'est pas pertinent de modéliser un mot inconnu dans ce cas (d'autant plus que les valeurs de perplexité ne varient pas de façon significative pour le lissage sans cette modélisation).

Corpus AGS

Les caractéristiques de 11 corpus construits à partir du corpus d'apprentissage AGS sont regroupées dans le tableau ci-dessous (les tailles de lexiques sont exprimées en nombre des mots :

Numéro du corpus	Taille du lexique du corpus d'apprentissage	Taille du lexique du corpus de test	Nombre de mots absents du corpus d'apprentissage
1	804	378	17
2	806	372	15
3	805	408	16
4	799	407	22
5	799	414	22
6	801	391	20
7	806	369	15
8	800	402	21
9	802	413	19
10	807	408	14
11	800	390	21

La comparaison des résultats du lissage des grammaires sans et avec la modélisation du mot inconnu dans le corpus AGS est présentée sur les figures 5.10 et 5.11. La perplexité baisse de façon significative : $PP(A(AGS)) = 42.4$ (avec $\lambda_{optimal} = 0.4$ et $\beta_{optimal} = 0.05$) contre une valeur de 47.2 pour le lissage à "lexique fermé".

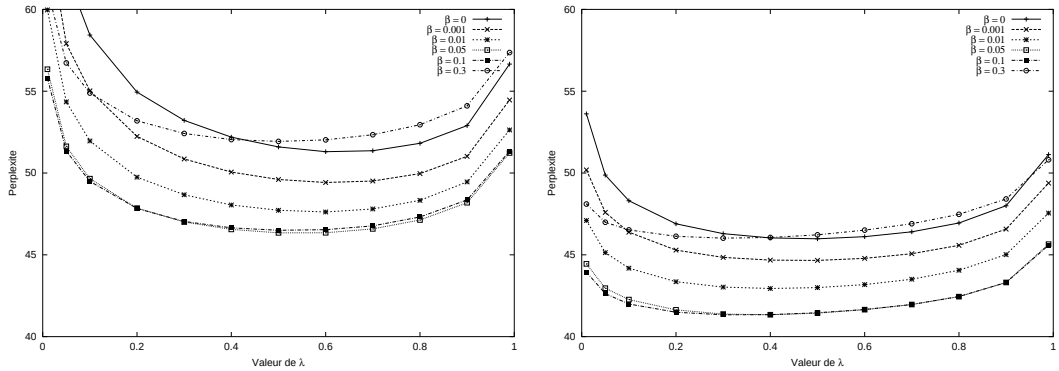


FIG. 5.10: Corpus AGS : courbes de perplexité en fonction du paramètre λ à β fixe. À gauche : lissage à lexique fermé, à droite : lissage avec modélisation du mot inconnu.

Corpus ATIS

Les caractéristiques de 11 corpus construits à partir du corpus d'apprentissage ATIS sont regroupées dans le tableau ci-dessous :

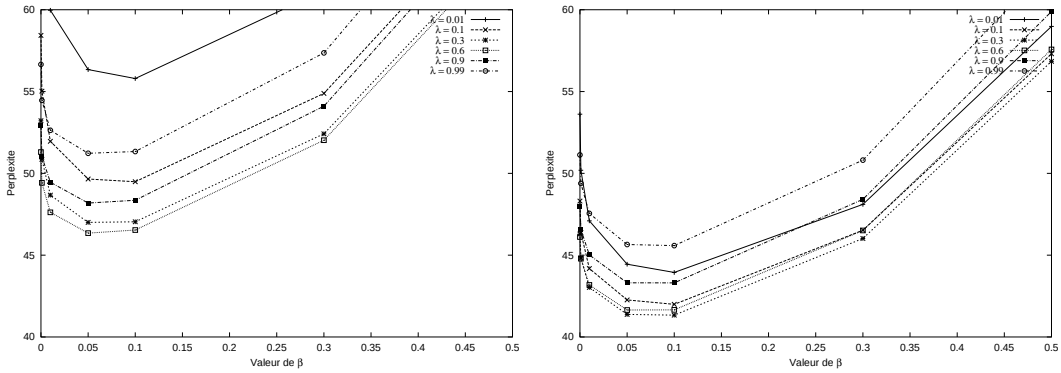


FIG. 5.11: Corpus AGS : courbes de perplexité en fonction du paramètre β à λ fixe. À gauche : lissage à lexique fermé, à droite : lissage avec modélisation du mot inconnu.

Numéro du corpus	Taille du lexique du corpus d'apprentissage	Taille du lexique du corpus de test	Nombre de mots absents du corpus d'apprentissage
1	1116	563	46
2	1128	549	34
3	1133	551	29
4	1136	542	26
5	1135	527	27
6	1115	579	47
7	1122	549	40
8	1132	537	30
9	1135	548	27
10	1131	541	31
11	1130	548	32

La comparaison des résultats du lissage des grammaires sans et avec la modélisation du mot inconnu dans le corpus ATIS est présentée sur les figures 5.12 et 5.13. La perplexité baisse de façon moins importante que dans le cas du corpus précédent, passant de $PP(A(ATIS)) = 29.7$ pour le lissage à "lexique fermé" à $PP(A(ATIS)) = 28.4$ pour le lissage avec modélisation du mot inconnu (avec $\lambda_{optimal} = 0.3$ et $\beta_{optimal} = 0.1$).

Commentaire

On constate une nette baisse de la perplexité lors de lissage de modèle avec la matrice qui prend en compte les fréquences d'utilisation des mots hors-vocabulaire. Nous l'interprétons de deux façons :

- L'apport de l'information sur le comportement du mot inconnu, c'est à dire sur les opérations : $(x \to UNK)$ et $(UNK \to x)$ avec $x \in \Sigma \cup \{\epsilon\}$ améliore la qualité du modèle. Certains mots du vocabulaire ont plus de chances que d'autres de jouer le rôle (au sens syntaxique) du mot que le modèle acoustique a estimé comme

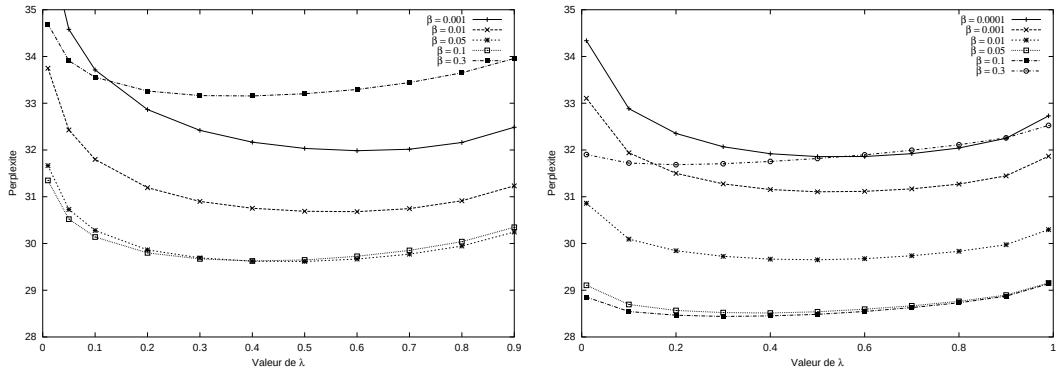


FIG. 5.12: Corpus ATIS : courbes de perplexité en fonction du paramètre λ pour un β fixe. À gauche : lissage à lexique fermé, à droite : lissage avec modélisation du mot inconnu.

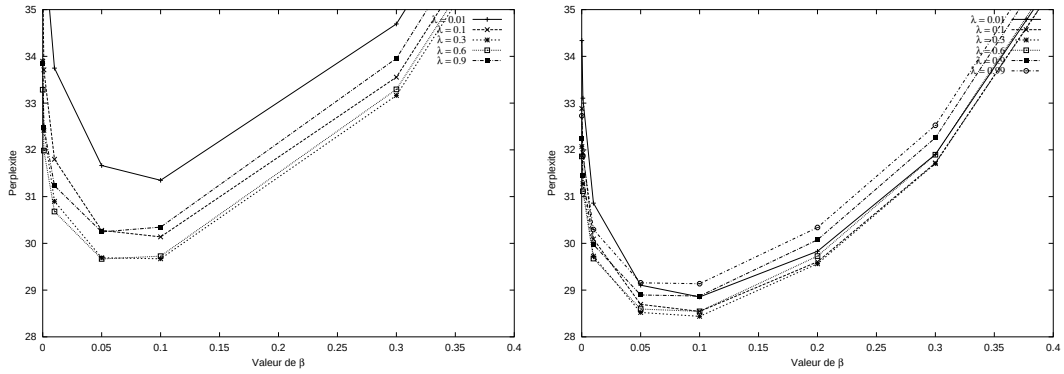


FIG. 5.13: Corpus ATIS : courbes de perplexité en fonction du paramètre β pour un λ fixe. À gauche : lissage à lexique fermé, à droite : lissage avec modélisation du mot inconnu.

"inconnu" (par exemple en revoyant une étiquette associée à cet événement).

- L'estimation des probabilités des opérations $(x \to y)$ avec $x, y \in \Sigma \cup \{\epsilon\}$ est meilleure. Les fréquences des opérations correctives les plus significatives sont renforcées par l'application itérative du calcul de la matrice M_{ECGI}.

5.5.3 Grammaires Stochastiques *versus* Bigrams

Nous comparons ici les perplexités des modèles bigram et des grammaires régulières stochastiques sur les trois corpus décrits en Annexe A.

Lors du calcul de la perplexité, les bigrams ont été lissés par une technique "Absolute Discountig", et les grammaires par celle proposée dans le Chapitre 5, issu de cette même technique. Le paramètre λ est variable pour le lissage des bigrams et des grammaires, le paramètre β est choisi en fonction des résultats obtenus dans les sections 5.5.1 et 5.5.2 :

$\beta_{optimal} = 0$ pour le langage L_0, $\beta_{optimal} = 0.05$ pour le corpus AGS et $\beta_{optimal} = 0.1$ pour le corpus ATIS.

Langage L_0

La comparaison faite sur les modèles appris sur un corpus du langage L_0 (cf. FIG. 5.14) donne des résultats peu significatifs vu la facilité de la tâche (la grammaire symbolique inférée reconnaît pratiquement toutes les phrases du corpus de test). Remarquons cependant que les grammaires lissées sont au même niveau de performances (voir legèrement supérieures) que les bigrams sur un langage non-trivial (malgré sa petite taille, la grammaire de L_0 est capable de générer environ quinze millions de phrases différentes). Il est raisonnable d'espérer qu'en présence d'un corpus de taille statistiquement satisfaisante, les grammaires seraient capables de rivaliser avec les modèles N-grams.

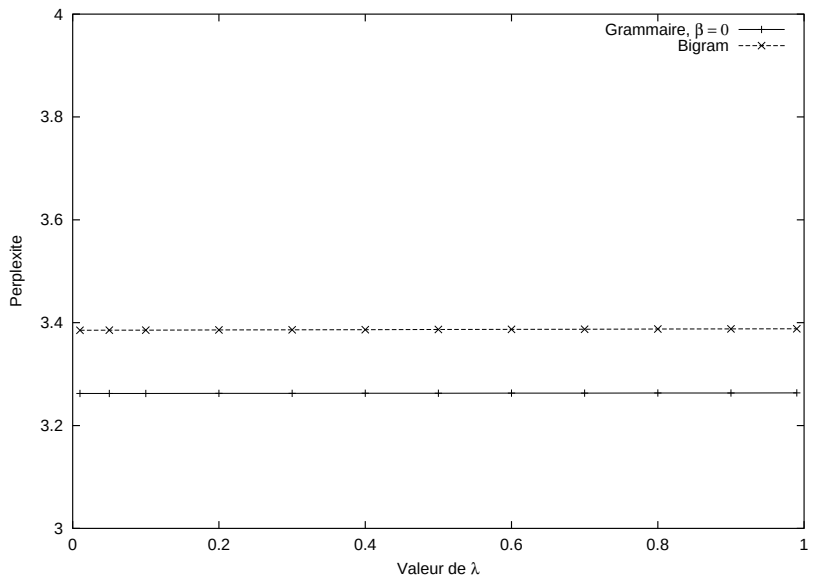


FIG. 5.14: Perplexité du bigram *versus* perplexité d'une grammaire stochastiques régulière apprise sur le corpus artificiel issu du langage L_0.

Corpus AGS

Pour le corpus AGS, la perplexité de la grammaire est environ deux fois plus grande que la perplexité du bigram (FIG. 5.15, courbe de gauche), ce qui est un résultat décevant. Les tests effectués sur un corpus construit à partir du brassage des séquences du AGS original (voir section 5.2.2) montrent que cette différence diminue de manière significative (FIG. 5.15, courbe de droite). Cela montre une sensibilité plus importante des grammaires à la représentativité statistique des données[3] : les grammaires étant

des modèles mixtes, symboliques et statistiques, l'absence de constructions syntaxiques importantes dans le corpus d'apprentissage pénalise fortement la qualité du modèle.

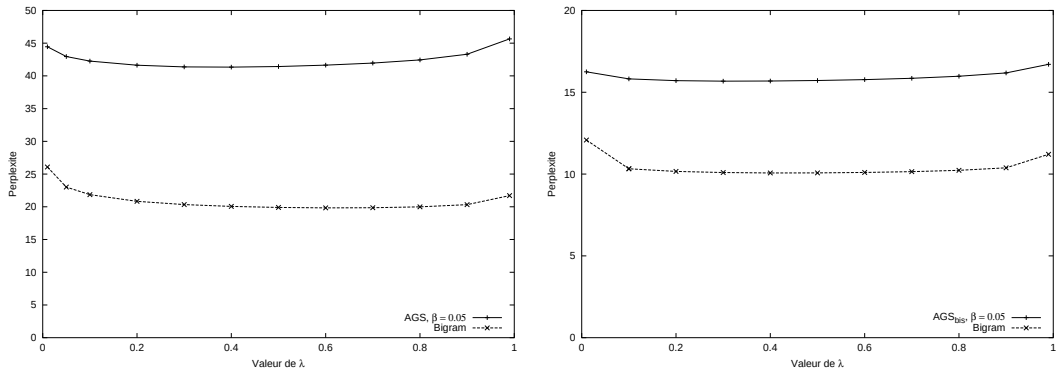


FIG. 5.15: Perplexité du bigram *versus* perplexité d'une grammaire stochastique régulière apprise sur le corpus AGS. À gauche : perplexité calculée sur le corpus de test original, à droite perplexité calculée sur le corpus AGS brassé.

Corpus ATIS

Les résultats obtenus sur le corpus ATIS (FIG. 5.16) sont comparables à ceux obtenus sur le corpus AGS brassé en termes de performances des bigrams ($PP = 20$) versus les celles des grammaires ($PP = 30$). La tâche ATIS semble cependant être plus difficile : le nombre de phrases distinctes est deux fois plus important, les séquences sont en moyenne deux fois plus longues, le lexique du langage représenté est plus riche. À l'examen, ce corpus semble également contenir un nombre de constructions syntaxiques plus riche. Cela explique la taille des grammaires apprises (environ 28 000 états et 42 000 transitions contre 7 500 états et 11 000 transitions en moyenne pour les grammaires apprises sur le corpus AGS).

5.6 Conclusion

Le lissage d'automates stochastiques, en particulier obtenus par inférence grammaticale, est un problème nouveau, dont l'importance a été montrée notamment dans [Thollard, 2000]. Nous avons proposé dans ce chapitre une méthode de lissage basée sur l'utilisation de la distance corrective. Ceci permet de quantifier la qualité de la généralisation obtenue en mesurant la perplexité de la grammaire sur l'ensemble de test. Cette méthode possède la caractéristique d'être homogène avec la méthode d'apprentissage : il nous semble que traiter par un concept commun (la correction d'erreur) l'apprentissage de l'automate, la réestimation des poids, l'affectation de probabilité aux règles apprises, puis le lissage[4] de l'automate appris est non seulement satisfaisant d'un

[3] ce qui est, en fin de compte, rassurant...

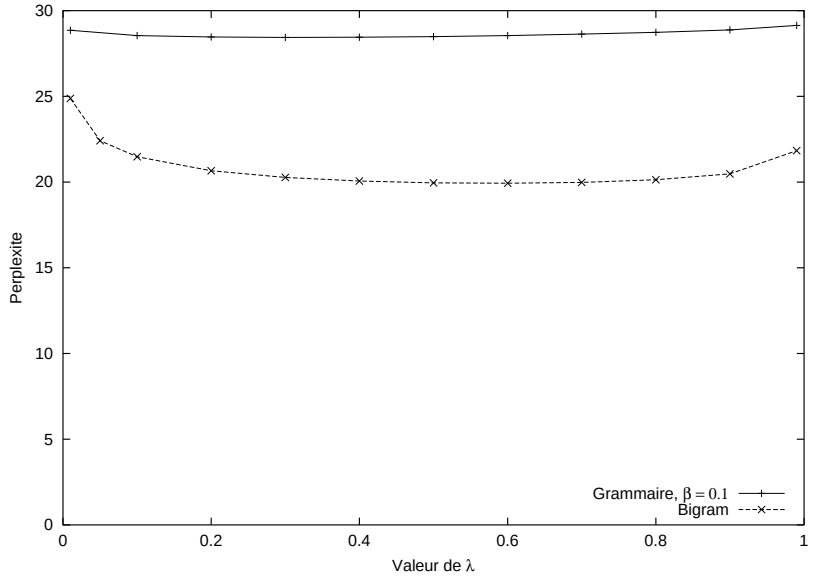


FIG. 5.16: Perplexité du bigram *versus* perplexité d'une grammaire stochastique régulière apprise sur le corpus ATIS.

point de vue de principe, mais donne des résultats intéressants sur un corpus réel de langue naturelle.

[4]Par communication interne au cours d'un séminaire informel (St-Etienne, 17 janvier 1999), nous avons appris l'existence de travaux similaires sur le lissage des automates stochastiques par correction d'erreurs. Les auteurs sont arrivés par une voie indépendante à des résultats analogues, en généralisant la méthode de lissage à des automates cycliques. Depuis, cette technique a fait sujet d'une publication : [Dupont & Amengual, 2000], elle est legèrement ultérieure à la publication de notre approche dans [Chodorowski & Miclet, 1999].

Chapitre 6

Modèles de langage à base de classes

Dans les applications TALN/TALP, sur les tâches réelles, il n'est pas rare de rencontrer des lexiques de taille importante (au delà de 1 000 mots). Cela suscite des problèmes à différents niveaux :

- *Validité statistique* : le chapitre précédent explique la nécessité du lissage du fait de l'existence d'événements non-vus. La taille du lexique influe directement sur leur nombre : à taille égale, le corpus avec le lexique plus grand est statistiquement moins "informatif".
- *Problème calculatoire* : la taille des modèles vus dans les chapitres précédents croît exponentiellement avec la taille de l'ensemble de symboles sur lequel ils sont construits. Ceci peut poser un problème pratique dans la gestion du compromis *espace de mémoire – temps de traitement*, particulièrement crucial en traitement de la parole.

Un des moyens de pallier ce problème est de réduire le nombre de symboles en les *regroupant en classes*. Dans ce chapitre, nous verrons comment construire et utiliser les classes de mots dans les modèles de langage, en particulier ceux utilisés pour la reconnaissance de la parole. Ainsi, la première partie du chapitre traite des N-grams à base de classes. Cette étude est suivie par l'application des techniques semblables en utilisant les grammaires stochastiques régulières : nous détaillons l'usage des classes dans les AFS et discutons d'une technique de construction des groupes de mots en employant ces mêmes modèles dans le critère de classement.

6.1 Utilisation des classes dans les modèles de langage

Nous écartons préalablement de notre étude le cas dans lequel un mot peut appartenir à plusieurs classes : il a été montré dans [Jelinek, 1998] que le processus de décodage à employer est complexe car il impose de supposer que le mot courant peut appartenir à chacune des classes et considérer l'ensemble de l'historique. Le nombre d'éléments à

estimer est trop important pour pouvoir utiliser cette approche de façon avantageuse.

Dans la suite de ce chapitre, nous utiliserons une surjection[1] $x_i \mapsto C_j$, avec $x_i \in \Sigma$ et $C_j \in \Sigma_C$, où Σ_C est un ensemble de symboles représentant des classes. On a donc : $|\Sigma| \geq |\Sigma_C|$. On accumule de cette manière certaines fréquences des événements impliquant les symboles de même classe, fournissant une information statistique plus satisfaisante, tout en réduisant le nombre de paramètres à estimer. Le problème est réorienté vers le choix des méthodes de classifications ainsi que vers l'estimation des probabilités $p(x_i \mapsto C_j)$.

6.1.1 Les N-grams

Les N-grams à base de classes sont un cas particulier des N-grams traditionnels : ils utilisent les fréquences de succession des classes de mots. La probabilité conditionnelle d'un mot w_n est définie par deux facteurs : la probabilité d'une classe étant données les classes précédentes et la probabilité du mot étant donnée une classe :

$$p(w_k|w_{k-N+1}\ldots w_{k-1}) \simeq p(w_k|C_k)p(C_k|C_{k-N+1}\ldots C_{k-1})$$

Ces deux facteurs sont déduits du corpus par l'estimation du maximum de vraisemblance :

$$p(w_k|C_k) = \frac{n_{occ}(w_k)}{n_{occ}(C_k)}$$

$$p(C_k|C_{k-N+1}\ldots C_{k-1}) = \frac{n_{occ}(C_{k-N+1}\ldots C_{k-1}C_k)}{\sum_{i=1}^{i=|C|} n_{occ}(C_{k-N+1}\ldots C_{k-1}C_i)}$$

La fonction $n_{occ}(x)$ retourne le nombre d'occurrences de l'événement x dans le corpus d'apprentissage.

Plusieurs travaux ([Ney & Knesser, 1993; Jelinek, 1990; Samuelsson & Reichl, 1999]) montrent que l'utilisation des N-grams à base de classes s'avère efficace dans la réduction de la perplexité et du taux d'erreur de reconnaissance. Ce constat nous a incité à utiliser les classes de symboles avec les grammaires stochastiques. La description de la méthode est contenue dans les paragraphes suivants.

6.1.2 Les grammaires régulières

Quasiment la totalité des travaux sur l'utilisation des classes dans les modèles de langage se rapporte aux modèles N-grams, présentés dans la section précédente. Des travaux récents ([Dupont & Chase, 1998]) se sont intéréssés à l'emploi des classes de mots avec les grammaires régulières, en montrant qu'il est possible par ce biais d'améliorer la qualité d'une grammaire apprise. Nous nous proposons de formaliser ici ce type d'approche, puis de l'associer aux résultats du chapitre précedent.

[1] nous employons également le mot anglais "mapping".

6.1.2.1 Construction du modèle

Dans cette partie du chapitre, nous supposons disposer d'une fonction $f_C : x_i \mapsto C_j$ qui associe à un symbole x_i une seule classe C_j. Le modèle est appris de manière habituelle (ici : avec l'algorithme ECGI) sur les séquences composées sur Σ_C. En pratique, cela consiste à étiqueter le corpus d'apprentissage par les symboles de Σ_C selon le mapping f_C. La grammaire ainsi apprise contient *a priori* moins de règles : des sous-séquences différentes sur Σ sont remplacées par une suite de symboles identiques sur Σ_C, comme le montre l'exemple suivant :

Exemple 1 :
Soient :
$\Sigma = \{je, j', il, lis, apprends, apprend, un, le, poème, jeu\}$,
$\Sigma_C = \{N, V, D\}$
et

$$f_C : \left\{ \begin{array}{l} je \mapsto N \\ j' \mapsto N \\ il \mapsto N \\ poème \mapsto N \\ jeu \mapsto N \\ apprends \mapsto V \\ apprend \mapsto V \\ lis \mapsto V \\ un \mapsto D \\ le \mapsto D \end{array} \right\}$$

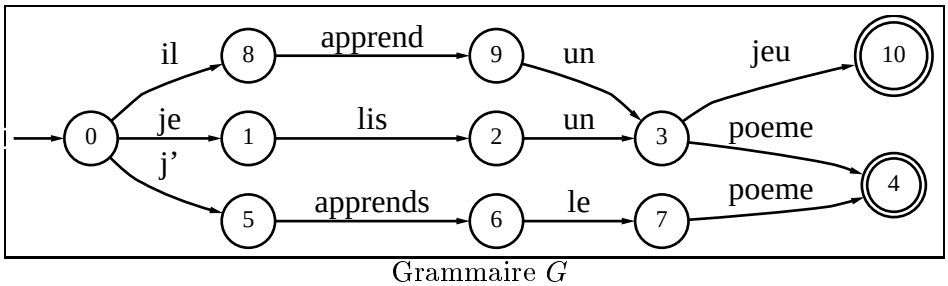
Soit le corpus d'apprentissage :

$$I_\Sigma = \left\{ \begin{array}{l} je\ lis\ un\ poème \\ j'\ apprends\ le\ poème \\ il\ apprend\ un\ jeu \end{array} \right\}$$

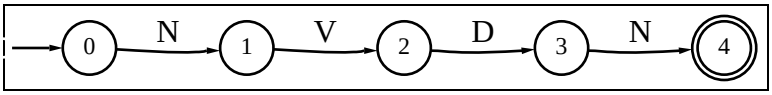
L'application de f_C sur les éléments des séquences de I_Σ donne un nouveau corpus :

$$I_{\Sigma_C} = \left\{ \begin{array}{l} N\ V\ D\ N \\ N\ V\ D\ N \\ N\ V\ D\ N \end{array} \right\}$$

Les grammaires G et G_C apprises respectivement sur I_Σ et sur I_{Σ_C} sont :



Grammaire G



Grammaire G_C

Dans cet exemple trivial, nous avons attribué implicitement des catégories grammaticales aux éléments de Σ_C : V = verbe, N = nom, D = déterminant. La simplicité des constructions syntaxiques présentes dans I_Σ est mise en relief par la grammaire apprise sur I_{Σ_C}. □

L'obtention des classes des mots est discutée plus loin dans ce chapitre.

6.1.2.2 Décodage des séquences

Soit une séquence $x = x_1x_2x_3 \ldots x_T \in \Sigma^*$. Le décodage probabiliste d'une séquence est fait de façon suivante :

$$P(x|G_C) = \sum_{\mathbf{d}(x)\in\{\mathbf{d}\}} (\prod_{i=1}^{T} p(f_C(x_i))p(x_i|f_C(x_i)))$$

où :

$\mathbf{d}(x)$ est une dérivation de x par G (définie dans la section 1.2.2.3).

$p(f_C(x_i))$ est la probabilité de la règle qui consomme le symbole $f_C(x_i)$.

$p(x_i|f_C(x_i))$ est la probabilité d'appartenance du symbole x_i à la classe que lui attribue le mapping f_C

Remarque 11 *Dans le cas des grammaires déterministes, il n'y a qu'une seule dérivation* $\mathbf{d}(x)$*, le terme* $\sum_{\mathbf{d}(x)\in\{\mathbf{d}\}}$ *est présent ici pour prendre en compte les modèles non-déterministes.* □

Les probabilités $p(f_C(x_i))$ sont calculées lors de l'apprentissage, les valeurs de $p(x_i|f_C(x_i))$ sont estimées de la même façon que dans le cas des modèles N-grams. Nous notons :

C_j : la classe de l'élément x_i, donnée par f_C ;

$|C_j|$: nombre d'éléments différents que contient la classe C_j ;

$\|C_j\|$: nombre d'occurrences d'éléments dans la classe C_j :

$$\|C_j\| = \sum_{i=1}^{|C_j|} n_{occ}(x_i), x_i \in C_j.$$

Alors :

$$p(x_i|C_j) = \frac{n_{occ}(x_i)}{\|C_j\|}$$

6.1.2.3 Constructions syntaxiques non vues

Bien que, comme nous l'avons vu plus haut, l'utilisation des classes de mots permet de faire une première généralisation sur les règles de construction des phrases, il est rare d'observer dans un corpus d'apprentissage réel tous les agencements possibles entre les catégories syntaxiques (ceci dépend bien sûr du nombre de classes choisies : dans le cas le plus extrême, une seule classe peut rassembler l'ensemble de mots — ceci revient à construire un *unigram*). De même, dans le cas pratique, la généralisation faite par l'algorithme d'apprentissage n'est pas suffisante pour construire un modèle capable de reconnaître tout le langage ciblé. La solution consiste ici encore à lisser la grammaire des classes. La méthode décrite dans le Chapitre 5 peut être directement appliquée dans ce cas. La possibilité d'occurrence d'un mot inconnu nécessite cependant la redéfinition de la fonction f_C.

Traitement du mot inconnu Le problème de décodage des séquences avec des symboles nouveaux a été traité dans la section 5.4.2. L'extension de cette méthode au cas des grammaires apprises sur les classes de symboles est effectuée comme décrit ci-dessous.

Pour prendre en compte des symboles absents du corpus d'apprentissage nous ajoutons à l'ensemble des classes une classe supplémentaire, notée C_{UNK} comportant un seul symbole UNK. Les probabilités des règles de correction impliquant cette classe sont estimées selon la méthode que nous avons proposée dans le chapitre précedent. Au cours du décodage des phrases, chaque mot hors-vocabulaire est remplacé par le symbole UNK. Afin de conserver la consistance du modèle, ce symbole est attribué à la classe C_{UNK} avec une probabilité de 1 (car : $|C_{UNK}| = 1$: voir Remarque 12 ci-dessous). C'est un biais heuristique : on considère que chaque "mot inconnu" se comporte de la même manière dans le même contexte, ce qui est, bien sûr, une approximation très arbitraire.

Remarque 12 *La formule de Bayes relie la probabilité conjointe d'un symbole x_i et de sa classe C_j à leurs probabilités conditionnelles :*

$$p(x_i, C_j) = p(x_i|C_j)p(C_j) = p(C_j|x_i)p(x_i)$$

d'où :

$$p(x_i|C_j) = \frac{p(C_j|x_i)p(x_i)}{p(C_j)} = \frac{p(x_i)}{p(C_j)}$$

car $p(C_j|x_i) = 1$: un mot n'appartient qu'à une seule classe. Or :

$$p(x_i) = \frac{n_{occ}(x_i)}{\sum_{C_j \in \{C\}} \|C_j\|} \text{ et } p(C_j) = \frac{\|C_j\|}{\sum_{C_j \in \{C\}} \|C_j\|}$$

donc :

$$p(x_i|C_j) = \frac{n_{occ}(x_i)}{\|C_j\|}$$

Dans le cas particulier où il n'y a qu'un symbole dans une classe ($|C_j| = 1$) on a : $n_{occ}(x_i) = \|C_j\|$ et donc $p(x_i|C_j) = 1$. □

Formellement, le calcul de la probabilité d'un symbole $x_i \in \{\Sigma \cup UNK\}$ connaissant sa classe C_j se présente comme suit :
1. La fonction f_C est remplacée par une fonction $f_C^{'}$ telle que :

$$f_C^{'}(x_i) = \begin{cases} f_C & si\ x_i\ \in\ \Sigma \\ C_{UNK} & sinon. \end{cases}$$

2. La valeur $p(x_i|C_j)$ est calculée par :

$$p(x_i|C_j) = \begin{cases} \frac{n_{occ}(x_i)}{\|C_j\|} & si\ x_i\ \in\ \Sigma \\ 1 & sinon. \end{cases}$$

Ce calcul est illustré dans l'exemple ci-dessous :
Exemple 2 : Soient $C_1 = \{a, b, c\}$, $C_2 = \{d, e\}$ et $C_{UNK} = \{UNK\}$. Les probabilités d'occurrences des symboles dans un corpus sont distribuées (de façon arbitraire dans cet exemple) comme suit :

$$\begin{aligned} p(a) &= 0.1 \\ p(b) &= 0.2 \\ p(c) &= 0.1 \\ p(d) &= 0.2 \\ p(e) &= 0.3 \end{aligned}$$

Une fraction de probabilités, égale à 0.1, est assignée à la classe C_{UNK} d'où :

$$p(UNK) = 0.1$$

Cette distribution satisfait :

$$\sum_{x_i \in \{C_1 \cup C_2 \cup C_{UNK}\}} p(x_i) = 1$$

Le résultat du calcul des valeurs de $p(x_i|C_j)$ est alors resumé dans le tableau suivant :

C_j	$p(C_j)$	x_i	$p(x_i)$	$p(x_i\|C_j)$
C_1	0.6	a	0.1	0.17
		b	0.2	0.33
		c	0.3	0.50
C_2	0.3	d	0.1	0.33
		e	0.2	0.67
C_{UNK}	0.1	UNK	0.1	1.00

□

6.2 Détermination des classes

Lorsque l'ensemble des symboles utilisés pour l'apprentissage est constitué par les mots d'un langage naturel, on peut bénéficier des connaissances *a priori* sur les classes grammaticales (verbe, nom, adjectif, etc.). Une telle approche est discutée dans la section 6.2.1. À l'opposé, il existe des méthodes statistiques pour construire des groupes de symboles, uniquement à partir de l'information contenue dans le corpus. Elles seront présentées dans la section 6.2.2. En fin de cette étude, nous discutons l'utilisation des grammaires stochastiques régulières dans le critère de classification automatique.

6.2.1 Classes de la langue naturelle

Les études sur les grammaires modernes des langues naturelles menées depuis plusieurs décennies ont abouti à une catégorisation précise de chacun des mots présents dans le lexique d'une langue donnée. Dans leur généralité les catégories grammaticales sont communes à toutes les langues[2]. La langue française en comporte neuf : *nom, article, adjectif, pronom, verbe, adverbe, préposition, conjonction, interjection* ; les cinq premières étant des catégories *variables* c'est-à-dire susceptibles d'être modifiées selon le contexte (par déclinaison, conjugaison, accord, ...). Ce nombre est souvent enrichi par l'ajout (ou affinement) d'autres catégories comme des noms propres, des onomatopées, etc.

L'étude des catégories grammaticales (en anglais : Parts-of-Speech) est un domaine très actif du TALN, il existe notamment un nombre important d'outils linguistiques destinés à assigner automatiquement des étiquettes grammaticales à des mots d'un corpus (cf. notamment [El-Bèze & Mérialdo, 1999]). Cependant, comme il a été mentionné plus haut, l'attribution des catégories grammaticales de la langue naturelle est souvent sujette aux ambiguïtés qui ne peuvent être résolues qu'en fonction du contexte (par exemple, le mot `cours` peut être un nom masculin mis au singulier, un nom masculin mis au pluriel ou un verbe mis en troisième personne en singulier). Leur utilisation est alors trop coûteuse dans le traitement de la parole (voir début de la section 6.1). Par ailleurs, le nombre de catégories grammaticales est figé, l'affinement n'est possible que par le biais des traits, par exemple des traits d'accord. Le lecteur peut consulter dans l'Annexe B une classification obtenue avec l'étiqueteur de Brill ([Brill, 1992]) sur les mots du corpus AGS.

6.2.2 Calcul automatique des classes

L'approche consistant en un apprentissage d'un nombre arbitraire de classes de mots à partir d'un corpus porte le nom du *clustering*. Elle consiste à partitionner l'alphabet Σ sur lequel ce corpus est défini. La distribution des mots entre les classes est effectuée selon un critère purement statistique. Le résultat du classement est une surjection : *un mot ne peut appartenir qu'à une seule classe*. C'est la fonction f_C, introduite dans la

[2] à notre connaissance et tout au moins dans la famille des langues indo-européennes.

section 6.1.2.1.

Algorithmes de catégorisation des mots

Deux algorithmes ont été utilisés pour obtenir les partitions de Σ : l'algorithme de Ney ([Ney & Knesser, 1993]) et l'algorithme de Brown ([Brown et al., 1992]).

Algorithme de Ney. L'algorithme de Ney est une approche gloutonne. Il part d'un nombre de classes K fixé *a priori* et d'une assignation initiale mot-classe faite de façon arbitraire[3]. Il consiste à essayer de déplacer chaque mot d'une classe à une autre. Le mot est assigné au final à la classe pour laquelle la perplexité du N-gram à base de classes est la plus faible. Ce processus est réitéré tant que la différence de la perplexité ne descend pas en dessous d'un certain seuil.

Algorithme de Brown. L'algorithme de Brown diffère de l'algorithme précédent par le critère d'optimisation d'une part et par sa structure hiérarchique de l'autre. En effet, le point de départ de l'algorithme consiste à associer chaque mot du vocabulaire à sa propre classe, puis à effectuer des fusions successives jusqu'à un nombre final K de classes fixé *a priori*. Le critère de fusion des classes dérive de l'*information mutuelle* moyenne entre les classes, définie par :

$$I(C_i, C_j) = \sum_{C_i} \sum_{C_j} p(C_i C_j) \log \frac{p(C_i C_j)}{p(C_i)p(C_j)}$$

Le fait de fusionner deux classes entraîne une diminution de l'information mutuelle moyenne. Le critère consiste ainsi à fusionner la paire de classes pour laquelle la perte d'information mutuelle moyenne est minimale.

Utilisation des grammaires régulières stochastiques dans le critère de regroupement des mots

Les deux algorithmes présentés plus haut permettent d'obtenir une classification indépendamment de la méthode d'apprentissage du modèle final. Si cela est satisfaisant du point de vue de la flexibilité de l'approche, il serait intéressant d'utiliser un critère de classement homogène avec le modèle appris. Plus précisemment, le critère du choix d'une classification serait la qualité de la grammaire inférée en utilisant les classes attribuées successivement à chaque mot. Cependant les méthodes qui permettent de recalculer rapidement un N-gram à base de classes à partir de la classification faite à l'étape précédente ne s'appliquent plus lors de l'apprentissage d'une grammaire à base de classes. Un algorithme "force brute" (où un pas consiste à inférer la grammaire correspondante sur chaque classification possible) est inexploitable : son coût calculatoire

[3] en général on assigne $K-1$ mots les plus fréquents aux $K-1$ premières classes, puis tous les mots restants à la dernière classe.

est prohibitif pour les corpus dont nous disposons. L'étude d'une approche qui optimise ce calcul reste à faire.

6.3 Résultats

Les classes utilisées lors de l'inférence ont été calculées sur le corpus AGS uniquement, en utilisant les deux algorithmes cités plus haut :

- algorithme de Brown (FIG. 6.1, courbe de gauche) ;
- algorithme de Ney (FIG. 6.1, courbe de droite).

Les modèles ont été inférés avec les coûts unitaires des opérations ($classe_i \rightarrow classe_j$), en faisant varier le nombre de classes pour chaque critère. Les calculs de perplexité ont été effectués en utilisant la technique de lissage présentée dans le Chapitre 5 : la matrice M_{ECGI} est construite en fonction du nombre de classes utilisées.

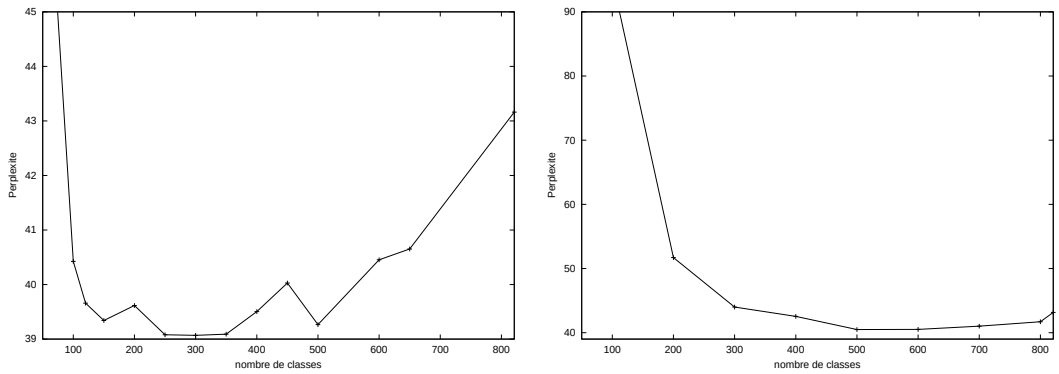


FIG. 6.1: Perplexité du modèle appris avec les classes des mots. À gauche : classes obtenues par algorithme de Brown, à droite : classes obtenues par algorithme de Ney.

La superposition des deux courbes (données ici pour visualiser distinctement le comportement de chaque modèle) est représentée sur la FIG. 6.2. Elle montre que le classement obtenu par le critère d'information mutuelle converge plus vite et donne un meilleur résultat. La perplexité minimale ($PP_{min} = 39.1$) est obtenue pour la grammaire inférée avec 300 classes.

6.4 Conclusion

Le modèle obtenu avec la méthode décrite dans la section 6.1.2 possède la perplexité la plus basse obtenue pour le corpus AGS, en comparaison avec d'autres approches proposées dans ce document. Ce résultat prouve que l'apport de l'information statistique contenue dans le regroupement des mots en classes améliore la qualité du modèle. Il nous est permis de penser qu'en utilisant les grammaires régulières stochastiques dans le critère du partitionnement du lexique, il serait possible d'obtenir un modèle encore

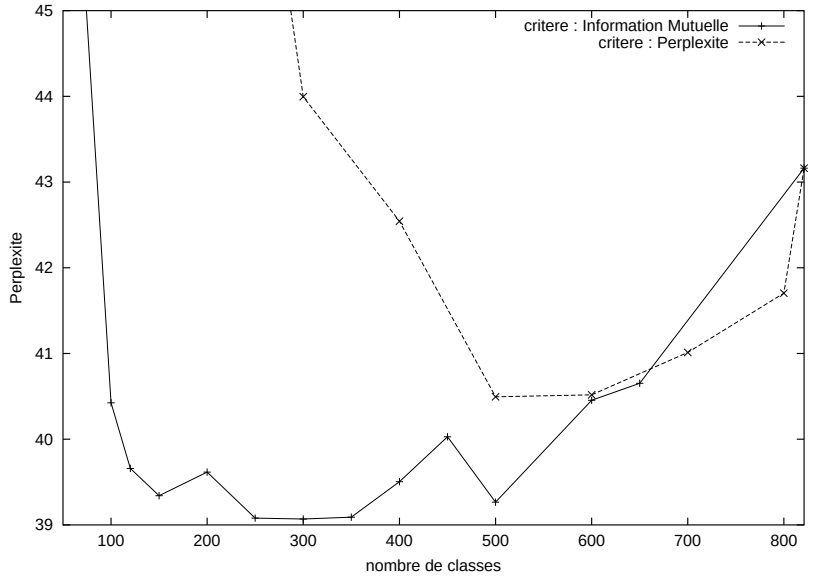


FIG. 6.2: Superposition des deux courbes précedentes.

meilleur et, sur le plan méthodologique, obtenir une approche très satisfaisante du fait de son homogénéité.

Chapitre 7

Réduction de la taille du modèle

Le nombre de paramètres qui interviennent dans le processus de décodage probabiliste de séquences est d'une grande importance car il conditionne l'efficacité de ce processus. Une première approche visant à réduire ce nombre a fait l'objet du chapitre précedent, avec la réduction de la taille de l'alphabet de l'ensemble d'apprentissage. Dans ce chapitre, nous proposons une étude prospective d'une méthode originale de classement des séquences puis de réduction de la taille des grammaires apprises par les techniques correctives. Elle est basée sur les mesures de similitude des séquences que nous avons décrites dans le Chapitre 3.

7.1 Motivation

L'apprentissage incrémental par correction d'erreurs présenté dans le Chapitre 4 construit des modèles dont la taille augmente avec le nombre de constructions syntaxiques présentes dans le corpus. Cet algorithme ne possède pas de paramètre susceptible de contrôler la taille du modèle appris : il est envisageable uniquement de diminuer la taille de l'ensemble d'apprentissage et donc de perdre de l'information statistique et structurelle qui y est contenue. Cette solution est bien sûr peu satisfaisante. Cependant, en traitement de la langue naturelle les corpus d'apprentissage possèdent des propriétés qui peuvent être prises en compte : d'une part, on dispose souvent de données bruitées qui sont traitées à égalité avec les données représentatives d'une tâche donnée ; d'autre part, certaines formes syntaxiques reviennent plus souvent que les autres, du fait de la spécificité du domaine traité. Par exemple, dans le corpus AGS, la phrase :

je voudrais un serveur météo

est ce qu'on aurait tendance à appeler un "patron syntaxique" pour toutes les phrases qui en dérivent comme :

euh je cherche un serveur météorologique
je voudrais la météo de la Bretagne
je voudrais un serveur de la météo locale etc.

Il est facile de constater que certains mots dans ces phrases comme *je, voudrais, serveur, météo* sont utilisées pratiquement à chaque fois, tandis que les autres (*euh, cherche,*

locale, Bretagne, ...) sont des remplacements ou des insertions greffés sur la phrase patron. Cela nous amène à faire l'hypothèse qu'il existe une séquence qui est la plus "représentative" d'un groupe de phrases. On peut la définir comme un "centre" : sa distance moyenne aux autres phrases est la plus petite. Si l'on veut réduire la taille d'une grammaire en la conservant la plus proche (selon la Définition 38 de la page 86) du corpus d'apprentissage il faut alors choisir les séquences les plus représentatives dans ce corpus. Une méthode qui vise à déterminer les *séquences centrales* dans un corpus est proposée dans la section 7.2.

Un autre constat vient du contenu des corpus d'apprentissage. Le corpus ATIS, ainsi que le corpus AGS sont des transcriptions de dialogue. En conséquence, les phrases qu'on y retrouve forment des groupes dont la syntaxe diffère de façon significative. Ainsi les phrases vues plus haut (qui constituent des *requêtes des services* : ce sont en général les premières phrases prononcées) figurent à coté d'autres éléments de dialogue comme par exemple des précisions géographiques :

dans les Ardennes
Bretagne
pour le Limousin

des précisions diverses :

oui
non
l'autre

ou autres requêtes :

pouvez vous répéter
au revoir
etc.

On peut alors imaginer apprendre un modèle pour chaque groupe possédant ses propres constructions syntaxiques. Ce type d'approche a été utilisé avec succès dans la reconnaissance optique de caractères ([Lucas et al., 1994])[1] où le nombre de groupes est connu (un alphabet latin, des chiffres arabes, etc.). Le choix du nombre de groupes dans un corpus de la langue naturelle n'est pas évident *a priori* (il peut être soumis à des contraintes linguistiques ou des contraintes techniques ...). La section suivante décrit, en parallèle avec la détection des séquences centrales, un critère qui permet de construire une hiérarchie puis de classer les séquences en un nombre de groupes souhaité.

[1] elle consiste à décomposer chaque symbole en un ensemble de vecteurs, dont les descripteurs forment des séquences, puis inférer des grammaires correspondant à chaque symbole. Lors de la reconnaissance, la séquence représentant un caractère est décodée par l'ensemble de grammaires : celle qui le reconnaît exactement fournit la classe du symbole (voir aussi le paragraphe 2.1.3).

7.2 Classement des séquences

Dans cette section, nous proposons une méthode qui vise à regrouper les séquences selon le critère de *distance d'édition* donné dans la Définition 29 (page 50). Tout d'abord nous expliquons la construction d'un tableau de distances entre les séquences. Nous montrons que ce tableau peut être utilisé comme donnée d'entrée d'un algorithme de classification ascendante hiérarchique. Nous utilisons finalement l'arborescence ainsi obtenue pour répartir les séquences en groupes par une coupure de l'arbre de classification.

7.2.1 Construction du tableau de distances

Nous considérons un ensemble de séquences : $C = \{x_1, \ldots, x_i, \ldots, x_T\}$ de taille $|C| = T$ et tel que $x_i \neq x_j$ $\forall i \neq j$. Pour chaque paire de séquences (x_i, x_j), nous calculons la distance $d(x_i, x_j)$, en utilisant l'algorithme présenté dans la section 3.3.1. Les valeurs obtenues sont stockées dans une matrice triangulaire M_T :

$$M_T = \begin{array}{llll} d(x_1, x_2) & & & \\ d(x_1, x_3) & d(x_2, x_3) & & \\ \vdots & \vdots & & \\ d(x_1, x_T) & d(x_2, x_T) & \ldots & d(x_{T-1}, x_T) \end{array}$$

M_T peut alors être utilisée avec des méthodes de Classification Ascendante Hiérarchique (CAH) classiques.

7.2.2 Classement et création des groupes

Le résultat d'un algorithme de CAH appliqué sur M_T peut être visualisé sous forme d'un *dendrogramme* (cf. Fig. 7.1) dont les feuilles sont constituées des séquences de C et dont la racine constitue une classe qui correspond à l'ensemble de ces séquences. Une coupure du dendrogramme à un niveau donné fournit des nœuds correspondants à des sous-ensembles de séquences dont les distances relatives sont optimales. Pour trouver les séquences centrales il faut alors effectuer un parcours en profondeur à partir de ces nœuds vers les feuilles de l'arbre qui sont regroupées par les nœuds du niveau le plus faible. Il est également possible de construire une hiérarchie de séquences en faisant un parcours en largeur de l'arbre, comme le montre l'exemple 1. Un nombre (maximal) de séquences par groupe peut être fixé pour limiter la taille du corpus tout en conservant les séquences syntaxiquement les plus représentatives.

Exemple 1 : extraction des séquences caractéristiques d'un corpus.

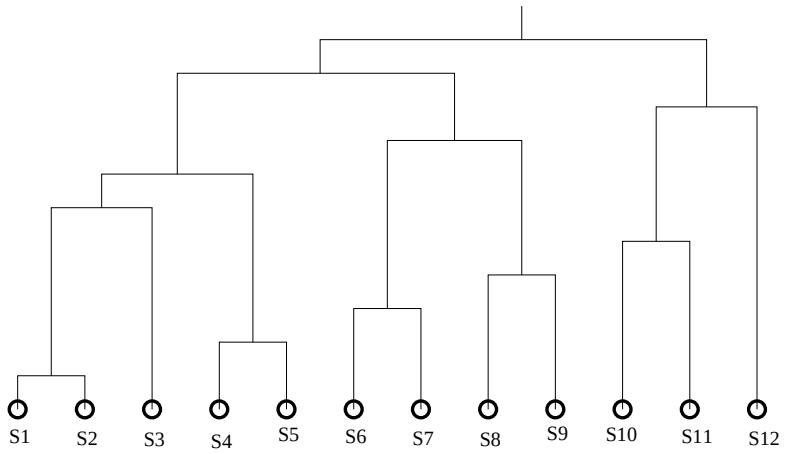
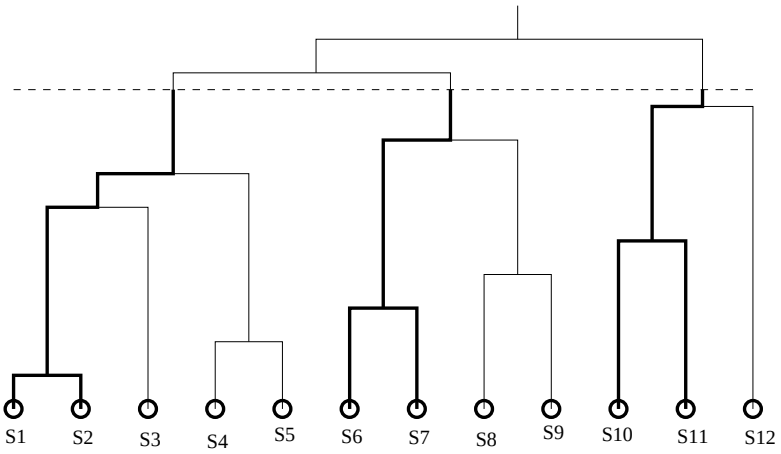
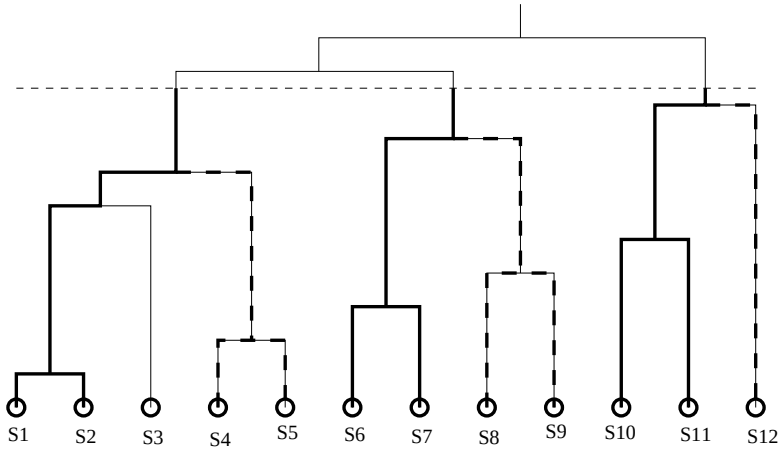


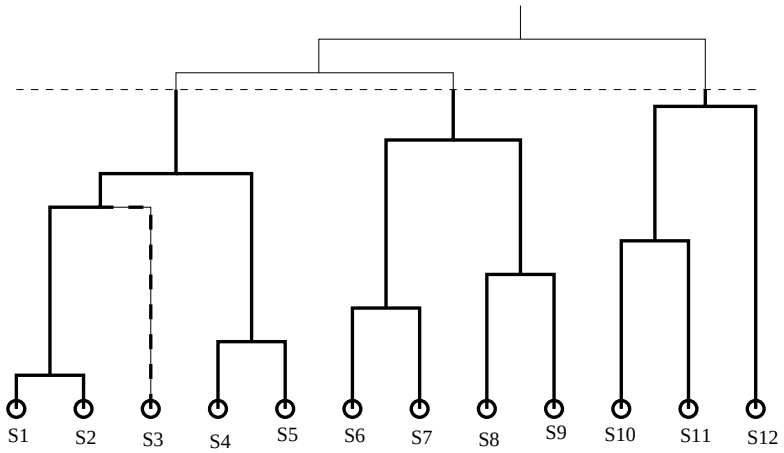
FIG. 7.1: Un dendrogramme obtenu à partir de la CAH appliquée sur une matrice triangulaire de distances calculées à partir d'un corpus $C = \{S_1, \ldots, S_{12}\}$.



La coupure statistique de l'arbre : 3 branches ont été choisies sur ce graphique. Les *séquences centrales* sont obtenues par le parcours en profondeur de chaque branche.



Enrichissement du corpus des *séquences centrales* : le nœud suivant dans chaque branche est celui dont la hauteur suit la hauteur du dernier nœud choisi (parcours en largeur du sous-arbre).



Toutes les séquences ont été choisies : le corpus initial C est divisé en trois sous-corpus $C_1 = \{S_1, S_2, S_3, S_4, S_5\}$, $C_2 = \{S_6, S_7, S_8, S_9\}$, $C_2 = \{S_{10}, S_{11}, S_{12}\}$. □

7.2.3 Apprentissage des "Grammaires Noyaux"

L'étape précédente a permis de diviser un corpus d'apprentissage en n sous-corpus selon un critère de ressemblance basé sur le calcul des distances entre séquences. En appliquant l'algorithme ECGI (ou tout autre algorithme d'inférence grammaticale) sur chacun de ces corpus nous obtenons n grammaires représentatives du corpus d'apprentissage d'origine : des "grammaires noyaux".

Remarque 13 *Cette démarche semble s'opposer à un des principes de l'apprentissage automatique :* plus de données assure un meilleur modèle. *Nous remarquons cepen-*

dant que notre approche permet de séparer l'information statistique de l'information structurelle :

1. *Il est tout à fait possible d'apprendre séparément la matrice des fréquences d'opérations correctionnelles (et donc appliquer la technique de lissage proposée dans le Chapitre 5) ;*
2. *Le corpus peut être probabilisé* après *l'apprentissage des règles par ECGI non-stochastique. La probabilisation peut faire intervenir l'analyse corrective : toute règle non-erreur (voir Définition 35 page 71) empruntée pour reconnaître une séquence du corpus d'apprentissage complet serait comptabilisée dans le calcul des fréquences (en accord avec la formule de probabilisation des règles présentée dans la section 4.3).*

□

7.2.4 Application sur les corpus réels

7.2.4.1 Conditions d'expérimentations

La puissance des outils statistiques dont nous disposions (StatLab, version antérieure à 1996) ne nous a pas permis de traiter l'ensemble des séquences. Nous avons limité nos expérimentations à 1 000 séquences d'apprentissage *distinctes* et 500 séquences de test également distinctes, retirées aléatoirement de chacun des deux corpus. Le tableau suivant montre la proportion des séquences distinctes par rapport au nombre de séquences au total dans les corpus AGS et ATIS :

Corpus	Nombre total de phrases	Nombre de phrases distinctes	Pourcentage de phrases distinctes
AGS	9 850	4 443	45%
ATIS	13 044	11 671	89%

Il apparaît ici que plus de la moitié de phrases du corpus AGS présentent des répétitions alors que la majorité des phrases du corpus ATIS sont uniques.

Six corpus d'apprentissage sont construits (à partir du corpus de 1 000 séquences uniques) : trois avec des séquences choisies et classés aléatoirement (respectivement 100, 500 et 1 000 séquences, pour référence nous les appellerons "aléatoires") et trois dont les séquences sont choisies par le parcours de l'arbre de classification (respectivement 100, 500 et 1 000 séquences caractéristiques, pour référence nous les appellerons "classés"). L'ordre de présentation des séquences correspond à l'ordre du parcours de l'arbre.

Les grammaires sont apprises avec l'algorithme ECGI non-stochastique. Elle sont évaluées par rapport au taux d'acceptation des phrases et par rapport à la distance cumulée au corpus de test (voir section 5.2, page 86).

7.2.4.2 Résultats

Les résultats présentés ici sont une moyenne réalisée sur 10 expériences suivant le même protocole.

Corpus AGS

	100 phrases	500 phrases	1000 phrases
Aléatoire	4	40	65
Classé	3	60	93
Classé puis brassé	3	58	64

Ce tableau représente le taux d'acceptation des phrases en fonction de la taille du corpus d'apprentissage. Il met en evidence l'avantage des corpus choisis selon le critère de distance entre les séquences. Si le corpus de 100 phrases semble être trop peu représentatif, celui avec 500 phrases permet de montrer que le choix des séquences caractéristiques permet d'augmenter d'environ 30% le taux de reconnaissance. Dans le corpus de 1 000 phrases, la différence constatée montre l'influence de l'ordre de présentation des exemples de l'algorithme ECGI. Ceci est mis en evidence par l'ajout de la troisième ligne au tableau : le modèle appris sur le corpus classé puis rebrassé présente sensiblement le même taux reconnaissance de phrases que le modèle appris sur le corpus aléatoire.

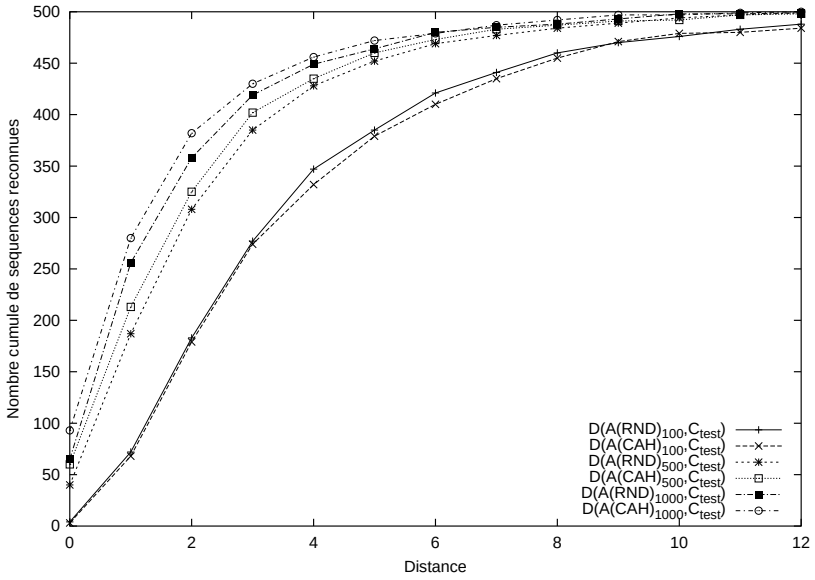


FIG. 7.2: Distance entre le corpus de test et les "grammaires noyaux" construites sur le corpus d'apprentissage issu du corpus AGS.

Les courbes de la FIG. 7.2 montrent que pour les corpus de 100 phrases il n'y a pas de différence notable entre les distances au corpus des deux modèles. Les modèles appris sur des corpus "classés" de 500 et de 1 000 phrases sont en revanche plus proches du corpus de test.

Corpus ATIS

Corpus	100 phrases	500 phrases	1 000 phrases
Aléatoire	6	33	45
Classé	14	40	51
Classé puis brassé	10	39	45

Ce tableau, similaire à celui décrivant les expérimentations avec le corpus AGS, confirme les résultats obtenus plus haut. Le taux de reconnaissance moyen, plus faible que celui constaté sur le corpus AGS est dû au nombre important des séquences uniques présentes dans le corpus ATIS (et donc à la plus faible représentativité de l'échantillon utilisé pour l'expérience). L'allure des courbes de la Fig. 7.3 confirme ce constat : la distance entre les modèles appris et le corpus de test est plus grande (la courbe monte moins vite). Ce graphique montre également que les modéles inférés sur les corpus classés sont plus proches du concept ciblé.

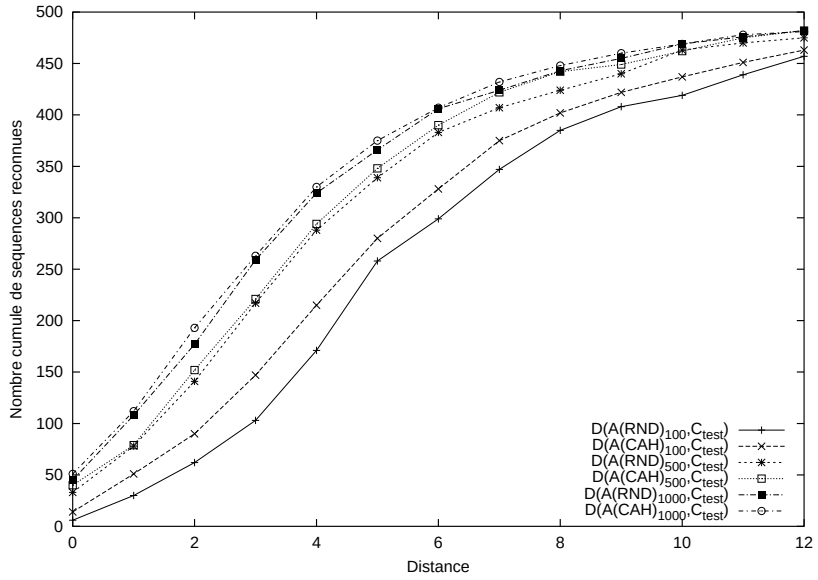


Fig. 7.3: Distance entre le corpus de test et les "grammaires noyaux" construites sur le corpus d'apprentissage issu du corpus ATIS.

Remarque 14 *Il est possible d'appliquer cette méthode récursivement pour traiter les corpus entiers : il suffit pour cela d'utiliser les corpus constitués des phrases caractéristiques obtenues à partir des ensembles des phrases caractéristiques des sous-corpus de taille plus petite.* □

7.2.4.3 Commentaires

Les résultats expérimentaux prouvent que les grammaires apprises sur des corpus construits à partir des séquences caractéristiques généralisent mieux : par rapport à

des modèles appris sur des phrases choisies aléatoirement, les “grammaires noyaux” reconnaissent 10% à 30% de phrases en plus. Ils ont permis également de montrer que l'ordre de présentation des séquences influe sur la qualité de la grammaire apprise par l'algorithme ECGI : disposant de la même quantité et qualité de données d'apprentissage, classer les phrases par la méthode proposée permet d'augmenter de 10% à 30% le taux de reconnaissance des phrases.

Reste à démontrer l'interêt d'apprentissage des “grammaires noyaux” en dehors d'une application précise qui consiste à diminuer la taille des modèles construits par l'algorithme ECGI. Cette technique peut être une aide à l'expert : par exemple, dans [Amengual, 1999] l'auteur propose d'enrichir automatiquement des modèles syntaxiques (grammaires régulières) à partir d'une grammaire construite manuellement par un expert. Les “grammaires noyaux” peuvent également être utilisées pour distribuer le calcul d'analyse syntaxique (avec ou sans diminution du corpus d'apprentissage).

7.3 Conclusion

Comme nous l'avons mentionné au début de ce chapitre, l'étude présentée ici n'est qu'une première revue des possibilités offertes par les techniques des correction d'erreurs en général et par l'algorithme ECGI en particulier, pour classer des séquences et pour modéliser un problème de façon approximative.

Cette approche utilise l'algorithme ECGI dans les versions présentées dans ce document. Une amélioration possible serait de suivre l'ordre de séquences donné par un parcours en profondeur du dendrogramme. L'algorithme d'apprentissage devrait alors être modifié, car le modèle courant ne serait plus uniquement amélioré en fonction d'une séquence, mais également en fonction d'un autre automate rencontré au cours de ce parcours et correspondant à un nœud regroupant plusieurs séquences.

Une autre approche consiste à appliquer directement le critère défini dans la Définition 38 (page 86). Un algorithme simple consiste à construire les n modèles $G_1, \ldots, G_n$ à partir de n séquences les plus représentatives, déterminées selon ce critère. Les itérations suivantes du processus d'apprentissage consistent alors à améliorer les modèles G_i initiaux selon la procédure ECGI, en respectant l'ordre de présentation des séquences donné par les distances respectives des séquences aux n modèles courants.

Conclusion

Conclusion

Le travail que nous avons décrit dans ce document vise à proposer l'utilisation des grammaires régulières comme une alternative aux modèles de type N-gram employés habituellement dans le cadre de la reconnaissance vocale. Il a été distribué selon quatre axes :

- Le premier propose l'utilisation d'un algorithme particulier (ECGIA) pour apprendre des modèles de langage à partir des corpus, par le biais de techniques de correction d'erreurs. Deux paramétrages originaux ont été proposés pour cet algorithme : l'un fondé sur les connaissances *a priori*, l'autre sur une approche entièrement automatique.
- Le deuxième consiste à élaborer une technique de *lissage* des grammaires apprises. Elle est indispensable à l'évaluation et à l'utilisation de ces modèles dans le cas d'insuffisance des données d'apprentissage (cas courant). Parallèlement, nous avons élaboré puis utilisé une méthode d'évaluation des grammaires non-stochastiques, fondée sur les techniques de correction d'erreurs ; elle permet de comparer leur pouvoir de généralisation.
- Le troisième transpose les techniques de *clustering* aux modèles étudiés ;
- Le quatrième est une approche originale dont l'objectif est d'extraire et de classer des *séquences caractéristiques* d'un corpus.

La présente étude n'a pas apporté des résultats surprenants : les approches proposées ne donnent pas de résultats meilleurs que ceux obtenus avec les modèles traditionnels. Ce résultat négatif est nuancé par le fait que lors de l'évaluation parallèle des deux types de modèles dans une tâche de reconnaissance vocale, les performances du système mesurées par le *taux d'erreur* étaient sensiblement proches[2]. Par ailleurs, nous arrivons à trois constats.

Premièrement, il apparaît que les grammaires régulières, dotées pourtant d'un pouvoir d'expression plus grand que les N-grams, s'avèrent moins performantes que ces derniers sur les corpus de parole spontanée. Cependant, parmi les trois corpus testés, allant du plus difficile (le corpus AGS, avec des phrases courtes et asyntaxiques) au plus facile (le corpus artificiel, avec des phrases comportant des régularités syntaxiques fréquentes et un lexique restreint) la différence de performances s'estompe. Nous en

[2] ces expériences ont été effectuées trop tard pour être présentées de façon detaillée dans ce document.

concluons que, d'une part, les grammaires régulières traitent avec succès les corpus représentatifs des langages régis par une syntaxe complexe (on le voit en particulier dans le cas des grammaires artificielles). D'autre part, les méthodes d'apprentissage des grammaires que nous avons étudiées sont sensibles à la "qualité syntaxique" des corpus — par ce terme, nous désignons le nombre de constructions grammaticales les plus représentatives du langage étudié, présentes dans les corpus. Ces deux observations mènent vers une discrimination de l'utilisation des types des modèles de langage : il n'est pas pertinent d'utiliser des grammaires capables de détecter des dépendances syntaxiques là où les modèles simples (de type N-gram) ont prouvé leur efficacité. Une utilisation conjointe peut être cependant envisagée : le modèle statistique peut déléguer le décodage du treillis accoustique à une grammaire (en l'occurrence une grammaire régulière stochastique) dans le cas où la syntaxe joue une rôle important (des phrases longues, avec des constructions caractéristiques, comme par exemple l'agencement des mots pour représenter les dates).

Deuxièmement, l'abstraction effectuée par l'algorithme d'inférence n'étant pas suffisante — le modèle appris n'accepte pas toutes les phrases envisageables — l'importance du lissage est cruciale. C'est un problème complexe, non seulement parce qu'il n'a été abordé que récemment, mais également par le fait qu'il concerne la structure du modèle au même titre que les valeurs numériques. La présente étude propose une solution qui hérite des techniques issues du lissage des modèles N-grams et qui, par extension, permet de traiter également le problème du mot inconnu. D'autre approches doivent être envisagées : les grammaires régulières sont pourvues d'un cadre théorique fort. L'exploration de leurs propriétés, comme par exemple une méthode efficace de parcours de la hiérarchie des modèles compatibles avec le corpus d'apprentissage, peut rendre ce type de modèles compétitif face aux approches traditionnelles. Par ailleurs, nous avons montré que la modélisation du comportement du mot inconnu contribue à l'amélioration du modèle appris. L'approche proposée peut être améliorée, notamment par l'attribution d'une classe (connue) à un mot hors vocabulaire (étude abordée notamment dans [Damnati, 2000]), en cohérence avec les techniques de correction d'erreurs présentées dans ce document.

Troisièmement, l'étude des "grammaires noyaux", bien que menée en marge de la problèmatique de la reconnaissance vocale, montre que les techniques de correction d'erreurs peuvent s'appliquer à la classification des séquences. Notre approche permet alors de découvrir des sous-langages disjoints dans un corpus de phrases. Un problème transversal, non-trivial, serait la modélisation des sous-langages, eux-mêmes régis par des règles de composition (le problème d'*enchâssement*). Les techniques correctives, par leur capacité du calcul de distances entre les modèles syntaxiques sont particulièrement adaptées pour résoudre ce problème.

Ajoutons enfin à ces conclusions celle du bien fondé de la convergence des techniques issues des domaines strictement formalisés (comme les statistiques et les grammaires formelles) et de ceux concernant les études du langage naturel. Les connaissances *a*

priori des seconds contribuent au renforcement des approches générales et automatisables rendus possibles par les premiers, notamment pour traiter les cas particuliers ou, comme il a été montré dans notre étude, pour initialiser un calcul de modélisation.

Annexes

Annexe A

Présentation des corpus d'apprentissage

A.1 Langage L_0

Le langage L_0 est un langage artificiel défini dans [Feldman et al., 1990], conçu à des fins expérimentales et proche (d'une partie) du langage naturel. Il décrit (en anglais) trois formes géométriques (*triangle, cercle et carré*) et leur positionnement relatif. Voici quelques exemples de phrases qui en font partie :

- a square and a circle touch a dark square and a light triangle
- a triangle is below a light triangle
- a square is far to the right of a circle
- a circle and a medium circle are below a large light circle
- a small square and a medium dark triangle touch a dark triangle
- the square which is to the right of the small circle is removed
- the medium triangle is removed
- the light triangle which is below the dark square is removed
- a light triangle and a circle touch a dark circle
- the triangle is removed

Le langage L_0 peut être défini par une grammaire régulière. Celle qui a servi à nos exepérimentations (que nous nommons G_{L_0}) est représentée sous forme d'un AFD sur la Fig. A.1. Elle possède 25 éléménts terminaux, 58 non-terminaux et 189 transitions. Le tableau suivant donne les caractéristiques des corpus d'apprentissage et de test générés par cette grammaire et utilisés pour les expériences (présentées dans les annexes qui suivent) :

Corpus	Nombre de phrases	Taille du lexique	Nombre moyen de mots par phrase
Apprentissage	10 000	25	8.73
Test	1 000	25	8.76

A.2 Corpus AGS

Le corpus AGS contient des phrases prononcées dans le cadre d'un dialogue entre un locuteur humain et un système automatique de France Télécom. Seules les requêtes des locuteurs sont transcrites. Les caractéristiques de ce corpus sont regroupées dans le tableau ci-dessous :

Corpus	Nombre de phrases	Taille du lexique	Nombre moyen de mots par phrase
Apprentissage	9 850	821	5.03
Test	724	373	4.95

Il s'agit de phrases courtes, souvent agrammaticales. Nous en donnons ici quelques exemples :

- je voudrais des offres d' emploi pour le métier de l'hôtellerie
- quelle région me proposez vous
- la métropole
- je voudrais le numéro de téléphone du serveur Audiotel
- oui pour la région Est
- un serveur d' emploi
- l' hôtellerie
- la France
- non
- je voudrais derniers numéros

Remarque 15 *Le corpus d'apprentissage et le corpus de test n'ont pas été conçus dans les mêmes conditions : ils ont été enregistrés au cours de sessions différentes, avec des locuteurs différents et des comportements différents. Ceci nous a incité à brasser les deux corpus, puis à les rediviser dans les mêmes proportions et refaire certains tests sur les deux. Les résultats obtenus (cf. section 5.5.1) indiquent la faible représentativité des données d'apprentissage par le corpus de test original.*

A.3 Corpus ATIS

Le corpus ATIS est un des corpus de référence dans la communauté internationale de TALN/TALP. Il contient des requêtes concernant les services de réservations des billets des vols :

- WHAT'S THE NUMBER OF THAT U S AIR FLIGHT
- DELTA FLIGHT SIX TWENTY EIGHT PLEASE
- BOOK THAT FLIGHT

- I WOULD LIKE TO LEAVE TOMORROW NOVEMBER THE SEVENTH ON A FLIGHT FROM SAN FRANCISCO TO PHILADELPHIA COULD YOU TELL ME WHAT FLIGHTS LEAVE EARLY IN THE MORNING
- PLEASE GIVE ME FLIGHTS FROM ATLANTA TO BOSTON ON WEDNESDAY MORNING AND THURSDAY AFTERNOON
- WHAT IS GROUND TRANSPORTATION BETWEEN THE SAN FRANCISCO AIRPORT AND THE CITY
- WASHINGTON TO SAN FRANCISCO WITH A STOPOVER IN FORT WORTH PLEASE THAT LEAVES IN THE AFTERNOON AND ARRIVES EARLY EVENING
- WHAT TYPE OF AIRPLANE IS USED FROM BOSTON TO ATLANTA AROUND NINE O'CLOCK IN THE MORNING
- SHOW ME ALL FLIGHTS FROM PITTSBURGH TO PHILADELPHIA WHICH ARRIVE BEFORE EIGHT O'CLOCK A M ON WEDNESDAY
- SHOW ME FLIGHTS FROM BALTIMORE TO DALLAS

Ses caractéristiques sont regroupées ci-dessous :

Corpus	Nombre de phrases	Taille du lexique	Nombre moyen de mots par phrase
Apprentissage	15 019	1 259	10.2
Test	1 001	552	11.7

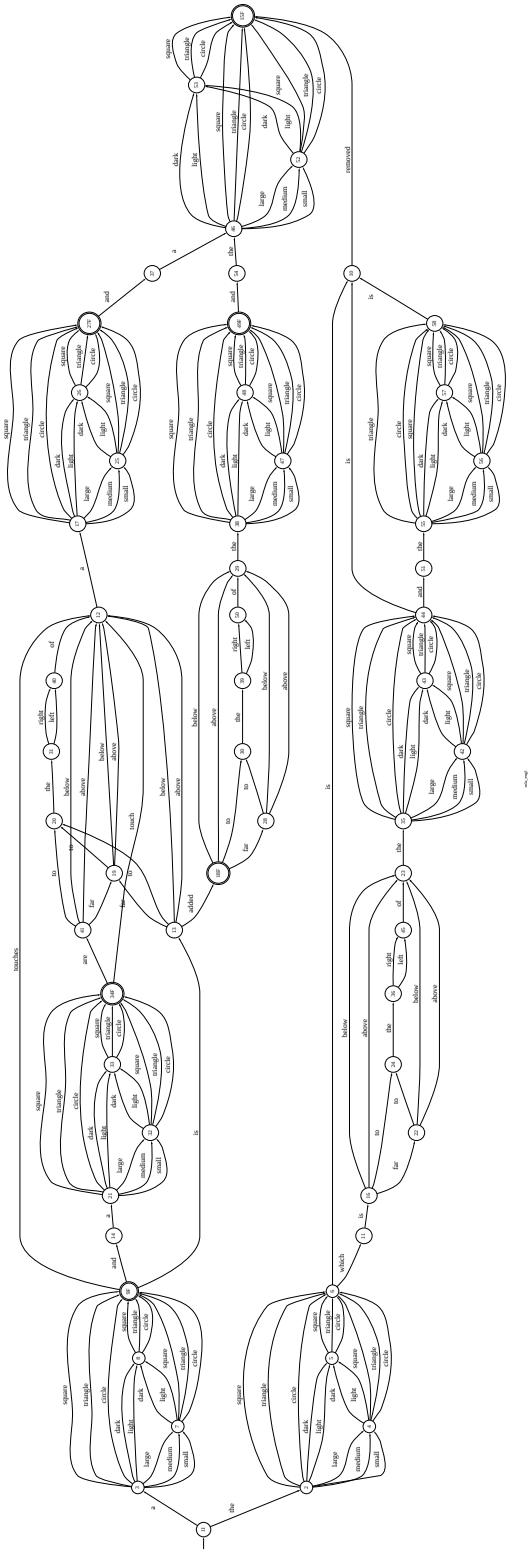


FIG. A.1: Grammaire du langage L_0.

Annexe B

Classes grammaticales de la langue française du corpus AGS

Les mots figurant dans le dictionnaire ci-dessous sont issus du corpus AGS (corpus d'apprentissage et de test).

Le lexique complet du corpus (apprentissage et test) possède 866 éléments. Les mots de ce lexique ont été marqués par un jeu de 25 étiquettes, codées sur 4 caractères designant la catégorie du mot. Elles peuvent être augmentées par deux traits supplémentaires : le genre (M ou F) et le nombre (P, S, 1, 2, 3, 4, 5, 6). Le tableau qui suit regroupe l'ensemble des étiquettes :

Catégorie	genre	nombre	exemple
ADJE	✓	✓	beau = ADJEMS
ADJI	✓	✓	chacun = ADJIMS
SUBS	✓	✓	chapeau = SUBSMS
ADVE	✗	✗	peu = ADVE
AUXE	✗	✓	suis = AUXE1
AUXA	✗	✓	as = AUXA2
CCOO	✗	✗	donc = CCOO
CHIF	✗	✗	deux = CHIF
CSUB	✗	✗	que = CSUB
DETR	✓	✓	le = DETRMS
PPOB	✓	✓	le = PPOBMS
PDEA	✗	✗	de = PDEA
PDET	✓	✓	celles = PDETFP
PHAT	✗	✗	hein = PHAT
PIND	✓	✓	aucune = PINDFS
PPAS	✓	✓	volée = PPASFS
PPRE	✗	✗	volant = PPRE
PPER	✗	✓	je = PPER1
PREL	✓	✓	qui = PRELMS
PREP	✗	✗	sans = PREP
VINF	✗	✗	manger = VINF
VERB	✗	✓	mangez = VERB5
NPRO	✗	✗	Paris = NPRO
NE_	✗	✗	ne = NE_
PAS_	✗	✗	pas = PAS_

Ci-dessous figure un échantillon extrait de ce lexique :

0	NBRE		00	NBRE	
ANPE	NPRO		ASSEDIC	NPRO	
BTS	NPRO		Bas-Rhin	NPRO	
Finistère	NPRO		Fusey	NPRO	
Vosges	NPRO		Yonne	NPRO	
a	AUXA3		à	PDEA	
ben	PHAT		bien	ADVE	
c'	PDETMS		cadres	SUBSMP	
de	PDEA		découvrir	VINF	
être	AUXE		ébéniste	SUBSMS	
fais	VERB1	VERB2	fait	SUBSMS	VERB3
garçons	SUBSMP		général	ADJEMS	
habite	VERB3	VERB1	hiver	SUBSMS	
hm-hm	PHAT		hôtellerie	SUBSFS	
ils	PPER6M		importe	VERB3	VERB1
j'	PPER1M	PPER1F	jamais	ADVE	
l-on	PPOBMS		la	DETRFS	PPOBFS
m'	PPOBMS		ma	DETRFS	
national	ADJEMS		nationale	ADJEFS	
occupe	VERB1	VERB3	océan	SUBSMS	
par	PREP		parce_que	CSUB	
quantité	SUBSFS		quatrième	ADJEMS	
rappeler	VINF		rappeliez	VERB5	
s'	PREFMS	PREFFS	s'il-vous-plaît	PHAT	
technicien	SUBSMS		techniques	ADJEFP	ADJEMP
un	DETRMS		une	DETRFS	
vacances	SUBSFP		vais	VERB1	
y	ADVE		zéro	ADJEMS	SUBSMS
				ADVE	

Bibliography

[AAAI, 1998] AAAI (1998). *Papers from the 1998 AAAI Symposium on Applying Machine Learning to Discourse Processing.* AAAI Press, Technical Report SS-98-01.

[Abeillé, 1988] Abeillé, A. (1988). Parsing french with a tree adjoining grammar. In *Proceedings of* 12th *COLING*, volume 1 (pp. 7–12).

[Abeillé, 1993] Abeillé, A. (1993). *Les Nouvelles Syntaxes.* Armand Colin.

[Aho & Ullman, 1972] Aho, A. & Ullman, J. D. (1972). *The theory of Parsing, Translation and Compiling, Vol 1 : Parsing.* Prentice-Hall.

[Ahonen & Mannila, 1994] Ahonen, H. & Mannila, H. (1994). Forming grammars for structured documents : an application of grammatical inference. In *Proceedings of ICGI'94*, volume 862 (pp. 153–167). Lecture Notes in Artificial Intelligence, Subseries of Lectures Notes in Computer Science.

[Alquézar et al., 1996] Alquézar, R., Sanfeliu, A., & Cueva, J. (1996). Learning of context-sensitive language acceptors through regular inference and constraint induction. In *Proceedings of ICGI'96*, volume 1147 (pp. 134–145). Lecture Notes in Artificial Intelligence, Subseries of Lectures Notes in Computer Science.

[Amengual, 1999] Amengual, J.-C. (1999). *Técnicas de Corrección de Errores y su Aplicación en Reconocimiento de Formas, Tratamiento del Lenguaje Natural y Traducción Automática.* PhD thesis, Universidad Politécnica de Valencia.

[Amengual & Vidal, 1998] Amengual, J.-C. & Vidal, E. (1998). Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10), 1109–1116.

[Angluin, 1978] Angluin, D. (1978). On the complexity of minimum inference of regular sets. *Information and Control*, 29(3), 741–765.

[Angluin, 1987] Angluin, D. (1987). Queries and concept learning. *Machine Learning*, 2, 319–342.

[Bahl et al., 1983] Bahl, L. R., Jelinek, F., & Mercer, R. L. (1983). A maximum likelihood approach to continious speech recognition. *IEEE trans. on Pattern Analysis and Machine Intelligence*, 5, 179–190.

[Barzdin, 1974] Barzdin, J. M. (1974). Two theorems on the limiting synthesis of functions. *Zapiski*, 210. Latv. Gos. Univ.

[Baum, 1972] Baum, L. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities*, 3, 1–8.

[Bimbot et al., 1994] Bimbot, F., Pieraccini, R., Levin, E., & Atal, B. (1994). Modèles de séquences à horizon variable : Multigrammes. In *Proceedings of the* 20th *JEP*.

[Bimbot et al., 1995] Bimbot, F., Pieraccini, R., Levin, E., & Atal, B. (1995). Variable-length sequence modeling : Multigrams. *IEEE Signal Processing Letters*, 2(6).

[Black et al., 1992] Black, E., Jelinek, F., Lafferty, J., Mercer, R., & Roukos, S. (1992). Decision tree models applied to the labeling of text with part-of-speech. In *Darpa Workshop on Speech and Natural Language* (pp. 117–121). New York, USA.

[Blin & Miclet, 2000] Blin, L. & Miclet, L. (2000). Generating synthetic speech prosody with lazy learning in tree structures. In *Fourth Conference on Computational Natural Language Learning and Second Learning Language in Logic Workshop (CoNLL-2000 and LLL-2000)* (pp. 87–90). Lisbon, Portugal.

[Bouloutas et al., 1991] Bouloutas, A., Hart, G., & Schwartz, M. (1991). Two extensions of the viterbi algorithm. In *IEEE Transactions on Information Theory* (pp. 430–436).

[Brazma et al., 1998] Brazma, A., Jonassen, I., Vilo, J., & Ukkonen, E. (1998). Pattern discovery in biosequences. In *Proceedings of* 4th *International Colloquium on Grammatical Inference* (pp. 257–270). Springer-Verlag.

[Brill, 1992] Brill, E. (1992). A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing* (pp. 152–155). Trento, Italy.

[Brown et al., 1992] Brown, P., Della-Pietra, V., de Souza, P., Lai, J., & Mercer, R. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, 18(4), 467–479.

[Cardie, 1995] Cardie, C. (1995). *Domain-Specific Knowledge Acquisition for Conceptual Sentence Analysis*. PhD thesis, University of Massachusetts.

[Cardie, 1996] Cardie, C. (1996). Embedded machine learning systems for natural language processing : A general framework. In Springer-Verlag (Ed.), *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing* (pp. 315–328). Berlin, Allemagne.

[Carrasco & Oncina, 1994] Carrasco, R. & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *Proceedings of ICGI'94*, volume Lecture Notes in Artificial Intelligence 862 (pp. 139–152). Subseries of Lectures Notes in Computer Science.

[Carrasco, 1997] Carrasco, R. C. (1997). Accurate computation of the relative entropy beetween stochastics regulars grammars. http ://www.dlsi.ua.es/∼carrasco/rcc.html.

[Castellanos et al., 1994] Castellanos, A., Galliano, I., & Vidal, E. (1994). Application of ostia to machine translation tasks. In *Proceedings of ICGI'94* (pp. 93–105).

[Chanod & Tapanainen, 1995] Chanod, J. & Tapanainen, P. (1995). Tagging french - comparing a statistical and a constraint based method. In *Proceedings of the* 7th *Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)* (pp. 149–156).

[Chodorowski & Miclet, 1998] Chodorowski, J. & Miclet, L. (1998). Applying grammar inference in learning a language model for oral dialogue. In *Proceedings of International Colloquium on Grammatical Inference* (pp. 102–113). Springer-Verlag.

[Chodorowski & Miclet, 1999] Chodorowski, J. & Miclet, L. (1999). Apprentissage et evaluation de modèles de langage par des techniques de correction d'erreurs. In *Actes de* 6e *conférence annuelle sur le Traitement Automatique des Langues Naturelles* (pp. 253–262).

[Chomsky, 1957] Chomsky, N. (1957). *Syntactic Structures.* La Haye, Mouton.

[Chomsky & Miller, 1957] Chomsky, N. & Miller, G. A. (1957). *Pattern conception.* Technical report, AFCRC-TN-5757. (ASTIA Document AD 110076).

[Chou & Fu, 1976] Chou, S. & Fu, K. (1976). Inference for transition network grammars. In *Proceedings the* 3rd *International Conference on Pattern Recognition* (pp. 79–84).

[Chow & Schwartz, 1989] Chow, Y. & Schwartz, R. (1989). The n-best algorithm : an efficient procedure for finding top n-sequences hypotheses. In *Proceedings of ARPA Workshop on Speech and Natural Language* (pp. 199–202).

[Coste & Nicolas, 1998] Coste, F. & Nicolas, J. (1998). How considering incompatible state merging may reduce the DFA induction search space. In *Proceedings of ICGI'98* (pp. 199–210).

[Cros, 1873] Cros, C. (1873). À un page bleu de la reine Isabeau. "Le coffret de santal".

[Cruz & Vidal, 1998] Cruz, P. & Vidal, E. (1998). Learning regular grammars to model musical style. In *Proceedings of ICGI'98* (pp. 211–222).

[Daelemans et al., 1996] Daelemans, W., Berck, W., & Gillis, S. (1996). Unsupervised discovery of phonological categories through supervised learning of morphological rules. In *Proceedings of the* 16th *International Conference on Computational Linguistics* (pp. 95–100).

[Damnati, 2000] Damnati, G. (2000). *Modèles de Langage et Classification Automatique pour la Reconnaissance de la Parole Continue dans un Contexte de Dialogue Oral Homme-machine.* PhD thesis, Université d'Avignon et des Pays du Vaucluse.

[Deligne, 1996] Deligne, S. (1996). *Modèles de séquences de longueurs variables : application au traitement du langage écrit et parlé.* PhD thesis, ENST.

[Dempster et al., 1977] Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likehood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(ser. B), 1–38.

[Deo, 1974] Deo, N. (1974). *Graph Theory with Application to Engineering and Computer Science.* Prentice-Hall.

[Dijkstra, 1959] Dijkstra, E. (1959). A note on two problems in connection with graph theory. *Numerische Mathematik*, 1, 269–271.

[Dupont, 1994] Dupont, P. (1994). Regular grammatical inference from positive and negative samples by genetic search : the GIG method. In *Proceedings of ICGI'94, Grammatical Inference and Applications* (pp. 236–245). Springer Verlag.

[Dupont, 1996] Dupont, P. (1996). *Utilisation et apprentissage de modèles de langages pour la reconnaissance de la parole continue.* PhD thesis, ENST.

[Dupont & Amengual, 2000] Dupont, P. & Amengual, J.-C. (2000). Smoothing probabilistic automata : An error-correcting approach. In *Proceedings of ICGI'00* (pp. 51–64). Springer Verlag.

[Dupont & Chase, 1998] Dupont, P. & Chase, L. (1998). Using symbol clustering to improve probabilistic automaton inference. In *Proceedings of ICGI'98* (pp. 233–243). Springer Verlag.

[Dupont & Miclet, 1998] Dupont, P. & Miclet, L. (1998). *Inférence grammaticale régulière : fondements théoriques et principaux algorithmes.* Technical Report 3449, IRISA. Rapport de recherche, thème 3.

[Dupont et al., 1994] Dupont, P., Miclet, L., & Vidal, E. (1994). What is the search space of the regular inference? In *Proceedings of ICGI'94, Grammatical Inference and Applications* (pp. 25–37). Springer Verlag.

[El-Bèze et al., 1997] El-Bèze, M., Jardino, M., & Bimbot, F. (1997). Une approche alternative pour le calcul de perplexité. In 1ères *JST Francil* (pp. 79–84).

[El-Bèze & Mérialdo, 1999] El-Bèze, M. & Mérialdo (1999). HMM based taggers. In *Syntactic Wordclass Tagging* (pp. 263–284). ed. Hans von Halteren, Kluwer Academic Publishers.

[Feldman et al., 1990] Feldman, J. A., Lakoff, G., Stolcke, A., & Weber, S. H. (1990). *Miniature Language Acquisition : A touchstone for cognitive science.* Technical report, ICSI. TR-90-009.

[Fred, 1996] Fred, A. L. N. (1996). Clustering of sequences using a minimum grammar complexity criterion. In *Proceedings of ICGI'96* (pp. 107–116). Springer Verlag.

[Fu, 1970] Fu, K. S. (1970). *Formal Langages and their Relation to Automata.* Addison Wesley.

[Gazdar et al., 1985] Gazdar, G., Klein, E., Pullum, G., & Sag, I. (1985). *Generalised Phrase Structure Grammar.* Harvard University Press.

[Giordano, 1994] Giordano, J.-Y. (1994). Grammatical inference on context-free grammars by enumeration : Structural containment as an ordering bias. In *Proceedings of ICGI'94* (pp. 212–221). Alicante, Spain.

[Giordano, 1996] Giordano, J.-Y. (1996). Grammatical inference using tabu search. In *Proceedings of ICGI'96* (pp. 292–312). Montpellier, France.

[Gold, 1967] Gold, E. M. (1967). Langage indentification in the limit. *Information and control*, 10(5), 447–474.

[Gold, 1978] Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and Control*, 37, 302–320.

[Good, 1953] Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3), 237–264.

[Hetherington et al., 1996] Hetherington, I. L., Philips, M. S., Glass, J. R., & Zue, V. (1996). A word network search for continuous speech recognition. In *Proceedings of Eurospeech'93* (pp. 21–24).

[Higuera (de la), 1996a] Higuera (de la), C. (1996a). Algorithmique de l'inférence grammaticale. Université de Montpellier 2. Habilitation à diriger les recherches.

[Higuera (de la), 1996b] Higuera (de la), C. (1996b). Characteristic sets for polynomial grammatical inference. In *Proceedings of ICGI'96* (pp. 59–71).

[Higuera (de la), 1997] Higuera (de la), C. (1997). Characteristic sets for polynomial grammatical inference. *Machine Learning*, 2(27), 1–14.

[Higuera (de la), 2000] Higuera (de la), C. (2000). World wide web home page for Colin de la Higuera. http ://www.univ-st-etienne.fr/eurise/cdlh.html.

[Higuera (de la) et al., 1996] Higuera (de la), C., Oncina, J., & Vidal, E. (1996). Identification of DFA : data-dependent versus data-independant algorithms. In *Proceedings of ICGI'96* (pp. 311–325). Montpellier, France.

[Hopcroft & Ullman, 1969] Hopcroft, J. E. & Ullman, J. D. (1969). *Formal Languages and their Relation to Automata.* Addison-Wesley, Reading.

[Hopcroft & Ullman, 1979] Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading.

[ICGI, 1994] ICGI (1994). *Grammatical Inference and Applications.* Springer-Verlag.

[ICGI, 1996] ICGI (1996). *Grammatical Inference : Learning Syntax from Sentences.* Springer-Verlag.

[ICGI, 1998] ICGI (1998). *Grammatical Inference.* Lecture Notes in Artificial Intelligence 1433, Springer-Verlag.

[ICGI, 2000] ICGI (2000). *Grammatical Inference.* Springer-Verlag.

[ICML, 1997] ICML (1997). *Empirical Learning of Natural Language Processing Tasks.*

[Itai, 1994] Itai, A. (1994). Learning morphology — practice makes good. In *Proceedings of ICGI'94* (pp. 5–15).

[Jelinek, 1985] Jelinek, F. (1985). Markov source modeling of text generation. In *Impact of Processing Techniques on Communications* : Ed. Nijhoff. J. K. Skwirzinski.

[Jelinek, 1990] Jelinek, F. (1990). Self-organized language modeling for speech recognition. In *Readings In Speech Recognition* (pp. 450–506). San Mateo, California : Morgan Kaufmann.

[Jelinek, 1998] Jelinek, F. (1998). *Statistical Methods for Speech Recognition.* MIT Press.

[Jones, 1996] Jones, D. (1996). *Analogical Natural Language Processing.* UCL Press.

[Jurafsky & Martin, 2000] Jurafsky, D. & Martin, J. H. (2000). *Speech and Language Processing.* Prentice Hall.

[Kaplan et al., 1996] Kaplan, R., Bod, R., Sima'an, K., & Scha, R. (1996). A data oriented approach to lexical-functionnal grammars. *Computational Linguisitcs in the Nandherlands.*

[Kaplan & Bresnan, 1982] Kaplan, R. & Bresnan, J. (1982). Lexical-functionnal grammar : a formal system for grammatical representation. In J. Bresnan (Ed.), *The Mental Representation of Grammatical Relations* Cambidge, Massachusetts : MIT Press.

[Katz, 1997] Katz, S. M. (1997). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3), 400.

[Kearns & Valiant, 1989] Kearns, M. & Valiant, G. (1989). Cryptocraphic limitations on learning boolean formulae and finite automata. In *Proceedings of the 21st annual ACM symposium on Theory of computating* (pp. 433–444). New York, USA : ACM ACM.

[Kenny et al., 1994] Kenny, P., Labute, P., Li, Z., & O'Shaughnessy, D. (1994). New graph search techniques for speech recognition. In *Proceedings of ICASSP'94* (pp. 553–556). Adelaide, Australie.

[Kohavi, 1970] Kohavi, Z. (1970). *Finite State Automata.* Addison-Wesley.

[Kruskal & Sankoff, 1983] Kruskal, J. B. & Sankoff, D. (1983). *Time Warps, String Edits, and Macromolecules : the Theory and Practice of Sequence Comparaison.* Addison-Wesley.

[Kuhn & Mori, 1995] Kuhn, R. & Mori, R. D. (1995). The application of semantic classification trees to natural language understanding. *IEEE Transactions on PAMI*, 17(5), 449–460.

[Lafferty et al., 1992] Lafferty, J., Sleator, D., & Temperley, D. (1992). *Grammatical trigrams : a probabilistic model of link grammar.* Technical report, Department of Computer Science, Carnegi Mellon University.

[Lang & Pearlmutter, 1997] Lang, K. & Pearlmutter, B. (1997). Abbadingo one competition. http ://abbadingo.cs.unm.edu/.

[Lang et al., 1998] Lang, K., Pearlmutter, B., & Coste, F. (1998). Gowachin competition. http ://www.irisa.fr/Gowachin/.

[Lang, 1992] Lang, K. J. (1992). Random DFA's can be approximately learned from sparse uniform examples. *Proceedings of the* 5th *ACM Workshop on Computation Learning Theory*, (pp. 45–52).

[Loupy et al., 2000] Loupy, C. D., El-Bèze, M., & Marteau, P.-F. (2000). Using semantic classification trees for WSD. *Computer and Humanities*, 34, 187–192.

[Lowrance & Wagner, 1975] Lowrance, R. & Wagner, R. A. (1975). An extension of string-to-string correction problem. *Journal of the ACM*, 22(2), 177–183.

[Lucas et al., 1994] Lucas, S., Vidal, E., Amiri, A., Hanlon, S., & Amengual, J. C. (1994). A comparison of syntactic and statistical techniques for off-line OCR. *ICGI'94*, Lecture Notes in Artificial Intelligence 862, 168–180.

[Magerman, 1995] Magerman, D. M. (1995). Statistical decision trees models for parsing. *Proceedings of the Association for Computational Linguistics.*

[Miclet, 1979] Miclet, L. (1979). *Inférence de Grammaires Régulières.* PhD thesis, Ecole Nationale Supérieure des Télécommunications.

[Miclet & Gentile (de), 1994] Miclet, L. & Gentile (de), C. (1994). Inférence grammaticale à partir d'exemples et de contre-exemples : deux algorithmes optimaux : (BIG et RIG) et une version heuristique (BRIG). *Journées Acquisition, Validation, Apprentissage*, (pp. F1–F13). Strasbourg, France.

[Muggleton & Raedt, 1994] Muggleton, S. & Raedt, L. D. (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20, 629–679.

[Natarajan, 1991] Natarajan, B. K. (1991). *Machine Learning : A Theoritical Approach.* San Mateo, CA : Morgan-Kaufmann.

[Ney & Essen, 1993] Ney, H. & Essen, U. (1993). Estimating "small" probabilities by leaving-one-out. In *European Conference on Speech Communication and Technology (Eurospeech'93)* (pp. 2239–2242). Berlin, Germany.

[Ney & Knesser, 1993] Ney, H. & Knesser, R. (1993). Improved clustering techniques for class-bases statistical language modelling. In *European Conference on Speech Communication and Technology (Eurospeech'93)* (pp. 973–976). Berlin, Germany.

[Nigam et al., 1998] Nigam, K., McCallum, A., Thrun, S., & Mitchell, T. (1998). Learning to classify text from labeled and unlabeled documents. In *Proceedings of the* 15th *National Conference on Artificial Intelligence (AAAI-98)* (pp. 792–799). Madison, Wisconsin, USA.

[Oncina & Garcia, 1992] Oncina, J. & Garcia, P. (1992). Inferring regular languages in polynomial update time. In N. P. de la Blanca, A. Sanfeliu, & E. Vidal (Eds.), *Pattern Recognition and Image Analysis*, volume 1 (pp. 49–61). World Scientific, Singapore.

[Pollard & Sag, 1987] Pollard, C. & Sag, I. (1987). *An Information-based Syntax and Semantics, Volume 1 : Fundamentals.* Standford University and Chicago University Press. CSLI Lectures Notes.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.

[Reilley & Sharkey, 1992] Reilley, R. G. & Sharkey, N. E. (1992). *Connectionist Approaches to Natural Language Processing.* Hillsdale.

[Ron et al., 1994] Ron, D., Singer, Y., & Tishby, N. (1994). Learning probabilistic automata with variable memory length. In A. Press (Ed.), *Proceedings of the Sevent Annual ACM Conference on Computational Learning Theory* (pp. 35–46). Brunswick, New Jersey, USA : Morgan Kaufmann Publishers.

[Rulot et al., 1989] Rulot, H., Prieto, N., & Vidal, E. (1989). Learning accurate finite-state structural models of words : the ECGI algorithm. In *ICASSP'89*, volume 1 (pp. 643–646).

[Rulot & Vidal, 1988] Rulot, H. & Vidal, E. (1988). An efficient algorithm for the inference of circuit-free automata. In G. Ferraté & others (Eds.), *Syntactic and Structural Pattern Recognition (Proc. of the NATO Advanced Research Workshop, held 1986 in Barcelona)*, volume 45 (pp. 173–184). Springer.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rummelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing : Exploration in the Microstructure of Cognition*, volume 1 (pp. 318–362).

[Sadek et al., 1996] Sadek, D., Ferrieux, A., Cozannet, A., Bretier, P., Panaget, F., & Simonin, J. (1996). Effective human-computer cooperative spoken dialogue : the AGS demonstrator. In *Proceedings of ICSLP* (pp. 542–545).

[Samuelsson & Reichl, 1999] Samuelsson, C. & Reichl, W. (1999). A class-based language model for large vocabulary speech recognition extracted from part-of-speech statistics. In *IEEE ICASSP'99* (pp. 537–540). : IEEE.

[Shabes, 1990] Shabes, Y. (1990). *Computational and Mathematical Proprieties of Lexicalised Grammars.* PhD thesis, University of Pennsylvanie, Philadelphia.

[Shabes, 1992] Shabes, Y. (1992). Stochastic lexicalized tree-adjoining grammars. In *Proceeding of the* 14th *International Conference on Computational Linguistics.*

[Shannon, 1951] Shannon, C. E. (1951). Prediction of entropy of printed english. *Bell System Technical Journal*, 30, 50–66.

[Smaïli et al., 1992] Smaïli, K., Zitouni, I., Charpillet, F., & Haton, J. P. (1992). An hybrid language model for a continuous dictation prototype. In *Proceeding of Eurospeech'97* (pp. 2723–2726).

[Spriet & El-Bèze, 1997] Spriet, T. & El-Bèze, M. (1997). Introduction of rules into a stochastic approach for language modeling. In *NATO ASI summer school* (pp. 350–355). : NATO book.

[Takada, 1996] Takada, K. (1996). An efficient recognition algorithm for multiple context-free languages. Master's Thesis, Osaka University.

[TALN, 1999] TALN (1999). *Workshop sur les Méthodes hybrides TALN/TALP pour le Traitement Robuste de la Langue, TALN'99.* J.-Y. Antoine and D. Genthial.

[Thollard, 2000] Thollard, F. (2000). *Inférence grammaticale pour l'apprentissage de la syntaxe en reconnaissance de la parole et dialogue oral.* PhD thesis, Université de Saint-Etienne.

[Thollard & Higuera (de la), 1998] Thollard, F. & Higuera (de la), C. (1998). The importance of smoothing in learning deterministic stochastic finite automata. ECML'98 — Poster session, Chemnitz, Germany.

[Thomason, 1974] Thomason, M. G. (1974). Errors in regular languages. *IEEE Transactions on Computers*, C-23(6).

[Tomita, 1982] Tomita, M. (1982). Dynamic construction of finite automata from exemples using hill-climbing. In *Proceednigs of* 4th *Annual Cognitive Science Conference* (pp. 105–108).

[Traber, 1995] Traber, C. (1995). SVOX : The implementation of a text-to-speech system for german. Diss. ETH Zürich NR. 11064, TIK-Schriftenreihe Nr. 7, vdf Hochschulverlag, Zürich.

[Turing, 1950] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 50, 433–460.

[Valiant, 1984] Valiant, L. (1984). A theory of the learnable. *Comunication of the ACM*, 27(11), 1134–1142.

[van den Bosch et al., 1996] van den Bosch, A., Daelemans, W., & Weijters, A. (1996). Morphological analysis as classiciation : an inductive-learning approach. In K. Oflazer & H. Somers (Eds.), *Proceedings of the Second International Conference on New Methods in Language Processing* (pp. 79–89).

[Vidal et al., 1993] Vidal, E., Rulot, H., Valiente, J., & Andreu, G. (1993). Recognition of planar shapes through the error-correcting grammatical inference algorithm (ecgi). In *First Int. Colloquium on Grammatical Inference : Theory, Application and Alternatives.*

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13, 260–269.

[Wagner, 1974] Wagner, R. A. (1974). Order-n correction for regular languages. *Communication of the ACM*, 17(5), 265–268.

[Wagner & Ficher, 1974] Wagner, R. A. & Ficher, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1), 168–173.

[Wharton, 1974] Wharton, C. (1974). Approximate language identification. *Information and Control*, 26, 236–255.

[Wolper, 1991] Wolper, P. (1991). *Introduction à la calculabilité.* InterEditions.

Index

algorithme
- de Baum-Welsh, 42
- de Brown, 118, 119
- de Dijkstra, 63
- de Good-Turing, 86
- de Ney, 118, 119
- de Viterbi, 10, 62–64, 91
- de Wagner, 60–62

alphabet, **10**, 17, **46**
- d'un modèle, 100
- fini, 43, 46
- infini, 10
- partitionnement d'un, 117
- taille d'un, **46**, 96

ambiguïté
- dans la l'attribution des catégories grammaticales, 76, 117
- dans la langue naturelle, 3, 7

analyse
- correctionnelle, 62, 96
  - optimale, 65
- d'une phrase, voir *analyse, d'une séquence*
- d'une séquence, 7, **12**, 27, 103, 115
  - probabiliste, 23, 89, 100, 114, 121

arbre accepteur de préfixes, **33**

automate, 11
- complet, 93

automate fini, 13–16
- acceptation d'un langage par, 15
  - exacte, **15**
- acceptation d'une chaîne par, **14**
- canonique, **16**
  - maximal, **33**
- configuration d'un, **14**
- déterministe, **13**, 14, 31, 37, 39
  - minimal, voir *automate fini, canonique*
- exécution d'un, **14**
- non-déterministe, 14, **14**, 15, 31, 70, 114
  - stochastique, 20
- universel, 15

bigram, **17**, 18, 100, 107–109

catégories grammaticales, 114, 117
- de la langue naturelle, 76–78, 103, 117

chaîne, **10**

chemin
- dans un automate
  - probabilité de, 20
- dans un graphe, **55**, 58, 59
- dans un treillis accoustique, 9, 10
- de coût minimal, 56, 62–64
- le plus court, 61
  - longueur de, 61

coûts
- "linguistiques", voir *coûts, déduits des catégories grammaticales*
- déduits des catégories grammaticales, 102
- fonction des, 49
- logarithmiques, 79, 81
- matrice de, **64**, 76, 77, 79, 80, 82
- unitaires, 51, **51**, 56, 59, 65, 77, 80, 87, 88, 101, 102, 104, 119

comparaison
- d'une séquence et d'un DAG, voir *distance, entre séquence et DAG*
- de deux DAG, voir *distance entre deux DAG*
- de deux séquences, voir *distance, d'édition*

des séquences, 43–45, 52, 65
dynamique, 52, 53
complexité
calculatoire, 12, 15, 16, 30, 42, 56, 59, 60, 62–64, 72, 96
du processus d'inférence, 38–39
des comparaison des séquences, 44
des constructions syntaxiques, 40
du langage, 60
consistance
d'un automate, voir *consistance, d'une grammaire*
d'un modèle, **74**
d'une grammaire, **19**, 20, 89, 90, 93, 96
contre-exemples, voir *exemples, négatifs*

décodage
correctionnel, voir *analyse, correctionnelle*
de séquence, voir *analyse, d'une séquence*
des catégories grammaticales, 111
du treillis acoustique, 134
dérivation, 18, 19, **19**, 71, 100, 114
correctionnelle, 92, 100
optimale, **71**, 72
d'un automate, 34, **34**
désambiguïsation
sémantique, 22, 23
statistique des phrases, 23
syntaxique, 9, 22
différence entre deux séquences, **49**
différence entre séquences, voir *distance, d'édition*
distance
correctionnelle, voir *distance, d'édition*
d'édition, 45, 47, **50**, 49–54, 58, 65, 70, 75, 109, 123
de Hamming, 44, 65
de Levenshtein, voir *distance, d'édition*
de Wagner et Ficher, voir *distance, d'édition*
entre deux DAG, 57–59
entre distribution des probabilités, 26
entre modèle et corpus, 39, **86**, 86–89, 126–129
entre modèle et séquence, 39, 70, 75, 86
entre séquence et DAG, 54–57
entre séquence et langage régulier, 59–65
pondérée, 62–65
entre séquences, 49–59, 123, 125, 127
euclidienne, 65
mathématique, 51
tableau de, 123, 124

ECGI, 41, 42, 69–84, 87, 88, 90, 92, 93, 97, 113, 125, 127, 129
"EM-like", 83
itératif, 96
non-stochastique, 70–74, 126
stochastique, 74–75
échantillon
caractéristique, 30, 38, **39**, 41
négatif, voir *exemples, négatifs*
positif, voir *exemples, positifs*
structurellement complet, **32**
croisée, 26, 40
évaluation
d'ECGIA, 75–76
d'un modèle, 25–27, 39–40, 88, 89, 100
avec une valeur unique, **87**
par cumul, **87**
par seuil, **87**
des coûts d'édition, 76, 78–80
des grammaires stochastiques, 96
exemples
apprendre des grammaires à partir de, 29
ensemble caractéristique de, voir *échantillon caractéristique*
négatifs, **16**, 27, 30, 32, 35, 39, 41, 42, 69, 87
"proches" du langage ciblé, 41

ordre de présentation, 73, 75, 126, 127, 129
positifs, **16**, 27, 30, 32–37, 39, 41, 70

grammaire, **11**
à structure de phrase, **11**
artificielle, 87
augmentée, 92, **92**
déterministe, 114
des classes, 115, 118
lissage d'une, voir *lissage, d'une grammaire régulière, des classes*
du langage L_0, 139
formelle, 11–13
hors-contexte, **12**
non-déterministe, 42, 71
régulière, **12**
lissage, voir *lissage, d'une grammaire régulière*
stochastique, 18–21, 117–119
sensible au contexte, **11**
type de, **11**
"grammaires noyaux", 125–129
graphe
acyclique orienté, 9, 43, 49, 54, **54**, 55, 56, 58, 83
cyclique, 62, 63
d'analyse, 62, 63
d'hypothèses, 57, 58
valué sur les arcs, 57, 71

inférence, 78, 87, 119
correcte, 36
critère de, 30, 36–38, 70
grammaticale, 24, 29–42, 69, 109, 125
stochastique, voir *inférence, stochastique*
méthode de, 29, 35
processus de, 29, 101
régulière, 32, **32**, 39, 41, 42
stochastique, **30**, 32
système de, 96

insertion, **47**

langage, **11**
infini, 11
régulier, 13, 34, 35
universel, 15
lexique, voir *alphabet*
lissage, **17**, 42, 77, 85, 90, 111, 119, 126
d'un bigram, 107
d'un N-gram, 79, 85–86, 90, 91, 94, 103
d'une grammaire régulière, 80, 85
des classes, 115
non-stochastique, 86
stochastique, 89–110
par Backing-Off, 18
par discounting, 18, 85, 90–91, 94, 102, 107
par interpolation, 17, 85, 91
listing, **48**, 47–49, 65

matrice
de coûts, voir *coût, matrice de*
de fréquences des opérations d'édition, **78**, 79, 84, 94, 95, 97, 101, 104, 107, 119
mesure de dissemblance, **54**
modèle
bigram, voir *bigram*
de langage, 7–27
à base de classes des mots, 78, 111–120
mot hors-vocabulaire, voir *mot inconnu*
mot inconnu, 96–100, 106, 115
mot nouveau, voir *mot inconnu*

N-gram, 8, **17**, 17–18, 32, 85, 108, 133, 134
à base de classes, 111–112, 114, 118
lissage, voir *lissage, d'un N-gram*
N-multigram, 18

opération élémentaire, voir *opération d'édition*
opération d'édition, 45, **45**, 47, **47**, 48–50, **50**, 51, 52, 56, 57, 59–62, 64, 65, 75, 76, 79, 87, 93, 106, 107

évaluation du coût de, voir *évaluation, des coûts d'édition*
coût, 49, 50, 77, 80, 84, 103
unitaire, 119
fréquence de, 65, 78, 79, 96, 97, 103, 104, 107
ordre de, 45, 48, 49, 65
pondérée, 65, 75, 80, 86
probabilité de, 85, 91–95, 99, 100, 104

perplexité, 26–27, 40, 76, 77, 80–83, 87–89, 100–109, 112, 118, 119
phrase, **10**, **46**
décodage d'une, voir *analyse, d'une séquence*
désambiguïsation statistique, voir *désambiguïsation, statistique des phrases*
syntaxiquement correcte, 17
taux d'acceptation, 126, 127
précision, **27**
préfixe, 14, 33, 47, **47**, 54, 56, 57, 59, 63, 65, 86
productions, voir *règle*

règle, **11**
d'erreur, 71, **71**, 72, 92
de correction d'erreurs, 92
non-erreur, 72, **72**, 126
rappel, **27**
reconnaissance
d'une séquence par un modèle de langage, voir *analyse, d'une séquence*
de la parole, 3, 7, 8, 22, 24, 27, 29, 49, 52, 69, 111, 133, 134, 159
des formes, 29, 30, 43
statistique, 30
des mots isolés, 70
optique des caractères, 31, 69, 122
processus de, 8, 159
système de, 3
taux d'erreur, 112
taux de, 27, 127–129
ressemblance
entre séquences, voir *distance, d'édition*

séquence, **10**, **46**
caractéristique, voir *séquence, centrale*
centrale, 122, **122**, 123–128
cible, 47–49
espace des, 46–47
générée par un graphe, 55
initiale, 55
longueur d'une, **26**, **46**
source, 47–49
séquences
ordre de présentation, voir *exemples, ordre de présentation*
sous-chaîne, voir *sous-séquence*
sous-chemin, 58
sous-classe, 36, 38, 41, 69
sous-couverture, **27**
sous-langage, 32
sous-phrase, voir *sous-séquence*
sous-séquence, **46**, 61, 65, 69, 70, 113
SRG, voir *grammaire, régulière, stochastique*
substitution, **47**
suffixe, 14, 54
suppression, **47**
sur-généralisation, **27**, 39, 41, 69, 87
symbole, 10

techniques de correction d'erreurs, 41, 43–65, 75, 86, 109, 121, 129
trace, **47**, 47–48, 53, 65
treillis, 34
acoustique, 8, 9, 22, 134
appris par ECGIA, 83
d'automates, 34, 35, 91
de mots, voir *treillis, acoustique*
trigram, 17, **17**, 18

unigram, **18**, 115

vocabulaire, voir *alphabet*

Résumé

Le contexte global de la thèse (réalisée dans le cadre de la convention CTI 97-1B-004, signée avec France Télécom) est l'apprentissage automatique de la syntaxe pour de la reconnaissance de la parole dans l'interaction orale entre l'homme et l'ordinateur. Le travail décrit dans ce document consiste à construire des *modèles de langage*, qui serviront de filtres syntaxiques à un ensemble d'hypothèses issues des étages situés en amont du processus de reconnaissance. Les modèles proposés sont des *grammaires régulires*. Elles constituent une alternative aux modèles utilisés couramment qui s'appuient uniquement sur la fréquence d'apparition des groupes des mots. Le travail de recherche est focalisé d'une part sur l'aspect algorithmique des techniques d'apprentissage appliqués aux grammaires formelles (*Inférence Grammaticale*), d'autre part sur la méthodologie d'intégration de ces modèles dans le processus de reconnaissance. Il est distribué selon quatre axes :

1. Utilisation d'un algorithme particulier (ECGIA) pour apprendre des modèles de langage à partir des corpus, par le biais de techniques de correction d'erreurs. Deux paramétrages originaux ont été proposés pour cet algorithme : l'un fondé sur les connaissances *a priori*, l'autre sur une approche entièrement automatique.

2. Élaboration d'une technique de *lissage* des grammaires apprises. Elle est indispensable à l'évaluation et à l'utilisation de ces modèles dans le cas d'insuffisance des données d'apprentissage (cas courant). Parallèlement, nous avons élaboré puis utilisé une méthode d'évaluation des grammaires non-stochastiques, fondée sur les techniques de correction d'erreurs ; elle permet de comparer leur pouvoir de généralisation.

3. Transposition des techniques de *clustering* au cas des modèles étudiés ;

4. Conception d'une approche originale dont l'objectif est d'extraire et de classer des *séquences caractéristiques* d'un corpus.

Ces approches ont été validées par voie expérimentale sur trois corpus. Deux d'entre eux ont été conçus à partir des cas réels (corpus AGS et ATIS), le troisième étant un corpus artificiel, proche de la langue naturelle.