Utilisation et apprentissage de modèles de langage pour la reconnaissance de la parole continue

Pierre Dupont

Document de thèse

Soutenue le 21 février 1996, à l'ENST Paris.

Rapporteurs : Jean-Paul Haton
Enrique Vidal
Examinateurs : Christophe Godin
Denis Jouvet
Laurent Miclet
Jacques Nicolas
Jean-Pierre Tubach

Table des matières

I Généralités sur la reconnaissance de la parole 1

1 Introduction 3
1.1 Reconnaissance et compréhension de la parole 3
1.2 Les modèles de langage 6
1.2.1 Objet des modèles de langage 6
1.2.2 Utilisation d'un modèle de langage 8
1.2.3 Apprentissage d'un modèle de langage 9

2 Reconnaissance de la parole continue 11
2.1 Concepts généraux 11
2.1.1 Les différents niveaux de représentation de la parole . . 11
2.1.2 Reconnaissance analytique ou intégrée 13
2.1.3 Le traitement acoustique 15
2.1.4 Le décodage acoustique 16
2.2 Les modèles de Markov 19
2.2.1 Définitions 19
2.2.2 Le critère du maximum de vraisemblance 22
2.2.3 Le critère de Viterbi 24
2.2.4 Apprentissage des modèles acoustiques 25
2.2.5 L'alignement Viterbi 29
2.2.6 Contrôle de l'espace de recherche 34

II Utilisation de modèles de langage 39

3 Typologie des modèles de langage 41
3.1 Modèles purement statistiques 42
3.1.1 Bigrams 42
3.1.2 N-grams 42
3.2 Modèles purement syntaxiques 44
3.2.1 Grammaires à structure de phrase 44

3.2.2 Grammaires d'unification 46
3.3 Modèles étendus ou combinés 47
3.3.1 N-grams définis au niveau de classes syntaxiques 47
3.3.2 Approximation statistique de grammaires 47
3.3.3 Arbres de classification 48
3.3.4 Grammaires hors-contexte stochastiques 48
3.4 Critère de qualité : la perplexité 50

4 Utilisation de modèles de langage hors-contexte (*) 53
4.1 Objectifs 53
4.2 Décodage Viterbi guidé par un automate fini 54
4.3 La méthode DYANE (*) 56
4.3.1 Représentation en réseaux récursifs 57
4.3.2 Développement dynamique d'un graphe fini (*) 58
4.3.3 Utilisation de la Forme Normale de Greibach (*) 59
4.3.4 Contrôle de l'espace de recherche (*) 65
4.3.5 Prédictions dialogiques (*) 66
4.4 Relations avec d'autres travaux 67

5 Evaluation expérimentale de la méthode DYANE (*) 71
5.1 Expériences sur la tâche de Feldman (*) 71
5.1.1 Description de la tâche et des modèles de langage . . . 71
5.1.2 Evaluations expérimentales (*) 73
5.2 Expériences complémentaires sur diverses bases de données (*) 78
5.2.1 Influence de la recherche en faisceau en mots isolés(*) . 78
5.2.2 Intérêt de l'intégration complète d'un modèle de langage (*) 79

III Apprentissage de modèles de langage 81

6 Apprentissage de modèles grammaticaux réguliers (*) 83
6.1 Introduction 83
6.2 Définitions et notations 84
6.2.1 Langages, automates et partitions 85
6.2.2 Echantillons d'un langage et automates associés 88
6.2.3 Treillis d'automates 91
6.3 Objet de l'inférence 92
6.3.1 La spécification d'un problème d'inférence grammaticale 92
6.3.2 Présentations admissibles 93
6.3.3 L'identification à la limite 94

6.3.4 Autres critères d'identification 95
6.3.5 Influence du type de présentations 96
6.3.6 Identification par utilisation de questions d'équivalence 97
6.3.7 Identification stochastique 97
6.3.8 Identification-PAC 98
6.4 Complexité calculatoire de l'identification d'un automate . . . 99
6.5 Espace de recherche (*) 100
6.5.1 Théorèmes fondamentaux (*) 101
6.5.2 Propriétés de l'espace de recherche (*) 103
6.6 Ensemble frontière (*) 105
6.7 Inférence régulière par recherche déterministe (*) 108

7 Algorithmes d'inférence régulière (*) 111
7.1 Inférence sans échantillon négatif 112
7.1.1 L'algorithme k-TSSI 113
7.1.2 Inférence de langages k-réversibles 115
7.1.3 L'algorithme MGGI 119
7.1.4 L'algorithme ECGI 122
7.2 Inférence avec un échantillon négatif 126
7.2.1 Les algorithmes RIG et BRIG 126
7.2.2 L'algorithme RPNI 129
7.2.3 L'algorithme GIG (*) 131
7.2.4 Le point de vue connexionniste 131
7.3 Inférence régulière semi-incrémentale (*) 132

8 Inférence régulière par optimisation génétique (*) 135
8.1 L'inférence régulière : un problème d'optimisation (*) 135
8.2 Les algorithmes génétiques 136
8.3 La méthode d'inférence GIG (*) 138
8.3.1 Initialisation de la population (*) 139
8.3.2 Représentation codée des solutions (*) 139
8.3.3 Opérateurs génétiques (*) 139
8.3.4 Fonctions d'évaluation (*) 141
8.3.5 L'algorithme GIG (*) 143
8.4 Comparaisons avec d'autres méthodes 144

9 Inférence caractérisable et incrémentale (*) 145
9.1 Caractérisation des données d'apprentissage (*) 146
9.1.1 Identification à la limite pour une présentation à données fixées (*) 146
9.1.2 Spécification d'un échantillon caractéristique 147

9.1.3 Identification à la limite de l'algorithme RPNI (*) . . . 155
9.1.4 Un élément remarquable de l'ensemble frontière 161
9.2 L'algorithme RPNI avec suppression des finales (*) 163
9.2.1 Présentation de l'algorithme (*) 163
9.2.2 Propriétés de l'algorithme RPNI avec suppression des finales (*) 167
9.3 Construction incrémentale de l'espace de recherche (*) 170
9.4 L'algorithme RPNI2 (*) 174
9.4.1 Comment garantir la compatibilité avec I_- (*) 174
9.4.2 Identification à la limite de l'algorithme RPNI2 (*) . . 179
9.4.3 Complexité de l'algorithme RPNI2 (*) 182

10 Evaluation expérimentale de l'inférence (*) 185

10.1 Historique et motivations 185
10.2 Le protocole expérimental (*) 186
10.3 La base d'évaluation 188
10.4 Résultats comparatifs de RPNI, GIG et BRIG (*) 189
10.4.1 Les trois algorithmes concurrents (*) 189
10.4.2 Sensibilité de la méthode GIG au réglage des paramètres (*) 194
10.4.3 Apport de l'inférence semi-incrémentale (*) 200
10.5 Evaluations sur la tâche de Feldman (*) 204
10.5.1 Adaptation du protocole expérimental (*) 204
10.5.2 Résultats comparatifs des algorithmes GIG, RPNI et RPNI2 (*) 205

11 Conclusions et perspectives 209

11.1 Bilan de notre étude 209
11.2 Inférence de grammaires régulières stochastiques 210
11.3 Inférence de grammaires hors-contexte 211

A Grammaires de la tâche de Feldman 213

A.1 Grammaire régulière française (*) 213
A.2 Grammaire régulière espagnole 214
A.3 Grammaire hors-contexte française (*) 215
A.4 Grammaire hors-contexte espagnole (*) 217

B Automates de la base d'évaluation 219

C La grammaire du cidre 223

Table des figures

2.1 Chaîne de Markov. 16
2.2 Modèle du mot “aller” : a , l , é. 17
2.3 Modèle de Markov du premier ordre. 21
2.4 Récurrence de Baum-Welch. 24
2.5 Récurrence de Viterbi. 25
2.6 Alignement entre vecteurs acoustiques et modèles de mots. . . 30
2.7 Transitions au sein d'un mot (a) et transitions entre mots (b). 32
2.8 Création et recombinaison des points. 36

4.1 Exemple de graphe syntaxique. 55
4.2 Exemple de graphe sans contrainte syntaxique. 56
4.3 Exemple de réseaux récursifs. 57
4.4 Exemple d'expansion dynamique du réseau initial. 58
4.5 Item d'un chart. 68

5.1 Modèle $LM1$: taux d'erreurs à seuil d'élagage variable. 74
5.2 Modèle $LM2$: taux d'erreurs à seuil d'élagage variable. 75
5.3 Modèle $LM2$: évolution du graphe syntaxique. 76
5.4 Modèle $LM2$: complexité temporelle de la méthode DYANE. 77
5.5 Modèle $LM2$: complexité en nombre d'expansions. 78

6.1 Une représentation de l'automate A_1. 86
6.2 Automate canonique du langage $L = a(aa)^*b(bb)^*$. 88
6.3 L'automate quotient A_1/π. 89
6.4 L'automate maximal canonique $MCA(I_+)$, avec $I_+ = \{a, ab, bab\}$. 90
6.5 L'arbre accepteur des préfixes $PTA(I_+)$, avec $I_+ = \{a, ab, bab\}$. 91
6.6 L'automate universel sur l'alphabet $\Sigma = \{a, b\}$. 91
6.7 L'automate A et le $MCA(I_+)$, avec $I_+ = \{ab\}$. 102
6.8 $AFN(L_1)$, $A(L_1)$ et $MCA(I_+)$, avec $L_1 = (ba^*a)^*$ et $I_+ = \{baa\}$.104
6.9 $MCA(I_+)$ et $PTA(I_+)$, avec $I_+ = \{ba, baa\}$. 105
6.10 $AFN(L_2)$ et $A(L_2)$, avec $L_2 = (ba + b(aa)^*)$. 105

6.11 Sup demi-treillis des AFDs dans $Lat(PTA(I_+))$, pour $I_+ = \{aaa\}$. 110

7.1 Exemple d'exécution de l'algorithme k-TSSI. 114
7.2 L'automate A et son inverse A^r. 116
7.3 Exemple d'exécution de l'algorithme k-RI. 118
7.4 Automate acceptant le langage local sur l'alphabet E'. 121
7.5 Automate inféré par l'algorithme MGGI. 121
7.6 Exemple d'exécution de l'algorithme ECGI. 124
7.7 Construction de l'automate acceptant $(aa)^* \cup bb$. 133

9.1 Automate acceptant un nombre pair de b sur l'alphabet $\{a, b\}$. 148
9.2 Arbre accepteur des préfixes de $\{\lambda, a, bb, bba, baab, baaaba\}$. . . 151
9.3 Automates quotient déterministe (a) et non-déterministe (b). . 152
9.4 Fusion pour déterminisation. 154
9.5 Exemple d'exécution de l'algorithme RPNI. 161
9.6 Exécution de l'algorithme RPNI avec suppression des finales. . 167
9.7 Incompatibilité vis-à-vis de l'échantillon positif par l'exécution de l'algorithme RPNI avec suppression des finales. 169
9.8 Extension par le plus court suffixe. 172
9.9 L'arbre des préfixes et un automate quotient déterministe. . . 175
9.10 Fission d'un automate quotient déterministe. 176
9.11 Fission pour déterminisation. 178

10.1 Performance de GIG en fonction de la taille de la population. . 196
10.2 Performance et temps calcul de GIG en fonction du nombre maximal d'évaluations. 197
10.3 Performance de GIG en fonction du taux de croisement. . . . 198
10.4 Performance de GIG en fonction du taux de mutation par bit. 199
10.5 Performance de GIG en fonction du taux de descente. 200
10.6 L'automate cible et les deux solutions proposées par l'algorithme GIG. 202
10.7 Exécution semi-incrémentale de l'algorithme GIG. 203
10.8 RPNI et RPNI2 : taux de classification sur les données de test. 207
10.9 Complexité comparative de RPNI2 et RPNI. 208

C.1 La grammaire du cidre. 225

"Que nous as-tu pondu de drôle ?"
demanda Racine à Corneille qui venait d'écrire Polyeucte[1].

[1] Extrait de la *Grammaire française et impertinente*, par Jean-Louis Fournier, Editions Payot, 1992.

Résumé

Le présent travail s'intéresse à la modélisation du langage pour la reconnaissance de la parole continue. Cette modélisation a pour objectif le contrôle du décodage acoustique par une information contraignant les séquences de mots à reconnaître.

Dans la première partie, nous définissons le contexte général d'utilisation d'un modèle de langage. Nous y abordons le cadre probabiliste et les aspects algorithmiques de l'utilisation de modèles de Markov cachés en reconnaissance de la parole continue. Nous détaillons l'algorithme de Viterbi et la stratégie de recherche en faisceau qui lui est associée.

La seconde partie de notre travail traite d'un type particulier de modèles de langage, ceux basés sur une grammaire hors-contexte probabiliste ou non. Nous proposons une approche originale pour l'intégration d'un tel modèle dans l'algorithme de Viterbi. Elle utilise un développement d'une structure récursive qui définit dynamiquement l'espace de recherche du décodage acoustique. A ce propos, nous montrons l'intérêt d'une mise sous forme normale de Greibach de la grammaire hors-contexte. Nous étudions également la complexité théorique d'un algorithme de transformation sous cette forme normale. Les performances de notre approche sont évaluées dans différents systèmes de reconnaissance de la parole en mettant l'accent sur sa complexité pratique.

Dans la troisième partie, nous nous intéressons à l'apprentissage automatique de modèles de langage définis par le biais d'une grammaire formelle. En particulier, nous nous concentrons sur l'inférence de grammaires régulières à partir d'échantillons positif et négatif d'un langage. Nous proposons une étude théorique détaillée de l'espace de recherche de ce problème et nous démontrons des propriétés originales permettant de guider la construction d'une solution. Nous introduisons également un nouveau point de vue qui consiste à traiter l'inférence régulière comme un problème d'optimisation combinatoire. Dans ce cadre, nous développons un algorithme d'inférence par optimisation génétique. Ensuite, nous étudions l'inférence régulière sur base d'une présentation séquentielle des données d'apprentissage. A cette fin, une extension incrémentale d'un algorithme connu est proposée. Nous en démontrons la convergence et nous étudions sa complexité théorique. Finalement, nous définissons deux protocoles d'évaluation de méthodes d'inférence et nous comparons les différents algorithmes étudiés.

Remerciements

A l'heure d'écrire ces lignes, je suis encore surpris par le nombre et la qualité des personnes qui m'ont considérablement aidé à la réalisation de ce travail.

La première de ces personnes est sans conteste Laurent Miclet, mon directeur de thèse. Il m'est assez difficile de résumer son aide tellement elle fut conséquente. Son enthousiasme, sa compétence et sa disponibilité permanente furent essentiels. Et puis, il y a quelque chose qui doit s'appeler une confiance donnée. Elle fut la meilleure des motivations.

J'ai eu également la très grande chance de collaborer avec Enrique Vidal, personnage étonnant par le champ spectaculaire de ses compétences. Je ne compte plus les discussions animées, les petits séminaires et autres idées astucieuses qui ont fait de mon séjour à l'université polytechnique de Valencia une expérience unique. Outre ses qualités techniques, il a fait preuve d'un sens de l'accueil, d'un humour et d'une simplicité rares.

Je remercie sincèrement Christophe Godin, Jean-Paul Haton, Denis Jouvet, Jacques Nicolas et Jean-Pierre Tubach pour leur participation au jury. Leurs commentaires communiqués lors d'une présentation préliminaire de ce travail m'ont été précieux et leurs analyses m'ont guidé pour en améliorer le contenu.

Je remercie également Christel Sorin pour m'avoir permis de mener ce travail, tout en étant membre du Centre National d'Etudes en Télécommunications, à l'heure où les préoccupations d'un "mathématicien aux mains blanches" (sic) ne sont pas forcément prioritaires.

La première partie de ce travail a débuté lorsque j'étais membre du laboratoire de Philips à Bruxelles. Mes "premiers pas" y furent si bien guidés par Yves Kamp et André Thayse qui m'ont communiqué leur goût de la recherche - une ligne claire délivrée sans contraintes. Dominique Snyers fut aussi l'artisan du début de ce travail et nous dissertâmes des "réseaux récursifs" jusque sur le chemin de la Pilatte, lieu quelque peu improbable.

De nombreux membres du Centre National d'Etudes en Télécommunications et de l'université polytechnique de Valencia m'ont aidé pendant ce travail. Je pense en particulier à Jacques Simonin, Thierry Moudenc et Chafic Mokbel qui m'ont supporté malgré l'état permanent, voire dégradé, de ma qualité de thésard. Je pense également à Victor Jiménez, pour les nombreuses heures tardives où nous nous sommes consacrés à la programmation fine et à l'étude d'algorithmes les plus récursivement définissables, et à Pedro García pour sa disponibilité à m'éclairer sur quelques propriétés qui me semblaient a priori des plus obscures.

Enfin, je remercie ma famille pour son soutien constant et je ferai grâce à mes parents de la démonstration d'une propriété triviale car tout ça, finalement, ne serait jamais arrivé sans eux. □

Première partie

Généralités sur la reconnaissance de la parole

Chapitre 1

Introduction

1.1 Reconnaissance et compréhension de la parole

La parole est l'un des moyens les plus naturels par lequel des personnes communiquent. Cependant, à ce jour, la commande de machines est en général effectuée par des gestes ou par le truchement de langages artificiels, tels les langages de programmation ou les commandes des automatismes. Par exemple, une machine à écrire peut être commandée par l'intermédiaire d'un clavier manuel ou d'une commande de télétype. Pour des raisons de facilité d'interaction, on a depuis longtemps été tenté de concevoir des machines dont les commandes seraient directement activées par la parole. Dans cette perspective, il s'agit de développer des systèmes capables de reconnaître la parole, de la comprendre et d'exécuter les actions résultant de la compréhension du message.

La *reconnaissance de la parole* a pour objet la transformation automatique d'un signal acoustique en une séquence de mots qui, idéalement, correspond à la phrase prononcée par un locuteur. La *reconnaissance* constitue donc un processus de *traduction* d'un signal acoustique en une phrase ou une séquence de phrases de la *langue naturelle*[1]. Ce processus pourra s'appuyer sur une description de la syntaxe de la langue naturelle - à tout le moins d'un fragment de cette langue -, puisque c'est la syntaxe qui permet de caractériser les phrases, c'est-à-dire les séquences de mots appartenant à la langue. Une

[1]Par défaut, l'appellation "langue naturelle" concerne ici la langue orale telle qu'elle est prononcée par un locuteur, retranscrite manuellement ou reconnue par un système automatique. Cependant, les techniques d'apprentissage présentées dans la suite de ce travail ne sont pas spécifiques à la langue orale, ni à la langue proprement dite. Elles peuvent, en principe, s'appliquer plus généralement à la modélisation de processus séquentiels.

description de la sémantique de la langue naturelle, c'est-à-dire une théorie qui vise à rendre compte des phénomènes signifiants du langage, peut également être utilisée en vue de restreindre la reconnaissance aux seules phrases porteuses de signification.

Le but de la *compréhension de la parole* est de percevoir la signification - le sens - de la phrase et de traduire cette signification dans un langage interprétable par une machine ; ce peut être, par exemple, un langage dont chacune des phrases est constituée d'une séquence d'ordres. La traduction s'effectue souvent en deux temps : une traduction de la langue naturelle dans un *langage formel* suivie d'une traduction de ce langage dans un formalisme directement interprétable par la machine. La *compréhension* fait donc appel au processus de perception de la signification de la phrase ; ce processus utilisera par conséquent une description de la sémantique de la langue.

Les applications de la reconnaissance et de la compréhension de la parole sont en général restreintes à des contextes particuliers qui n'utilisent qu'un fragment de la langue. Que ce soient des systèmes d'interrogation d'une base de données, des systèmes de dictée automatique de lettres d'affaires, des systèmes de commande vocale d'éditeur de texte ou de robots, tous ces systèmes utilisent des fragments de la langue naturelle adaptés à leur application particulière. La traduction des phrases de ces fragments dans un formalisme interprétable par la machine permettra alors d'effectuer directement les opérations qui résultent de la signification de ces phrases.

Certains systèmes de reconnaissance [Jelinek85, Rabiner et al.87, Baker93] sont basés sur le traitement de *mots isolés* ; ceci implique que le locuteur doive marquer une pause entre chaque mot prononcé. Cette pause (caractérisée par une période de faible énergie du signal acoustique) est utilisée pour segmenter le signal acoustique et supprime les effets de coarticulation entre mots ; il restera alors à identifier chacun des segments ainsi obtenus à un mot du vocabulaire. Cette façon de procéder, si elle permet un traitement plus aisé, est cependant fort artificielle pour le locuteur.

D'autres systèmes [Bahl et al.78, Lee88, Murveit et al.88] ont pour but de traiter la *parole continue*, c'est-à-dire des phrases prononcées de façon naturelle sans marquer de pause entre les mots. A ce jour, les systèmes de reconnaissance de parole continue sont, en outre, caractérisés par des vocabulaires pouvant atteindre plusieurs milliers de mots [Steinbiss et al.90, Gauvain et al.93]. L'accroissement du nombre de mots du vocabulaire va de pair avec un accroissement du risque de confusion lors de la reconnaissance. Ces systèmes sont confrontés au problème difficile de traduire un signal acoustique en une séquence de mots, dont on ignore a priori le nombre ainsi que la localisation dans l'intervalle de temps défini par le début et la fin du signal acoustique. Finalement, les effets de coarticulation entre mots et de vitesse d'élocution accrue (par rapport au mode d'élocution en mots isolés) rendent la tâche de la reconnaissance de la *parole continue* encore plus complexe.

De nombreux travaux ont porté sur la définition et l'utilisation de *modèles acoustiques,* c'est-à-dire une représentation, généralement statistique, des sons d'une langue. Notre travail ne concerne pas le développement de tels modèles. Cependant, nous donnons une présentation générale d'un système de reconnaissance de parole[2] au chapitre 2 et nous y exposons brièvement les *modèles de Markov cachés.* Ils sont utilisés comme modèles acoustiques dans la très grande majorité des systèmes de reconnaissance actuels.

L'objectif de la première partie de ce travail est l'utilisation d'un *modèle de langage,* c'est-à-dire une information permettant de guider la reconnaissance par une définition des séquences de mots considérées comme correctes ou probables. Nous présentons au chapitre 3 les différents modèles de langage généralement utilisés et nous traitons au chapitre 4 de l'utilisation d'un type particulier de modèle, celui basé sur une grammaire hors-contexte. La complexité pratique de la méthode proposée et les performances obtenues pour un système de reconnaissance de parole continue sont présentées au chapitre 5.

La seconde partie de ce travail concerne l'apprentissage automatique d'un modèle de langage. Dans la mesure où nous nous intéressons ici à des modèles structurels, c'est-à-dire qui peuvent être représentés par une grammaire formelle, ce problème relève de l'*inférence grammaticale.* Nous définissons formellement l'objet de l'inférence au chapitre 6. En particulier, nous y détaillons les propriétés générales de l'inférence de grammaires régulières, ou *inférence régulière.* Dans ce cadre théorique, nous présentons au chapitre 7 une sélection de méthodes exactes ou heuristiques. Ensuite, nous proposons une nouvelle méthode d'inférence par optimisation génétique. Cette méthode est exposée au chapitre 8. Nous proposons également une extension incrémentale d'un algorithme d'inférence existant. Elle consiste à tirer parti de la solution construite précédemment lorsque les données sont reçues une à une et dans un ordre quelconque. Une caractérisation formelle de l'algorithme original et de son extension incrémentale est exposée au chapitre 9. Finalement, nous présentons au chapitre 10 des résultats expérimentaux comparatifs de différentes techniques d'inférence régulière.

[2]Les chapitres et paragraphes marqués d'un astérisque (*) sont ceux pour lesquels nous estimons avoir apporté une contribution originale. Les autres parties de ce document sont présentées à des fins introductives ou comparatives.

1.2 Les modèles de langage

1.2.1 Objet des modèles de langage

Malgré le recours à des modèles acoustiques complexes, la phrase reconnue par un système automatique peut être partiellement erronée. Ainsi, la phrase

"Jean s'endort et rêve"

pourrait être reconnue comme

"Jean sent doré rêve".

L'introduction de contraintes sur les séquences de mots a pour effet de restreindre la reconnaissance à des phrases syntaxiquement correctes ou à des séquences probables, diminuant par ce biais le taux d'erreur [Young et al.89]. Ce type de contraintes nécessite une information particulière, en l'occurrence un modèle de langage.

Différents types de modèles langages ont été proposés dans le but de restreindre les séquences de mots envisagées au cours d'un processus de reconnaissance de la parole. Nous les présentons en détails au chapitre 3.

Les modèles les plus élémentaires sont de nature purement statistique. Ils sont basés sur les propriétés statistiques d'un fragment observé de la langue naturelle plutôt que sur une théorie formelle de celle-ci. Ainsi, les *N-grams* [Bahl et al.83] attribuent une probabilité aux séquences de mots selon leurs fréquences d'apparition dans un corpus d'estimation. Ils représentent des contraintes *locales* sur les séquences de mots.

Les *modèles syntaxiques,* ou *structurels*, sont basés sur une théorie formelle de la langue telle qu'en fournissent les langages logiques ou les grammaires formelles. Celles-ci définissent au moyen de règles les structures syntaxiques admises et, par conséquent, les séquences de mots correspondant à ces structures. Parmi les formalisations possibles, celle des "grammaires à structure de phrase" due à Chomsky a été largement utilisée [Chomsky57]. Les modèles ainsi conçus déterminent de façon univoque les séquences de mots correspondant aux structures syntaxiques admises. Le fragment de la langue naturelle couvert syntaxiquement sera d'autant plus petit que le nombre de structures syntaxiques est limité, ce qui imposera d'autant plus de contraintes au locuteur qui est supposé respecter la syntaxe définie. Par contre, plus restrictif sera le modèle, plus grande sera la possibilité d'éviter des erreurs de reconnaissance.

Dans la hiérarchie des "grammaires à structure de phrase", les "grammaires hors-contexte" occupent une place importante aussi bien en traitement de la langue naturelle [Gazdar et al.89] que dans la conception des langages de programmation [Aho et al.72]. De plus, ces grammaires peuvent être considérées comme la base de modèles de langage plus complexes tels que, par exemple, "les grammaires à structure de phrase généralisées" [Gazdar et al.85], "les grammaires lexico-fonctionnelles" [Bresnan82] et "les grammaires à clauses définies" [Pereira et al.80]. Ces formalismes permettent de rendre compte de manière souple de contraintes syntaxiques dépendant du contexte, ainsi que de certaines contraintes sémantiques. La phrase

"Jean sent doré"

est syntaxiquement correcte quoique probablement dépourvue de sens. Pour éviter ce type d'erreur, il faut faire appel à un modèle de langage acceptant exclusivement les phrases sémantiquement correctes. Ceci suppose une connaissance sémantique du "domaine du discours", c'est-à-dire du contexte dans lequel est utilisé le fragment de la langue naturelle traité. Cette connaissance est alors formalisée et incluse dans le modèle.

Nous ne détaillerons pas ici les nombreux formalismes introduits en *Traitement de la Langue Naturelle*[3]. Le lecteur intéressé pourra, par exemple, consulter les ouvrages [Gazdar et al.89, Miller et al.90]. Ces formalismes ont généralement un pouvoir d'expression bien supérieur à celui des grammaires hors-contexte. Néanmoins, l'utilisation de ces modèles pour guider un système de reconnaissance s'avère problématique. En effet, nous verrons que l'utilisation d'une grammaire formelle dans un système de reconnaissance peut donner lieu à une complexité calculatoire critique. A ce titre, les grammaires hors-contexte offrent un compromis intéressant entre leur pouvoir d'expression et la complexité calculatoire de leur utilisation.

L'avantage des modèles structurels sur les N-grams est qu'ils représentent généralement des contraintes *globales* sur les séquences de mots constituant la phrase prononcée. La notion de contrainte globale n'implique pas nécessairement une modélisation complète et détaillée de chaque constituant d'un énoncé mais elle suppose de pouvoir prendre en compte des dépendances à long terme. L'inconvénient majeur des modèles structurels est qu'ils reposent sur une théorie qui doit être adaptée pour chaque fragment de la langue à traiter. De plus, la parole spontanée ne respecte généralement pas la syntaxe écrite. Ces observations motivent le développement de techniques

[3] Ces formalismes ont été développés essentiellement pour le traitement de la langue naturelle écrite.

d'apprentissage de modèles structurels à partir d'exemples. Un modèle combinant les informations structurelles et statistiques est, par exemple, celui des grammaires hors-contexte probabilistes [Gonzalez et al.78, Jelinek et al.92]. L'objectif serait, à terme, de disposer d'une procédure automatique d'apprentissage des règles et d'estimation des probabilités de ce genre de modèle.

Remarquons que si l'objet de la reconnaissance se limite à la traduction correcte entre le signal acoustique observé et la séquence de mots prononcée, le problème plus général du dialogue oral consiste à gérer une séquence d'échanges entre une machine et son utilisateur. Une modélisation logique du dialogue a été proposée, par exemple, par Sadek [Sadek94]. La question d'extraire le contenu pragmatico-sémantique d'une phrase éventuellement mal reconnue, afin de tirer complètement parti de ce type de modèle, nous semble délicate.

1.2.2 Utilisation d'un modèle de langage

Une approche possible, utilisant un modèle de langage pour contrôler la reconnaissance de la parole, procède en deux étapes. Dans ce cas, le *décodage acoustique* a pour objet de transformer le signal acoustique reçu en un *treillis de mots*[4], c'est-à-dire en un ensemble d'hypothèses de mots auxquelles est attribuée une localisation temporelle dans la phrase. Ce treillis est ensuite analysé sur base du modèle de langage en vue d'en extraire la phrase correcte [Quinton82, Tomita86, Fissore et al.88, Chow et al.89]. L'inconvénient de cette méthode réside dans la perte d'information lors de la conversion du signal acoustique en un ensemble de symboles - le treillis de mots -, ce qui diminue la possibilité de correction d'erreurs. En effet, l'obtention du treillis suppose, pour pouvoir être réalisée pratiquement, une sélection parmi l'ensemble des hypothèses envisagées au cours du décodage acoustique. Ces hypothèses, relatives à la localisation et à l'identification des différents mots (voire même des phonèmes) de la phrase, constituent *l'espace de recherche*. Or cette sélection est effectuée sur base du seul contenu acoustique du signal, c'est-à-dire avant qu'aucune connaissance sur les séquences de mots n'ait été utilisée.

L'approche que nous envisagerons par la suite est *intégrée* en ce sens qu'elle utilise le modèle de langage pendant le décodage acoustique [Ney84, Godin90, Dupont et al.90]. La phrase reconnue résulte donc de la sélection dans l'espace de recherche de la meilleure séquence de mots, eu égard au contenu acoustique du signal et au modèle de langage utilisé. L'approche

[4] Quinton utilise le terme de spectre lexical [Quinton82]. Le décodage acoustique, tel que nous l'avons défini, peut lui-même être effectué en plusieurs étapes (voir paragraphe 2.1.2).

intégrée donne le plus souvent un meilleur taux de reconnaissance. Dans ce cas, le décodage acoustique s'effectue sur un espace de recherche défini par le modèle de langage.

Si une approche intégrée est généralement meilleure en termes de taux de reconnaissance, la question de son coût doit être étudiée avec soin. Le coût concerne ici les ressources de calcul requises pour effectuer la reconnaissance. Disposant de ressources caractérisées par une vitesse de calcul et une capacité de mémorisation données, l'objectif est d'effectuer la reconnaissance en un temps minimum tout en limitant l'espace mémoire nécessaire. Nous étudierons cette question plus en détails au chapitre 4.

1.2.3 Apprentissage d'un modèle de langage

La seconde partie de ce travail concerne l'apprentissage d'un modèle de langage pour un système de reconnaissance de parole continue.

L'apprentissage dépend directement du type de modèle utilisé. Pour un modèle purement statistique, comme celui des N-grams, il s'agit d'un problème d'estimation statistique. Le manque de données par rapport au nombre de paramètres à estimer peut être traité par diverses techniques de lissage [Jelinek et al.80, Katz87, Brown et al.92b]. Une extension intéressante des N-grams est un modèle "cache" [Kuhn et al.90] qui a pour objet de modifier dynamiquement les probabilités associées aux séquences de mots en fonction des séquences récemment observées. Il s'agit en quelque sorte d'une technique de réestimation continue. D'autres techniques permettent de relâcher l'aspect purement local des N-grams usuels et tentent de traiter les dépendances à plus long terme [Ney et al.94, Della Pietra et al.94].

Dans la mesure où les modèles structurels permettent de représenter directement des contraintes globales sur la séquence de mots à reconnaître, ils nous semblent potentiellement plus intéressants. Lorsque ces modèles sont représentés par le biais d'une grammaire formelle, leur apprentissage relève de l'inférence grammaticale [Biermann et al.72, Fu et al.75a, Fu et al.75b, Miclet79, Gregor94]. Il s'agit de découvrir les règles d'une grammaire formelle à partir d'exemples du fragment de la langue à modéliser. Des contre-exemples, c'est-à-dire des séquences de mots n'appartenant pas au fragment considéré, peuvent également guider le processus d'apprentissage.

Dans cette seconde partie de notre travail nous revoyons le problème de l'inférence d'une grammaire. Nous concentrons ici notre attention sur les grammaires régulières. En effet, c'est pour cette classe de grammaires que la plupart des techniques d'inférences ont été développées. L'extension à l'inférence de grammaires hors-contexte sera évoquée à la fin de ce document. Notre étude permet de caractériser l'espace de recherche de l'inférence régulière et de formaliser certaines de ses propriétés. Ensuite, nous développons le point de vue qui consiste à traiter l'inférence régulière comme un problème d'optimisation explicite. Finalement, nous nous intéressons au traitement de données d'apprentissage reçues séquentiellement.

Chapitre 2

Reconnaissance de la parole continue

2.1 Concepts généraux

2.1.1 Les différents niveaux de représentation de la parole

La mise en œuvre d'un algorithme de reconnaissance de la parole fait appel à différentes informations, généralement désignées comme les *sources de connaissance.* Chacune est associée à un niveau de représentation de la parole : les niveaux acoustique, phonétique, lexical, syntaxique et sémantique, et pragmatique[1]. Nous exposons brièvement ces différents niveaux ci-après.

Le *traitement acoustique* regroupe l'acquisition du signal vocal, son filtrage, son échantillonnage et l'extraction des coefficients caractéristiques de ce signal. Au niveau *acoustique*, la parole est donc représentée par une suite de *vecteurs* constitués des coefficients qui caractérisent le signal.

Le niveau *phonétique* représente le signal de parole comme une séquence de *phonèmes.* Ceux-ci sont les unités de base des sons d'une langue à partir desquelles sont construits les mots. Les langues européennes (français, anglais, allemand, etc.) comportent une quarantaine de phonèmes[2]. A ces unités

[1]La définition de chaque niveau de représentation répond à un certain arbitraire. On peut d'ailleurs raffiner ces représentations en différenciant, par exemple, les informations *phonétique* et *phonologique* ou en distinguant l'information *morphologique* dans l'information lexicale. Certaines sources de connaissance, comme la *prosodie,* peuvent d'ailleurs porter sur plusieurs niveaux de représentation (la durée des phonèmes et leur intonation affectant la signification, par exemple).

[2]Le nombre et la définition des unités phonétiques peut être adapté d'un système à l'autre, même pour une langue donnée. Lorsque l'on a recours a une modélisation statis-

de base sont généralement associés des modèles statistiques permettant de rendre compte de la variabilité du signal de parole [Jelinek76, Bahl et al.83, Bourlard et al.85b]. Nous donnerons un premier aperçu des caractéristiques de ces modèles statistiques au paragraphe 2.1.4. Dans le cas particulier où les mots font partie d'un vocabulaire de petite taille (quelques dizaines de mots), on préfère souvent utiliser les mots comme unités de base plutôt que les phonèmes [Rabiner et al.87, Jouvet88]. Dans ce cas, les modèles statistiques, associés aux mots et non plus aux phonèmes, incluent directement les effets de coarticulation entre phonèmes au sein d'un mot. Par contre, l'utilisation des phonèmes comme unités de base permet de décrire un vocabulaire, en principe, "illimité". En effet, l'ajout d'un mot au vocabulaire se réalise simplement en donnant sa transcription phonétique, ce qui ne nécessite pas la définition d'un nouveau modèle[3].

La connaissance *lexicale* détermine, parmi l'ensemble des séquences de phonèmes possibles, celles qui constituent des mots du vocabulaire admis. Cette connaissance repose sur un lexique reprenant l'ensemble des mots de l'application traitée ainsi que leur transcription phonétique. Un mot peut avoir diverses prononciations auxquelles correspondent diverses transcriptions phonétiques.

La connaissance *syntaxique* et *sémantique* est basée sur un modèle du fragment considéré de la langue. Elle permet de restreindre les séquences de mots envisagées à des phrases syntaxiquement et sémantiquement correctes ou de favoriser les séquences les plus probables. La *syntaxe* concerne la structure de la langue et détermine les constructions grammaticales tandis que la *sémantique* s'intéresse à son contenu en éliminant les phrases dénuées de sens qui pourraient cependant être syntaxiquement correctes.

Dans une application donnée, certaines phrases, quoique porteuses d'une signification, d'un message raisonnable dans l'absolu, sont simplement en dehors du sujet : une connaissance *pragmatique*, qui examine la cohérence de la phrase reconnue avec le sujet en cours, permettra de se dispenser d'entreprendre la reconnaissance de phrases en dehors du sujet.

---

tique, la définition de ces unités peut reposer à la fois sur une connaissance phonologique et sur des propriétés statistiques de la base d'estimation (fréquences observées des différentes unités, proximité des distributions associées, interpolation entre différents jeux d'unités) [Lee88, Jouvet et al.91].

[3]A l'heure actuelle, les modèles statistiques utilisés sont généralement des "phonèmes en contexte". L'extension, en principe illimitée, à de nouveaux mots, suppose que nous disposions d'une base de données statistiquement représentative pour estimer les paramètres de ces modèles. En particulier, même si l'ensemble des contextes phonétiques est représenté dans la base d'estimation, il faut encore que les conditions de test soient effectivement similaires à celles de la base d'estimation (bruit ambiant, paramètres prosodiques, etc).

Les sources de connaissance évoquées ci-dessus peuvent être décrites sur base d'une connaissance a priori de la langue, du domaine voire de l'application traitée. Depuis le succès des modèles statistiques au niveau acoustique, de plus en plus de connaissances relatives aux autres niveaux sont également estimées à partir de données d'apprentissage. Les connaissances a priori peuvent alors servir de guide ou biais d'apprentissage. Cette méthodologie offre deux avantages. D'une part, elle tend à définir un cadre commun, en général statistique, pour la plupart des sources de connaissances utilisées et, d'autre part, elle rend plus aisée l'adaptation à une nouvelle tâche, voire à une nouvelle langue. C'est la raison pour laquelle, nous consacrons une part importante de ce travail à l'apprentissage automatique d'une connaissance syntaxique.

2.1.2 Reconnaissance analytique ou intégrée

Les différents niveaux de représentation de la parole traitent des unités couvrant des fractions plus ou moins étendues du signal de parole. Un vecteur acoustique est associé à chaque tranche de parole d'une durée de l'ordre de 10 ms. Un phonème couvre environ un dixième de seconde alors qu'un mot comporte typiquement de un à dix phonèmes. C'est pourquoi le niveau acoustique est généralement considéré comme inférieur au niveau phonétique, lui-même inférieur au niveau lexical et ainsi de suite. Cette hiérarchie est donc établie sur base de la durée moyenne des unités associées à chaque niveau.

Partant de la séquence de vecteurs acoustiques, résultat du traitement du signal de parole, deux familles de méthodes de reconnaissance sont envisageables : nous désignerons les premières comme les méthodes *analytiques* et les secondes comme les méthodes *intégrées*.

Les méthodes *analytiques* reposent sur une séparation entre les différents niveaux de représentation de la parole et à chaque niveau correspond un module de traitement spécifique [Lesser et al.75, Mercier et al.78, Haton et al.78, Méloni83]. Par exemple, dans une approche analytique ascendante, chaque niveau de représentation a comme données le résultat du niveau précédent ainsi que sa propre source de connaissance, tel un modèle de langage pour le niveau syntaxique. A partir des vecteurs acoustiques, le niveau phonétique assure la segmentation du signal en phonèmes. Cette segmentation s'exprime sous la forme d'une séquence, voire d'un *treillis phonétique*[4]. Un treillis phonétique est un ensemble d'hypothèses de phonèmes auxquelles on attribue une localisation temporelle dans la phrase. Cette information est alors convertie en une séquence, ou en un treillis, de mots sur

[4]Quinton utilise le terme de *spectre phonétique* [Quinton82].

base de la connaissance lexicale. La séquence ou le treillis de mots est traité par une syntaxe et une sémantique de la langue en vue d'en extraire une ou plusieurs phrases en accord avec ces données linguistiques.

Les méthodes *intégrées* [Ney84, Jouvet88, Godin90] font interagir les différentes connaissances plus directement. Une source de connaissance d'un niveau donné impose des contraintes au niveau inférieur, restreignant ainsi l'ensemble des hypothèses envisageables à ce niveau. Lors de la transformation de la séquence de vecteurs acoustiques en une séquence de phonèmes, les contraintes lexicales imposent aux séquences de phonèmes produites l'obligation de constituer des mots du vocabulaire. On réalise donc l'intégration des niveaux phonétique et lexical. Poussant la démarche plus loin, les contraintes syntaxiques obligent les séquences de phonèmes (formant effectivement des séquences de mots, de par les contraintes lexicales) à correspondre à une structure grammaticale admise. Partant des vecteurs acoustiques, l'intégration des niveaux phonétique, lexical et syntaxique permet de produire, en une seule passe, une suite de modèles de phonèmes formant une phrase syntaxiquement correcte. La même démarche peut être appliquée dans le but d'intégrer également une connaissance sémantique.

Les systèmes analytiques et intégrés se distinguent également sur un autre point. Les premiers utilisent des connaissances a priori à chacun des niveaux. C'est la raison pour laquelle, on parle habituellement de systèmes à base de règles (en général, non apprises). Les systèmes intégrés évoqués ci-dessus reposent sur une modélisation acoustique statistique[5]. Ce type de modélisation permet de considérer les sources de connaissances des autres niveaux comme des contraintes agissant sur le niveau acoustique. Les systèmes intégrés reposant sur une modélisation acoustique statistique sont encore appelés *globaux*[6] [Jouvet88].

L'avantage de l'approche *intégrée* sur l'approche *analytique* est d'incorporer dès que possible plusieurs sources de connaissance. Les niveaux supérieurs de connaissance guident la reconnaissance des unités acoustiques de base au lieu d'essayer de corriger a posteriori les erreurs commises. C'est la raison pour laquelle les systèmes conçus dans cette optique ont atteint des taux de reconnaissance sensiblement meilleurs que ceux basés sur une approche

[5]Ces distinctions ne sont pas toujours nettes. Par exemple, une technique statistique basée sur des fonctions de discrimination linéaire a été inclue dans le décodage acoustico-phonétique du système analytique KEAL [Mercier et al.89]. Par ailleurs, des règles phonologiques ont été utilisées pour la définition des unités phonétiques contextuelles dans le système intégré PHIL90 [Jouvet et al.91].

[6]Cette dénomination ne doit pas être confondue avec celle de Pierrel [Pierrel82] qui désigne comme globaux des systèmes, antérieurs aux systèmes analytiques, n'ayant pas pour but une analyse "fine" du signal vocal.

analytique.

Les paragraphes suivants présentent quelques aspects du traitement acoustique et de l'intégration des niveaux phonétique et lexical.

2.1.3 Le traitement acoustique

Pour obtenir une représentation numérique, on échantillonne le signal acoustique de parole. La fréquence d'échantillonnage est choisie conformément au théorème d'échantillonnage (au-delà de 8 kHz pour un signal dont la bande passante aurait été limitée par la qualité téléphonique).

Le signal acoustique est aléatoire et non stationnaire, c'est-à-dire que ses moments statistiques dépendent de l'intervalle de temps sur lequel ils sont observés. En d'autres termes, le mécanisme de production du signal de parole varie dans le temps. Pour cette raison, on divise le signal en fenêtres d'analyse, ou *trames,* de parole de longueur fixe[7] durant lesquelles il peut être considéré comme stationnaire. A partir des échantillons de chaque trame, un traitement numérique extrait des coefficients caractéristiques qui sont rassemblés en un vecteur descriptif de cette tranche, désigné sous le nom de *vecteur acoustique.* Ces coefficients représentent, par exemple, l'énergie du signal acoustique dans différentes bandes de fréquence. La définition et le nombre des coefficients diffèrent selon les systèmes de reconnaissance : on rencontre principalement les paramètres LPC[8], les coefficients cepstraux et spectraux [Rabiner et al.78].

De multiples variantes des coefficients acoustiques ont été étudiées. Les coefficients mentionnés ci-dessus correspondent à des mesures instantanées du signal, c'est-à-dire qu'ils sont calculés sur une trame donnée. Par ailleurs, on prend souvent en compte des estimations des dérivées de ces coefficients. Ces dérivées sont estimées par différences finies sur quelques trames voisines de la trame courante. Les *vecteurs acoustiques* sont alors constitués d'un ensemble de paramètres instantanés et différentiels. L'ajout des paramètres différentiels a permis d'améliorer sensiblement les performances de reconnaissance, en rendant compte de la caractéristique dynamique du signal de parole [Furui86, Lee88].

---

[7]La durée des fenêtres d'analyse est typiquement de l'ordre de 20 à 30 ms. Dans la mesure où il y a un recouvrement des fenêtres, le décalage entre les trames est de l'ordre de 10 à 20ms.

[8]L'abréviation **LPC** provient de la terminologie anglaise pour “Linear Predictive Coding”.

2.1.4 Le décodage acoustique

Nous donnons ici un premier aperçu de l'opération centrale d'un système de reconnaissance. Les différentes notions et techniques évoquées seront détaillées dans les paragraphes suivants. Plaçons-nous dans le cadre d'une approche intégrant les niveaux phonétique et lexical. Dans cette approche intégrée, le *décodage acoustique* concerne la transformation d'une séquence de vecteurs acoustiques, issus du traitement acoustique, en une séquence de mots. On admet, en outre, que les modèles de référence, ou *modèles acoustiques*, sont des modèles de phonèmes ; ce choix s'impose dès que le vocabulaire du fragment de la langue à reconnaître dépasse quelques dizaines de mots. Interrogeons-nous tout d'abord sur les caractéristiques du signal de parole dont il faudra tenir compte lors de l'estimation des modèles.

Le signal de parole présente une variabilité élevée : un même locuteur utilise différentes vitesses d'élocution et diverses intonations, l'environnement acoustique (bruit de fond, effet du canal, etc) peut aussi varier d'un moment à l'autre. Cette variabilité du signal de parole suggère l'utilisation de modèles incluant des paramètres statistiques. Une approche intéressante à ce problème de modélisation est basée sur les chaînes de Markov [Bahl et al.75, Baker75, Jelinek76, Levinson et al.83].

Un exemple typique de chaîne de Markov, appelé modèle de Bakis, est présenté à la figure 2.1. Elle est formée d'une suite finie d'états et les séquences d'états admises sont définies par l'ensemble des transitions complétant le modèle. Dans l'exemple considéré, trois types de transitions sont possibles : le bouclage sur un état (donnant lieu à la répétition de cet état), le passage à l'état suivant et le passage vers le second état suivant (donnant lieu à l'omission d'un état)[9]. Une séquence de transitions de l'état initial à l'état final est appelée un *chemin*. Une chaîne de Markov est généralement utilisée comme modèle de phonème. Un modèle de mot est alors obtenu par la concaténation des modèles des phonèmes constituant ce mot (figure 2.2).

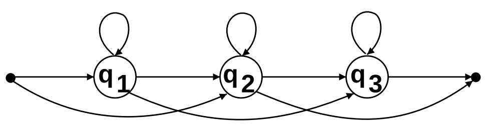


FIG. 2.1: Chaîne de Markov.

[9]Pour une modélisation par une chaîne de Markov, il n'y a pas à proprement parler de répétition ou d'omission d'état, puisqu'il n'existe pas de chemin prédéfini à travers la chaîne. Nous avons conservé ces termes par analogie avec l'alignement sur des formes acoustiques.

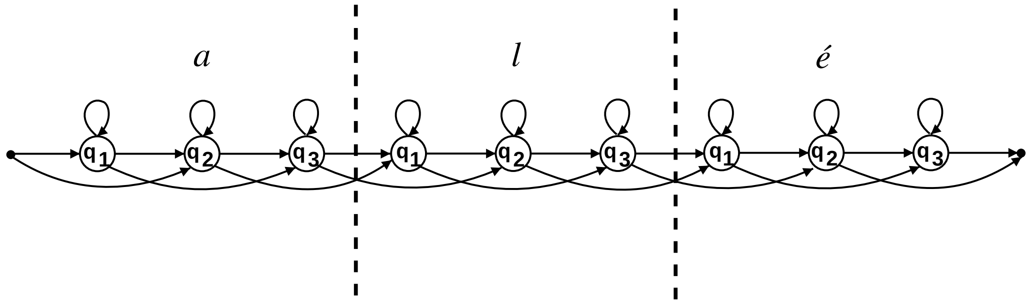


FIG. 2.2: Modèle du mot "aller" : a , l , é.

En vue de rendre compte des variations d'intonation, on associe une probabilité d'émission à chaque état ou à chaque transition[10]. Un état atteint, respectivement une transition empruntée, émettra donc aléatoirement un vecteur acoustique selon une distribution de probabilité spécifique - par exemple, une distribution gaussienne [Jelinek76].

En outre, il faudra pouvoir rendre compte des différentes vitesses d'élocution. C'est la raison pour laquelle on attachera à chaque transition de la chaîne une probabilité de passage par cette transition. Un chemin comportant une majorité de répétitions d'état correspondra à une prononciation lente. Inversement, un chemin comportant une majorité d'omissions d'état correspondra à une prononciation rapide. Associant une chaîne de Markov à chaque phonème, le chemin le plus court, c'est-à-dire celui constitué du plus petit nombre d'états visités, définit la durée minimale imposée du phonème.

Le modèle de Markov est donc caractérisé par deux ensembles de paramètres : les paramètres des probabilités d'émission qui rendent compte de la variabilité dans l'intonation et les probabilités de transition qui formalisent la variabilité dans la vitesse d'élocution. Cette modélisation soulève deux types de problèmes. Il s'agit tout d'abord d'estimer les valeurs optimales des paramètres qui seront incorporés au modèle. Il s'agit ensuite d'utiliser le modèle ainsi estimé en vue de reconnaître - autrement dit, de décoder - une émission inconnue.

L'estimation des paramètres se fait par la voie d'une *procédure d'entraînement* portant sur de nombreuses versions du phonème. Ces différentes versions doivent refléter au mieux les variations d'intonation et de vitesse d'élocution. On obtiendra alors un modèle statistiquement représentatif des diverses réalisations possibles d'un phonème. A cette fin, on utilise un en-

[10]Les modèles avec probabilités d'émission attachées aux transitions sont décrits dans [Jelinek76, Jouvet88, Lee88].

semble de phrases constituant la *base d'entraînement*. La séquence de mots - et donc de phonèmes - de chaque phrase est connue. Partant de cette information et d'une initialisation préalable des modèles (voir paragraphe 2.2.4), on réestime les paramètres par un *algorithme d'entraînement*. On distingue généralement l'algorithme de *Viterbi* [Forney73, Brown et al.83] et l'algorithme de *Baum-Welch* [Baum72, Liporace82]. Chacun d'eux est basé sur une opération qui maximise un certain critère, appelé *critère d'entraînement* (voir paragraphes 2.2.2 et 2.2.3).

Quel que soit le critère retenu pour l'entraînement des modèles, la base d'entraînement doit être suffisamment étendue pour permettre une estimation correcte des paramètres des modèles. Les systèmes de reconnaissance *dépendants du locuteur* [Lowerre76] disposent de modèles spécifiques à chaque locuteur. Il s'agit d'une limitation importante, car chaque nouveau locuteur doit effectuer une phase d'entraînement des modèles acoustiques. Par contre, la base d'entraînement peut être plus restreinte car elle ne doit rendre compte de la variabilité du signal de parole que pour un seul locuteur.

Pour pouvoir estimer des modèles acoustiques *indépendants du locuteur*, il est primordial de disposer d'une base d'entraînement reflétant au mieux la variabilité de la parole sur plusieurs locuteurs [Lee88, Jouvet88, Murveit et al.88]. En règle générale, plusieurs centaines de locuteurs permettent de constituer une telle base d'entraînement. Ceci fait, le système est, en principe, capable de reconnaître correctement n'importe quel locuteur. Dans les systèmes de reconnaissance *indépendants du locuteur* les modèles acoustiques sont non seulement entraînés sur une base de données plus grande, mais en outre ils sont souvent plus sophistiqués - le nombre de leurs paramètres est plus élevé.

Les systèmes de reconnaissance avec *adaptation vis-à-vis du locuteur* [Schwartz et al.87, Lee et al.91, Leggetter et al.94] réalisent un compromis intéressant entre les deux systèmes précédents. L'adaptation vis-à-vis du locuteur suppose un ensemble de paramètres préalablement entraînés. Ces paramètres sont alors ajustés sur quelques phrases prononcées par le nouveau locuteur.

Disposant d'un certain nombre de modèles de référence correspondant aux phonèmes, le problème de la reconnaissance d'un signal de parole consiste à trouver la séquence de modèles qui correspond le mieux au signal. En d'autres termes, il s'agit de maximiser la probabilité de produire une séquence d'états des modèles de Markov étant donné une séquence de vecteurs acoustiques (voir paragraphe 2.2.5). La probabilité d'une séquence d'états, c'est-à-dire un chemin, est le produit des probabilités de transition et d'émission rencontrées. Le chemin optimal est celui de plus grande probabilité et peut s'obtenir par un algorithme de programmation dynamique [Bellman57] appliqué à un formalisme stochastique [Forney73]. Cet algorithme est, dans ce cadre, souvent appelé *décodage de Viterbi* [Bahl et al.83]. Le chemin optimal définit une séquence d'états à laquelle correspond une séquence de phonèmes et donc de mots.

2.2 Les modèles de Markov

Le but des paragraphes suivants est de décrire quelques techniques générales utilisées en reconnaissance de la parole, en particulier dans les systèmes intégrés, et de présenter de manière plus détaillée le *décodage acoustique* introduit au paragraphe 2.1.4. De nombreuses variantes, extensions ou adaptations de ces techniques sont encore étudiées. Nous nous sommes limités ici à donner au lecteur une vision générale, illustrative des systèmes de reconnaissance actuels, afin de pouvoir aborder les problèmes soulevés par l'utilisation d'un modèle de langage.

Le décodage acoustique a pour objet l'obtention d'une séquence de mots à partir d'une séquence de vecteurs acoustiques qui résultent d'un premier traitement du signal de parole. Cette opération de décodage repose sur un ensemble de modèles acoustiques qui utilisent une représentation en chaînes de Markov. Dans le type de décodage que nous allons développer, nous associons, à chaque phonème de la langue considérée, un modèle acoustique qui lui est propre.

Nous introduisons au paragraphe 2.2.1 un exemple typique de modèle de Markov et nous y définissons les paramètres qui le caractérisent. Ceux-ci doivent être estimés au moyen d'une procédure d'entraînement. Cette procédure est une technique d'apprentissage automatique des paramètres qui est détaillée au paragraphe 2.2.4.

Les modèles acoustiques étant définis et leurs paramètres estimés, nous pouvons procéder au décodage acoustique. Nous montrons au paragraphe 2.2.5 comment résoudre ce problème par un algorithme classique dû à Bellman, la *programmation dynamique* [Bellman57], que nous appliquons ici à la reconnaissance de la parole.

2.2.1 Définitions

En vue de rendre compte de la variabilité du signal de parole, on a recours à des modèles acoustiques - généralement associés aux phonèmes - incluant des paramètres statistiques. Un exemple typique de modèle, basé sur une chaîne de Markov, a été présenté au paragraphe 2.1.4. Ces modèles acoustiques sont utilisés afin de reconnaître un signal de parole qui se présente sous la forme d'une séquence de vecteurs issus du traitement acoustique (voir paragraphe 2.1.3). A chaque trame du signal est associé un vecteur acoustique.

Disposant des chaînes de Markov, d'une part, et des vecteurs acoustiques, d'autre part, nous allons voir comment définir formellement les paramètres des modèles acoustiques, c'est-à-dire les probabilités d'émission d'un vecteur acoustique et les probabilités de transition. Dans ce but, nous introduisons les notations suivantes :

- Soit x_n le vecteur acoustique observé au temps n, où $1 \leq n \leq N$; N est le nombre de trames du signal de parole.

  La séquence $(x_1, x_2, \ldots, x_n)$ des n premiers vecteurs acoustiques est notée $X_{1 \ldots n}$.
- Soit $\{q_i | i = 1, 2, \ldots, S\}$ un ensemble de S états d'une chaîne de Markov, q_I l'état initial de la chaîne et q_F l'état final ; q_i^n représente l'état q_i visité au temps n.

  La séquence $(q_I^0, q_{i_1}^1, q_{i_2}^2, \ldots, q_{i_n}^n)$ d'états est notée $Q_{1 \ldots n}$.

A un instant n donné, la probabilité de visiter l'état q_l et d'observer le vecteur acoustique x_n, est donnée par[11] :

$$P(q_l^n, x_n | X_{1 \ldots n-1}, Q_{1 \ldots n-1}). \tag{2.1}$$

Cette probabilité, appelée *contribution locale*, peut dépendre de $X_{1 \ldots n-1}$, la séquence de vecteurs observés précédemment, et de $Q_{1 \ldots n-1}$, la séquence d'états visités antérieurement. On admettra, cependant, un certain nombre d'hypothèses simplificatrices.

La contribution locale est supposée indépendante des vecteurs observés précédemment. Dès lors, l'expression (2.1) s'écrit :

$$P(q_l^n, x_n | Q_{1 \ldots n-1}). \tag{2.2}$$

Cette expression se décompose en un produit de deux facteurs :

$$P(q_l^n, x_n | Q_{1 \ldots n-1}) = P(q_l^n | Q_{1 \ldots n-1}).P(x_n | q_l^n, Q_{1 \ldots n-1}). \tag{2.3}$$

Le premier facteur représente la probabilité de transition vers l'état q_l qui, pour *un modèle de Markov du premier ordre*, ne dépend que de l'état antérieur ; ce premier facteur peut donc s'écrire sous la forme :

$$P(q_l^n | q_k^{n-1}). \tag{2.4}$$

[11] Si A et B désignent deux variables aléatoires, la probabilité que la variable A prenne la valeur a étant donnée que la variable B a la valeur b s'écrit généralement $P(A = a | B = b)$. Dans la suite de ce document, nous utilisons la notation simplifiée $P(a|b)$, les variables aléatoires étant sous-entendues.

Comme, en outre, le modèle est supposé *stationnaire*, c'est-à-dire indépendant du temps, on peut remplacer l'expression (2.4) par :

$$P(q_l|q_k) \tag{2.5}$$

où q_l et q_k sont deux états visités à des instants consécutifs (indépendamment de ces instants eux-mêmes).

Le second facteur de l'expression (2.3) représente la probabilité d'émission du vecteur x_n conditionnée par le fait que l'on a visité l'état q_l à l'instant n. Cette probabilité est supposée indépendante de cet instant et de la séquence d'états précédents. Elle peut donc s'écrire sous la forme :

$$P(x_n|q_l). \tag{2.6}$$

Les états q_I et q_F sont non émetteurs - aucun vecteur acoustique ne peut être émis sur ces états. Ils identifient simplement l'entrée et la sortie du modèle.

Les hypothèses simplificatrices d'un modèle de Markov stationnaire du premier ordre (figure 2.3) conduisent à la définition de deux ensembles de paramètres : les probabilités de transition entre deux états visités à des instants consécutifs (2.5) et les probabilités d'émission sur un état visité (2.6). Ce type de modèle est appelé “modèle de Markov caché”, ou encore *HMM*[12]. En effet, bien que les vecteurs acoustiques soient observables, les états des modèles ne le sont pas ; seule la probabilité d'observer un vecteur pour chaque état est connue.

Si une chaîne de Markov est associée à chaque phonème, un modèle de mot s'obtient simplement par concaténation des différents modèles des phonèmes dont le mot est constitué (voir paragraphe 2.1.4, figure 2.2). Une séquence de transitions de l'état q_I à l'état q_F constitue un *chemin*. Ce chemin représente une réalisation possible du mot. La *longueur* d'un chemin est le nombre de vecteurs acoustiques émis en parcourant ce chemin.

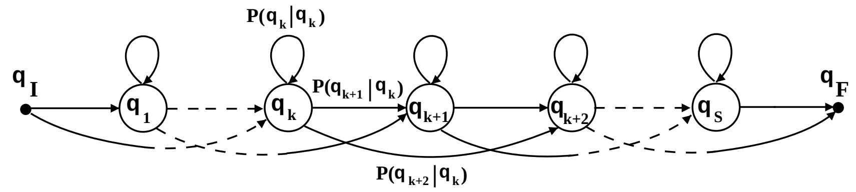


FIG. 2.3: Modèle de Markov du premier ordre.

[12]L'abréviation **HMM** pour “Hidden Markov Model” provient de la terminologie anglaise.

2.2.2 Le critère du maximum de vraisemblance

Voyons comment utiliser les modèles de Markov pour reconnaître un signal de parole correspondant à un mot. Il s'agit de traduire une séquence $X_{1 \dots N}$ de vecteurs acoustiques en une séquence d'états représentant ce mot.

Considérons un lexique de M mots $W^1, W^2, \dots, W^M$. La séquence X de vecteurs acoustiques correspondra à une réalisation du mot W^i si la probabilité $P(W^i|X)$ est maximale, c'est-à-dire si :

$$P(W^i|X) \geq P(W^k|X) \ , \ \forall k = 1, \dots, M. \tag{2.7}$$

Cette probabilité *a posteriori* du mot W^i, la séquence X étant donnée, peut être calculée en utilisant la règle de Bayes :

$$P(W^i|X) = \frac{P(X|W^i).P(W^i)}{P(X)}. \tag{2.8}$$

L'expression $P(W^i)$ est la probabilité *a priori* du mot W^i ; elle satisfait la condition $\sum_{i=1}^{M} P(W^i) = 1$. Cette probabilité peut dépendre du modèle de langage considéré. Si nous supposons tous les mots équiprobables, l'expression $P(W^i)$ n'intervient pas dans l'inégalité (2.7). Par ailleurs, pour une séquence donnée de vecteurs acoustiques, la probabilité $P(X)$ est une constante. En conséquence, la condition (2.7) revient à maximiser l'expression $P(X|W^i)$ qui est la probabilité d'engendrer une séquence X de vecteurs en prononçant le mot W^i. L'expression $P(X|W^i)$ est encore appelée *vraisemblance* des données X relativement au modèle du mot W^i. Cette probabilité peut s'obtenir aisément à partir du modèle de mot que nous nous sommes fixés.

Soit $C = Q_{1 \dots N} = (q_I^0, q_{i_1}^1, q_{i_2}^2, \dots, q_{i_N}^N, q_F^{N+1})$ un chemin de longueur N à travers le modèle du mot W^i, avec $1 \leq i_1 \leq i_2 \leq \dots \leq i_N \leq S$. La probabilité de produire la séquence $X_{1 \dots N}$ le long du chemin $Q_{1 \dots N}$ est donnée par le produit des probabilités d'émission et de transition rencontrées. En effet, en utilisant les hypothèses simplificatrices mentionnées ci-dessus, nous obtenons successivement :

$$\begin{aligned} P(X_{1 \dots N}, Q_{1 \dots N}) &= P(Q_{1 \dots N}).P(X_{1 \dots N}|Q_{1 \dots N}) \\ &= \prod_{n=1}^{N+1} P(q_{i_n}|q_{i_{n-1}}). \prod_{m=1}^{N} P(x_m|q_{i_m}), \end{aligned} \tag{2.9}$$

où $i_0 = I$ et $i_{N+1} = F$.

Soit $\{C_1, \dots, C_K\}$ l'ensemble des K chemins de longueur N à travers le modèle du mot W^i.

Nous pouvons obtenir la probabilité $P(X|W^i)$ au moyen de la relation suivante :

$$P(X|W^i) = P(\{(X, C_1) \vee \ldots \vee (X, C_K)\}). \tag{2.10}$$

Si les chemins sont mutuellement exclusifs, c'est-à-dire si

$$P(C_p, C_q) = 0 \text{ pour } p \neq q, \tag{2.11}$$

nous pouvons écrire :

$$P(X|W^i) = \sum_{p=1}^{K} P(X_{1\ldots N}, C_p) = \sum_{p=1}^{K} P(X_{1\ldots N}, Q_{1\ldots N}), \tag{2.12}$$

où la sommation porte sur tous les chemins de longueur N à travers le modèle du mot W^i. Le membre de droite de cette égalité est calculable récursivement. La probabilité d'atteindre à l'instant $n+1$ l'état q_i, compte tenu qu'on a déjà émis précédemment les n vecteurs $X_{1\ldots n}$ est donnée par :

$$P(q_i^{n+1}, X_{1\ldots n}) = \sum_{j=1}^{S} P(q_j^n, X_{1\ldots n-1}).P(x_n|q_j).P(q_i|q_j)\,, \quad \forall n = 1, \ldots, N. \tag{2.13}$$

Les probabilités intervenant dans l'expression (2.13) sont soumises aux contraintes suivantes :

$$\begin{aligned} P(q_i^1, X_{1\ldots 0}) &= P(q_i|q_I) : , \quad \forall i = 1, \ldots, S, \\ P(q_F^{N+1}, X_{1\ldots N}) &= P(X|W^i). \end{aligned}$$

La sommation dans l'expression (2.13) porte sur les S états du modèle. Cependant, le facteur $P(q_i|q_j)$ est nul pour tous les états non reliés par une transition. Dès lors, tous les chemins possibles - et uniquement ceux-là - sont pris en compte. L'équation (2.13) est connue sous l'appellation anglaise de "*forward recurrence*" de *Baum-Welch.*

Considérons, à la figure 2.4, les états $q_{j_1}, \ldots, q_{j_l}$ reliés par une transition à un état q_i donné (dans notre exemple, $q_{j_l} = q_i$). Tenant compte des émissions du vecteur x_n sur les états $q_{j_1}, \ldots, q_{j_l}$ et des transitions des états $q_{j_1}, \ldots, q_{j_l}$ vers l'état q_i, on somme toutes les contributions pour obtenir la probabilité d'atteindre cet état q_i à l'instant $n+1$.

En fonction du *critère du maximum de vraisemblance*, le mot reconnu est celui pour lequel la probabilité $P(X|W^i)$ est maximale. Ce critère revient, via l'expression (2.8) et les hypothèses mentionnées, à maximiser la probabilité *a posteriori* (équation 2.7).

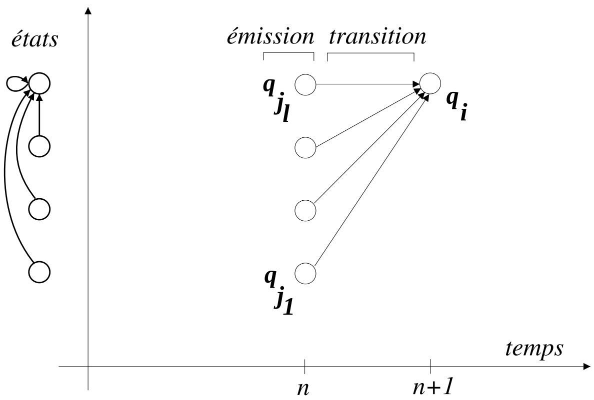


FIG. 2.4: Récurrence de Baum-Welch.

2.2.3 Le critère de Viterbi

Un version simplifiée du *critère du maximum de vraisemblance* est connue sous le nom de *critère de Viterbi*. Ce critère retient seulement le chemin le plus probable - au lieu de tous les chemins de longueur N, comme exprimé dans la relation (2.12) - dans chaque modèle de mot pour le calcul de la probabilité d'observer les vecteurs X en prononçant le mot W^i. Une conséquence importante de cette simplification est que la séquence de mots reconnue ne correspond pas nécessairement à celle de vraisemblance maximale mais correspond à la séquence d'états de probabilité maximale. Nous écrivons donc formellement :

$$P(X|W^i) \cong P(X, \hat{c}|W^i) = \max P(X_{1 \ldots N}, Q_{1 \ldots N}). \tag{2.14}$$

Le chemin $\hat{c}$, appelé *chemin optimal*, est celui de probabilité maximale et s'obtient par la récurrence :

$$P(q_i^{n+1}, X_{1 \ldots n}) = \max_{j=1,\ldots,S} \Big(P(q_j^n, X_{1 \ldots n-1}).P(x_n|q_j).P(q_i|q_j) \Big) \ , \ \forall n = 1, \ldots, N. \tag{2.15}$$

Cette équation constitue la base de l'algorithme de Viterbi qui sera abordé plus en détail au paragraphe 2.2.5.

Dans la méthode illustrée à la figure 2.5, au lieu de sommer les différentes contributions (comme on l'a fait à la figure 2.4), on ne retient que la contribution maximale. Ceci définit, parmi les états $q_{j_1}, \ldots, q_{j_l}$, l'état optimal q_{opt} associé à cette contribution maximale. La succession de ces états pour les différents instants constitue un chemin. Parmi l'ensemble des chemins passant à travers le modèle du mot W^i, celui de probabilité maximale à l'instant N est le *chemin optimal*.

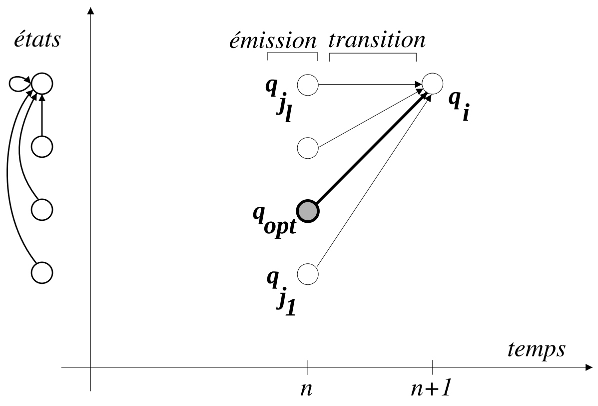


FIG. 2.5: Récurrence de Viterbi.

Finalement, on associe à chaque instant n l'état q_{opt} du chemin optimal. Le début d'un phonème correspondra à l'instant auquel le chemin optimal passe par le premier état appartenant au modèle de ce phonème. La *segmentation* en phonèmes du signal de parole, c'est-à-dire la division temporelle du signal en segments associés à des phonèmes, ainsi que l'*étiquetage*, c'est-à-dire l'identification de chaque segment à un phonème, résultent donc du chemin optimal obtenu grâce à la récurrence (2.15).

Notons que la reconnaissance d'une séquence de mots - au lieu d'un mot isolé - nécessite une généralisation de la théorie présentée ci-dessus. La formulation présentée ci-dessus reste correcte pour peu que W^i désigne un modèle, obtenu par la concaténation des modèles des mots de la séquence. La probabilité a priori $P(W^i)$ de ces séquences dépend du modèle de langage (voir chapitre 3).

2.2.4 Apprentissage des modèles acoustiques

Principes

Nous avons décrit au paragraphe 2.2.1 un type de modèles acoustiques et nous avons montré comment nous pouvions reconnaître des mots isolés à l'aide de ces modèles. La reconnaissance d'un signal de parole continue sera abordée plus en détail au paragraphe 2.2.5. Voyons à présent comment estimer les valeurs optimales des paramètres qui seront incorporés aux modèles acoustiques. Il s'agit ici d'estimer les probabilités de transition et les paramètres des probabilités d'émission associées à chaque modèle de phonème. Le contenu de ce paragraphe n'est pas indispensable à la compréhension de la suite.

Considérons, par exemple, une probabilité d'émission sur l'état q définie par une distribution multivariée, continue et gaussienne [Jelinek76]. Les paramètres associés à cette distribution sont le vecteur moyen μ et la matrice de covariance Σ. La probabilité d'émission est donnée par :

$$P(x|q) = \frac{1}{\sqrt{(2\pi)^d.|\Sigma|}}.\exp\left[-\frac{(x-\mu)^T\,\Sigma^{-1}\,(x-\mu)}{2}\right], \tag{2.16}$$

où $|\Sigma|$ désigne le déterminant de la matrice Σ et où d est la dimension du vecteur x. Sous l'hypothèse restrictive supplémentaire d'une matrice de covariance diagonale, la distribution multivariée devient un produit de distributions univariées :

$$P(x|q) = \prod_{l=1}^{d} \frac{1}{\sqrt{2\pi}.\sigma^l}.\exp\left[-\frac{1}{2}\left(\frac{x^l-\mu^l}{\sigma^l}\right)^2\right], \tag{2.17}$$

où x^l, μ^l et $(\sigma^l)^2$ désignent respectivement la *l-ième* composante de x et μ et le *l-ième* élément diagonal de Σ.

Dans notre exemple, les paramètres à estimer sont donc, d'une part, les probabilités de transition entre deux états reliés dans le modèle et, d'autre part, le vecteur moyen et la matrice de covariance des probabilités d'émission sur chaque état. Nous disposons à cette fin d'une *base d'entraînement*, c'est-à-dire d'un ensemble de phrases prononcées par un ou plusieurs locuteurs (voir paragraphe 2.1.4). Les phrases d'entraînement contiennent différentes réalisations de chaque phonème en vue d'obtenir un modèle statistiquement représentatif de ce phonème. Pour chaque phrase d'entraînement, nous connaissons la séquence de mots, et donc de phonèmes, et nous disposons de la séquence des vecteurs acoustiques émis par le locuteur lors de la prononciation de cette phrase.

- Soit W, la séquence des phonèmes constituant une phrase d'entraînement.
- Soit $X = (x_1, x_2, \ldots, x_N)$ la séquence des vecteurs acoustiques correspondant à une prononciation de cette phrase.
- Soit θ l'ensemble des paramètres des modèles.

Il s'agit de trouver l'estimation $\hat{\theta}$ de θ qui maximise la probabilité de la séquence de phonèmes W, étant donné X et les paramètres θ. L'on a :

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(W|X,\theta). \tag{2.18}$$

Par la règle de Bayes, on obtient :

$$P(W|X,\theta) = \frac{P(X|W,\theta).P(W)}{P(X|\theta)}, \tag{2.19}$$

où la probabilité *a priori* de W est supposée indépendante des paramètres θ. Si l'on utilise comme *critère d'entraînement* le critère du maximum de vraisemblance (voir paragraphe 2.2.2), l'équation (2.18) devient :

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(X|W,\theta). \tag{2.20}$$

Ayant initialisé les paramètres θ (voir ci-après), une *procédure itérative d'entraînement* est utilisée afin d'obtenir les valeurs optimales de ces paramètres.

Une procédure possible repose sur la construction de tous les chemins de longueur N à travers le modèle de W en vue de réestimer les paramètres θ. Cette procédure n'est autre que l'algorithme de Baum-Welch basé notamment sur la récurrence (2.13). Cet algorithme est applicable à des distributions de probabilité continues [Liporace82, Bourlard et al.85a] comme à des distributions discrètes [Rabiner et al.83, Lee88].

L'apprentissage Viterbi

Nous nous limitons ici à la procédure d'entraînement de Viterbi issue du critère simplifié (2.14). La réestimation des paramètres θ résulte, dans ce cas, de la récurrence de Viterbi (2.15) définissant le chemin optimal à travers le modèle d'une séquence de phonèmes W qui constitue un mot ou une phrase.

- Soit J le nombre de phrases d'entraînement.
- Soit W^j, pour $j = 1, \ldots, J$, la séquence des phonèmes de la *j-ième* phrase d'entraînement.
- Soit X^j, pour $j = 1, \ldots, J$, la séquence de vecteurs acoustiques de la *j-ième* phrase d'entraînement.

Pour chaque W^j, nous déterminons par la récurrence de Viterbi le chemin le plus probable à travers le modèle de W^j. Soit $P(X^j|W^j,\theta)$, la probabilité de ce chemin optimal. Il faut trouver la valeur optimale $\hat{\theta}$ de θ telle que l'expression

$$\prod_{j=1}^{J} P(X^j|W^j,\theta) \tag{2.21}$$

soit maximale. La procédure d'entraînement Viterbi permet de converger vers un optimum local des valeurs des paramètres θ. Chaque itération de cette procédure a lieu en deux temps.

Dans un premier temps, nous déterminons le chemin optimal sur base des anciennes valeurs des paramètres. La segmentation résultant du chemin optimal définit l'état atteint à chaque instant.

Dans un second temps, nous utilisons la segmentation obtenue pour réestimer les paramètres :

- Soit $n_{k,k+i}$, pour $i = 0, 1, 2$, le nombre d'occurrences de la transition de l'état q_k à l'état q_{k+i} rencontrées au cours des chemins optimaux sur l'ensemble des phrases d'entraînement (nous nous plaçons dans le cadre des modèles présentés à la figure 2.3 pour lesquels au plus trois transitions sont issues de chaque état).
- Soit $N_k = n_{k,k} + n_{k,k+1} + n_{k,k+2}$.
- Soit n_{q_k} le nombre de vecteurs acoustiques x_m émis sur l'état q_k le long des chemins optimaux.

Les nouvelles valeurs des paramètres sont données par les expressions

$$\begin{aligned} P(q_{k+i}|q_k) &= \frac{n_{k,k+i}}{N_k}, \\ \mu^l &= \sum_{m=1}^{n_{q_k}} \frac{x_m^l}{n_{q_k}}, \\ \sigma^l &= \sqrt{\sum_{m=1}^{n_{q_k}} \frac{(x_m^l - \mu^l)^2}{n_{q_k}}}, \end{aligned} \tag{2.22}$$

sous l'hypothèse d'une matrice de covariance diagonale pour la densité de probabilité d'émission associée à l'état q_k.

Disposant de ces nouvelles valeurs de paramètres, nous déterminons à nouveau les chemins optimaux pour les différentes phrases d'entraînement. Nous itérons cette procédure jusqu'à obtenir la stabilisation des valeurs des paramètres.

Voyons comment initialiser les paramètres des modèles. Partant d'une segmentation, nous pouvons estimer les paramètres au moyen des relations (2.22). Il s'agit donc de se donner une segmentation initiale.

La méthode la plus simple consiste à choisir une segmentation linéaire. Pour chaque phrase d'entraînement, nous connaissons la séquence des phonèmes, notée W^j. Notons qu'un phonème donné peut apparaître à plusieurs reprises dans la séquence W^j. Nous choisissons un nombre identique de vecteurs acoustiques émis sur chaque état de chaque phonème.

2.2.5 L'alignement Viterbi

Le décodage, ou *alignement*, Viterbi constitue la partie principale de l'algorithme de reconnaissance. Nous disposons d'une séquence X de vecteurs acoustiques qui doit être traduite en un mot, dans le cas de mots isolés, ou une séquence de mots, dans le cas de la parole continue.

Lors de la reconnaissance nous utilisons le critère de Viterbi (2.14). Dans le cas de mots isolés, nous pouvons calculer, pour chaque mot du vocabulaire, la probabilité de la meilleure séquence d'états de produire la séquence X (par la récurrence (2.15)). Le mot reconnu est celui pour lequel cette probabilité est maximale.

Voyons, à présent, comment reconnaître un signal de parole continue. Dans ce qui suit, W^i désigne une séquence de mots plutôt qu'un seul mot. La récurrence de Viterbi (2.15) maximise un produit de facteurs. Si nous décidons de minimiser une somme (chaque terme étant l'opposé du logarithme d'une probabilité), plutôt que de maximiser un produit, le problème peut être résolu par une *programmation dynamique* [Bellman57] :

$$-\log P(q_i^{n+1}, \quad X_{1\ldots n}) = \min_{j=1,\ldots,S} \Big(-\log P(q_j^n, X_{1\ldots n-1}) - \log P(x_n|q_j) - \log P(q_i|q_j)\Big), \tag{2.23}$$

où $n = 1, \ldots, N$.

L'expression

$$-\log P(X|W^i) = -\log P(q_F^{N+1}, X_{1\ldots N})$$

représente le *score de correspondance* de la séquence X avec la séquence d'états de W^i. A une probabilité maximale correspond donc un score (de correspondance) minimal.

Nous détaillons ci-après l'algorithme de programmation dynamique appliqué à la reconnaissance de la parole[13]. Introduisons les notations suivantes.

- Un *point* de la programmation dynamique est un triplet (n, j, k), où n est un instant et où j est l'indice d'un état appartenant au modèle du mot k. Le dernier état du mot k est noté $J(k)$ et nous supposons que le lexique contient M mots.

[13]Nous présentons ici l'algorithme "*One-Pass*" [Bridle et al.82, Ney84]. D'autres algorithmes, tels le "*Two-Level*" [Sakoe79] et le "*Level-Building*" [Myers et al.81], ont également été développés. Godin a proposé un cadre théorique unifiant ces 3 algorithmes [Godin et al.89, Godin90]. En particulier, Godin montre que, moyennant l'introduction de contraintes syntaxiques, une formulation synchrone du "Level-Building" et le "One-Pass" correspondent à un seul et même algorithme [Godin90].

- La *distance locale* du vecteur x_n, émis sur l'état q_j appartenant au modèle du mot k, est notée $d(n, j, k)$ et vaut $-\log P(x_n|q_j)$.
- La *pénalité de distorsion temporelle* de l'état q_j à l'état q_i (dans le modèle du mot k), est notée $T(j, i)$ et vaut $-\log P(q_i|q_j)$.

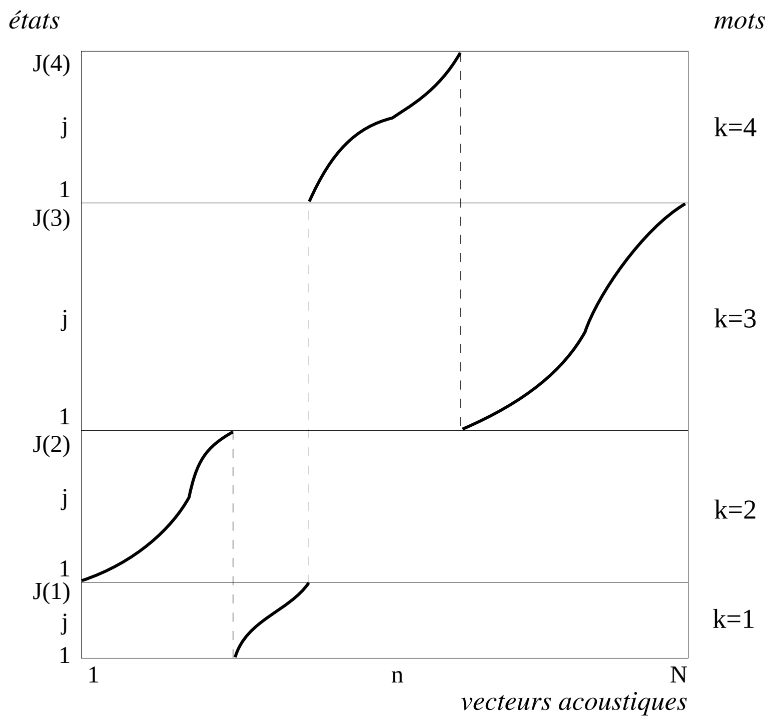


FIG. 2.6: Alignement entre vecteurs acoustiques et modèles de mots.

La programmation dynamique calcule le meilleur alignement entre la séquence de vecteurs acoustiques et la séquence d'états des modèles de mots [Ney84], c'est-à-dire le chemin optimal à travers les modèles de mot (voir figure 2.6). Ce chemin associe à chaque instant n un état particulier d'un modèle de mot. Tout chemin est défini par une séquence de *points* et nous nous proposons de trouver le meilleur chemin. Dans ce but, nous définissons la *distance cumulée* $D(n, i, k)$, qui est le score du meilleur chemin jusqu'au *point* (n, i, k). Nous calculons les distances cumulées jusqu'aux différents points de la programmation dynamique au moyen des deux récurrences (2.24) et (2.26) présentées ci-après. Ainsi, nous développons un ensemble de chemins partiels caractérisés chacun par une distance cumulée. La distance cumulée minimale à l'instant N (correspondant au dernier vecteur acoustique) identifie le dernier point du chemin optimal. Nous pouvons alors retrouver la séquence de points définissant ce chemin. Toutefois, nous ne nous intéressons généralement qu'à la séquence des mots associée au chemin optimal. A ce propos, nous introduisons les notations suivantes.

- La *table des mots* M, où $M(n)$ est défini comme le mot k dont la distance cumulée jusqu'au dernier état $J(k)$ est minimale à l'instant n.
- La *table des instants* I, où $I(n)$ est défini comme l'instant correspondant à la fin du mot précédant le mot $M(n)$.
- L'instant correspondant à la fin du mot à partir duquel le meilleur chemin partiel est arrivé au point (n, i, k) est noté $B(n, i, k)$.

Détaillons à présent les deux récurrences constitutives de la programmation dynamique.

Pour tenir compte des transitions entre états au sein d'un modèle de mot, nous définissons la distance cumulée par la formule de récurrence

$$D(n, i, k) = d(n, i, k) + \min_j (D(n-1, j, k) + T(j, i)), \tag{2.24}$$

et nous pouvons écrire

$$\begin{aligned} B(n, i, k) &= B(n-1, j', k), \\ \text{où } j' &= \operatorname*{argmin}_j (D(n-1, j, k) + T(j, i)). \end{aligned} \tag{2.25}$$

Ces équations sont évaluées à chaque instant n pour tous les points (n, i, k) qui vérifient les inégalités suivantes :

$$1 < i \leq J(k)\ ,\ 1 \leq k \leq M.$$

Par conséquent, si nous désignons par t_{ik} le nombre de transitions arrivant sur l'état i du modèle du mot k, et par T le nombre total de transitions du modèle, $T = \sum_{k=1}^{M} \sum_{i=1}^{J(k)} t_{ik}$, la complexité calculatoire de la récurrence (2.24) appliquée sur N trames est $\mathcal{O}(T.N)$.

La distance cumulée, ou le *score*, jusqu'au point (n, i, k) dépend de la *distance locale* et des *scores* obtenus à l'instant précédent pour tous les points $(n-1, j, k)$ correspondant à des états j reliés à l'état i (figure 2.7 (a)). On ne retient parmi ces scores que la contribution minimale. Celle-ci identifie le point précédent et définit, point par point, le meilleur chemin jusqu'au point (n, i, k) (voir aussi paragraphe 2.2.1 et figure 2.5). Nous construisons, à chaque instant et pour chaque point, un chemin partiel à travers les modèles. La récurrence (2.24) met en évidence le *principe d'optimalité* de la programmation dynamique : le chemin optimal est composé de chemins partiels qui sont eux-mêmes optimaux. En effet, la contribution minimale est calculée à chaque étape de la récurrence. Cette récurrence met également en évidence le caractère *gauche-droite* et *synchrone* du décodage Viterbi, puisque que l'on progresse par l'indice n, strictement croissant de 1 à N.

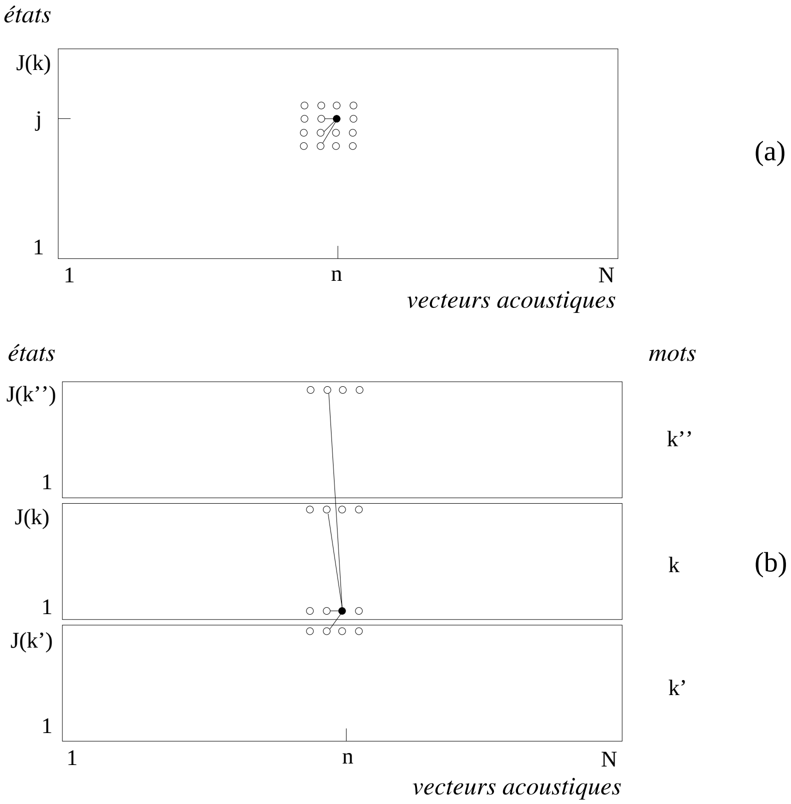


FIG. 2.7: Transitions au sein d'un mot (a) et transitions entre mots (b).

Pour tenir compte des transitions entre mots, nous définissons la distance cumulée jusqu'au début de chaque mot par la formule de récurrence

$$D(n,1,k) = d(n,1,k) + \min \left\{ \begin{array}{c} D(n-1,1,k) + T(1,1) \ ; \\ \min\limits_{k'=1,\ldots,M} \left(D(n-1,J(k'),k') + T(J(k'),k) \right) \end{array} \right\}. \tag{2.26}$$

Le terme $T(J(k'),k)$ désigne la pénalité associée à la transition du dernier état du mot k' au premier état du mot k. Elle dépend généralement d'un modèle de langage (voir chapitre 3). Dans la mesure où la récurrence (2.26) ne s'applique que sur les transitions entre mots, sa complexité calculatoire est asymptotiquement négligeable devant celle de la récurrence (2.24). La complexité globale de la programmation dynamique est donc $\mathcal{O}(T.N)$.

A nouveau, la récurrence (2.26) exprime le *principe d'optimalité* de la programmation dynamique puisque nous ne retenons que la contribution minimale. Si celle-ci est associée au point $(n-1,1,k)$, nous pouvons écrire

$$B(n,1,k) = B(n-1,1,k);$$

sinon, nous avons

$$B(n,1,k) = n-1. \tag{2.27}$$

Notons qu'en l'absence de contraintes syntaxiques, chaque mot peut être précédé de tout mot du lexique, y compris de lui-même (figure 2.7 (b)).

Après chaque application des récurrences (2.24) et (2.26), nous mettons à jour la *table des mots* et la *table des instants* comme suit :

$$\begin{aligned} M(n) &= \operatorname*{argmin}_k D(n,J(k),k) \ , \ 1 \le k \le M, \\ I(n) &= B(n,J(M(n)),M(n)). \end{aligned} \tag{2.28}$$

Nous appliquons les deux récurrences de la programmation dynamique à partir de l'instant 1 jusqu'à l'instant N. La distance cumulée minimale identifie le dernier état du chemin optimal. Nous déduisons la séquence des mots reconnus le long du chemin optimal à partir des deux tables définies ci-dessus. Cette opération, connue sous l'appellation anglaise de *backtracking*, peut être schématisée comme suit. L'indice i identifie les instants où le chemin optimal passe par des fins de mot.

Procédure backtracking

entrée $M_{1,\ldots,N}, I_{1,\ldots,N}$ // La table des mots et la table des instants
sortie S // La séquence de mots reconnus

début

```
S ← λ                              // λ désigne la chaîne vide
i ← N                              // i désigne l'instant courant
tant que (i > 0) faire
    S ← concaténation (M(i), S)
    i ← I(i)
fait
retourner S
fin
```

2.2.6 Contrôle de l'espace de recherche

A chaque instant, la programmation dynamique calcule les distances cumulées pour tous les points associés à chaque état de chaque modèle de mot du lexique. Ces points définissent différents chemins partiels à travers les modèles et constituent *l'espace de recherche*. En réalité, seul le chemin optimal nous intéresse, mais celui-ci n'est connu qu'en fin d'exécution de la programmation dynamique (la distance cumulée minimale associée au dernier échantillon du signal identifie le dernier point du chemin optimal).

Une stratégie "d'élagage" [Lowerre76, Ney et al.87] de l'espace de recherche repose sur l'observation que des chemins partiels qui ont un score largement supérieur au score minimal à un instant donné ne contribueront vraisemblablement pas au chemin optimal. C'est la raison pour laquelle on décide d'éliminer ces chemins de l'espace de recherche, à chaque instant. Après chaque application des récurrences (2.24) et (2.26), nous éliminons de l'ensemble des points actifs (c'est-à-dire de l'espace de recherche), ceux qui ont un score $D(n,i,k)$ dépassant, d'un certain seuil Δ, le score minimum $D_{min}(n)$ à cet instant :

$$(n,i,k) \notin \{\text{points actifs}\} \Leftrightarrow D(n,i,k) > D_{min}(n) + \Delta. \tag{2.29}$$

Cette comparaison a un sens car tous les chemins partiels à un instant donné ont un score basé sur la même portion du signal (la même séquence $X_{1\ldots n}$ de vecteurs acoustiques). C'est donc le caractère *synchrone* du décodage qui rend cette stratégie d'élagage aisée.

La première récurrence (2.24) de la programmation dynamique devient à présent :

$$D(n,i,k) = d(n,i,k) + \min_{\hat{\jmath}} \left(D(n-1,\hat{\jmath},k) + T(\hat{\jmath},i)\right), \tag{2.30}$$

où la minimisation porte sur tous les points *actifs* correspondant à des états $\hat{\jmath}$ reliés à l'état i.

La récurrence (2.26) est modifiée de manière similaire :

$$\begin{aligned} D(n,1,k) &= d(n,1,k) + \\ &\min \left\{ \begin{array}{c} D(n-1,1,k) + T(1,1)\ ; \\ \min\limits_{k'=1,\ldots,\hat{M}} \left(D(n-1,\hat{J}(k'),k') + T(\hat{J}(k'),k) \right) \end{array} \right\}. \end{aligned} \tag{2.31}$$

où la minimisation interne porte sur les derniers états $\hat{J}\ (k')$ des $\hat{M}$ mots pour lesquels ces points sont actifs[14].

Cette stratégie présente le risque d'une perte d'optimalité, c'est-à-dire de la suppression d'un chemin partiel dont le prolongement aurait conduit au chemin optimal. Par contre, elle s'avère indispensable, d'un point de vue pratique, pour diminuer la charge de calcul de l'algorithme (particulièrement dès que la taille du vocabulaire dépasse quelques centaines de mots).

Le nombre de chemins partiels retenus dépend des écarts entre les scores respectifs de ces chemins. Comme ces scores sont fonction des données, nous obtenons un élagage de l'espace de recherche guidé par les données. Tel que nous l'avons présenté cet élagage s'effectue sur base des scores dits *acoustiques,* c'est-à-dire le scores d'alignement entre les vecteurs acoustiques et les modèles de Markov, éventuellement combinés avec des *pénalités* introduites par le *modèle de langage* (termes $T(\hat{J}\ (k'),k)$). Nous pensons qu'il s'agit là d'une intégration idéale dans le décodage acoustique des connaissances syntaxiques, lexicales et acoustiques. Dans la mesure où le seuil Δ est généralement fixe, le nombre de points effectivement calculés fluctue au cours du décodage. En particulier, lorsque la combinaison des connaissances acoustiques, lexicales et syntaxiques ne permet pas une bonne discrimination (partie confuse du signal), un plus grand nombre d'états ont un score proche du score minimal et, par conséquent, l'espace de recherche effectif est augmenté. Cet ajustement dynamique de l'espace de recherche nous semble constituer un point essentiel de l'efficacité pratique de cet algorithme. Nous reviendrons sur ce point important au chapitre 4.

[14] Le terme $D(n-1,1,k)$ vaut $+\infty$ si le point $(n-1,1,k)$ est inactif.

Signalons que la mise en œuvre de cet algorithme proposée par Ney et ses coauteurs [Ney et al.87] permet de restreindre l'espace de recherche strictement aux points actifs sans devoir tester si chaque point est actif ou non. Disposant d'une liste des seuls points actifs à l'instant n, nous construisons une nouvelle liste de points actifs à l'instant suivant en trois opérations.

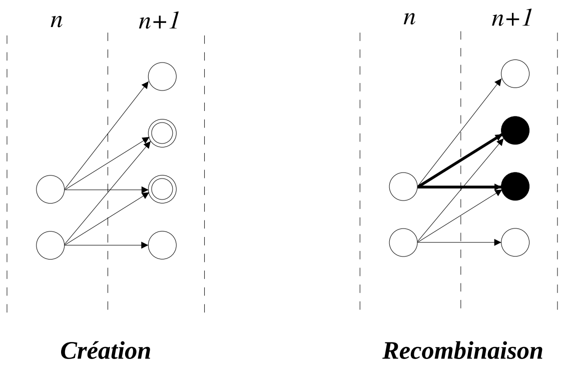


FIG. 2.8: Création et recombinaison des points.

Création de nouveaux points

Pour chaque point actif, nous déterminons l'ensemble des prolongements du chemin partiel à partir de ce point. En d'autres mots, nous établissons l'ensemble des points qui peuvent lui succéder en fonction des transitions possibles au sein d'un modèle acoustique. Lorsqu'un point actif correspond au dernier état d'un mot, c'est-à-dire qu'il s'agit d'un *nœud syntaxique*, le modèle de langage définit les transitions vers l'état initial d'autres mots, appelés *mots successeurs*. Nous calculons la distance cumulée relative à chaque point créé.

Recombinaison des points

Etant donné les transitions envisagées à l'étape de création, nous pouvons créer le même point à plusieurs reprises (ces points "multiples" sont représentés par des doubles cercles à la figure 2.8). Plusieurs scores sont relatifs à chaque point multiple. En vertu du principe du décodage de Viterbi, nous choisissons le score minimal associé à chaque point. Ce choix détermine la transition retenue, c'est-à-dire définit le chemin partiel jusqu'à ce nouveau point (figure 2.8). Nous pouvons alors mettre à jour les tables nécessaires à l'opération de *backtracking*. Remarquons que les opérations de création, recombinaison et mises à jour des tables s'effectuent, en réalité, simultanément. Pour chaque point que l'on crée, on vérifie s'il n'a pas déjà été créé, à la trame courante, auquel cas, si le dernier score trouvé est meilleur, les tables sont mises à jour.

Elagage de l'espace de recherche

Grâce à la recombinaison des points, un seul score est attaché à chaque point créé. Nous calculons la valeur minimale de ces scores et nous mettons à jour la liste des points actifs en éliminant ceux dont le score dépasse cette valeur minimale de plus d'un certain seuil Δ.

Répétant ces trois opérations depuis l'instant initial jusqu'à l'instant final, nous développons un ensemble, ou *faisceau*, de chemins partiels "proches" du chemin optimal ; les scores de ces chemins sont proches du score minimal à cet instant. Nous effectuons ensuite l'opération de *backtracking*, telle que nous l'avons présentée au paragraphe 2.2.5, afin d'obtenir la séquence des mots reconnus.

Complexité calculatoire

La réduction de complexité introduite par la stratégie d'élagage ne peut se mesurer asymptotiquement. En effet, pour un seuil Δ donné, le nombre de points actifs peut, dans le pire des cas, être égal au nombre total de points et la complexité est donc également $\mathcal{O}(T.N)$, où T désigne le nombre total de transitions du modèle. Cependant, nous verrons qu'en pratique cette stratégie s'avère particulièrement efficace.

Deuxième partie

Utilisation de modèles de langage

Chapitre 3

Typologie des modèles de langage

La théorie présentée au chapitre 2 s'applique à la reconnaissance de séquence de mots, pour peu que la notation W^i désigne une séquence particulière de mots, plutôt qu'un mot particulier. Dans la suite, nous désignons par W une séquence quelconque de mots, par W_l le *l-ième* mot de la séquence et par $W_{1 \ldots k}$ la séquence des k mots constituant W. La séquence W de mots reconnus, étant donné la séquence X des vecteurs acoustiques observés, sera celle pour laquelle la probabilité $P(W|X)$ est maximale. Conformément à la règle de Bayes (2.8) nous avons

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)}. \tag{3.1}$$

En particulier, si nous ne disposons pas de modèle de langage, nous pouvons considérer toutes les séquences W comme étant équiprobables et, par conséquent, seul le facteur $P(X|W)$ intervient dans la maximisation. Par contre, un modèle de langage fournit une estimation de la probabilité a priori $P(W)$ de la séquence W et nous pouvons écrire

$$P(W_{1 \ldots k}) = P(W_1). \prod_{l=2}^{k} P(W_l | W_{1 \ldots l-1}). \tag{3.2}$$

Remarquons que cette décomposition en probabilités conditionnelles n'est pas la seule possible. En particulier, lorsque le modèle de langage est basé sur une grammaire formelle, une autre décomposition de $P(W_{1 \ldots k})$ est utilisée (voir paragraphe 3.3).

Dans les paragraphes suivants, nous présentons les différents modèles de langage qui peuvent être utilisés pour améliorer les performances d'un système de parole continue. Dès la présentation du modèle des N-grams (paragraphe 3.1.2), nous insistons sur les difficultés que peut présenter l'utilisation d'un modèle de langage. Ce point important sera détaillé au chapitre 4.

La typologie des modèles de langage, telle que nous la présentons, reflète des distinctions de nature historique. Nous mentionnons au paragraphe 3.3.4 que les N-grams constituent en réalité un cas particulier de grammaires stochastiques. Cette propriété sera détaillée au paragraphe 7.1.1.

3.1 Modèles purement statistiques

Les modèles de langage les plus couramment utilisés sont purement statistiques[1]. Ils ont pour objet d'accorder une probabilité aux séquences de mots proportionnellement à leur fréquence d'apparition dans un ensemble de phrases ou *corpus d'apprentissage*. Ces corpus sont idéalement composés d'une retranscription de phrases prononcées. Ils pourront dès lors prendre en compte, statistiquement, les particularités du discours oral.

3.1.1 Bigrams

Un *bigram* est un modèle statistique où l'on approxime la probabilité des séquences de mots en considérant que la probabilité d'un mot de la séquence ne dépend que de son prédécesseur. Formellement, l'égalité (3.2) devient

$$P(W_{1\ldots k}) \cong P(W_1).\prod_{l=2}^{k} P(W_l|W_{l-1}). \tag{3.3}$$

L'intérêt essentiel des bigrams est leur nombre limité de paramètres et leur facilité d'utilisation dans la reconnaissance. En particulier, le terme $T(J(k'),k)$ de l'expression (2.26) correspond par exemple à une transition de l'état $J(k')$, dernier état du modèle du mot $W^{k'}$, au premier état du mot W^k. Ce terme est simplement obtenu par l'expression suivante

$$T(J(k'),k) = -\log P(W^k|W^{k'}). \tag{3.4}$$

3.1.2 N-grams

Dans le cas général d'un modèle *N-gram,* la probabilité d'un mot dans une séquence dépend de ses $N-1$ prédécesseurs. Formellement, l'égalité (3.2)

[1] Plus précisément, la structure des règles représentées par ce type de modèles est fixée dès que l'ordre du modèle (i.e. la valeur de N pour un *N-gram*) est choisi. Leur apprentissage relève alors d'un problème d'estimation purement statistique.

devient

$$P(W_{1\ldots k}) = \\ P(W_1).\,P(W_2|W_1).\cdots.P(W_{N-1}|W_{1\ldots N-2}).\prod_{l=N}^{k} P(W_l|W_{l-(N-1)\ldots l-1}). \tag{3.5}$$

Deux questions sont soulevées par l'extension des bigrams aux N-grams, avec $N > 2$.

D'une part, si le lexique comporte M mots, un N-gram possède M^N paramètres à estimer. Généralement, les corpus d'apprentissage disponibles ne sont pas de taille suffisante dès que le lexique comporte un millier de mots ou plus. Dans ce cas, il faut recourir à des techniques de lissage des paramètres, techniques déjà évoquées au paragraphe 1.2.

D'autre part, les formules de récurrence (2.24) et (2.26) ne sont plus valables telles quelles. La raison en est que la probabilité d'atteindre le premier état d'un mot ne dépend plus seulement du dernier état du mot précédent, dès que $N > 2$. Plus précisément, nous ne disposons plus d'un modèle du premier ordre et, dans ce sens, nous ne respectons plus l'hypothèse permettant le passage du facteur $P(q_l^n|Q_{1\ldots n-1})$ de l'expression (2.3) à $P(q_l^n|q_k^{n-1})$ (expression (2.4)). Une autre formulation du même argument consiste à dire que le principe d'optimalité de la programmation dynamique (voir paragraphe 2.2.5) ne peut plus s'appliquer tel quel. Cette difficulté théorique apparaît également pour les autres modèles de langage que nous allons décrire ci-après. La solution généralement retenue consiste à dupliquer les séquences d'états, que nous avons désignées par $Q_{1\ldots n-1}$, pour chaque *contexte syntaxique* CS, dans lequel cette séquence apparaît. La notion de contexte syntaxique sera détaillée au chapitre 4. Dans le cas particulier d'un N-gram, le contexte syntaxique du premier état du mot W_l correspond exactement à la séquence $W_{l-(N-1)\ldots l-1}$. Plus généralement, nous remplaçons dans l'expression (2.3) le facteur $P(q_l^n|Q_{1\ldots n-1})$ par $P(q_l^n|Q_{1\ldots n-1}, CS)$ et l'hypothèse de modèle de Markov du premier ordre et stationnaire s'exprime à présent sous la forme

$$P(q_l^n|Q_{1\ldots n-1}, CS) \cong P(q_l|q_k, CS), \tag{3.6}$$

où q_l et q_k sont deux états atteints à des instants consécutifs, indépendamment de ces instants eux-mêmes, et dans le contexte CS. Nous étudierons les conséquences algorithmiques de la relation (3.6) au chapitre 4.

3.2 Modèles purement syntaxiques

Nous présentons dans ce paragraphe les modèles de langage purement syntaxiques[2], c'est-à-dire ceux reposant sur une grammaire formelle. Dans ce cas, la grammaire définit l'ensemble des séquences de mots correctes et donc contraint les enchaînements de modèles acoustiques aux frontières de mots. Notons que les grammaires constituent un formalisme général. Les terminaux correspondent généralement aux mots d'un lexique connu. Néanmoins, ils peuvent également identifier d'autres types d'unités tels que, par exemple, des classes de mots ou des syntagmes. Moyennant l'introduction d'un terminal particulier représentant tous les mots d'un lexique, il est possible de ne contraindre que certaines parties des phrases du langage modélisé.

3.2.1 Grammaires à structure de phrase

Les grammaires à structure de phrase [Chomsky57] présentent un grand intérêt dans la mesure où elles ont été largement utilisées dans de nombreux domaines incluant la Théorie des Langages, le Traitement de la Langue Naturelle et la Reconnaissance des Formes Syntaxiques et Structurelles. Nous rappelons brièvement les notions principales ci-dessous afin de définir les notations utilisées par la suite. Pour plus de détails, le lecteur pourra consulter les ouvrages de référence [Aho et al.72, Harrison78].

Définition 3.1 Une *grammaire à structure de phrase* G est un quadruplet $G = (V, \Sigma, P, S)$ où

- V est un ensemble fini de (*symboles*) *non-terminaux* (encore appelés *variables* ou *catégories syntaxiques*),
- Σ est un ensemble fini de (*symboles*) *terminaux*, disjoint de V,
- P est un sous-ensemble fini de $(V \cup \Sigma)^* V (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ dont un élément (α, β) est noté $\alpha \to \beta$ et appelé *production*, de *partie gauche* α et de *partie droite* β ,
- S est un élément particulier de V, appelé *axiome* ou *symbole d'entrée*.

Définition 3.2 Une *proto-phrase*[3] d'une grammaire $G = (V, \Sigma, P, S)$ est une chaîne de longueur finie sur $(V \cup \Sigma)$, c'est-à-dire un élément de $(V \cup \Sigma)^*$, définie récursivement comme suit

[2] Le qualificatif *syntaxique* est utilisé ici dans le sens classique de la Reconnaissance des Formes. Il ne suppose donc pas nécessairement que les modèles définis soient basés sur une connaissance linguistique.

[3] En anglais, "*sentential form*".

- S est une proto-phrase,
- Si $\alpha\beta\gamma$ est une proto-phrase et $\beta \to \delta$ une production de G, alors $\alpha\delta\gamma$ est aussi une proto-phrase.

On dit également que $\alpha\delta\gamma$ *dérive directement* de $\alpha\beta\gamma$ et la relation de dérivation est notée $\underset{G}{\Rightarrow}$, ou simplement $\Rightarrow$, lorsqu'il n'y a pas d'ambiguïté, sa fermeture transitive est notée $\overset{+}{\Rightarrow}$ et sa fermeture réflexive et transitive est notée $\overset{*}{\Rightarrow}$.

Définition 3.3 La longueur $|\alpha|$ d'une proto-phrase α est le nombre de symboles de α.

Définition 3.4 Une *phrase* engendrée ou acceptée par une grammaire G est une proto-phrase de G ne contenant aucun non-terminal.

Définition 3.5 Une *dérivation gauche* d'une chaîne x de Σ^* à partir d'une grammaire G est une suite de proto-phrases, $S = \alpha_0 \Rightarrow \ldots \Rightarrow \alpha_m = x$, telle que α_i est obtenu en dérivant le non-terminal le plus à gauche de α_{i-1} conformément à une production commençant par ce non-terminal.

Définition 3.6 Une grammaire est *ambiguë* si au moins une chaîne x de Σ^* admet plusieurs dérivations gauches.

En guise d'exemple, considérons la grammaire définie par les productions suivantes

$$\begin{array}{ll} S & \to AB \\ S & \to abB \\ A & \to ab \\ B & \to b \end{array}$$

Elle admet deux dérivations gauches de la chaîne abb : $S \Rightarrow AB \Rightarrow abB \Rightarrow abb$ et $S \Rightarrow abB \Rightarrow abb$.

Définition 3.7 Le *langage* L engendré, ou accepté, par une grammaire $G = (V, \Sigma, P, S)$ est l'ensemble des phrases engendrées, ou acceptées, par G. $L(G) = \{x | x \in \Sigma^* \text{ et } S \underset{G}{\overset{*}{\Rightarrow}} x\}$.

Il existe une hiérarchie des grammaires à structure de phrase, chaque classe de la hiérarchie étant identifiée par certaines restrictions sur la forme des productions. Nous étudierons en particulier les deux classes au bas de la hiérarchie, celle des grammaires *linéaires à droite* ou *régulières* et celle des grammaires *hors-contexte* ou *algébriques*.

Définition 3.8 Une grammaire $G = (V, \Sigma, P, S)$ est *de type 3, ou* linéaire à droite (RLG[4]), si chaque production de P est de la forme $A \to aB$ ou $A \to a$, avec $A, B \in V$ et $a \in \Sigma$.

Définition 3.9 Une grammaire $G = (V, \Sigma, P, S)$ est *de type 2, ou* hors-contexte (CFG[5]), si chaque production de P est de la forme $A \to \alpha$, avec $A \in V$ et $\alpha \in (V \cup \Sigma)^*$.

Définition 3.10 Une grammaire $G = (V, \Sigma, P, S)$ est de type 1, ou sensible au contexte (CSG[6]), si chaque production de P est de la forme $\alpha \to \beta$, avec $\alpha, \beta \in (V \cup \Sigma)^*$ et $|\alpha| \leq |\beta|$.

Définition 3.11 Une grammaire sans restriction est de *type 0.*

Une autre manière de spécifier un langage est de le décrire par le biais d'un *automate*. Chaque classe de grammaires possède un type d'automates équivalents. Par exemple, les langages réguliers sont les langages définis par une grammaire régulière ou un *automate fini* alors que les langages hors-contexte sont définis par une grammaire hors-contexte ou un *automate à pile* [Aho et al.72]. Nous ne détaillons pas ici la théorie des automates. Néanmoins, la seconde partie de ce travail sera consacrée à l'apprentissage de grammaires régulières représentées sous forme d'automates finis. Par conséquent, le lecteur trouvera une description plus complète de ces automates au paragraphe 6.2.

Etant donné un modèle de langage exprimé par une grammaire G, les phrases admises sont celles qui correspondent à un élément du langage défini par G. Nous verrons au chapitre 4, comment résoudre les problèmes posés par l'utilisation d'un tel modèle.

3.2.2 Grammaires d'unification

Malgré le succès des grammaires à structure de phrase, un grand nombre de formalisme grammaticaux ont été proposés. Ils sont souvent basés sur une opération dite d'*unification* qui permet d'exprimer aisément des dépendances syntaxiques ou sémantiques complexes [Gazdar et al.89, Miller et al.90]. Nous n'étudierons pas ici ce type de modèles car la complexité calculatoire d'un

[4] L'abréviation **RLG** pour "Right Linear Grammar" provient de la terminologie anglaise. Nous utiliserons également l'abréviation **RG**, pour "Regular Grammar".

[5] L'abréviation **CFG** pour "Context-Free Grammar" provient de la terminologie anglaise.

[6] L'abréviation **CSG** pour "Context-Sensitive Grammar" provient de la terminologie anglaise.

algorithme d'analyse incluant des unifications ne permet généralement pas d'utiliser ces modèles pour contrôler le décodage acoustique. Cela oblige donc à renoncer à l'intégration du modèle de langage dans le processus de reconnaissance proprement dit. De plus, il est généralement admis que la grande majorité des structures syntaxiques de la langue naturelle peuvent être représentées par les grammaires hors-contexte[7][Gazdar et al.89].

3.3 Modèles étendus ou combinés

Dans ce paragraphe, nous mentionnons un certain nombre de variantes des modèles présentés dans les paragraphes précédents.

3.3.1 N-grams définis au niveau de classes syntaxiques

Derouault a proposé de combiner un 3-gram et une connaissance syntaxique représentée par des catégories ou classes de mots [Derouault85]. Dans ce modèle, on dispose d'une base d'estimation où chaque mot a été étiqueté par une certaine classe syntaxique. Si nous désignons par $P(W|C)$ la probabilité qu'un mot W appartienne à une classe C donnée, nous obtenons la formulation suivante

$$P(W_l|W_{l-2}W_{l-1}) \cong \sum_C P(W_l|C).P(C|W_{l-2}W_{l-1}) \ . \tag{3.7}$$

Le modèle de langage final combine deux modèles : un N-gram au niveau des classes et un N-gram au niveau du mot. Ces deux modèles sont pondérés par une technique de lissage des paramètres [Jelinek et al.80].

3.3.2 Approximation statistique de grammaires

Une autre manière de combiner un modèle syntaxique et statistique consiste à estimer un N-gram à partir d'un corpus généré automatiquement par une grammaire. En effet, si nous disposons d'une grammaire G, nous pouvons l'utiliser pour générer un corpus de phrases appartenant à $L(G)$. A partir de ce corpus nous pouvons estimer, par exemple, un bigram par calcul des fréquences relatives de chaque paire de mots. Cette approximation statistique suppose, en outre, de définir une distribution de probabilité sur les règles

[7]Il s'agit d'un argument formel concernant le pouvoir d'expression minimal qu'une grammaire doit posséder pour représenter des structures syntaxiques. Néanmoins, l'écriture manuelle d'une grammaire peut s'avérer d'autant plus aisée que le formalisme utilisé a un pouvoir d'expression important.

de la grammaire. Faute de disposer d'aucune information statistique, nous pouvons par exemple supposer équiprobables toutes les règles commençant par un non-terminal donné[8].

Une approximation statistique d'une grammaire, en particulier sous la forme d'un bigram, rend l'utilisation du modèle de langage plus aisée (voir paragraphe 3.1). Par contre, elle présente l'inconvénient de ne plus rendre compte de dépendances à long terme qui pouvaient être modélisées par la grammaire.

3.3.3 Arbres de classification

Bahl et ses coauteurs [Bahl et al.89] ont proposé une extension des N-grams, basée sur un arbre de classification [Breiman et al.84]. L'utilisation d'un arbre de classification binaire permet de définir un ensemble de classes d'équivalence, chacune assignée à une feuille de l'arbre. Partant de la racine, une question binaire permet de choisir une branche à chaque nœud de l'arbre, jusqu'à atteindre une feuille, par récursion. Une question binaire serait par exemple : "le mot W_{l-1} appartient-il à l'ensemble des adjectifs ?" ou "le mot W_l est-il présent dans la séquence $W_{1\ldots l-1}$?". A chaque feuille de l'arbre est attachée une distribution de probabilité d'observer le mot suivant. Plus précisément, si nous désignons par $f_{1\ldots l-1}$ la feuille de l'arbre à laquelle nous arrivons sur base des questions posées et des $l-1$ mots précédents, nous avons

$$P(W_l|W_{1\ldots l-1}) \cong P(W_l|f_{1\ldots l-1}). \tag{3.8}$$

La technique décrite permet de définir les questions à poser et la structure de l'arbre de classification conformément à un critère d'entropie minimale sur les distributions attachées aux feuilles.

3.3.4 Grammaires hors-contexte stochastiques

Une extension classique des grammaires à structure de phrase est celle obtenue par adjonction d'une probabilité sur les règles. Ceci définit une distribution de probabilité sur l'ensemble des phrases engendrées par la grammaire. Ce type de modèle est particulièrement intéressant car il offre un cadre formel cohérent combinant les modèles purement statistiques, d'une part, et purement syntaxiques, d'autre part.

[8]Disposant d'une *grammaire hors-contexte stochastique* (voir paragraphe 3.3.4), un algorithme de calcul exact d'un N-gram a été développé [Stolcke et al.94b]. Il permet d'éviter la génération d'un corpus préalable à l'estimation du N-gram.

Définition 3.12 Une grammaire G hors-contexte stochastique (SCFG[9]) est un quadruplet (V, Σ, P_s, S) où P_s est un ensemble de productions hors-contexte du type $A_i \overset{p_{ij}}{\to} \alpha_j$ $(A_i \in V, \alpha_j \in (V \cup \Sigma)^*)$ et p_{ij} désigne la probabilité d'appliquer cette production [Gonzalez et al.78].

Définition 3.13 Si nous désignons par $\{A_i \overset{p_{i1}}{\to} \alpha_1, \ldots, A_i \overset{p_{in_i}}{\to} \alpha_{n_i}\}$ l'ensemble des n_i productions commençant par le non-terminal A_i, la grammaire hors-contexte stochastique G est *propre* si et seulement si

$$\sum_{j=1}^{n_i} p_{ij} = 1, \; \forall A_i \in V.$$

Définition 3.14 La probabilité $p_{1\ldots m}(x|G)$ d'une dérivation gauche d'une chaîne x de $\sum^*$, faisant intervenir la séquence de productions $r_1, \ldots, r_m$ de la grammaire G, est le produit des probabilités des productions utilisées lors de cette dérivation

$$p_{1\ldots m}(x|G) = p(r_1)p(r_2|r_1)\ldots p(r_m|r_1, \ldots, r_{m-1}) \triangleq p(r_1)p(r_2)\ldots p(r_m). \tag{3.9}$$

Il s'agit de l'extension au cas stochastique de l'indépendance d'application des règles hors-contexte.

Définition 3.15 Pour une grammaire stochastique ambiguë, nous désignons par m_x le nombre de dérivations gauches $p_g(x|G)$ de la chaîne x. La probabilité $p(x|G)$ de la chaîne x conformément à la grammaire G est donnée par

$$p(x|G) = \sum_{g=1}^{m_x} p_g(x|G). \tag{3.10}$$

Définition 3.16 Une grammaire G est *consistante*[10] si et seulement si

$$\sum_{x \in \Sigma^*} p(x|G) = 1. \tag{3.11}$$

[9] L'abréviation **SCFG** pour "Stochastic Context-Free Grammar" provient de la terminologie anglaise.

[10] Par définition, les chaînes de Σ^* qui ne peuvent pas être engendrées par la grammaire G ont une probabilité nulle. La condition de consistance s'exprime donc également par $\sum_{x \in L(G)} p(x|G) = 1$.

Finalement, si nous utilisons une SCFG comme modèle de langage, l'expression (3.2) devient

$$P(W_{1\ldots k}) = P(x = W_{1\ldots k}|G). \tag{3.12}$$

Cette relation est applicable pour tout modèle de langage basé sur une grammaire formelle pour peu que l'on se fixe une distribution de probabilité sur les séquences. Faute de données statistiques, une hypothèse déjà mentionnée consiste à poser

$$p_{ij} = 1/n_i,\ 1 \leq j \leq n_i,\ \forall A_i \in V. \tag{3.13}$$

L'expression (3.13) illustre le fait qu'une CFG peut être considérée comme un cas particulier d'une SCFG[11]. De plus, nous verrons au paragraphe 7.1.1 qu'un N-gram est équivalent à l'extension stochastique d'un *langage k-testable au sens strict*, cas particulier d'un langage régulier stochastique, lui-même cas particulier d'un langage hors-contexte stochastique. C'est la raison pour laquelle les SCFGs fournissent un cadre général incluant les deux classes de modèles exposés aux paragraphes 3.1 et 3.2.

Finalement, l'algorithme "Inside-Outside" permettant d'estimer les probabilités des productions d'une SCFG, a été proposé par Baker [Baker79] et reformulé dans [Lari et al.90]. Des améliorations récentes de cet algorithme sont décrites dans [Pereira et al.92, Young et al.94].

3.4 Critère de qualité : la perplexité

Les différents modèles de langage présentés dans les paragraphes précédents peuvent être comparés selon plusieurs critères. Parmi les critères déjà évoqués nous avons : la possibilité de prendre en compte des dépendances syntaxiques à long terme, la possibilité d'apprentissage automatique du modèle à partir d'un corpus représentatif du fragment de langue, la complexité d'intégration de cette source de connaissance dans le décodage acoustique, la possibilité d'adaptation progressive du modèle, etc.

L'objectif principal d'un modèle de langage est de réduire le taux d'erreur du système de reconnaissance par le biais d'une estimation correcte de la probabilité a priori des séquences de mots. Plus précisément, pour un langage inconnu L, nous désirons trouver un modèle M tel que le langage $L(M)$ accepté par le modèle, approxime aussi bien que possible L. Par conséquent, il faut que les probabilités accordées aux séquences conformément à $L(M)$

[11] L'hypothèse d'équiprobabilité des productions d'une CFG n'est strictement valable que si la SCFG résultante est consistante. Un algorithme d'évaluation de la consistance d'une SCFG est proposée dans [Wetherell80].

correspondent aux probabilités réelles de ces séquences (conformément au langage L). Nous avons vu que les N-grams comme les CFGs constituent des cas particuliers de modèles basés sur les SCFGs. Nous supposons donc, de façon générale, le modèle de langage représenté par une grammaire G de ce dernier type.

Deux problèmes dans l'estimation du modèle peuvent apparaître : d'une part, la *sur-généralisation* qui consiste à attribuer des probabilités élevées à des séquences de probabilités faibles[12], d'autre part, la *sous-couverture* qui consiste à attribuer des probabilités faibles à des séquences de probabilités élevées[13]. Si nous disposons d'une modèle consistant ($\sum_{x\in\Sigma^*} p(x|G) = 1$), il existe une mesure combinée de la sur-généralisation et de la sous-couverture du modèle G qui est l'*entropie croisée par symbole* $H(L, L(G))$ entre $L(G)$ et L. Comme nous ne pouvons disposer, en pratique, que d'un *échantillon de test*, $S = \{x_1, \ldots, x_T\}$ du langage L, nous approximons l'entropie croisée réelle par l'entropie calculée sur cet échantillon. Elle est définie comme suit

$$H(L, L(G)) \simeq H(S, L(G)) = \frac{-\sum_{x=x_1}^{x_T} \log p(x|G)}{\sum_{x=x_1}^{x_T} |x|}, \tag{3.14}$$

où $|x|$ désigne le nombre de symboles de la chaîne x. En particulier, si l'échantillon S est représentatif du langage L, chaque chaîne x, ou séquence de mots, apparaîtra dans S avec une fréquence proportionnelle à sa probabilité $p(x|L)$. Lorsque le logarithme est exprimé en base 2, l'entropie croisée par symbole est exprimée en bits. La *perplexité* $Q(S, L(G))$ *de l'échantillon de test* S est définie comme suit

$$Q(S, L(G)) = 2^{H(S,L(G))}. \tag{3.15}$$

L'entropie croisée d'un modèle de langage et d'un ensemble de test constitue également une mesure du pouvoir de prédiction du modèle et les modèles de langage pour lesquels l'entropie croisée (ou la perplexité) est plus faible conduisent directement à de meilleurs performances de reconnaissance [Brown et al.92a].

Pour illustrer ce point nous allons considérer le modèle d'un unigram, c'est-à-dire

$$P(x = W_{1\ldots k}) \simeq P(W_1)\ldots.P(W_k), \tag{3.16}$$

---

[12]Dans le cas d'un modèle purement syntaxique, la sur-généralisation consiste à inclure dans $L(G)$ des chaînes exclues de L.

[13]Dans le cas d'un modèle purement syntaxique, la sous-couverture consiste à ne pas inclure dans $L(G)$ des chaînes appartenant à L.

et, de plus, nous supposons toutes les probabilités égales

$$P(W^j) \triangleq 1/M, 1 \leq j \leq M, \tag{3.17}$$

où M désigne le nombre de mots du lexique. Donc, $p(x) = M^{-|x|}$. Dans ce cas particulier, l'expression (3.14) devient

$$\begin{aligned} H(S, L(G)) &= \frac{-\sum_{x=x_1}^{x_T} \log p(x|G)}{\sum_{x=x_1}^{x_T} |x|} \overset{(3.17)}{=} \frac{-\sum_{x=x_1}^{x_T} \log M^{-|x|}}{\sum_{x=x_1}^{x_T} |x|} \\ &= \frac{\log M . \sum_{x=x_1}^{x_T} |x|}{\sum_{x=x_1}^{x_T} |x|} = \log M, \end{aligned} \tag{3.18}$$

et la perplexité est égale à M, la taille du lexique. Il s'agit de la valeur maximale de la perplexité. Il est évident que, dans ce cas, le modèle de langage n'a aucun pouvoir de prédiction, c'est-à-dire n'apporte aucune information permettant de guider la reconnaissance. A l'inverse un modèle prédisant parfaitement la phrase x à reconnaître serait celui attribuant une probabilité égale à l'unité à cette unique phrase. Dans ce cas extrême, l'entropie est nulle et la perplexité égale à l'unité. Nous avons donc en général

$$1 \leq Q(S, L(G)) \leq M. \tag{3.19}$$

C'est la raison pour laquelle la perplexité est parfois considérée comme le *facteur de branchement moyen*, c'est-à-dire le nombre moyen de successeurs d'un mot donné. Cette interprétation n'est strictement valable que dans les deux cas extrêmes décrits ci-dessus.

Chapitre 4

Utilisation de modèles de langage hors-contexte (*)

4.1 Objectifs

Nous avons vu qu'un modèle de langage permet d'améliorer le processus de reconnaissance dans la mesure où il fournit une estimation de la probabilité a priori des séquences de mots. Au paragraphe 3.1.2, nous avons déjà évoqué le problème posé par l'utilisation d'un modèle de langage. En résumé, pour pouvoir appliquer les hypothèses d'un modèle du premier ordre et effectuer le décodage en utilisant le principe d'optimalité de la programmation dynamique, il faut dupliquer les séquences d'états apparaissant dans des contextes syntaxiques différents.

Nous allons préciser tout d'abord la notion de contexte syntaxique et voir comment le décodage de Viterbi peut traiter des grammaires régulières équivalentes aux automates finis. Nous allons ensuite étendre cet algorithme afin de traiter une grammaire hors-contexte générale donnant lieu à un algorithme *optimal* de décodage incluant ce niveau de contraintes. Le caractère éventuellement stochastique du modèle n'influencera pas la complexité du problème. En d'autres termes, nous pouvons étudier le problème d'utilisation d'une CFG pour définir les seules séquences de mots correctes et considérer l'utilisation d'une SCFG comme une extension triviale de l'algorithme de décodage (les transitions entre mots sont simplement affectées d'une pénalité qui dépend de la probabilité de la règle correspondante).

L'algorithme que nous proposons repose sur deux idées que nous pouvons résumer comme suit. La récursivité propre à une grammaire hors-contexte peut être bornée par la longueur de la phrase à analyser. C'est en particulier le cas lorsque la récursivité à gauche de la grammaire a été éliminée lors d'une phase de compilation préalable à la reconnaissance. Notons qu'il ne s'agit pas d'une limitation de notre approche, dans la mesure où un algorithme de mise sous la forme normale de Greibach permet de transformer, en temps polynomial (voir propriété 4.1), une quelconque CFG en une grammaire équivalente non récursive à gauche. En outre, cette transformation ne doit être effectuée qu'une seule fois pour une grammaire donnée. Le recours à la forme normale de Greibach présente un second avantage algorithmique qui sera exposé au paragraphe 4.3.3. Tel quel l'algorithme de décodage est optimal ; mais nous pouvons réduire sa complexité pratique en utilisant la technique de contrôle de l'espace de recherche décrite au paragraphe 2.2.6. Comme nous l'avons indiqué, l'algorithme résultant ne peut plus être garanti optimal. L'intérêt de notre approche est que le recours à un modèle de langage hors-contexte n'introduit pas de "sous-optimalité supplémentaire". En d'autres termes, pour toutes les séquences d'états développées en faisceau, la contrainte apportée par le modèle de langage hors-contexte respecte l'optimalité de la programmation dynamique.

4.2 Décodage Viterbi guidé par un automate fini

Dans ce paragraphe, nous présentons l'extension classique de l'algorithme de Viterbi sous contrainte d'un automate fini, décrivant au niveau syntaxique les séquences de mots admises. Cet automate fini est équivalent à une grammaire régulière [Aho et al.72].

Dans la suite, nous désignons par des lettres minuscules $a, b, c, \ldots$ les terminaux d'une grammaire, c'est-à-dire les mots du lexique[1]. A chaque mot est associé son modèle de Markov, éventuellement obtenu par concaténation des modèles attachés aux phonèmes qui le composent. Un automate fini peut être représenté par son *diagramme de transition,* que nous désignerons dans ce contexte comme un *graphe syntaxique*, tel celui représenté à la figure 4.1. Chaque transition est étiquetée par une lettre correspondant à un mot. Une séquence de transitions constitue un chemin et tous les chemins partant d'un état d'entrée et arrivant sur un état de sortie identifient les phrases acceptables par l'automate fini. Nous avons déjà rencontré ce type de représentation, par exemple à la figure 2.3. Si la grammaire régulière équivalente n'est pas stochastique, l'automate n'inclut simplement pas de probabilité. Pour une étude détaillée des propriétés des automates finis, le lecteur peut

[1]Une généralisation possible consiste à considérer des catégories de mots, encore appelées classes pré-terminales.

se référer au chapitre 6. Nous nous limiterons ici aux notions utiles à notre propos.

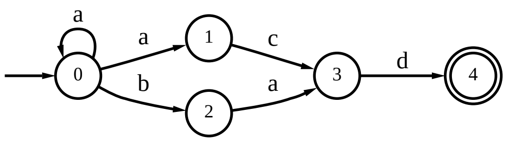


FIG. 4.1: Exemple de graphe syntaxique.

Nous avons vu au paragraphe 3.1.2 que l'utilisation d'un modèle de langage pouvait introduire des contraintes ne dépendant pas seulement du mot prédécesseur. En particulier, dans notre exemple, le mot a peut être suivi des mots a ou b, du mot c ou du mot d, selon la place de ce mot a dans la phrase. Afin de respecter l'hypothèse de modèle de premier ordre et de pouvoir appliquer le principe d'optimalité de la programmation dynamique, nous avons introduit la notion de contexte syntaxique et l'expression $p(q_l|q_k, CS)$ rend compte de la probabilité de la transition de l'état q_k à l'état q_l dans le contexte CS. Dans le cas du contrôle du décodage par un automate fini, un contexte syntaxique est identifié par chacun des nœuds du graphe[2]. Par conséquent, le principe d'optimalité peut s'appliquer au niveau syntaxique en dupliquant les états d'un modèle de Markov attaché, par exemple, au mot a pour chaque transition du graphe syntaxique étiquetée par a. En particulier, parmi tous les chemins arrivant à un instant n donné, à un nœud syntaxique CS, seul le meilleur chemin (celui affecté du score minimal) devra être poursuivi et l'information correspondante mémorisée (table des instants et tables des mots). Dans notre exemple, les chemins arrivant au nœud 3 peuvent correspondre aux séquences de mots suivantes : $ac, ba, aac, aba, aaac, aaba, \ldots$. Seule la séquence affectée du meilleur score sera poursuivie. Une manière usuelle de caractériser ces séquences consiste à dire, qu'à partir du nœud 3, elles se poursuivent toutes par le même ensemble de séquences, c'est-à-dire qu'elles partagent un "futur" commun. Dès lors, seule la meilleure séquence partielle jusqu'à chaque nœud doit être mémorisée et poursuivie.

Formellement, un point de la programmation dynamique devient un triplet (n, j, tr_k) où l'indice tr_k désigne une transition syntaxique particulière étiquetée par le mot k dans le graphe syntaxique. Nous désignons également par $J(tr_k)$ le nombre d'états du modèle (acoustique) de Markov associé à cette transition syntaxique. La première récurrence (2.24) devient simplement

$$D(n, i, tr_k) = d(n, i, tr_k) + \min_j(D(n-1, j, tr_k) + T(j, i)), \qquad (4.1)$$

[2]Un contexte syntaxique étant l'ensemble des séquences de transitions menant à un nœud du graphe, il y a un contexte syntaxique par nœud.

pour $1 < i \leq J(tr_k), 1 \leq tr_k \leq TR$, avec TR le nombre total de transitions du graphe syntaxique.

De façon similaire, nous obtenons la seconde récurrence

$$D(n,1,tr_k) = d(n,1,tr_k) + \min \left\{ \begin{array}{c} D(n-1,1,tr_k) + T(1,1) \; ; \\ \min\limits_{tr'_k=1,\ldots,TR} \left(D(n-1, J(tr'_k), tr'_k) + T(J(tr'_k), tr_k) \right) \end{array} \right\}. \tag{4.2}$$

Le terme $T(J(tr'_k), tr_k)$ désigne la pénalité associée à la transition du dernier état du mot attaché à la transition syntaxique tr'_k, au premier état du mot attaché à la transition syntaxique tr_k. Si le modèle de langage est stochastique, ce terme dépend directement de la probabilité de la production associée.

La complexité calculatoire du décodage est également $\mathcal{O}(T.N)$, où T désigne le nombre total de transitions du modèle. Dans ce cas, si nous désignons par t_{i,tr_k} le nombre de transitions arrivant sur l'état i du modèle de Markov associé à la transition syntaxique tr_k, nous avons

$$T = \sum_{tr_k=1}^{TR} \sum_{i=1}^{J(tr_k)} t_{i,tr_k}.$$

Remarquons finalement qu'en l'absence de contraintes syntaxiques, nous obtenons le graphe syntaxique représenté à la figure 4.2. Dans ce graphe, puisque nous avons une transition syntaxique par mot, nous pouvons poser $tr_k = k$ et nous retrouvons exactement les récurrences (2.24) et (2.26).

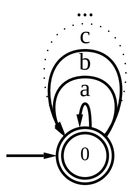


FIG. 4.2: Exemple de graphe sans contrainte syntaxique.

4.3 La méthode DYANE (*)

La méthode[3] présentée dans les paragraphes suivants assure une intégration complète dans le décodage Viterbi d'un modèle de langage reposant sur une grammaire hors-contexte[4].

[3] L'abréviation **DYANE** pour "DYnamic Adaptive Network Expansion" provient de la terminologie anglaise.

[4] Le caractère stochastique de la grammaire importe peu pour le problème présent. La généralisation au traitement d'une SCFG est triviale.

4.3.1 Représentation en réseaux récursifs

Une grammaire hors-contexte $G = (V, \Sigma, P, S)$ peut être représentée par un ensemble de réseaux récursifs, ou RTNs[5] [Woods70]. Insistons sur le fait qu'il s'agit d'un mode de représentation plutôt que d'une technique d'analyse hors-contexte. De plus, nous n'utilisons pas les RTNs comme une structure servant à l'analyse, mais plutôt à la *prédiction.* Dans ce contexte, l'opération de prédiction consiste à définir à partir d'une séquence partielle de mots (l'ensemble des mots commençant une phrase), les mots qui peuvent suivre directement. Nous appelons ces mots les *successeurs* (directs) de la séquence partielle.

Un RTN constitue l'équivalent pour un automate à pile d'un diagramme de transition associé à un automate fini (voir paragraphe 4.2). Outre les transitions *terminales,* celles étiquetées par des terminaux ou mots du lexique, certaines transitions, dites *non-terminales,* sont étiquetées par des non-terminaux. De plus, à chaque non-terminal est associé un RTN.

A titre d'exemple, nous présentons à la figure 4.3, les RTNs associés à la grammaire suivante :

$$\begin{aligned} S &\rightarrow aSB \\ S &\rightarrow aCB \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

Cette grammaire engendre le langage hors-contexte $a^n cb^n$.

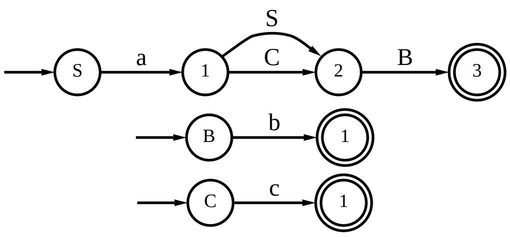


FIG. 4.3: Exemple de réseaux récursifs.

En général, à chaque non-terminal est associé un RTN qui représente l'ensemble des règles ayant ce non-terminal pour partie gauche.

La récursivité des RTNs apparaît dans notre exemple où la transition S appartient au réseau associé à S. En général, il y a une récursivité dès qu'il existe un non-terminal A tel que $\exists \alpha, \beta \in (\Sigma \cup V)^*$ et $A \stackrel{+}{\Rightarrow} \alpha A \beta$. En

[5] L'abréviation **RTN** pour "Recursive Transition Network" provient de la terminologie anglaise.

particulier, si $|\alpha|$ et $|\beta| \geq 1$, il n'est pas possible de construire un automate fini équivalent.

4.3.2 Développement dynamique d'un graphe fini (*)

Partant du RTN attaché à l'axiome S, nous pouvons progressivement développer un graphe en remplaçant chaque transition non-terminale par son réseau associé. Cette opération consiste à copier à la place de la transition non-terminale le réseau correspondant. Si nous itérons ce processus de développement dans le réseau résultant, nous obtenons un graphe de taille infinie étant donné la récursivité du modèle.

L'idée centrale à la méthode DYANE consiste à remarquer que le développement complet de ce graphe infini n'est pas nécessaire pour guider, *de façon optimale*, le décodage Viterbi. Plus précisément, la longueur des séquences de mots construites au cours du décodage est bornée par le nombre de trames de la phrase à reconnaître et par le fait que les réalisations de chaque mot "consomment" au moins un nombre minimal de trames. Ce nombre minimal de trames correspond à la longueur du chemin de durée minimale dans le modèle de Markov associé. Le développement du graphe syntaxique est donc borné.

Considérons, l'exemple de la figure 4.3. L'objectif du modèle de langage est de prédire les mots successeurs (et, en général, d'ajouter une pénalité correspondant à la probabilité de la production associée (voir paragraphe 4.2)). Partant du RTN attaché à l'axiome S, le mot a est le seul à pouvoir commencer une phrase. En ce sens, il s'agit du mot issu du nœud initial du réseau S. Tant qu'aucun chemin du décodage acoustique n'a atteint le dernier état du mot a, le nœud 1 du RTN initial reste inactif. Lorsque ce nœud devient actif le RTN initial est développé pour obtenir le réseau représenté à la figure 4.4.

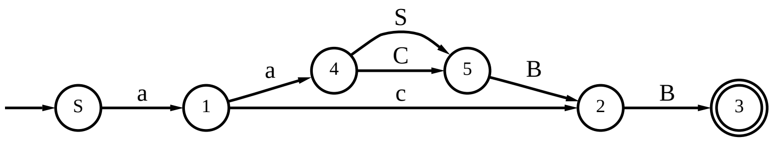


FIG. 4.4: Exemple d'expansion dynamique du réseau initial.

Le processus de copie des RTNs associés aux transitions non-terminales rencontrées est poursuivi dynamiquement selon les nœuds activés au cours du décodage. L'optimalité du décodage est garantie puisque, grâce aux copies, il existe une duplication des mots pour chacun des contextes syntaxiques. Par ailleurs, lorsque plusieurs chemins arrivent au même nœud du graphe syntaxique (par exemple, le nœud 2 du réseau développé à la figure 4.4), seul le meilleur chemin sera mémorisé et poursuivi. En conclusion, le développement dynamique ne correspond pas à l'arbre des préfixes des phrases possibles mais à un graphe qui permet d'appliquer le principe d'optimalité.

Une caractéristique intéressante du développement dynamique est que le traitement syntaxique effectué (la copie des réseaux associés aux transitions non-terminales), ne doit pas être réitéré pour les différentes possibilités d'alignement d'un mot attaché à une même transition du graphe. Par exemple, pour tous les chemins arrivant sur le dernier état du mot initial a, les mots successeurs sont connus après la première opération de copies. Dans la mesure où, en reconnaissance de la parole continue, de nombreuses hypothèses correspondant à la même séquence de transitions syntaxiques sont calculées pour des trames successives (en raison de l'incertitude sur la localisation temporelle des mots), il est particulièrement avantageux de ne pas recommencer le traitement syntaxique dans ces cas.

La complexité calculatoire du décodage acoustique intégrant le modèle de langage hors-contexte est $\mathcal{O}(\sum_{i=1}^{N} T^i)$, si T^i désigne le nombre de transitions du modèle développé jusqu'à la trame i, $1 \leq i \leq N$. A chaque trame, la complexité calculatoire est donc exactement celle du décodage Viterbi contrôlé par un automate fini. Par contre, le nombre de transitions du modèle croît avec le temps. Cette croissance dépend des caractéristiques de la grammaire (nombre de non-terminaux, nombre de productions et de structures récursives imbriquées). Elle peut, dans le pire des cas, s'avérer exponentielle mais bornée par la longueur de la phrase.

4.3.3 Utilisation de la Forme Normale de Greibach (*)

Définition 4.1 Une CFG $G = (V, \Sigma, P, S)$ est sous la forme normale de Greibach, ou GNF[6], si G est ε-libre[7] et si toute non-ε-production de P est de la forme $A \rightarrow a\alpha$, avec $a \in \Sigma, \alpha \in V^*$.

Remarquons que toute CFG G peut être transformée en une grammaire équivalente G' sous GNF. Nous traitons donc de la classe générale des langages hors-contexte. La mise sous GNF peut être effectuée préalablement à la phase de reconnaissance pour un modèle de langage donné.

[6]L'abréviation **GNF** pour “Greibach Normal Form” provient de la terminologie anglaise.

[7]Une ε-production est de la forme $A \rightarrow \varepsilon$, où ε désigne le symbole vide ; sinon il s'agit d'une non-ε-production. Une CFG est ε-libre si P ne comprend pas d'ε-production ou si la seule ε-production est $S \rightarrow \varepsilon$. Toute CFG peut être transformée en une grammaire équivalente et ε-libre [Aho et al.72].

La grammaire de l'exemple de la page 57 est sous GNF. Il en résulte deux propriétés intéressantes. D'une part, toute possibilité de récursivité à gauche est éliminée. La récursivité à gauche s'exprime comme suit $\exists A \in V, \exists \alpha \in (V \cup \Sigma)^* | \; A \stackrel{+}{\Rightarrow} A\alpha$. Elle entraînerait un développement dynamique non borné du RTN correspondant, puisqu'une transition non-terminale issue du nœud initial du réseau A est étiquetée par A (éventuellement, après un nombre fini de copies). La mise sous GNF de la grammaire présente le second avantage que toutes les transitions issues du nœud initial de chaque RTN sont nécessairement terminales. Par conséquent, l'opération de copie des RTNs ne doit pas être réitérée à partir d'un nœud donné du graphe. Dans notre exemple, après que les transitions non-terminales S et C aient été remplacées par leur RTN correspondant, nous avons obtenu directement les successeurs possibles, les mots a et c, du mot a initial. En résumé, la mise sous GNF assure qu'il y ait exactement une copie pour chaque transition non-terminale issue des nœuds actifs du graphe développé.

Complexité théorique de la transformation en GNF (*)

La transformation d'une CFG en GNF peut s'effectuer selon différents algorithmes [Aho et al.72] qui présentent des variations quant au nombre de règles et de non-terminaux de la grammaire résultante. Nous allons détailler l'algorithme dû à Rosenkrantz et montrer qu'il est de complexité polynomiale. A cet effet, nous introduisons la notion d'*équations de définition* [Rosenkrantz67].

Par exemple, la grammaire $G = (V, \Sigma, P, S)$ suivante

$$\begin{array}{ll} S \rightarrow & SaB \\ S \rightarrow & BB \\ S \rightarrow & b \\ B \rightarrow & aS \\ B \rightarrow & BSa \\ B \rightarrow & Bd \\ B \rightarrow & c \end{array}$$

peut être représentée par les équations

$$\begin{array}{ll} S = & SaB + BB + b \\ B = & aS + BSa + Bd + c \end{array} \tag{4.3}$$

où S et B sont des inconnues représentant des ensembles.

Définition 4.2 Soient Σ et Δ deux alphabets disjoints. Un ensemble d'*équations de définition* sur Σ et Δ est un ensemble d'équations de la forme

$A = \alpha_1 + \alpha_2 + \ldots + \alpha_k$, où $A \in \Delta$ et chaque α_i est une chaîne de $(\Sigma \cup \Delta)^*$. Si $k = 0$, l'équation correspondante est $A = \emptyset$.

Définition 4.3 Une *solution* à un ensemble d'équations de définition est une fonction $f : \Delta \to 2^{\Sigma^*}$ telle que si $f(A)$ est substitué pour chaque A dans Δ, les équations deviennent des égalités d'ensembles.

Définition 4.4 Une solution f est un *point fixe minimal* si pour tout A de Δ et pour toute solution g, $f(A) \subseteq g(A)$.

La grammaire G correspondant à un ensemble d'équations de définition est obtenue en créant les règles $A \to \alpha_1, \ldots, A \to \alpha_k$ pour chaque équation $A = \alpha_1 + \alpha_2 + \ldots + \alpha_k$. Par conséquent, Δ est l'ensemble des non-terminaux de G et Σ son ensemble de terminaux[8].

Lemme 4.1 *Le point fixe minimal d'un ensemble d'équations de définition sur Σ et Δ est unique et est donné par $f(A) = \{w | A \overset{*}{\underset{G}{\Rightarrow}} w$ avec $w \in \Sigma^*\}$, où G est la grammaire correspondante [Aho et al.72].*

Par conséquent, si f est le point fixe minimal et S l'axiome de G, $f(S)$ est le langage engendré par la grammaire G.

Nous pouvons exprimer l'ensemble d'équations de définition, associées à une grammaire G, sous la forme matricielle suivante.
Soit $\Delta = \{A_1, A_2, \ldots, A_n\}$ les non-terminaux de G, nous pouvons écrire l'équation matricielle $\underline{A} = \underline{A}\,\underline{\underline{R}} + \underline{B}$, où $\underline{A}$ est un vecteur ligne $[A_1, A_2, ..., A_n]$, $\underline{\underline{R}}$ est une matrice de $n \times n$ expressions régulières et $\underline{B}$ est un vecteur ligne constitués de n expressions régulières[9]. L'expression régulière à la position (j, i) dans la matrice $\underline{\underline{R}}$ est $\alpha_1 + \ldots + \alpha_k$ si $A_j\alpha_1, \ldots, A_j\alpha_k$ sont les termes commençant par A_j dans l'équation définissant A_i. La composante d'indice i dans le vecteur $\underline{B}$ est l'ensemble des termes commençant par des terminaux dans l'équation définissant A_i.

Les équations de définition (4.3) s'écrivent donc

$$[S, B] = [S, B] \begin{bmatrix} aB & \emptyset \\ B & Sa + d \end{bmatrix} + [b, aS + c]. \qquad (4.4)$$

L'idée centrale de la transformation de Rosenkrantz est de poser l'existence de la forme de Greibach par l'introduction d'inconnues supplémentaires

[8] Pour que la définition de la grammaire soit complète, on suppose que la première équation de définition est relative à l'axiome de la grammaire.

[9] Nous supposons le lecteur familier avec la notion d'expression régulière (voir, par exemple, [Hopcroft et al.79]).

dans le système d'équations de définition. Ces inconnues correspondent à de nouveaux non-terminaux dans la grammaire. Cette transformation repose sur le théorème suivant.

Théorème 4.2 *Soit* $\underline{A} = \underline{A}\,\underline{\underline{R}} + \underline{B}$ *un système d'équations de définition sur* Σ *et* Δ, *et* $\underline{\underline{C}}$ *une matrice de la taille de* $\underline{\underline{R}}$ *, telle que chaque élément de* $\underline{\underline{C}}$ *soit un nouveau symbole. Le double système d'équations représenté par* $\underline{A} = \underline{B}\,\underline{\underline{C}} + \underline{B}$ *et* $\underline{\underline{C}} = \underline{\underline{R}}\,\underline{\underline{C}} + \underline{\underline{R}}$ *a un point fixe minimal qui correspond, sur* Δ, *à celui du système d'équations* $\underline{A} = \underline{A}\,\underline{\underline{R}} + \underline{B}$ *[Aho et al.72].*

Soit $G = (V, \Sigma, P, S)$ une CFG[10], la grammaire $G' = (V', \Sigma, P', S)$ sous GNF est obtenue par l'algorithme de transformation décrit ci-après. Afin d'analyser sa complexité, nous posons $n = |V|$ et p_{max} le nombre maximal de règles ayant un non-terminal donné en partie gauche.

1. Nous écrivons l'ensemble d'équations de définition $\underline{A} = \underline{A}\,\underline{\underline{R}} + \underline{B}$ correspondant à la grammaire G.

2. Soit $\underline{\underline{C}}$ une matrice de $n \times n$ éléments. Nous écrivons le double système d'équations $\underline{A} = \underline{B}\,\underline{\underline{C}} + \underline{B}$ et $\underline{\underline{C}} = \underline{\underline{R}}\,\underline{\underline{C}} + \underline{\underline{R}}$ en introduisant l'ensemble des n^2 nouveaux non-terminaux qui constituent les éléments de la matrice $\underline{\underline{C}}$. Nous désignons par C l'ensemble des nouveaux non-terminaux et nous posons $\hat{V} = V \cup C$. Le système d'équations $\underline{A} = \underline{B}\,\underline{\underline{C}} + \underline{B}$ décrit les règles associées aux non-terminaux de V.

   Dans notre exemple, nous obtenons
   $$[S, B] = [b, aS + c] \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} + [b, aS + c]. \tag{4.5}$$

   Pour chacun des n non-terminaux de V, nous obtenons des règles dont les parties droites commencent nécessairement par un terminal. Il y a au plus p_{max} règles pour chacune des composantes de $\underline{B}$. Chaque non-terminal de V a $\mathcal{O}(np_{max})$ règles. Le premier système d'équations correspond donc à $\mathcal{O}(n^2 p_{max})$ règles pour n non-terminaux en partie gauche.

   Le système d'équations $\underline{\underline{C}} = \underline{\underline{R}}\,\underline{\underline{C}} + \underline{\underline{R}}$ relatif aux nouveaux non-terminaux est dans ce cas
   $$\begin{bmatrix} W & X \\ Y & Z \end{bmatrix} = \begin{bmatrix} aB & \emptyset \\ B & Sa + d \end{bmatrix} \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} + \begin{bmatrix} aB & \emptyset \\ B & Sa + d \end{bmatrix}. \tag{4.6}$$

[10] En toute rigueur, la grammaire est supposée *propre*, c'est-à-dire sans symbole inutile ou inaccessible (voir [Aho et al.72]), et ne contenant pas la règle $S \to \varepsilon$.

Chacun des n^2 nouveaux non-terminaux est défini par une équation qui comporte également au plus np_{max} termes. Chaque non-terminal de C ayant $\mathcal{O}(np_{max})$ règles, le système d'équations (4.6) représente $\mathcal{O}(n^3 p_{max})$ règles. Par exemple, pour le non-terminal Y nous obtenons les règles suivantes.

$$\begin{array}{ll} Y \to & BW \\ Y \to & SaY \\ Y \to & dY \\ Y \to & B \end{array}$$

Par construction, les parties droites de ces règles commencent nécessairement par un terminal ou par un non-terminal de V.

3. Pour chaque règle du type $X \to A\alpha$, avec $X \in C$, $A \in V$ et $\alpha \in (\Sigma \cup \hat{V})^*$, nous remplaçons A par les parties droites des règles qui lui sont associées. Il s'agit de l'*expansion* du non-terminal A dans les règles associées à X. Comme il y a $\mathcal{O}(np_{max})$ règles pour chaque X et $\mathcal{O}(np_{max})$ règles pour chaque non-terminal de V, nous obtenons $\mathcal{O}(n^2 p^2_{max})$ règles pour chaque élément de C. Après cette transformation, la grammaire comporte donc $\mathcal{O}(n^2)$ non-terminaux et $\mathcal{O}(n^4 p^2_{max})$ règles. Les parties droites de ces règles commencent nécessairement par des terminaux puisque cette propriété est déjà garantie pour les règles associées aux éléments de V.

   Dans notre exemple, les règles associées au non-terminal Y deviennent les suivantes.

$$\begin{array}{ll} Y \to & bXW \\ Y \to & aSZW \\ Y \to & cZW \\ Y \to & aSW \\ Y \to & cW \\ Y \to & bWaY \\ Y \to & aSYaY \\ Y \to & cYaY \\ Y \to & baY \\ Y \to & dY \\ Y \to & bX \\ Y \to & aSZ \\ Y \to & cZ \\ Y \to & aS \\ Y \to & c \end{array}$$

4. Nous remplaçons chaque terminal a apparaissant en partie droite d'une règle, sauf s'il s'agit du premier symbole, par un nouveau non-terminal A' et nous introduisons la règle $A' \rightarrow a$. Si nous désignons par D l'ensemble des nouveaux non-terminaux introduits à cette étape, nous obtenons l'ensemble final de non-terminaux $V' = \hat{V} \cup D$. Cette dernière opération garantit la mise sous GNF moyennant l'ajout de $\mathcal{O}(|\Sigma|)$ non-terminaux et $\mathcal{O}(|\Sigma|)$ règles. A titre d'exemple, la règle $Y \rightarrow bWaY$ devient $Y \rightarrow bWA'Y$.

Cette analyse nous conduit à l'énoncé de la propriété suivante.

Propriété 4.1 *Soit $G = (V, \Sigma, P, S)$ une CFG, avec $n = |V|$ et p_{max} le nombre maximal de règles ayant un non-terminal donné en partie gauche.*

Il existe une grammaire G' sous GNF et équivalente à G, qui comporte $\mathcal{O}(n^2 + |\Sigma|)$ non-terminaux et $\mathcal{O}(n^4 p_{max}^2 + |\Sigma|)$ règles. Chacune des étapes de l'algorithme de transformation nécessitant une opération élémentaire pour chaque règle créée, la complexité temporelle de cet algorithme est $\mathcal{O}(n^4 p_{max}^2 + |\Sigma|)$. □

Complexité pratique de la transformation en GNF(*)

La complexité pratique de la mise sous GNF est généralement très inférieure à la borne théorique que nous venons de calculer. En particulier, pour chaque non-terminal A de la grammaire originale, tel qu'il n'y ait aucune règle du type $A \rightarrow a\alpha$, le vecteur $\underline{B}$ a une composante nulle. Chaque composante nulle de $\underline{B}$ réduit de n le nombre de nouveaux non-terminaux à introduire. Remarquons également que seule la troisième étape, c'est-à-dire l'expansion d'un non-terminal, peut provoquer un allongement des règles. De plus, dans la transformation de Rosenkrantz, cette expansion n'a nécessairement lieu qu'une seule fois. En résumé, cette transformation a tendance à augmenter le nombre de non-terminaux tout en allongeant modérément les règles. A l'inverse, la transformation classique (voir [Aho et al.72], pages 153-159) a tendance à conserver le nombre de non-terminaux (la borne théorique est dans ce cas $\mathcal{O}(2n + |\Sigma|)$) tout en allongeant et en multipliant les règles car plusieurs expansions successives sont généralement nécessaires.

Les avantages respectifs des deux algorithmes de transformation ne sont pas directement corrélés avec la complexité pratique de l'expansion dynamique propre à la méthode DYANE. D'une part, la représentation en RTNs permet de factoriser les règles, c'est-à-dire de regrouper des transitions identiques correspondant au début ou à la fin des parties droites des règles associées à un non-terminal donné. D'autre part, indépendamment du nombre total de RTNs, qui correspond au nombre de non-terminaux de la grammaire sous GNF, la croissance du graphe syntaxique dépend du nombre effectif d'hypothèses envisagées au cours du décodage acoustique. Ce point sera détaillé au paragraphe 4.3.4.

Traitement des SCFGs et des grammaires attribuées (*)

La méthode DYANE s'applique également au traitement des SCFGs. Lors de l'expansion dynamique du graphe syntaxique, les probabilités associées aux règles sont attachées aux transitions correspondantes. La méthode DYANE offre d'ailleurs une technique d'estimation aisée de ces probabilités. Il suffit d'effectuer le développement dynamique du graphe syntaxique en analysant un corpus d'apprentissage. Une estimation des probabilités des transitions est alors obtenue par comptage des transitions utilisées[11].

Par ailleurs, la mise sous GNF permet de conserver la distribution de probabilité définie sur Σ^* par une SCFG. Ce résultat a été démontré par Fu et Huang pour la transformation classique [Fu et al.71] et par Kochman et Kupin pour la transformation de Rosenkrantz [Kochman et al.92]. Ces derniers auteurs démontrent l'existence d'une correspondance bijective entre les arbres de dérivation de la grammaire originale et de la grammaire transformée. Cette transformation particulière est donc "réversible", car il est possible de retrouver la structure syntaxique originale à partir de la structure syntaxique transformée. Cette propriété garantit l'équivalence de la probabilité affectée à chaque dérivation. Dans la mesure où la grammaire originale serait attribuée (s'il s'agit d'une DCG[12], par exemple), cette propriété permet de préserver également l'information contenue dans les attributs.

4.3.4 Contrôle de l'espace de recherche (*)

Dans la plupart des utilisations pratiques d'un système de reconnaissance comportant éventuellement plusieurs milliers de mots, une stratégie d'élagage est appliquée (voir paragraphe 2.2.6). Une propriété intéressante de la méthode DYANE est que l'élagage, effectué sur les chemins parcourus au niveau acoustique, limite directement la taille du graphe syntaxique développé. En d'autres termes, la sous-optimalité introduite au niveau acoustique

[11] L'estimation exacte des probabilités, selon le critère du maximum de vraisemblance, dans le cas d'une grammaire ambiguë peut être obtenue par l'algorithme "Inside-Outside" [Baker79, Lari et al.90]. La technique d'estimation proposée constitue une application du critère de Viterbi (voir paragraphe 2.2.3).

[12] L'abréviation DCG pour "Definite Clause Grammar" provient de la terminologie anglaise. Nous avons mentionné ce type de formalisme au paragraphe 1.2.1, sous le nom de grammaire à clauses définies.

permet également de réduire à une croissance, en pratique linéaire (voir paragraphe 5.1.2), la taille du graphe syntaxique. Le recours à un modèle de langage hors-contexte n'introduit pas de "sous-optimalité supplémentaire". Il y a donc un contrôle partagé et cohérent de l'espace de recherche : la structure syntaxique guide le décodage acoustique en prédisant à chaque nœud activé les successeurs corrects, et les chemins réellement parcourus au niveau acoustique déterminent le nombre de nœuds syntaxiques activés. Cela nous semble constituer une intégration idéale entre les connaissances acoustiques, lexicales et celles modélisées par un modèle de langage hors-contexte.

4.3.5 Prédictions dialogiques (*)

A l'issue de la reconnaissance d'une phrase, nous disposons d'un graphe syntaxique de taille finie mais pouvant encore être développé récursivement. Deux stratégies peuvent alors s'appliquer. La première consiste à réinitialiser le graphe syntaxique en repartant du RTN associé à l'axiome de la grammaire. Cela présente l'avantage de libérer l'espace mémoire correspondant au graphe développé. La seconde possibilité consiste à contrôler la reconnaissance de la phrase courante à partir du graphe développé à la fin de la phrase précédente. Pour peu que l'on ait recours à une liste des états actifs (voir paragraphe 2.2.6), cette seconde stratégie permet de ne pas recommencer les traitements syntaxiques (copies des RTNs) déjà effectués, tout en ne parcourant que la partie utile du graphe. De nouvelles extensions du graphe sont évidemment permises.

Un ensemble de variantes des deux stratégies évoquées est envisageable. Une approche intéressante consiste à choisir la stratégie optimale en fonction de l'état du dialogue. Par exemple, si le système demande à l'utilisateur "Veuillez répéter, s'il-vous-plaît.", la seconde stratégie sera clairement préférable.

L'état du dialogue peut contraindre les développements du graphe syntaxique. En particulier, si des non-terminaux désignent des classes ou concepts sémantiques, il est aisé de mettre en œuvre des *prédictions dialogiques* : seul le développement des non-terminaux pertinents selon l'état du dialogue sera autorisé. L'usage de prédictions dialogiques offre le double avantage de réduire la complexité calculatoire du décodage et d'améliorer le taux de reconnaissance, en supposant que l'utilisateur se conforme aux prédictions du système.

4.4 Relations avec d'autres travaux

Le développement dynamique d'un graphe fini, principe de base de la méthode DYANE, a été exposé dans [Dupont et al.89]. Nous avons repris ces idées et présenté les liens avec le décodage Viterbi dans [Dupont et al.90, Dupont et al.91]. Quelques-uns des résultats expérimentaux, exposés au paragraphe 5, ont été présentés dans [Dupont93a, Dupont93b].

La représentation en réseaux récursifs pour l'analyse de la langue naturelle est due à Woods [Woods70]. Il propose également les *réseaux augmentés*, ou ATNs[13], qui comportent des registres supplémentaires et des actions appliquées à ces registres. Ces derniers modèles sont équivalents à une machine de Turing et l'adjonction de registres pourrait constituer une extension intéressante de notre approche. Ils permettent de traiter de façon aisée des dépendances contextuelles comme l'accord des genres voire des dépendances sémantiques. Une représentation en ATNs est présente, par exemple, dans le système Myrtille II [Pierrel82], mais le modèle de langage y est utilisé de façon non intégrée dans le décodage acoustique.

Godin a introduit la notion d'*automate fini hiérarchisé* [Godin90]. Une représentation similaire est présentée dans [Young et al.88]. Bien que ces automates comportent des transitions étiquetées par des non-terminaux, la récursivité imbriquée en est exclue. Ils sont donc équivalents aux grammaires régulières.

Plus récemment, différents systèmes utilisent une représentation en réseaux récursifs [Kai et al.92, Lee93]. En particulier, la notion de développement dynamique d'un réseau fini sous contrôle d'une recherche en faisceau est présentée explicitement dans [Kai et al.92]. Aucun de ces travaux ne mentionne la nécessité d'éliminer la récursivité à gauche et les avantages qui résultent de la mise préalable sous GNF. La notion de contrôle de l'expansion dynamique en fonction de l'état du dialogue (voir paragraphe 4.3.5) est, par contre, évoquée dans [Lee93].

Un certain nombre de travaux reposent sur l'idée que les algorithmes classiques d'analyse hors-contexte sont adaptables au problème présent. Notons que cela nécessite d'ajouter à la complexité calculatoire du décodage acoustique celle de l'algorithme d'analyse. Dans notre cas, la seule opération qui relève proprement de l'utilisation d'une CFG est la copie des réseaux associés aux transitions non terminales activées. La complexité globale dépend du parcours de l'espace de recherche activé, lequel parcours doit être effectué quelque soit l'algorithme d'analyse.

[13] L'abréviation **ATN** pour "Augmented Transition Network", provient de la terminologie anglaise.

Les techniques efficaces d'analyse des CFGs sans restriction sont les algorithmes d'Earley [Earley70] et CYK [Younger67]. Il s'agit de méthodes dites de *chart parsing* parce qu'elles utilisent une structure intermédiaire, appelée *chart.* Un chart est une table d'objets, appelés *items*, du type $< A \to \alpha, i, j >$ où $A \to \alpha$ désigne une production particulière et i, j les positions, dans la phrase, de la séquence $W_{i \ldots j}$ dérivée du non-terminal A (voir figure 4.5). Ces méthodes d'analyse tirent leur efficacité du fait que chaque item ne doit être construit qu'une seule fois (pour $A \to \alpha, i, j$ fixés) et peut être utilisé à plusieurs reprises pour construire différents arbres de dérivation partiels ou globaux (cas d'ambiguïté locale ou globale).

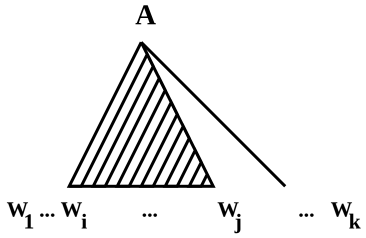


FIG. 4.5: Item d'un chart.

Nous considérons que les techniques de *chart parsing* ne sont pas bien adaptées au contrôle du décodage Viterbi, tel que nous l'avons présenté, pour les raisons suivantes. Premièrement, le contrôle syntaxique du décodage ne fait pas appel à un processus d'*analyse* proprement dit, mais de *prédiction.* Il s'agit, pour chaque séquence partielle de mots, de prédire les mots qui peuvent suivre et idéalement d'appliquer le principe d'optimalité. En particulier, l'utilisation d'un chart permet de traiter efficacement les ambiguïtés locales ou globales, c'est-à-dire l'indéterminisme au cours de l'analyse. Mais, comme Godin l'a souligné [Godin90], l'indéterminisme ne gène pas la génération d'hypothèses, c'est-à-dire la prédiction. Cette propriété est également évoquée par Ney qui remarque que la complexité calculatoire de l'algorithme CYK intégré dans le décodage acoustique est la même pour une grammaire ambiguë ou non ambiguë [Ney91]. Deuxièmement, l'intégration complète dans le décodage acoustique d'un chart et l'application du principe d'optimalité suppose de considérer des items étendus $< A \to \alpha, t_i, t_j, s >$, $t_i \leq t_j$, où s désigne le score d'alignement entre la trame t_i et la trame t_j [Ney91]. En effet, bien que la construction d'un item correspondant à une séquence $W_{i \ldots j}$ dérivée du non-terminal A, ne doive pas être reproduite pour différents alignements de cette séquence, il est cependant essentiel de dupliquer les items pour chaque alignement possible si l'on désire garantir l'optimalité (combinaison optimale des scores à chaque trame pour chaque contexte syntaxique). Cela pose plusieurs problèmes. D'une part, si l'on désire conserver le caractère *synchrone* de l'algorithme de décodage, nous ne pouvons que considérer des items du type $< A \rightarrow \alpha, t_i, t_j, s >$, pour $1 \leq t_i \leq t_j \leq n$, jusqu'à l'instant n. Cela nécessite une formulation synchrone de l'algorithme CYK et du décodage acoustique [Ney91, Jelinek et al.92]. Cet algorithme doit calculer et mémoriser également les scores des alignements commençant aux trames t_i, pour $1 < t_i \leq t_j$[14]. D'autre part, la taille du chart ne dépend plus du nombre (a priori inconnu) de mots de la phrase, mais bien du nombre de trames. Ces deux modifications peuvent s'avérer particulièrement coûteuses en espace mémoire et en temps (notamment, l'accès à chaque item). Finalement, même dans les formulations synchrones des techniques de chart parsing, tous les items ne couvrent pas la même portion du signal. Cela complique particulièrement l'application d'une stratégie d'élagage, sur base des scores acoustiques, des items à considérer. En d'autres termes, l'intégration du modèle de langage dans le décodage de Viterbi est idéale par la méthode DYANE parce que le faisceau qui contrôle le développement de l'espace de recherche contrôle, simultanément, le développement de la structure syntaxique. Cette propriété ne peut s'appliquer directement aux techniques de chart parsing.

Godin propose l'adaptation de l'analyse LR[15] pour contrôler le décodage acoustique [Godin90]. Il soulève le problème qu'un automate LR utilisé en prédiction peut générer des hypothèses (séquences partielles de mots) incorrectes qui ne peuvent être éliminées qu'au moment de l'opération de *réduction* propre à ce type d'analyse. Plus fondamentalement, l'algorithme proposé n'est pas optimal même sans stratégie d'élagage. Godin présente également une intégration de l'algorithme d'Earley avec une adaptation synchrone du "Two-Level". Il en résulte une complexité accrue du décodage acoustique ainsi que les inconvénients déjà évoqués du chart parsing.

L'utilisation de tables LR généralisées est étudiée dans [Kita et al.89]. Au niveau de décodage acoustique, cet algorithme s'apparente au "Level-Building" non synchrone. Il est donc de complexité accrue par rapport au décodage que nous avons présenté (i.e. le "One-Pass"). De plus, les auteurs utilisent une structure intermédiaire, appelée *cell*, similaire à un item d'un chart parser avec les inconvénients cités.

Hemphill et Picone ont développé une approche intégrée qui repose sur une grammaire d'unification stochastique [Hemphill et al.89]. L'originalité de leur approche réside dans le fait que les sources de connaissance de tous

[14] Il s'agit en quelque sorte d'un algorithme qui fonctionne par niveaux comme le "Level-Building". Il a donc une complexité plus élevée que celle du "One-Pass".

[15] L'abréviation **LR** pour "Left to right scanning, Rightmost derivation in reverse" provient de la terminologie anglaise.

les niveaux sont exprimées à l'aide de ce formalisme unique. Il en résulte une hiérarchie de grammaires d'unification. Malgré le pouvoir d'expression important des grammaires d'unification, il semble que le modèle testé soit équivalent à une grammaire régulière stochastique (en particulier, la récursivité est exclue des modèles). La technique d'analyse est un chart parsing, où chaque item comporte 8 champs. Comme les auteurs le soulignent, le coût (en particulier, en espace mémoire) d'une telle approche constitue une limitation forte quant à son utilisation dans un système de reconnaissance de parole continue. De plus, nous pensons que si l'on désirait tirer tout le parti d'une grammaire d'unification (non réductible à un modèle hors-contexte, voire régulier), le coût calculatoire de l'opération d'unification deviendrait lui-même prohibitif.

Dans cette étude comparative, nous n'avons mentionné que les approches visant à l'intégration complète des modèles acoustiques, de l'information lexicale et du modèle de langage. De nombreux autres travaux utilisent ces informations de façon plus ou moins découplée avec les inconvénients mentionnés au paragraphe 1.2. Rappelons, en particulier, la perte d'information liée à la représentation intermédiaire, tel un treillis de mots, propre aux approches découplées [Tomita86].

Chapitre 5

Evaluation expérimentale de la méthode DYANE (*)

Nous exposons dans ce chapitre un ensemble d'expériences qui ont été menées afin d'évaluer la méthode DYANE. Les expériences les plus représentatives sont décrites au paragraphe 5.1 car elles concernent la reconnaissance de parole continue et illustrent une comparaison de différents modèles de langage, en particulier régulier ou hors-contexte. En outre, la tâche décrite, bien que de taille relativement réduite, constitue une base de référence pour des travaux menés dans différents domaines (recherches en sciences cognitives [Feldman et al.90], apprentissage de grammaires probabilistes attribuées [Stolcke94], apprentissage de transducteurs [Castellanos et al.94]). Cette tâche servira également de base d'évaluation pour la deuxième partie de notre travail.

Les expériences complémentaires, résumées au paragraphe 5.2, permettent de confirmer, sur diverses bases de données, les résultats obtenus.

5.1 Expériences sur la tâche de Feldman (*)

5.1.1 Description de la tâche et des modèles de langage

La tâche de Feldman [Feldman et al.90] a été conçue afin d'étudier l'acquisition du langage. Elle n'a pas pour objet d'évaluer le processus complet de l'apprentissage du langage mais elle définit un cadre restreint et modifiable à souhait, dans lequel l'apprentissage peut être étudié. Dans sa formulation originale, cette tâche consiste à construire un système automatique capable de décider comme vraie ou fausse une assertion concernant la position d'objets sur un écran. Par exemple, nous avons les assertions suivantes :

"Un cercle est à droite d'un triangle",
"Un triangle et un cercle sont au-dessus d'un carré".

Dans cette partie de notre travail, nous utilisons la tâche de Feldman comme base d'évaluation d'un module de reconnaissance. Ce module a été intégré dans un système complet capable de traduire un énoncé en langue espagnole vers un énoncé équivalent en allemand, en anglais ou en une représentation logique [Jiménez et al.94]. Par conséquent, le module de reconnaissance reçoit en entrée un énoncé espagnol[1].

Nous comparons ici la reconnaissance contrôlée par trois modèles de langage différents. Le modèle de langage $LM0$ n'introduit aucune contrainte sur les séquences de mots. Il permet d'obtenir le taux de la reconnaissance basée uniquement sur l'information acoustique et lexicale. Le modèle de langage $LM1$ repose sur une grammaire régulière qui accepte, dans ce cas précis, un langage fini comportant de l'ordre de $1,7 \times 10^7$ phrases. Ce langage est celui de la tâche originale enrichie d'actions portant sur les objets. Cette grammaire est décrite complètement en annexe (voir paragraphes A.1 et A.2). Outre les positions d'objets, elle permet de modéliser des phrases du type

"Eliminer le triangle à droite du carré",
"Ajouter un carré de taille moyenne au-dessus du cercle clair et du petit triangle sombre".

Nous avons défini un modèle de langage $LM2$ basé sur une grammaire hors-contexte (voir paragraphes A.3 et A.4) qui accepte un langage plus riche que celui accepté par $LM1$. En particulier, il permet de tenir compte d'inversion dans l'ordre des compléments telle que

"Ajouter, au dessus du carré clair, un triangle",

d'associer une relative à un substantif, comme par exemple

"Un triangle, qui touche un carré clair, est au-dessus d'un cercle",

et de traiter des relatives imbriquées telles que

"Un triangle, qui touche un carré qui est au-dessus du cercle, est en-dessous d'un petit carré".

Les récursivités introduites dans le modèle $LM2$ rendent le langage accepté de cardinalité infinie.

[1]Pour la clarté de l'exposé, nous donnons des exemples d'énoncé en français. L'implémentation du module de reconnaissance réellement utilisé concerne la langue espagnole. Les grammaires françaises et espagnoles sont détaillées en annexe A.

5.1.2 Evaluations expérimentales (*)

Description des modèles

Le module de reconnaissance, utilisé pour les expériences sur la tâche de Feldman, est basé sur des HMMs discrets associés à 26 phonèmes indépendant du contexte et un dictionnaire[2] à 128 symboles. Le lexique comporte 29 mots dont les modèles sont obtenus par concaténation des modèles de phonèmes. Un ensemble de 120 phrases prononcées par dix locuteurs constitue la base d'apprentissage des modèles acoustiques. La base de test est faite de 100 phrases indépendantes et prononcées par 4 locuteurs distincts. L'évaluation est réalisée en mode indépendant du locuteur et de la tâche[3].

Le modèle de langage $LM0$ est décrit par un automate comportant 1 état et 29 transitions. Le modèle de langage $LM1$, correspondant à la grammaire détaillée dans l'annexe A.2, est décrit par un automate comportant 57 états et 149 transitions. La grammaire originale décrivant le modèle $LM2$ est détaillée dans l'annexe A.4. Elle comporte 75 règles et 39 non-terminaux. Après mise sous GNF, la grammaire résultante comporte 144 règles et 37 non-terminaux. Ils correspondent à 37 RTNs représentant un ensemble de 159 états, 65 transitions terminales et 123 transitions non-terminales. Pour tous les modèles de langage testés, toutes les transitions issues d'un état donné sont posées équiprobables.

Résultats expérimentaux

La table 5.1 donne la perplexité[4] des modèles de langage, les taux d'erreur au niveau mot et au niveau phrase. Le modèle $LM2$ étant plus général que le modèle $LM1$,il n'est pas surprenant d'obtenir des taux d'erreur supérieur. Notons cependant que la différence de perplexité calculée sur la base de test est faible, mais celle-ci mesure le pouvoir de prédiction du modèle de langage indépendamment des possibilités de confusion acoustique entre les mots.

[2]Dans le cas d'une modélisation discrète, le *dictionnaire* désigne l'ensemble des vecteurs prototypes auxquels sont assignés les vecteurs acoustiques (chaque vecteur étant remplacé par son prototype le plus proche dans l'espace acoustique).

[3]Bien qu'il s'agisse en principe d'une expérimentation indépendante du locuteur et de la tâche, la simplicité des modèles et le volume restreint de données d'apprentissage des modèles acoustiques rend la modélisation acoustique perfectible.

[4]La perplexité du modèle $LM0$ est de 30 et non de 29 (la taille du lexique). Ceci est dû au fait que le modèle de langage a également pour objet de prédire si la fin d'une phrase est atteinte ou non. Il y a, par conséquent, un élément implicite supplémentaire dans le lexique. Cet élément correspond au symbole de fin de phrase.

Modèle	Perplexité	Erreurs mots	Erreurs phrases
LM0	30,00	42,88%	100,00%
LM1	2,53	1,88%	17,50%
LM2	3,35	4,31%	24,25%

TAB. 5.1: Résultats de reconnaissance sur la tâche de Feldman.

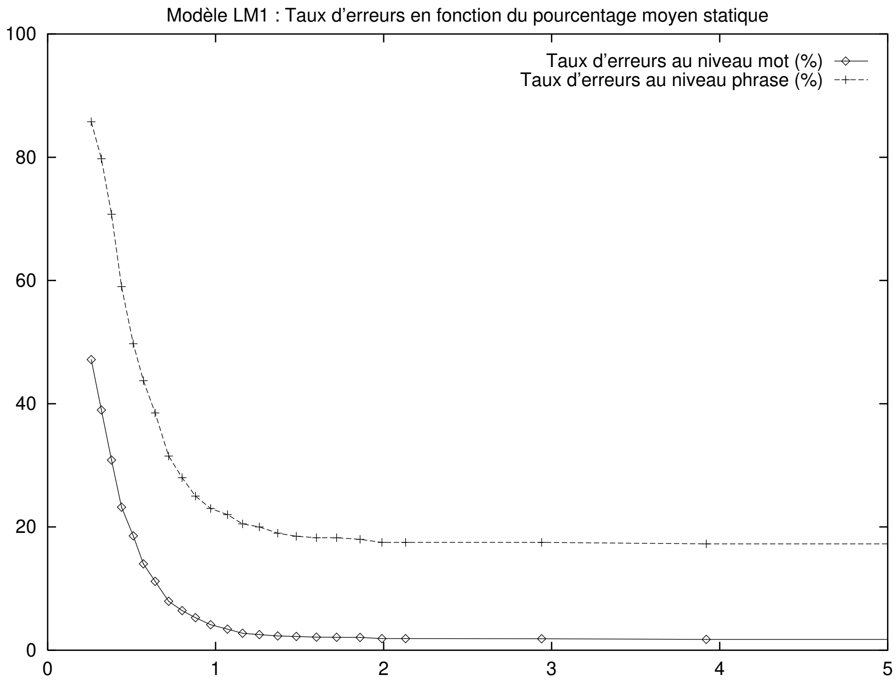


FIG. 5.1: Modèle $LM1$: taux d'erreurs à seuil d'élagage variable.

Nous allons détailler l'influence du seuil d'élagage sur les performances de reconnaissance. Dans le cas du modèle $LM1$, le graphe complet est un réseau fini de taille connue et nous pouvons calculer le nombre d'états actifs en moyenne à chaque trame, par rapport au nombre total d'états. Nous appelons ce rapport le *pourcentage moyen statique* d'états actifs. Lorsque l'on diminue le seuil Δ (voir paragraphe 2.2.6), le taux d'erreur[5] est susceptible d'être augmenté en raison de la sous-optimalité de la recherche. Nous constatons, en pratique, que nous pouvons réduire sensiblement l'espace de recherche sans modifier significativement le taux d'erreur. Nous appellerons *seuil critique* $\Delta_{critique}$, la valeur minimale du seuil Δ pour laquelle le taux d'erreur n'est pas significativement augmenté[6]. La figure 5.1 donne la variation des taux

[5]Les taux d'erreur au niveau mot correspondent à l'addition des taux d'insertion, de substitution et d'omission de mots.

[6]Les résultats présentés à la table 5.1 ont été obtenus en ajustant Δ à sa valeur critique,

d'erreurs au niveau mot et phrase en fonction de la taille de l'espace de recherche. Au seuil critique, l'espace de recherche actif est de l'ordre de 2 % de l'espace total. Il y correspond un taux d'erreurs de 1,88 % au niveau mot alors que le taux d'erreur obtenu sans élagage est de 1,76 %. Notons que l'espace de recherche exploré sans élagage représente 84,4 % du graphe complet et non 100 %. Cette propriété est due à la topologie des modèles qui ne permet pas d'activer tous les états dès les premières trames du décodage.

Dans le cas du modèle $LM2$, le graphe complet n'est borné que par la longueur de la phrase (voir paragraphe 4.3.2). Nous calculons ici le rapport moyen de l'espace de recherche activé au cours du décodage par rapport à la taille de l'espace développé par expansion dynamique. Nous appelons ce rapport le *pourcentage moyen dynamique*. Remarquons que dans le cas d'un modèle ne présentant aucune possibilité d'expansion dynamique, le pourcentage moyen dynamique correspond exactement au pourcentage moyen statique. La figure 5.2 décrit l'évolution des taux d'erreurs en fonction du pourcentage moyen dynamique.

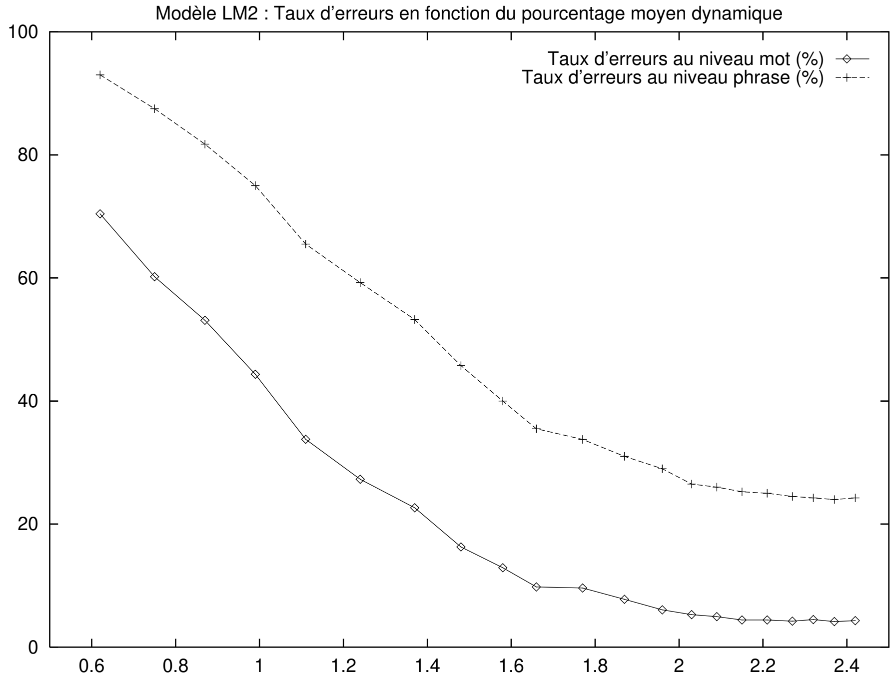


FIG. 5.2: Modèle $LM2$: taux d'erreurs à seuil d'élagage variable.

Les temps de reconnaissance[7] sont directement proportionnels à la taille de l'espace de recherche activé sous contrôle de l'élagage, comme l'illustrent

pour le modèle $LM1$ et en conservant cette valeur pour les autres modèles.

[7]Il s'agit du *facteur temps-réel* obtenu sur station IBM/RS6000 3AT. Ce facteur corres-

Modèle	Etats actifs	Temps calcul	Etats actifs/unité de temps
LM0	237	0,08	2962
LM1	47	0,02	2350
LM2	554	0,21	2638

TAB. 5.2: Temps de reconnaissance.

les résultats présentés à la table 5.2. La seconde colonne y présente le nombre moyen d'états actifs par trame, pour un seuil Δ ajusté à sa valeur critique, et la troisième colonne présente le temps calcul. La dernière colonne est le rapport des deux précédentes. Elle indique le nombre moyen d'états qui sont traitables par unité de temps[8]. Nous constatons que l'expansion dynamique, propre à l'utilisation du modèle $LM2$, ne diminue pas le nombre moyen d'états actifs traitables par unité de temps. Cette propriété a été confirmée par une analyse fine du temps consacré à chaque opération du décodage acoustique. Il s'avère que l'expansion dynamique représente 0,4 % de la charge de calcul de la reconnaissance.

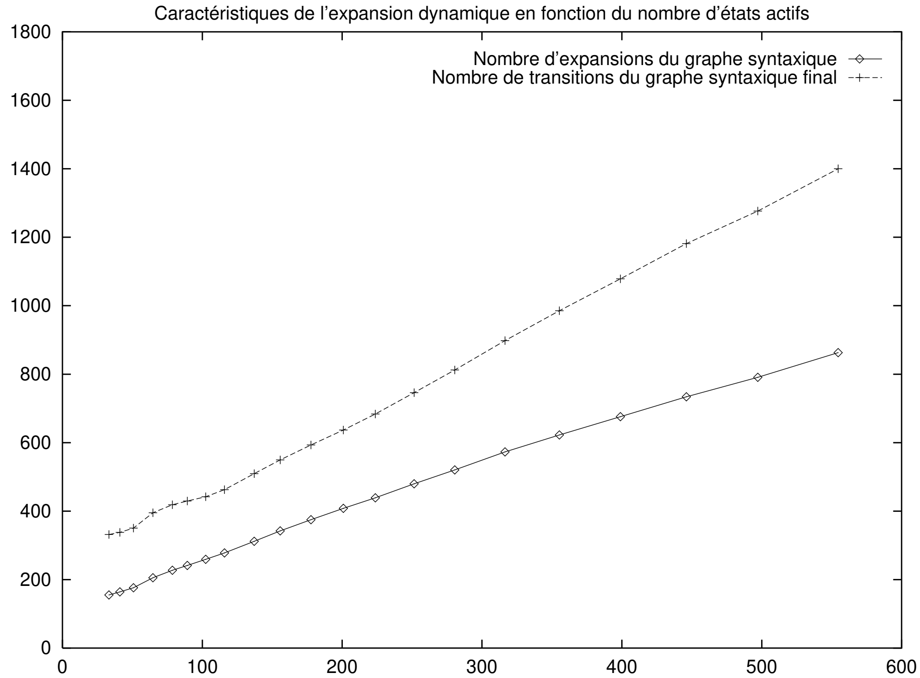


FIG. 5.3: Modèle $LM2$: évolution du graphe syntaxique.

pond au temps de la reconnaissance par rapport au nombre de trames observées. L'analyse acoustique et la quantification vectorielle ne sont pas pris en compte dans cette mesure. Les tests étant effectués en mode différé, il est possible d'obtenir des facteurs temps-réel inférieur à l'unité.

[8]L'unité de temps correspond ici à la longueur d'une trame acoustique qui, dans ce module de reconnaissance, est de 10 ms.

La figure 5.3 présente les caractéristiques du développement du graphe syntaxique. Il y apparaît une dépendance linéaire du nombre d'expansions effectuées et de la taille du graphe obtenu à la dernière trame, en fonction du nombre moyen d'états actifs.

Finalement, nous avons étudié la dépendance du nombre d'expansions du graphe syntaxique et le temps calcul en fonction de la durée du signal de parole. Nous avons vu au paragraphe 4.3.2 que la complexité du décodage avec développement dynamique du graphe syntaxique est donnée par $\mathcal{O}(\sum_{i=1}^{N} T^i)$, où T^i désigne le nombre de transitions du modèle développé jusqu'à la trame i. Il est intéressant de mesurer l'impact sur le temps calcul de l'évolution du graphe syntaxique en fonction de N, le nombre de trames du signal de parole. Ces résultats[9], présentés aux figures 5.4 et 5.5, sont obtenus avec un seuil d'élagage ajusté à la valeur critique. Nous constatons que la complexité pratique de la méthode DYANE se rapproche d'un comportement linéaire.

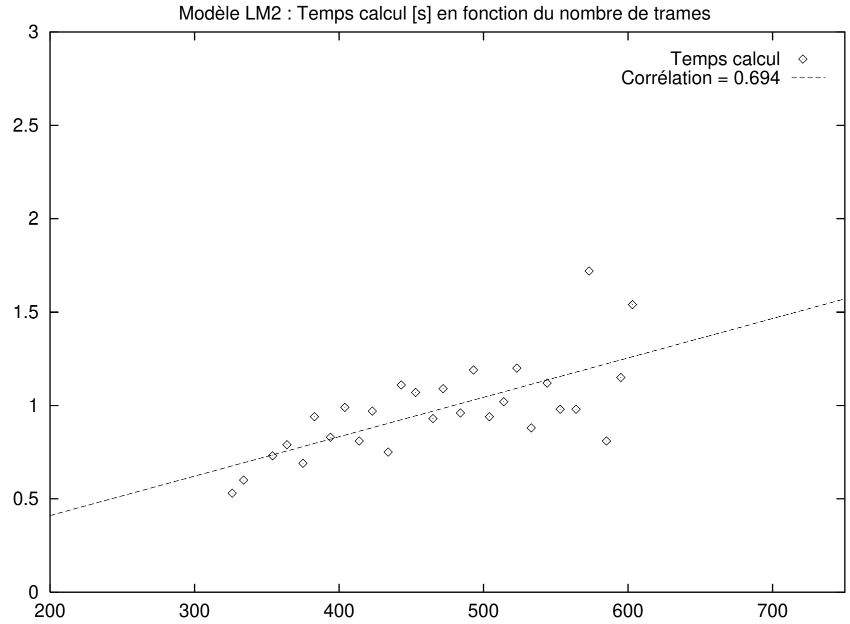


FIG. 5.4: Modèle $LM2$: complexité temporelle de la méthode DYANE.

[9]Nous divisons l'échelle des temps en fenêtres de 10 trames. Chaque point des figures 5.4 et 5.5 représente une moyenne pour tous les enregistrements de parole dans une fenêtre donnée. Seuls les points représentant au moins 5 enregistrements sont considérés. Une droite de régression linéaire est calculée et le coefficient de corrélation de la régression est mentionné.

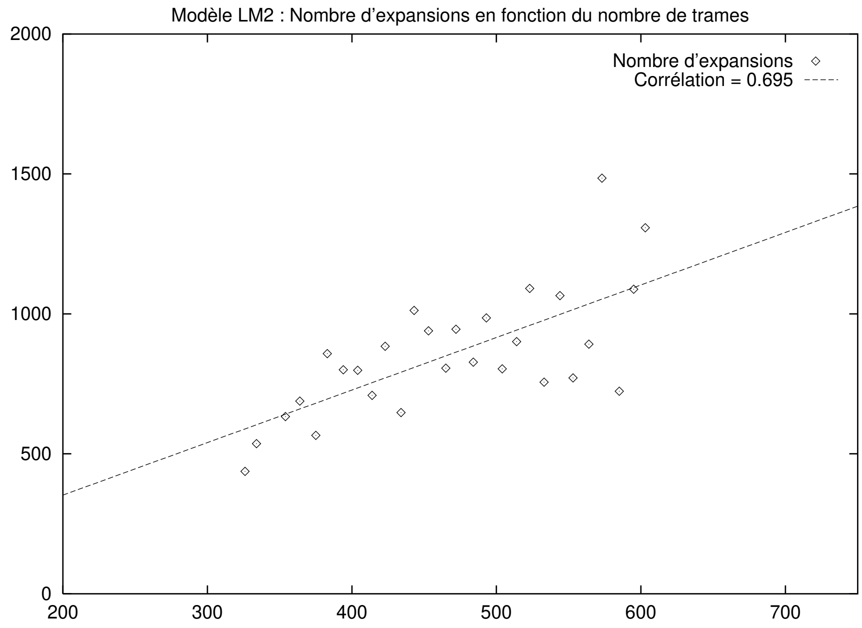


FIG. 5.5: Modèle $LM2$: complexité en nombre d'expansions.

5.2 Expériences complémentaires sur diverses bases de données (*)

5.2.1 Influence de la recherche en faisceau en mots isolés(*)

Les expériences mentionnées dans ce paragraphe concerne la reconnaissance de mots isolés, car nous désirons estimer l'influence de l'élagage au niveau acoustique, sans effet d'un modèle de langage. Les conclusions tirées sont identiques à celle présentées en parole continue.

La première expérience concerne la reconnaissance, indépendante du locuteur, à travers le réseau téléphonique, en mots isolés dans un lexique de 36 mots. La modélisation acoustique est basée sur des HMMs associés aux mots du lexique. Ces résultats ont été évalués sur un ensemble de test comprenant 12844 prononciations réparties sur environ 400 locuteurs. Le taux d'erreur au niveau du mot passe de 1,09 % sans élagage à 1,28 % lorsque le seuil d'élagage est ajusté à sa valeur critique. Dans ce cas, le pourcentage moyen statique est de 8,9 % de l'espace complet.

La seconde application étudiée concerne la reconnaissance plurilocuteur[10], à travers le réseau téléphonique, de mots isolés dans un lexique de 500 mots. Dans ce cas, la modélisation acoustique repose sur des HMMs associés à des allophones (phonèmes en contexte [Jouvet et al.91]), et les modèles associés aux mots sont obtenus à partir des allophones constitutifs. Par l'introduction de règles phonologiques (variantes de prononciations), chaque mot est représenté par un graphe de phonèmes et chaque phonème en contexte par son HMM associé. L'ensemble de test comprend 4883 prononciations.

Nous retrouvons les mêmes propriétés que celles observées dans les expériences précédentes. En particulier, l'espace de recherche actif peut être réduit à 12,5 % de l'espace complet sans modifier le taux d'erreur de 6,66 % au niveau du mot. Une autre propriété intéressante est que la valeur du seuil critique $\Delta_{critique}$ est pratiquement égale pour ces deux applications bien que les espaces de recherche et les modélisations acoustiques soient totalement différents. Cette propriété s'est avérée reproductible, pour un système de reconnaissance donné, sur d'autres bases d'évaluation avec diverses modélisations acoustiques et divers modèles de langage. Cela peut s'expliquer par le fait que le seuil Δ correspond à un écart de score, c'est-à-dire à un rapport de probabilités. Avec la modélisation acoustique reposant sur un modèle de Markov, il semble donc exister un rapport critique, relativement indépendant de l'application, sur les produits de probabilités des événements à considérer.

5.2.2 Intérêt de l'intégration complète d'un modèle de langage (*)

Une autre expérience a pour objet de vérifier l'utilité d'une approche intégrée par rapport à une utilisation différée de l'information contenue dans un modèle de langage. En outre, cette expérience permet d'évaluer la méthode DYANE pour une grammaire de taille importante [Dupont93a].

Nous disposons d'une base de données d'épellations de noms propres (en particulier, des noms de villes). Pour cette application le lexique correspond aux 26 lettres de l'alphabet et quelques "mots" supplémentaires (silences et chiffre 2 utilisé dans l'épellation des lettres doublées). Le lexique complet comporte 30 mots. A chacun des mots est associé un HMM. L'ensemble de test est constitué de 1641 épellations de noms de ville (correspondant à 11400 mots). Il s'agit d'un système de reconnaissance, indépendant du locuteur, de parole téléphonique, prononcée de façon continue. De plus, nous disposons

[10] En pratique, 10 locuteurs ont prononcé 3 répétitions du lexique de 500 mots. Les deux premières prononciations ont servi à l'apprentissage d'un modèle regroupant tous les locuteurs et la troisième prononciation a constitué la base de test.

d'une liste finie de 5000 villes à partir de laquelle nous pouvons construire un modèle de langage. Dans ce cas particulier, le langage est fini et nous pourrions donc, en principe, représenter ce modèle par un seul automate fini. Néanmoins, nous avons défini une grammaire équivalente mise sous GNF, comportant 55 non-terminaux et 5054 productions. En moyenne, la partie droite de chaque production comporte 6 non-terminaux, chacun correspondant à une lettre suivie d'un silence optionnel.

Sans modèle de langage, le taux d'erreur défini au niveau des séquences de lettres est de 57,2 %. Grâce au contrôle de l'espace de recherche par un graphe dynamiquement développé, le taux d'erreur au niveau des séquences passe à 3,0 %. Alors que le nombre maximal d'expansions de RTNs par séquence reconnue est de l'ordre de 30000, le nombre moyen d'expansions est de 1642.

Le développement dynamique à partir du réseau initial permet de ne considérer que le fragment utile du graphe complet, qui, dans ce cas particulier est de taille finie. Remarquons, que du point de vue du développement dynamique, il n'y a pas de réelle différence entre un modèle réductible à un automate fini et un modèle véritablement hors-contexte. Dans les deux cas, la méthode DYANE permet de développer un graphe fini de taille bornée en fonction de la longueur de la phrase et sous contrôle du seuil d'élagage.

Cet ensemble de test nous permet d'illustrer l'intérêt d'une intégration complète du modèle de langage. Nous comparons notre approche avec une approche découplée où la reconnaissance est effectué sans contrainte syntaxique (toutes les séquences de lettres sont permises) suivie d'une technique d'analyse corrective statistique. Un HMM discret est appliqué sur la séquence reconnue pour trouver la séquence la plus probable parmi les 5000 séquences possibles [Jouvet et al.93a]. Le taux d'erreur obtenu après l'analyse corrective est de 6,6 %. Une amélioration de cette approche découplée consiste à utiliser les N meilleures séquences reconnues (typiquement $N = 8$) lors de la première étape et à appliquer l'analyse corrective sur ces N séquences [Jouvet et al.93b]. Le taux d'erreur est alors de 4,1 %. Ce taux d'erreur est supérieur de 36,6 % (en relatif) à celui obtenu par l'intégration complète du modèle de langage, soit 3,0 %.

Troisième partie

Apprentissage de modèles de langage

Chapitre 6

Apprentissage de modèles grammaticaux réguliers (*)

6.1 Introduction

Nous avons décrit au chapitre 3 les différents types de modèles de langages qui peuvent être utilisés dans un système de reconnaissance de la parole. Nous nous intéressons ici à l'apprentissage de ces modèles. Dans cette partie de notre travail, nous optons pour des modèles purement syntaxiques, c'est-à-dire ceux pouvant être décrits par le biais d'une grammaire formelle. Ce problème relève alors de l'*inférence grammaticale* dans le sens classique du terme. Elle consiste en l'apprentissage d'une grammaire représentant un langage à partir d'un ensemble d'exemples. Cet ensemble est formé au moins d'un *échantillon positif*, c'est-à-dire un sous-ensemble fini d'un langage. Nous pouvons également disposer d'un *échantillon négatif*, c'est-à-dire un ensemble fini de chaînes n'appartenant pas au langage.

L'inférence grammaticale a été étudiée depuis le développement de la théorie des grammaires formelles. Outre son intérêt théorique, elle offre un ensemble d'applications potentielles, en particulier dans les domaines de la *Reconnaissance des Formes Syntaxiques et Structurelles*, le *Traitement de la Langue Naturelle* et, dans notre cas, le *Traitement de la Parole*. Nous nous limiterons ici à l'*inférence régulière*, c'est-à-dire l'apprentissage d'une grammaire représentant un langage supposé régulier. En effet, c'est dans ce cadre qu'ont été menés la plupart des travaux jusqu'à ce jour. Nous exposerons à la fin de ce travail quelques extensions existantes ou envisageables.

Nous rappelons au paragraphe 6.2 les notions algébriques de base qui nous permettent de définir l'espace des solutions possibles de l'inférence régulière : langages réguliers, automates finis, partitions d'un ensemble fini, dérivation d'un automate relativement à une partition de son ensemble d'états, treillis d'automates. En particulier, nous étendons la notion classique de *complétude structurelle* d'un échantillon relatif à un automate.

La caractérisation et l'évaluation d'une méthode d'inférence requiert un cadre formel définissant l'objet de l'apprentissage. Nous présentons au paragraphe 6.3 quelques critères d'inférence correcte. En particulier, nous détaillons l'identification exacte d'un langage telle qu'elle a été définie par Gold [Gold67, Gold78].

Nous rappelons au paragraphe 6.4 les principaux résultats de la complexité théorique d'identification d'un langage régulier [Pitt89]. Bien que l'identification exacte d'un langage régulier à partir d'échantillons positif et négatif constitue un problème NP-difficile, nous insistons sur le fait qu'une identification quasi-exacte puisse être obtenue en temps polynomial [Lang92, Oncina et al.92].

Nous consacrons le paragraphe 6.5 à une description formelle de l'inférence régulière. Elle est vue comme un problème de *recherche explicite* d'une généralisation de l'échantillon positif. Nous revoyons ici les théorèmes fondamentaux conformément à notre définition de complétude structurelle. Il en résulte que l'espace de recherche est un treillis d'automates, construit sur la base de l'échantillon positif. Nous étudions certaines propriétés de ce treillis. En particulier, nous montrons que le treillis construit à partir de l'*automate canonique maximal* est plus général que celui construit à partir de l'*accepteur des préfixes*.

Nous définissons ensuite la notion d'*ensemble frontière* (paragraphe 6.6), c'est-à-dire l'ensemble des solutions compatibles les plus générales, et nous étudions quelques propriétés de cet ensemble. Notons que nous avons présenté les principaux résultats des paragraphes 6.5 et 6.6 dans [Dupont et al.94].

Finalement, nous démontrons au paragraphe 6.7 un autre résultat original. Il s'agit de l'existence d'un demi-treillis constitué de tous les automates déterministes du treillis associé à l'accepteur des préfixes.

6.2 Définitions et notations

Le lecteur familier avec les définitions classiques de la théorie des automates peut passer le paragraphe 6.2.1. Nous présentons ces notions par souci de complétude et afin de spécifier les notations utilisées par la suite.

6.2.1 Langages, automates et partitions

Notions de base

Soit Σ un *alphabet fini* et les *lettres* a, b, c des éléments de Σ. Nous désignons par u, v, w, x des éléments de Σ^*, c'est-à-dire des chaînes de longueur finie sur Σ, et par λ la *chaîne vide*. $|u|$ désigne la longueur de la chaîne u.

Définition 6.1 Nous disons que u est un *préfixe* de v s'il existe w tel que $uw = v$.

Définition 6.2 Nous disons que u est un *suffixe* de v s'il existe w tel que $wu = v$.

Définition 6.3 Un langage L est un sous-ensemble quelconque de Σ^*. Les éléments de L sont des chaînes qui, dans ce contexte, sont généralement appelées *mots*[1].

Définition 6.4 Soit $Pr(L) = \{u|\exists v, uv \in L\}$ l'ensemble des préfixes de L. Nous désignons par $L/u = \{v|uv \in L\}$ le *quotient-droit* de L par u, encore appelé *ensemble des finales* de u. Nous avons donc $L/u \neq \phi$ si et seulement si $u \in Pr(L)$.

Automates finis

Définition 6.5 Un *automate fini* est un quintuple $(Q, \Sigma, \delta, q_0, F)$ où Q est un ensemble fini d'*états*, Σ est un *alphabet fini*, δ est une *fonction de transition*, c'est-à-dire une application de $Q \times \Sigma \rightarrow 2^Q$, q_0 est l'*état initial* et F est un sous-ensemble de Q identifiant les *états finaux* ou *d'acceptation*.

Définition 6.6 Si, pour tout q de Q et tout a de Σ, $\delta(q, a)$ contient au plus un élément (respectivement exactement un élément), l'automate A est dit *déterministe* (respectivement *complet*). Par la suite, nous utiliserons l'abréviation **AFD** pour un "automate fini déterministe" et **AFN** pour un "automate fini non-déterministe".

Considérons, par exemple, l'automate A_1, représenté à la figure 6.1. Il comporte 5 états, $Q = \{0, 1, 2, 3, 4\}$. Il est défini sur un alphabet de 2 lettres,

[1]Ces désignations sont classiques en théorie des langages. Elles peuvent néanmoins induire une confusion. Si nous nous référons aux appelations classiques en traitement de la langue naturelle, les mots ou entrées lexicales correspondent ici aux lettres et les phrases correspondent ici aux mots.

$\Sigma = \{a, b\}$. L'état initial[2] q_0 est l'état 0 et les états 3 et 4 sont finaux[3], $F = \{3, 4\}$. La fonction de transition est représentée par les arcs reliant les états. Il s'agit ici d'un automate non déterministe car il a deux arcs étiquetés a à partir de l'état 0. En d'autres termes, $\delta(0, a) = \{1, 2\}$.

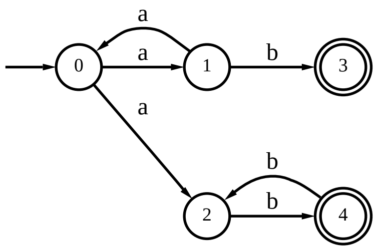


FIG. 6.1: Une représentation de l'automate A_1.

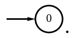
Langage accepté par un automate fini

Définition 6.7 Une *acceptation* d'une chaîne $u = a_1 \ldots a_l$ par un automate A, éventuellement non-déterministe, définit une séquence, éventuellement non unique, de $l + 1$ états $(q^0, \ldots, q^l)$ telle que $q^0 = q_0$, $q^l \in F$ et $q^{i+1} \in \delta(q^i, a_{i+1})$, pour $0 \leq i \leq l - 1$. Les $l + 1$ états sont dits *atteints* pour cette acceptation et l'état q^l est *utilisé comme état d'acceptation*. De façon similaire, les l transitions, c'est-à-dire des éléments de δ, sont dites *exercées* par cette acceptation.

Par exemple, la séquence des états $(0, 1, 0, 2, 4, 2, 4)$ correspond à une acceptation de la chaîne *aaabbb* dans l'automate A_1.

Définition 6.8 Un automate A est *ambigu* s'il existe au moins une chaîne u pour laquelle il y a plusieurs acceptations. Dans ce cas, l'automate A est nécessairement non-déterministe. Un ensemble de transitions exercées par un ensemble de chaînes S_s est l'union des ensembles de transitions *exercées* par une acceptation de chaque chaîne de S_s. Nous dirons que les états associés sont *atteints* par une acceptation de S_s.

L'automate A_1 est ambigu car la chaîne *ab* peut être acceptée par les séquences d'états $(0, 1, 3)$ et $(0, 2, 4)$.

[2]L'état initial est représenté graphiquement par .

[3]Les états finaux sont représentés graphiquement par .

Définition 6.9 Le langage $L(A)$ accepté par un automate A est l'ensemble des chaînes acceptées par A. Le langage L est accepté par un automate A si et seulement s'il constitue un *ensemble régulier*, c'est-à-dire qu'il peut être défini par une *expression régulière*[4] [Aho et al.72].

Dans la suite de ce document, un automate signifiera, par défaut, un automate *fini* et un langage signifiera, par défaut, un langage *régulier*.

Définition 6.10 La différence symétrique $L1 \oplus L2$ des langages $L1$ et $L2$ est définie comme suit :

$$L1 \oplus L2 = (L1 \cup L2) \setminus (L1 \cap L2).$$

Définition 6.11 Un état q d'un automate A est *utile* s'il existe une chaîne u de $L(A)$ pour laquelle l'état q peut être atteint pour une acceptation de u. Sinon, l'état q est *inutile*. Un automate *élagué* ne contient aucun état inutile.

Définition 6.12 L'automate $A(L) = (Q, \Sigma, \delta, q_0, F)$ désigne l'AFD qui possède le nombre minimal d'états pour un langage L donné. $A(L)$ est généralement appelé *automate déterministe minimal* ou *automate canonique* de L, et peut être défini comme suit :

$Q = \{L/u \mid u \in Pr(L)\}$,
$q_0 = L/\lambda$,
$F = \{L/u \mid u \in L\}$,
$\delta(L/u, a) = L/ua$ où $u, ua \in Pr(L)$.

On démontre que $A(L)$ est unique à une permutation près des indices de ses états. Tout automate déterministe acceptant exactement L et possédant le nombre minimal d'états pour ce langage, est donc *isomorphe* à $A(L)$ [Aho et al.72]. Par la suite, nous parlerons indifféremment de l'automate canonique ou de l'automate déterministe minimal de L.

Par exemple, l'automate canonique représenté à la figure 6.2 accepte le langage défini par l'expression régulière $L = a(aa)^*b(bb)^*$. Il s'agit du langage accepté également par l'automate A_1 de la figure 6.1. Il n'existe pas d'automates déterministes comportant moins d'états et acceptant ce langage.

[4]Nous supposons le lecteur familier avec les notations utilisées pour spécifier les expressions régulières [Aho et al.72].

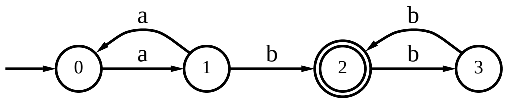


FIG. 6.2: Automate canonique du langage $L = a(aa)^*b(bb)^*$.

Automates dérivés

Définition 6.13 Pour tout ensemble S, une partition π est un ensemble de sous-ensembles de S, non vides et disjoints deux à deux, dont l'union est S. Si s désigne un élément de S, $B(s, \pi)$ désigne l'unique élément, ou *bloc*, de π comprenant s. Une partition π_i *raffine*, ou *est plus fine que*, une partition π_j si et seulement si tout bloc de π_j est un bloc de π_i ou est l'union de plusieurs blocs de π_i.

Définition 6.14 Si $A = (Q, \Sigma, \delta, q_0, F)$ est un automate, l'automate $A/\pi = (Q', \Sigma, \delta', B(q_0, \pi), F')$ *dérivé de* A *relativement à la partition* π de Q, aussi appelé l'*automate quotient* A/π, est défini comme suit :

$Q' = Q/\pi = \{B(q, \pi) | q \in Q\}$,
$F' = \{B \in Q' | B \cap F \neq \phi\}$,
$\delta' : Q' \times \Sigma \to 2^{Q'} : \forall B, B' \in Q', \forall a \in \Sigma, B' \in \delta'(B, a)$ si et seulement si $\exists q, q' \in Q, q \in B, q' \in B'$ et $q' \in \delta(q, a)$.

Nous dirons que les états de Q appartenant au même bloc B de la partition π sont *fusionnés*.

Si nous reprenons l'automate A_1, représenté à la figure 6.1, et que nous définissons la partition de son ensemble d'états, $\pi = \{\{0, 1\}, \{2\}, \{3, 4\}\}$. L'automate quotient A_1/π, obtenu en fusionnant tous les états appartenant à un même bloc de π, est représenté à la figure 6.3[5].

6.2.2 Echantillons d'un langage et automates associés

Définition 6.15 Nous désignons par I_+ un sous-ensemble fini, appelé *échantillon positif*, d'un langage L quelconque. Nous désignons par I_- un sous-ensemble fini, appelé *échantillon négatif*, du langage complémentaire $\Sigma^* - L$. Par conséquent, I_+ et I_- sont des sous-ensembles finis et disjoints de Σ^*.

[5]Remarquons qu'il ne s'agit pas d'un automate quotient de l'automate canonique, acceptant le même langage, représenté à la figure 6.2.

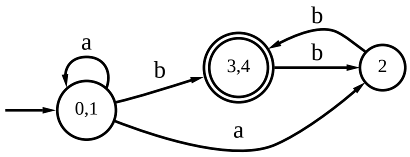


FIG. 6.3: L'automate quotient A_1/π.

Complétude structurelle (*)

Définition 6.16 Un échantillon I_+ est *structurellement complet* relativement à un automate A acceptant L, s'il existe une acceptation $\mathcal{AC}(I_+, A)$ de I_+ telle que :

(1) toute transition de A soit exercée.
(2) tout élément de F (l'ensemble des états finaux de A) soit utilisé comme état d'acceptation.

Les définitions classiques de la complétude structurelle [Fu et al.75a, Miclet79, Miclet80, Angluin82] n'incluent pas la seconde condition. Nous verrons au paragraphe 6.5.1 qu'elle est cependant nécessaire.

A titre d'exemple, l'échantillon $I_+ = \{aab, ab, abbbbb\}$ est structurellement complet relativement à l'automate A_1.

Automate Canonique Maximal, Arbre Accepteur des Préfixes et Automate Universel

Soit $I_+ = \{u_1, \ldots, u_M\}$ un échantillon positif, où $u_i = a_{i,1} \ldots a_{i,|u_i|}, 1 \leq i \leq M = |I_+|$.

Définition 6.17 Nous désignons par $MCA(I_+) = (Q, \Sigma, \delta, q_0, F)$ l'*automate maximal canonique*[6] *relatif à* I_+ [Miclet79]. Il est construit comme suit :

Σ est l'alphabet sur lequel I_+ est défini.
$Q = \{v_{i,j} | 1 \leq i \leq M, 1 \leq j \leq |u_i|, v_{i,j} = a_{i,1} \ldots a_{i,j}\} \cup \{\lambda\}$,
$q_0 = \lambda$,
$F = I_+$,
$\delta(\lambda, a) = \{a_{i,1} | a_{i,1} = a, 1 \leq i \leq M\}$,

[6] L'abréviation **MCA** pour "Maximal Canonical Automaton" provient de la terminologie anglaise. Le qualificatif *canonique* se rapporte ici à un échantillon. Le MCA ne doit pas être confondu avec l'automate canonique d'un langage (voir déf. 6.12).

$\delta(v_{i,j}, a) = \{v_{i,j+1} | v_{i,j+1} = v_{i,j}a\}$, $1 \leq i \leq M, 1 \leq j \leq |u_i| - 1$.

Par conséquent, $L(MCA(I_+)) = I_+$ et $MCA(I_+)$ est le plus grand automate, c'est-à-dire l'automate ayant le plus grand nombre d'états, pour lequel I_+ est structurellement complet. Remarquons que $MCA(I_+)$ est généralement non-déterministe.

A titre d'exemple, l'automate représenté à la figure 6.4 est l'automate maximal canonique relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

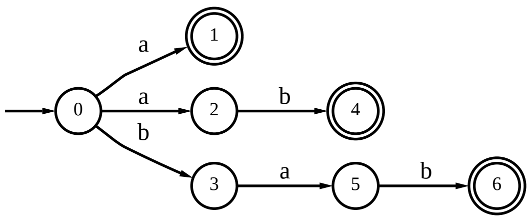


FIG. 6.4: L'automate maximal canonique $MCA(I_+)$, avec $I_+ = \{a, ab, bab\}$.

Définition 6.18 Nous désignons par $PTA(I_+)$ l'*arbre accepteur des préfixes*[7] de I_+ [Angluin82]. Il s'agit de l'automate quotient $MCA(I_+)/\pi_{I_+}$ où la partition π_{I_+} est définie comme suit :

$$B(q, \pi_{I_+}) = B(q', \pi_{I_+}) \text{ si et seulement si } Pr(q) = Pr(q').$$

En d'autres termes, le $PTA(I_+)$ peut être obtenu à partir du $MCA(I_+)$ en fusionnant les états partageant les mêmes préfixes. Remarquons que le $PTA(I_+)$ est déterministe[8].

A titre d'exemple, l'automate représenté à la figure 6.5 est l'arbre accepteur des préfixes relatif à l'échantillon $I_+ = \{a, ab, bab\}$.

Définition 6.19 Nous désignons par UA *l'automate universel*[9]. Il accepte toutes les chaînes définies sur l'alphabet Σ, c'est-à-dire $L(UA) = \Sigma^*$. Il s'agit donc du plus petit automate pour lequel des échantillons de Σ^* peuvent être structurellement complets.

[7] L'abréviation **PTA** pour "Prefix Tree Acceptor" provient de la terminologie anglaise.

[8] Puisque le $PTA(I_+)$ est l'automate déterministe acceptant I_+ et comprenant le plus grand nombre d'états, il pourrait être appelé *automate maximal déterministe*.

[9] L'abréviation **UA** pour "Universal Automaton" provient de la terminologie anglaise.

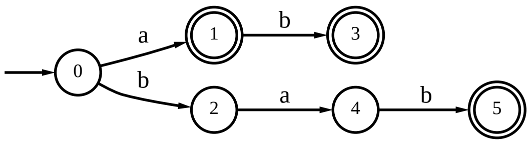


FIG. 6.5: L'arbre accepteur des préfixes $PTA(I_+)$, avec $I_+ = \{a, ab, bab\}$.

L'automate universel défini sur l'alphabet $\Sigma = \{a, b\}$ est représenté à la figure 6.6.

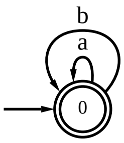


FIG. 6.6: L'automate universel sur l'alphabet $\Sigma = \{a, b\}$.

6.2.3 Treillis d'automates

Définition 6.20 Nous désignons par $P(A)$ l'ensemble des partitions de l'ensemble des états d'un automate A. Soit $r(\pi_i)$, ou simplement r_i, le nombre de blocs de la partition π_i. Supposons que $\pi_1 = \{B_{11}, \ldots, B_{1r_1}\}$ et π_2 soient deux éléments de $P(A)$. La partition π_2 *dérive directement de* π_1 s'il existe j et k, compris entre 1 et r_1 $(j \neq k)$, tels que

$$\pi_2 = \{B_{1j} \cup B_{1k}\} \cup (\pi_1 \setminus \{B_{1j}, B_{1k}\}),$$

Par conséquent, $r_2 = r_1 - 1$.

Cette opération de dérivation définit une relation d'ordre partiel sur $P(A)$. Nous désignons cette relation par $\preceq$. En particulier, nous pouvons écrire $\pi_1 \preceq \pi_2$. Nous désignons également la fermeture transitive de cette relation par $\ll$. En d'autres termes, nous avons $\pi_i \ll \pi_j$ si et seulement si π_i est plus fine que π_j. Par extension, nous disons que A/π_i est *plus fin que* A/π_j ou, réciproquement, que A/π_j *dérive de* A/π_i.

Par construction d'un automate quotient, nous pouvons à présent énoncer la propriété d'inclusion des langages réguliers [Fu et al.75a] :

Propriété 6.1 *Si un automate A/π_j dérive d'un automate A/π_i, alors le langage accepté par A/π_i est inclus dans celui accepté par A/π_j. Formellement, nous avons :*

$$\textit{Si } A/\pi_i \ll A/\pi_j \textit{ alors } L(A/\pi_i) \subseteq L(A/\pi_j).$$

Définition 6.21 L'ensemble des automates partiellement ordonné par la relation $\preceq$ est un *treillis complet complémenté* [10], que nous désignons par $Lat(A)$[11][Hartmanis et al.66]. Les automates A et UA, l'automate universel, en sont respectivement les éléments nul et universel.

Définition 6.22 La *profondeur* d'un automate A/π dans $Lat(A)$ est donnée par la formule $N - r(\pi)$, où N désigne le nombre d'états de A. Dès lors, la profondeur de l'automate A dans $Lat(A)$ est nulle tandis que la profondeur de l'automate universel UA est égale à $N - 1$.

6.3 Objet de l'inférence

Une définition informelle de l'*apprentissage* est, par exemple, "l'acquisition d'une capacité à réaliser une tâche". L'*induction,* ou *inférence inductive,* constitue un type particulier d'apprentissage où un "raisonnement" sur quelques cas observés est appliqué à un ensemble "plus général" de cas. Une définition précise de l'induction requiert une caractérisation des généralisations recherchées. Celles-ci répondent à des critères formellement établis qui définissent ce qu'est un apprentissage correct ou réussi.

L'inférence grammaticale est une instance de l'inférence inductive. Il s'agit d'apprendre une grammaire qui constitue une représentation possible d'un langage, à partir d'exemples de ce langage. En d'autres termes, les cas observés sont ici des exemples, ou chaînes, appartenant à un langage inconnu et la généralisation est le langage complet représenté par la grammaire.

Dans la suite de ce paragraphe, nous nous inspirons de l'article de Angluin et Smith sur les méthodes d'inférence inductive [Angluin et al.83].

6.3.1 La spécification d'un problème d'inférence grammaticale

Afin de définir un problème d'inférence grammaticale, cinq points doivent être spécifiés :

[10] Un *treillis* est un ensemble ordonné E dans lequel tout couple d'éléments admet un plus petit majorant et un plus grand minorant.
Un treillis E est *complet* si tout sous-ensemble E' de E constitue également un treillis.
Un treillis E est *complémenté* s'il possède un élément nul et un élément universel et si tout élément x de E admet au moins un complément.

[11] L'abréviation *Lat* pour "Lattice" provient de la terminologie anglaise.

- La *classe de grammaires* à inférer. Dans cette partie de notre travail, il s'agit essentiellement de la classe des grammaires *régulières*[12] ou d'une de ses sous-classes.

- L'*espace d'hypothèses*, c'est-à-dire l'ensemble des descriptions tel que chaque grammaire de la classe considérée possède au moins une description dans l'espace. Pour les grammaires régulières, il peut s'agir de l'espace des *automates*, des *expressions régulières* ou des *ensembles réguliers*.

- Un *ensemble d'exemples,* pour chaque grammaire à inférer et un protocole de présentation de ces exemples. En particulier, nous définissons au paragraphe 6.3.2 les présentations dites *admissibles*.

- La *classe des méthodes d'inférence* en considération. Il s'agit en particulier des méthodes constructives ou heuristiques (voir paragraphe 6.1).

- Les critères d'une inférence réussie, définis aux paragraphes 6.3.3 et 6.3.4.

Une *méthode d'inférence* est alors un processus calculable qui lit des exemples conformément à une présentation admissible et propose des solutions dans l'espace des hypothèses.

6.3.2 Présentations admissibles

Nous allons à présent définir trois protocoles de présentation d'exemples d'un langage. Remarquons que ces définitions supposent que les langages considérés soient de cardinalité infinie.

Définition 6.23 Une *présentation positive* d'un langage L est une séquence infinie d'exemples, comportant au moins une occurrence de chaque élément de L. On parle également d'*exemples positifs* du langage L.

Définition 6.24 Une *présentation négative* d'un langage L est une séquence infinie d'exemples comportant au moins une occurrence de chaque élément de $\Sigma^* - L$. On parle indifféremment d'*exemples négatifs* ou de *contre-exemples* du langage L.

[12] Les grammaires régulières sont celles qui, par définition, engendrent ou acceptent un langage régulier (voir paragraphe 6.2).

Définition 6.25 Une *présentation complète* d'un langage L est une séquence infinie ordonnée de paires (x, d) de $\Sigma^* \times \{0, 1\}$. Tout élément x de Σ^* apparaît comme premier argument d'une paire de la séquence et $d = 1$ si et seulement si $x \in L$, $d = 0$ sinon.

Un échantillon positif étant un ensemble fini d'éléments d'un langage, nous pouvons considérer une présentation positive comme une séquence infinie d'échantillons positifs inclus et de taille croissante. Une méthode d'inférence séquentielle lit les exemples présentés un par un et propose une nouvelle solution dans l'espace d'hypothèses après chaque nouvel exemple lu. Notons que telles qu'elles ont été définies les présentations ne supposent pas d'ordre particulier sur les exemples. Ces présentations sont dénommées *admissibles*. Une présentation *non admissible* pourrait alors être définie récursivement de la manière suivante : toute séquence d'exemples telle que l'exemple suivant présenté contredise nécessairement la solution proposée par la méthode d'inférence, sur base des exemples déjà lus.

Une autre présentation admissible d'exemples d'un langage est la *présentation par oracle*. Un oracle est, par définition, capable de répondre à des questions concernant le langage. Par exemple, la méthode d'inférence propose un exemple et l'oracle répond à une *question d'appartenance*, c'est-à-dire s'il s'agit ou non d'un élément du langage. En théorie, cette présentation par oracle est équivalente à une présentation complète mais ce type de présentation modifie profondément la conception des algorithmes d'inférence. En effet, nous ne disposons en pratique que d'un sous-ensemble *fini* d'une présentation positive ou complète. L'existence d'un oracle permet alors de classifier, à la demande, des exemples n'appartenant pas aux échantillons observés.

6.3.3 L'identification à la limite

L'*identification à la limite* a été proposée par Gold [Gold67]. Il s'agit d'un critère d'inférence correcte défini par le biais d'un protocole théorique d'inférence. De nombreux résultats découlent de ce critère, tant sur la possibilité d'identifier ou non un langage par induction, que de la complexité calculatoire de ce processus d'identification [Gold78, Pitt89]. En outre, ces travaux constituent une justification de l'utilisation d'échantillons négatifs.

Le protocole d'identification à la limite est défini comme suit. Soit M une méthode d'inférence et $S = x_1, x_2, \ldots$ une présentation admissible d'exemples d'un langage L. Nous notons S_i le sous-ensemble fini des i premiers éléments de S. La méthode d'inférence doit proposer une solution dans l'espace d'hypothèses après chaque nouvel exemple lu. Soit $H(S_i)$, l'hypothèse proposée par la méthode M après avoir lu les i premiers exemples présentés.

Définition 6.26 La méthode d'inférence M *identifie à la limite* le langage L, s'il existe un indice fini j, tel que

- $H(S_i) = H(S_j)$, pour tout $i \geq j$.
- $L(H(S_j)) = L$.

La première condition impose que la méthode converge à partir d'un indice fini, la seconde qu'elle converge vers **une** représentation de la solution correcte. Nous appelons *point de convergence* pour la présentation S, l'indice fini j à partir duquel la convergence est atteinte.

Définition 6.27 Une méthode d'inférence M identifie à la limite une classe $\mathcal{C}$ de langages, si M identifie à la limite n'importe quel langage L de la classe $\mathcal{C}$ pour n'importe quelle présentation admissible d'exemples de L.

Ces définitions correspondent à ce que nous désignerons par la suite comme l'*identification exacte* d'un langage ou d'une classe de langages. Nous verrons au paragraphe 6.3.5 que le type de présentation des exemples influencent les possibilités d'identification.

L'abréviation EX désigne la classe de tous les ensembles U de fonctions récursives totales[13] telle qu'il existe une méthode d'inférence qui identifie exactement U à la limite. Remarquons que l'ensemble de toutes les fonctions récursives totales ne peut pas être identifié à la limite [Gold67].

6.3.4 Autres critères d'identification

Malgré la popularité du critère d'identification à la limite, de nombreux autres critères d'identification ont été proposés. Ils constituent généralement une relaxation de l'identification exacte. Nous en présentons quelques-uns ci-après.

BC-identification

L'appellation *BC-identification* provient de la terminologie anglaise pour "Behaviorally Correct Identification". Ce critère n'impose pas que la méthode d'inférence converge vers une seule hypothèse dans l'espace des hypothèses mais, qu'à partir du point de convergence, toutes les hypothèses proposées constituent une description possible de la solution correcte. La classe BC est une sous-classe stricte de la classe EX [Barzdin74, Case et al.83].

[13]Les fonctions récursives totales sont les fonctions entières calculables par des Machines de Turing qui s'arrêtent pour toute entrée.

ε-identification

Ce critère d'identification est dû à Wharton [Wharton74]. Il définit une fonction de pondération des chaînes $w : \Sigma^* \to \mathbb{R}$. Chaque chaîne x de Σ^* est affectée d'un *poids* de telle sorte que la somme des poids vaut l'unité. On définit alors une distance $d(L_1, L_2)$ entre deux langages comme la somme des poids des chaînes dans leur différence symétrique $L_1 \oplus L_2$. Cette distance définit une métrique sur l'espace des langages.

Définition 6.28 Pour tout $\varepsilon > 0$, une méthode d'inférence M *ε-identifie* un langage L, si et seulement si elle converge vers une grammaire G telle que $d(L(G), L) < \varepsilon$.

Un résultat positif de cette théorie est que pour tout ε, la classe de tous les langages définis sur un alphabet fixé est ε-identifiable à la limite à partir de présentations positives par une méthode énumérative qui produit des grammaires engendrant des langages finis [Wharton77].

EX^k-identification

Cette relaxation de l'identification exacte concerne le nombre maximal d'*anomalies* entre le langage proposé après convergence et le langage correct.

Définition 6.29 Pour tout entier naturel k, une méthode d'inférence M identifie à la limite un langage L avec au plus k anomalies, si et seulement si elle converge vers une grammaire G, telle que la différence symétrique $L(G) \oplus L$ contienne au plus k éléments.

L'abréviation EX^k désigne la classe de tous les ensembles U de fonctions récursives telle qu'il existe une méthode d'inférence qui identifie U à la limite avec au plus k anomalies. Par conséquent, $EX^0 = EX$. De façon similaire, EX^* désigne la classe de tous les ensembles U de fonctions récursives telle qu'il existe une méthode d'inférence qui identifie U à la limite avec un nombre fini d'anomalies. Case et Smith ont montré que $EX^0, EX^1, EX^2, \ldots$ est une hiérarchie stricte dont l'union est strictement inclue dans EX^* [Case et al.83].

6.3.5 Influence du type de présentations

Nous étudions dans ce paragraphe l'effet des différents types de présentation sur les possibilités d'identification exacte d'un langage [Gold67].

Définition 6.30 Une classe *superfinie* de langages est une classe qui contient tous les langages de cardinalité finie et au moins un langage de cardinalité infinie.

Propriété 6.2 *Aucune classe superfinie de langages n'est identifiable à la limite à partir de présentations positives [Gold67].*

Cette propriété implique en particulier que la classe des langages réguliers n'est pas identifiable à la limite à partir de présentations positives.

Propriété 6.3 *Toute classe énumérable de langages récursifs est identifiable à la limite à partir de présentations complètes [Gold67].*

Cette classe inclut la classe des langages sensibles au contexte et donc également celle des langages réguliers. Ce résultat justifie l'utilisation d'échantillons négatifs car il assure que, pour peu que les échantillons soit suffisamment grands, une identification exacte de n'importe quel langage régulier est possible. Remarquons cependant que l'identification exacte a une complexité calculatoire non-polynomiale. Cette question sera étudiée plus en détail au paragraphe 6.4.

6.3.6 Identification par utilisation de questions d'équivalence

Angluin a proposé un autre protocole théorique d'identification d'un langage L sur base d'une présentation par oracle. A chaque étape, la méthode d'inférence propose une hypothèse H qui est une description possible d'un langage, désigné par $L(H)$. Une *question d'équivalence* est alors posée à l'oracle : si $L(H) = L$, l'oracle répond "oui" et l'identification est atteinte, sinon l'oracle propose un contre-exemple, c'est-à-dire un élément de $L(H) \oplus L$ [Angluin87].

Les résultats sur la complexité de l'identification par oracle avec questions d'équivalence ou d'appartenance sont présentés dans [Pitt89]. Nous ne développerons pas plus ces résultats, étant donné leur limitation qui consiste à disposer, en pratique, d'un oracle.

6.3.7 Identification stochastique

Un langage *stochastique* est un langage L_s muni d'une distribution de probabilité D sur ses éléments [Fu et al.75b, Gonzalez et al.78]. L'espace d'hypothèses d'un problème d'inférence stochastique est généralement la classe des *grammaires stochastiques*. Une *présentation stochastique* relative à un langage stochastique L_s est une séquence infinie d'exemples qui sont supposés avoir été générés conformément à la probabilité attribuée à ses éléments. Ce type de présentation ne comporte donc pas de contre-exemples explicites. Dans cette partie de notre exposé, nous ne détaillerons pas l'inférence d'une grammaire stochastique. Signalons simplement qu'elle répond à des critères d'identification qui lui sont propres. Angluin a démontré que, pour une distribution de probabilité D quelconque, si l'on désire garantir l'identification avec une probabilité 1, le problème se réduit à un problème d'identification classique [Angluin88]. Par contre, elle démontre que si l'on se restreint à des distributions rationnelles[14] l'identification exacte peut être obtenue, avec probabilité 1, à partir d'une présentation stochastique. En particulier, Horning montre que les langages engendrables par les grammaires hors-contexte stochastiques, dont les règles sont munies de probabilités rationnelles, peuvent être identifiés à la limite, avec une probabilité 1, à partir d'une présentation stochastique [Horning69].

6.3.8 Identification-PAC

Valiant a introduit la notion d'apprentissage PAC, pour *probablement approximativement correct* [Valiant84]. Dans le cas particulier de l'apprentissage d'automates déterministes[15], l'*identification-PAC* peut être définie comme suit[16] :

Définition 6.31 Les AFDs sont PAC-identifiables si et seulement s'il existe un algorithme A (éventuellement aléatoire) tel que, pour tout $\varepsilon > 0$, $\delta > 0$, pour tout AFD M de taille n, pour tout entier naturel m et pour toute distribution de probabilité D définie sur les chaînes de Σ^* de longueur au plus égale à m, si A reçoit en entrée des exemples classifiés relativement à M et générés aléatoirement conformément à la distribution D, alors A produit un AFD M' tel que, avec une probabilité au moins égale à $1-\delta$, la probabilité de l'ensemble des chaînes de la différence symétrique $L(M) \oplus L(M')$ est au plus égale à ε. La complexité temporelle de A doit être polynomiale en fonction de $n, m, \frac{1}{\varepsilon}, \frac{1}{\delta}$, et $|\Sigma|$ [Pitt89].

[14] Plus précisément, Angluin démontre que l'on peut identifier à la limite des distributions supposées provenir d'une séquence de distributions approximables par des fonctions rationnelles [Angluin88].

[15] La taille d'un automate déterministe est généralement définie comme son nombre d'états.

[16] La définition présentée dans [Natarajan91] ne requiert pas la complexité polynomiale évoquée dans la définition de Pitt. Néanmoins, on s'intéresse, en pratique, aux algorithmes de complexité polynomiale.

Il s'agit donc d'une relaxation de l'identification exacte, dans un double sens. D'une part, l'identification ne doit être garantie qu'avec une probabilité bornée par $1 - \delta$ et, d'autre part, un ensemble de chaînes peuvent être mal classifiées pour peu que la probabilité de l'ensemble de ces chaînes, conformément à la distribution D, soit bornée par ε. Par contre, le résultat doit être valable pour toute distribution D.

Ce cadre théorique est encore en plein développement. Remarquons que les résultats actuels sont plutôt négatifs [Pitt et al.89]. Par ailleurs, les AFDs sont PAC-identifiables sur base d'une présentation par oracle et questions d'appartenance [Natarajan91].

6.4 Complexité calculatoire de l'identification d'un automate

Pitt a proposé un excellent article de synthèse sur la complexité calculatoire de l'inférence inductive des automates [Pitt89]. Nous ne détaillerons pas ces aspects ici mais nous présentons quelques résultats qui devraient clarifier la complexité théorique et pratique de l'inférence d'un AFD à partir d'échantillons positif I_+ et négatif I_-.

Le problème du *plus petit AFD compatible* est défini comme suit. Il s'agit de trouver un AFD A

tel que :

- $L(A) \supseteq I_+$,
- $L(A) \cap I_- = \emptyset$,
- A est un AFD comportant le nombre minimal d'états.

Nous pouvons énoncer les principaux résultats théoriques négatifs :

- Le problème du plus petit AFD compatible est NP-difficile [Angluin78, Gold78].
- Si $P \neq NP$, il n'existe pas d'algorithme polynomial d'approximation qui garantirait de produire un AFD compatible dont la taille serait bornée par un quelconque polynôme en fonction de la taille de la solution optimale [Pitt et al.89].
- L'inférence approchée d'automates à partir d'un sous-ensemble fini quelconque d'une présentation complète est NP-difficile si un adversaire choisit la machine à identifier et les données [Kearns et al.89].

Par ailleurs, Trakhtenbrot et Barzdin proposent un algorithme par fusion[17] d'états de complexité $\mathcal{O}(mn^2)$ [Trakhtenbrot et al.73] pour construire le plus petit AFD compatible avec un échantillon *complet*, où m est la taille du $PTA(I_+)$ et n la taille de l'AFD *cible*, c'est-a-dire l'automate à identifier. Cet échantillon complet est constitué de toutes les chaînes de Σ^* de longueur au plus égal à l, chacune étant étiquetée comme exemple positif ou négatif. La longueur l dépend de certaines caractéristiques[18] de l'automate cible. Elle est au plus égale à $2n-1$ [Trakhtenbrot et al.73]. Dans le pire des cas, la taille d'un échantillon complet croît donc exponentiellement avec la taille de l'automate cible. De plus, Angluin a démontré que, dans le pire des cas, l'identification exacte est impossible dès qu'une fraction arbitrairement petite de l'échantillon complet est manquante [Angluin78].

Malgré les résultats négatifs énoncés, la complexité calculatoire semble être meilleure *en moyenne*. Pour des AFDs générés aléatoirement conformément à une distribution de probabilité prédéfinie, la valeur estimée de la taille d'un échantillon complet est [Trakhtenbrot et al.73] :

$$|S|_{complet} = (|\Sigma|^2 n^C \log_2(n-1))/(|\Sigma|-1)$$

où C ne dépend que de $|\Sigma|$.

La valeur estimée de la taille d'un échantillon complet pourrait suggérer que l'identification exacte d'AFDs est en moyenne envisageable, du moins pour des alphabets restreints. Cependant, il n'y a aucune garantie de disposer, en pratique, d'un échantillon complet, quelqu'en soit la taille. Seule la disponibilité de données *éparses* peut être supposée. Dans ce cadre, Lang a montré expérimentalement [Lang92] qu'une identification quasi-exacte peut être obtenue en n'utilisant qu'une faible fraction (aléatoirement générée) d'un échantillon complet. De plus, les résultats de Lang montrent que cette fraction décroît, en proportion, avec la croissance de la taille de l'automate cible.

L'algorithme polynomial proposé par Lang [Lang92], et indépendamment par Oncina et García [Oncina et al.92], est similaire à celui proposé par Trakhtenbrot and Barzdin [Trakhtenbrot et al.73]. Cet algorithme sera présenté plus en détail au paragraphe 7.2.2 et au chapitre 9.

6.5 Espace de recherche (*)

Nous pouvons définir l'inférence régulière comme la recherche d'un automate A inconnu à partir duquel un échantillon positif I_+ est supposé avoir

[17] Ce type d'algorithme sera exposé aux paragraphes 7.1 et 7.2

[18] Ces caractéristiques sont le *degré de distingabilité* de l'automate et sa *profondeur* dans le sens défini dans [Trakhtenbrot et al.73].

été généré. Moyennant l'hypothèse supplémentaire de complétude structurelle de I_+ relativement à l'automate A, nous pouvons considérer ce problème comme la recherche à travers un treillis construit sur la base de I_+. Cette propriété résulte de deux théorèmes partiellement démontrés, respectivement dans [Miclet79] et [Angluin82], que nous avons revus dans [Dupont et al.94]. En particulier, nous montrons la nécessité de la seconde condition de la complétude structurelle de l'échantillon I_+ (voir définition 6.16).

6.5.1 Théorèmes fondamentaux (*)

Théorème 6.1 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit A n'importe quel automate acceptant exactement L. Si I_+ est structurellement complet relativement à A alors A appartient à $Lat(MCA(I_+))$.*

Preuve :
Nous allons construire le $MCA(I_+)$ à partir d'une acceptation de I_+ par A. Cette construction définit une partition π telle que A soit isomorphe à $MCA(I_+)/\pi$.
Soit $I_+ = \{u_1, \ldots, u_M\}$, où $u_i = a_{i,1} \ldots a_{i,|u_i|}$, $1 \leq i \leq M$, un échantillon positif structurellement complet relativement à A. L'acceptation $\mathcal{AC}(I_+, A)$ définit pour chaque chaîne u_i une séquence $(q_A^{i,0}, \ldots, q_A^{i,|u_i|})$ de $|u_i|+1$ états, où $q_A^{i,0} = q_{0_A}$, $q_A^{i,|u_i|} \in F_A$ et $q_A^{i,j+1} \in \delta_A(q_A^{i,j}, a_{i,j+1})$, $1 \leq i \leq M, 0 \leq j \leq |u_i| - 1$.
Nous commençons la construction du MCA en considérant son état initial, $q_{0_{MCA}}$ avec la définition : $q_{MCA}^{i,0} = q_{0_{MCA}}$, $1 \leq i \leq M$. Chaque fois qu'une transition de A est exercée, nous ajoutons un nouvel état dans Q_{MCA} et nous adaptons la fonction de transition δ_{MCA} comme suit :

$$q_A^{i,j+1} \in \delta_A(q_A^{i,j}, a_{i,j+1}) \Leftrightarrow q_{MCA}^{i,j+1} \in \delta_{MCA}(q_{MCA}^{i,j}, a_{i,j+1}),$$

pour $1 \leq i \leq M, 0 \leq j \leq |u_i| - 1$.
De plus, nous construisons l'ensemble des états d'acceptation F_{MCA} comme suit :

$$F_{MCA} = \{q_{MCA}^{i,|u_i|}, 1 \leq i \leq M\}.$$

Nous définissons également une fonction φ qui fait correspondre les états du MCA à ceux de A :

$$\varphi : Q_{MCA} \rightarrow Q_A : \varphi(q_{MCA}^{i,j}) = q_A, \text{ lorsque } q_A = q_A^{i,j},\ 1 \leq i \leq M, 0 \leq j \leq |u_i|.$$

Finalement, nous définissons la partition π comme suit :

$$B(q_{MCA}^l, \pi) = B'(q_{MCA}^k, \pi) \Leftrightarrow \varphi(q_{MCA}^l) = \varphi(q_{MCA}^k).$$

La définition même de la partition π implique que A soit isomorphe à MCA/π, puisque la complétude structurelle de I_+ implique que $\delta_{MCA/\pi}$ corresponde exactement à δ_A. La seconde condition requise par la complétude structurelle de I_+ impose que

$$\forall q_A \in F_A, \exists i, 1 \leq i \leq M \text{ tel que } q_A^{i,|u_i|} = q_A.$$

Par conséquent, $F_{MCA} = \{q_{MCA} | \varphi(q_{MCA}) \cap F_A \neq \phi\}$ et $F_{MCA/\pi}$ correspond exactement à F_A. □

FIG. 6.7: L'automate A et le $MCA(I_+)$, avec $I_+ = \{ab\}$.

En guise d'illustration de la dernière partie de cette démonstration, nous considérons l'exemple suivant. Conformément à la définition 6.16, l'échantillon $I_+ = \{ab\}$ n'est pas structurellement complet relativement à l'automate A de la figure 6.7, où q_1 et q_2 sont des états d'acceptation. Pour cette raison, l'automate A ne peut pas être dérivé du $MCA(I_+)$, et évidemment pas non plus du $PTA(I_+)$. En effet, il n'est pas possible de définir une partition π de l'ensemble des états Q_{MCA} telle que A corresponde à $MCA(I_+)/\pi$ avec $q_1 \in F_{MCA/\pi}$. Au contraire, l'échantillon $\{ab, a\}$ est structurellement complet relativement à l'automate A qui peut donc être dérivé du MCA, ou du PTA, associés à cet échantillon.

Théorème 6.2 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit $A(L)$ l'automate* ***canonique*** *acceptant L. Si I_+ est structurellement complet relativement à $A(L)$ alors $A(L)$ appartient à $Lat(PTA(I_+))$.*

Preuve :
Nous pouvons faire appel au raisonnement utilisé dans la preuve du théorème 6.1, à l'exception de l'acceptation de I_+ par $A(L)$ qui, à présent, est unique puisque $A(L)$ est déterministe. De façon similaire, chaque entrée de la fonction de transition du $PTA(I_+)$ contient au plus un élément, puisque le $PTA(I_+)$ est déterministe. □

Théorème 6.3 *Soit $\mathcal{A}$ l'ensemble des automates relativement auxquels un échantillon positif I_+ est structurellement complet.*
L'ensemble $\mathcal{A}$ est $Lat(MCA(I_+))$.

Preuve :
(1) *Si I_+ est structurellement complet relativement à un automate A, alors A appartient à $Lat(MCA(I_+))$.*
Il s'agit exactement du théorème 6.1.
(2) *Si un automate A appartient à $Lat(MCA(I_+))$ alors I_+ est structurellement complet relativement à A* [Fu et al.75a].
Par construction, I_+ est structurellement complet relativement au $MCA(I_+)$. Puisque n'importe quel automate appartenant à $Lat(MCA(I_+))$ peut être dérivé du $MCA(I_+)$ pour une certaine partition π, le résultat est une conséquence directe de la définition 6.14 d'un automate quotient. □

6.5.2 Propriétés de l'espace de recherche (*)

Grâce aux résultats présentés au paragraphe 6.5.1, nous savons que si nous disposons d'un échantillon positif I_+ d'un langage inconnu L et si nous supposons que I_+ est structurellement complet relativement à un automate inconnu A acceptant exactement L, alors nous pouvons dériver A pour une certaine partition π de l'ensemble des états du $MCA(I_+)$. Nous pouvons dès lors considérer l'inférence régulière comme un problème de recherche, dans un treillis, de la partition π.

Signalons, tout d'abord, que le théorème 6.1 est plus général que le théorème 6.2. En effet, le premier suppose seulement la complétude structurelle de I_+ relativement à un quelconque automate acceptant un langage L, et non nécessairement relativement à l'automate canonique[19] de L. De plus, nous pouvons énoncer la propriété d'inclusion des treillis :

Propriété 6.4 $Lat(PTA(I_+)) \subseteq Lat(MCA(I_+))$.

Cette propriété découle directement de la définition 6.18 du $PTA(I_+)$ qui est un automate quotient du $MCA(I_+)$. Comme le treillis $Lat(PTA(I_+))$ est généralement *strictement* inclus dans le treillis $Lat(MCA(I_+))$, rechercher une solution dans $Lat(PTA(I_+))$ au lieu de $Lat(MCA(I_+))$ permet de considérer un espace de recherche plus restreint.

La taille de l'espace de recherche est le nombre de partitions $|\mathcal{P}(N)|$ d'un ensemble à N éléments où N est ici le nombre d'états de l'élément nul du

[19] Un échantillon structurellement complet relativement à $A(L)$ est parfois appelé *représentatif pour L* [Muggleton84].

treillis, c'est-à-dire $PTA(I_+)$ ou $MCA(I_+)$. Ce nombre peut être calculé par la formule suivante :

$$|\mathcal{P}(N)| = \sum_{i=1}^{N} S(N,i)$$

où $S(N,i)$ désigne le nombre de partitions comportant i blocs d'un ensemble à N éléments. $S(N,i)$ est le nombre de Stirling de deuxième espèce qui est donné par l'application de la formule de récurrence [David et al.62] :

$S(N,1) = 1,\ N \geq 1,$
$S(N,i) = 0$, si $i > N$,
$S(N,i) = i \cdot S(N-1,i) + S(N-1,i-1)$, pour $2 \leq i \leq N$.

A titre d'exemple, $|\mathcal{P}(30)| = 8.5 \times 10^{23}$.

Nous détaillons dans la suite de ce paragraphe d'autres propriétés des treillis, que nous avons établies [Dupont et al.94]. La première caractéristique importante est que chaque élément des treillis correspond à un automate particulier mais que beaucoup d'automates différents peuvent représenter, c'est-à-dire accepter, le même langage. Donc, nous pourrions envisager de restreindre la recherche à des automates déterministes. Cependant, nous avons la propriété suivante :

Propriété 6.5 *Il existe des échantillons positifs I_+ pour lesquels certains langages ne sont représentés que par des AFNs dans $Lat(MCA(I_+))$.*

Notons que cette propriété s'applique également à $Lat(PTA(I_+))$ en conséquence de l'inclusion des treillis (propriété 6.4).

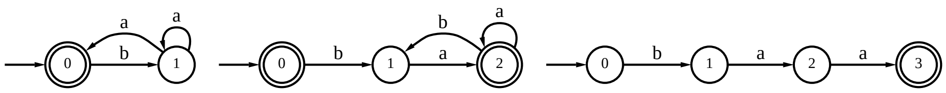


FIG. 6.8: $AFN(L_1)$, $A(L_1)$ et $MCA(I_+)$, avec $L_1 = (ba^*a)^*$ et $I_+ = \{baa\}$.

Considérons l'automate non-déterministe de la figure 6.8. Il accepte le langage L_1 défini par l'expression régulière $(ba^*a)^*$. L'échantillon $I_+ = \{baa\}$ est structurellement complet relativement à cet automate. Cependant, cet échantillon n'est pas structurellement complet relativement à l'automate déterministe minimal (définition 6.12) du même langage et donc relativement à aucun automate déterministe acceptant le langage L_1. Par conséquent, nous ne pouvons pas identifier le langage L_1 si nous restreignons la recherche à des automates déterministes.

Propriété 6.6 *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 $Lat(MCA(I_+))$.*

Par la propriété 6.4 d'inclusion des treillis, nous savons que l'ensemble des automates qui peuvent être dérivés de $MCA(I_+)$ inclut l'ensemble de ceux qui peuvent être dérivés de $PTA(I_+)$. La propriété 6.6 résulte du fait que, d'une part, cette inclusion peut être stricte et que, d'autre part, les automates qui appartiennent à $Lat(MCA(I_+))$ sans appartenir à $Lat(PTA(I_+))$ n'ont pas d'automate équivalent, c'est-à-dire n'importe quel automate acceptant le même langage, dans $Lat(PTA(I_+))$.

Considérons le langage $L_2 = (ba + b(aa)^*)$, qui peut être représenté par l'automate non-déterministe $AFN(L_2)$ de la figure 6.10 et par son automate canonique $A(L_2)$. Supposons que l'échantillon positif I_+ soit $\{ba, baa\}$. Le $MCA(I_+)$ et le $PTA(I_+)$ sont représentés à la figure 6.9. Nous pouvons clairement dériver $AFN(L_2)$ à partir du $MCA(I_+)$ et non du $PTA(I_+)$. Cela provient du fait que les états q_1 et q_2 du MCA, respectivement q_3 et q_4, sont fusionnés dans le PTA. De plus, puisque I_+ est structurellement complet relativement à $AFN(L_2)$ mais pas relativement à $A(L_2)$, ce dernier automate ne peut être dérivé du PTA ni aucun automate déterministe acceptant le même langage.

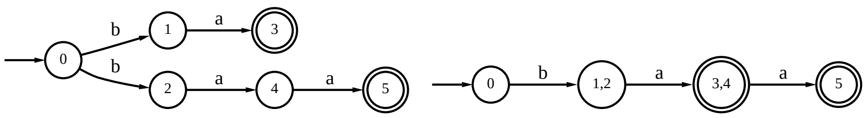


FIG. 6.9: $MCA(I_+)$ et $PTA(I_+)$, avec $I_+ = \{ba, baa\}$.

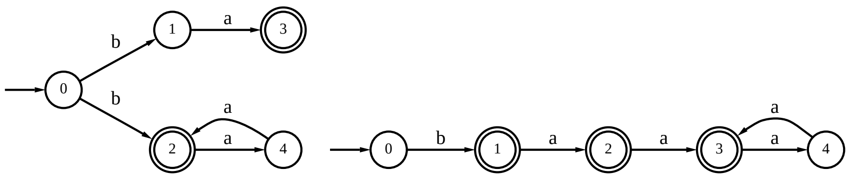


FIG. 6.10: $AFN(L_2)$ et $A(L_2)$, avec $L_2 = (ba + b(aa)^*)$.

6.6 Ensemble frontière (*)

Supposons que nous disposions d'un échantillon négatif du langage inconnu L. Dans ce cas, nous savons que nous pouvons, en principe, identifier au moins n'importe quel langage régulier (voir paragraphe 6.3.5). L'inférence peut être alors considérée comme la découverte d'un automate compatible avec les échantillons positif et négatif (voir paragraphe 6.4). Rappelons que cela consiste à chercher un automate A tel que $I_+ \subseteq L(A)$ et $I_- \cap L(A) = \phi$. Il existe un grand nombre d'automates compatibles appartenant à $Lat(MCA(I_+))$ et, en particulier, le $MCA(I_+)$ satisfait ces conditions mais ne généralise pas l'échantillon positif. Par conséquent, nous pouvons chercher une solution plus générale sous le contrôle de l'échantillon négatif.

Si nous choisissons la simplicité de l'automate inféré comme critère de généralité et que nous restreignons la recherche à des automates déterministes, la solution cherchée est alors l'AFD compatible comportant le nombre minimal d'états. Il s'agit du problème du plus petit AFD compatible, déjà évoqué au paragraphe 6.4. Nous avons vu qu'il n'existe pas d'algorithme polynomial résolvant ce problème dans tous les cas mais qu'il existe un algorithme polynomial qui permet, en général, de trouver une solution quasi-exacte. Nous donnerons un aperçu de cet algorithme au paragraphe 7.2.2. Deux autres algorithmes seront abordés aux paragraphes 7.2.1 et 7.2.3. Afin de comparer ces trois algorithmes et d'étudier le problème du plus petit AFD compatible, nous poursuivons la caractérisation de l'espace de recherche introduite au paragraphe 6.5.

La notion d'ensemble frontière [Miclet et al.94] a été proposée, dans le domaine de l'inférence grammaticale, par Miclet. Cet ensemble constitue la limite de la généralisation possible à partir de l'échantillon positif sous contrôle de l'échantillon négatif. En ce sens, l'ensemble frontière est l'ensemble des *généralisations les plus générales* conformément à la terminologie propre à l'étude de l'*espace des versions* [Mitchell78, Nicolas93]. Cet espace est, dans notre cas, l'ensemble de tous les automates quotient du $MCA(I_+)$ et compatibles avec I_-. Notons que l'ensemble des *généralisations les plus spécifiques* se réduit, dans $Lat(MCA(I_+))$, au $MCA(I_+)$.

Nous avons étudié dans [Dupont et al.94] quelques-unes des propriétés de l'ensemble frontière que nous rappelons ci-après.

Définition 6.32 Une *antichaîne* $\overline{AS}$ dans un treillis d'automates est un ensemble d'automates tel que chaque élément de $\overline{AS}$ n'est relié par la relation $\preceq$ avec aucun autre élément de $\overline{AS}$.

Définition 6.33 Nous disons qu'un automate A est *à une profondeur maximale* dans un treillis d'automates s'il n'existe pas, dans ce treillis, d'automate A', différent de A, tel que A' puisse être dérivé de A et tel que $L(A') \cap I_- = \phi$.

Définition 6.34 L'*ensemble frontière*[20] $BS_{MCA}(I_+, I_-)$ est l'antichaîne dont chaque élément est à une profondeur maximale dans $Lat(MCA(I_+))$.

Définition 6.35 L'*ensemble frontière* $BS_{PTA}(I_+, I_-)$ est l'antichaîne dont chaque élément est à une profondeur maximale dans $Lat(PTA(I_+))$.

Par conséquent, l'ensemble frontière d'un treillis est l'antichaîne des automates les plus généraux sous le contrôle de l'échantillon négatif. Le problème du plus petit AFD compatible consiste à découvrir le plus petit AFD appartenant à l'ensemble frontière[21].

Propriété 6.7 $BS_{PTA}(I_+, I_-) \subseteq BS_{MCA}(I_+, I_-)$.

Il s'agit d'une conséquence directe de l'inclusion des treillis (propriété 6.4).

Propriété 6.8 *Il peut y avoir plusieurs langages distincts représentés par des éléments du* $BS_{MCA}(I_+, I_-)$.

Cette propriété résulte du fait que la relation d'ordre partiel introduite au paragraphe 6.2.3 s'applique aux automates et non aux langages.

Propriété 6.9 *Il peut exister des AFNs appartenant à* $BS_{MCA}(I_+, I_-)$ *contenant moins d'états que le plus petit AFD compatible.*

Par exemple, ce cas se présente lorsque le plus petit AFD compatible correspond à un langage qui peut être représenté, dans le treillis considéré, par un AFN comportant moins d'états.

Propriété 6.10 *Tous les AFDs appartenant à* $BS_{MCA}(I_+, I_-)$ *sont nécessairement minimaux pour le langage qu'ils acceptent.*

[20] L'abréviation **BS** pour "Border Set" provient de la terminologie anglaise.

[21] On pourrait également introduire une notion d'*ensemble frontière déterministe*, c'est-à-dire constitué uniquement d'AFDs. Au sens strict, un ensemble frontière déterministe BS_1 peut être distinct d'une restriction à des automates déterministes d'un ensemble frontière quelconque BS_2. En effet, BS_1 peut inclure des AFDs dont on pourrait dériver des AFNs compatibles. Ces AFDs n'appartiendraient donc pas à la restriction déterministe de BS_2.

Remarquons que la même propriété s'applique à $BS_{PTA}(I_+, I_-)$ comme conséquence de la propriété 6.7.

Preuve :
Soit A un AFD acceptant le langage L et compatible avec I_+ et I_-. Supposons que A ne soit pas isomorphe à l'automate canonique $A(L)$. Nous pouvons construire la partition π de l'ensemble des états de A de telle sorte que A/π soit obtenu à partir de A en fusionnant les états partageant les mêmes ensembles de finales. Par construction, $A/\pi = A(L)$. Donc, $A \ll A(L)$ et A ne se trouve pas à une profondeur maximale. Finalement, quoi qu'il en soit de l'appartenance de $A(L)$ à $BS_{MCA}(I_+, I_-)$, A ne peut en être un élément. □

Cette propriété établit qu'un automate déterministe non minimal ne peut pas appartenir à l'ensemble frontière $BS_{MCA}(I_+, I_-)$. Par ailleurs, il existe des automates canoniques qui n'appartiennent pas à $BS_{MCA}(I_+, I_-)$. En effet, tous les automates canoniques du treillis ne sont pas nécessairement compatibles avec I_- et, par ailleurs, il est parfois possible de dériver à partir d'un automate canonique un autre automate compatible.

6.7 Inférence régulière par recherche déterministe (*)

Le théorème 6.2 stipule que nous pouvons identifier n'importe quel langage régulier L en découvrant l'automate canonique $A(L)$ comme automate quotient du $PTA(I_+)$. Rappelons que cela nécessite la complétude structurelle de I_+ relativement à l'automate $A(L)$ recherché. Sous cette hypothèse, nous pouvons explorer le treillis $Lat(PTA(I_+))$. Le $PTA(I_+)$ et l'automate cible sont déterministes. Néanmoins, certains automates construits au cours de la recherche de $A(L)$ peuvent être non déterministes. La possibilité de restreindre la recherche à des automates déterministes est basée sur le théorème 6.4.

Théorème 6.4 *Soit I_+ un échantillon positif d'un quelconque langage régulier L et soit $A(L)$ l'automate canonique acceptant L. Si I_+ est structurellement complet relativement à $A(L)$ alors il existe une séquence d'AFDs telle que*

$$PTA(I_+) = PTA(I_+)/\pi_0 \preceq PTA(I_+)/\pi_1 \preceq \ldots \preceq PTA(I_+)/\pi_n = A(L)$$

Preuve :
Le théorème 6.2 nous assure que la partition π_n existe, avec $0 < n \leq |Q_{PTA(I_+)}| - 1$. Nous allons démontrer qu'il existe une séquence de partitions

$\pi_0 \preceq ... \preceq \pi_n$ telle que chaque automate quotient associé soit déterministe. Par hypothèse, $PTA(I_+)/\pi_n$ est un AFD. Nous allons démontrer par récurrence que si $PTA(I_+)/\pi_j$ est un AFD, pour $0 < j \leq n$, alors il existe une partition π_{j-1}, telle que $\pi_{j-1} \preceq \pi_j$ et $PTA(I_+)/\pi_{j-1}$ est un AFD.
Nous considérons les $r(j)$ blocs de la partition $\pi_j = \{B_1, ..., B_{r(j)}\}$. Chaque bloc reçoit l'indice de son état de rang minimal conformément à l'ordre standard des états du $PTA(I_+)$. Par conséquent, $B_1 < ... < B_{r(j)}$. Nous désignons par k l'indice, compris entre 1 et $r(j)$, du premier bloc comportant au moins 2 états, avec $B_k = \{q_k^0, ..., q_k^p\}$. Les états de chaque bloc, en particulier ceux du bloc B_k, sont également triés par ordre standard : $q_k^0 < ... < q_k^p$. La partition π_{j-1} est alors obtenue à partir de la partition π_j par la formule suivante :

$$\pi_{j-1} = \pi_j \backslash B_k \cup \{q_k^0\} \cup B_k \backslash q_k^0$$

Par construction : $\pi_{j-1} \preceq \pi_j$.
Supposons que $PTA(I_+)/\pi_{j-1}$ soit non déterministe alors que $PTA(I_+)/\pi_j$ est déterministe (par hypothèse de récurrence). Tous les blocs de rang inférieur à k correspondent nécessairement à un seul état du $PTA(I_+)$. Cela implique qu'il existe un état q du $PTA(I_+)$, avec $q < q_k^0$, et une lettre $a \in \Sigma$, tels que $\exists q' \in B_k \backslash q_k^0$ avec q' et q_k^0 tous les deux a-successeurs de q. Dans ce cas $PTA(I_+)$ serait non déterministe, ce qui est faux. Donc $PTA(I_+)/\pi_{j-1}$ est un automate déterministe. $\square$

Corollaire 6.1 *Tout AFD $A \in Lat(PTA(I_+))$ peut être obtenu par fusions successives en suivant une certaine séquence d'AFDs quotient du $PTA(I_+)$.*

Nous pouvons nous demander s'il existe un théorème dual du théorème 6.4 qui consisterait à trouver une séquence d'automates déterministes entre un automate déterministe quelconque dans $Lat(PTA(I_+))$ et l'automate universel. Il n'en est rien. En guise de contre-exemple, à partir de l'échantillon $I_+ = \{aaa\}$ nous pouvons obtenir le $PTA(I_+)$, les automates A_1, A_2, A_3, A_4, A_5 et l'automate universel UA représentés à la figure 6.11. Ces 7 automates sont déterministes mais toute fusion d'états de l'automate A_1 introduit un non déterminisme. Il n'existe donc pas de séquence d'automates déterministes et quotient de A_1 qui mènerait à UA. Cela nous amène à conclure que la restriction du treillis $Lat(PTA(I_+))$ aux automates déterministes est un *sup demi-treillis* dont l'élément nul est le $PTA(I_+)$. Nous avons représenté à la figure 6.11 le sup demi-treillis de l'exemple évoqué.

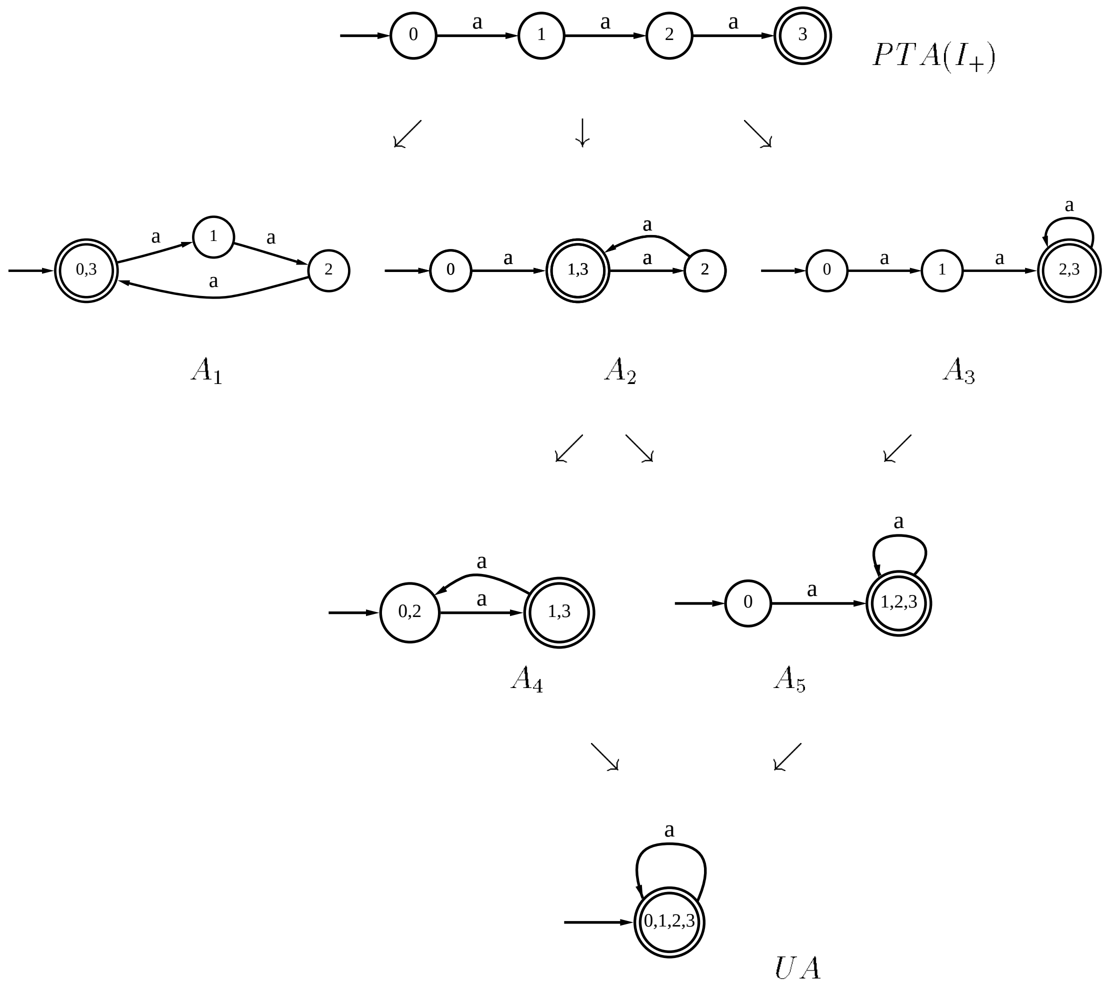


FIG. 6.11: Sup demi-treillis des AFDs dans $Lat(PTA(I_+))$, pour $I_+ = \{aaa\}$.

Nous pouvons contraindre les algorithmes de recherche heuristiques, en particulier BRIG, à n'explorer que le sup demi-treillis déterministe. Notons que cette contrainte peut être appliquée par une inspection efficace de la table de transition en utilisant des structures d'ensembles disjoints [Cormen et al.90].

Chapitre 7

Algorithmes d'inférence régulière (*)

De nombreux algorithmes d'inférence ont été développés. Certains sont *constructifs* ou *caractérisables* dans le sens qu'ils identifient *avec certitude* n'importe quel langage appartenant à une *classe définie.* Nous qualifions les autres d'algorithmes *heuristiques,* soit parce qu'ils n'identifient pas avec certitude n'importe quel langage d'une classe définie, soit parce qu'il n'existe pas de caractérisation de la classe de langages qu'ils peuvent identifier. Ces algorithmes utilisent un *biais d'apprentissage* ou heuristique pour guider la généralisation effectuée à partir des données.

La majorité des algorithmes d'inférence régulière n'utilisent que des échantillons positifs. La plupart de ces algorithmes sont décrits dans les articles de synthèse [Biermann et al.72, Angluin et al.83, Miclet90, Gregor94]. Nous présentons au paragraphe 7.1 quelques algorithmes constructifs ou heuristiques, qui utilisent ce type de données.

Si nous disposons également d'un échantillon négatif, l'inférence peut consister à construire *le plus petit automate compatible* avec les échantillons positif et négatif. Nous avons étudié le cadre formel de ce problème au paragraphe 6.6. Nous comparons au paragraphe 7.2 trois méthodes d'inférence avec échantillon négatif.

Nous terminons ce chapitre par l'exposé d'une technique originale d'inférence semi-incrémentale. Il s'agit d'une technique heuristique ayant pour objet de réduire la complexité pratique d'une inférence sous contrôle d'un échantillon négatif. Elle est détaillée au paragraphe 7.3.

7.1 Inférence sans échantillon négatif

Etant donné un échantillon positif I_+ d'un langage régulier L, nous avons à définir le critère qui guidera la recherche d'un automate A acceptant I_+. En particulier, le $MCA(I_+)$ et le $PTA(I_+)$ acceptent tous deux l'échantillon positif. Cependant, ils n'acceptent aucune autre phrase du langage L et, donc, ne généralisent pas l'information contenue dans les données d'apprentissage.

Nous savons, par la propriété 6.1, que n'importe quel automate dérivé du $MCA(I_+)$ accepte un langage incluant I_+. En ce sens, n'importe quel automate appartenant à $Lat(MCA(I_+))$ constitue une généralisation possible de l'échantillon positif. Conformément au résultat de Gold présenté au paragraphe 6.3.5, nous savons qu'il n'est pas possible d'identifier la classe entière des langages réguliers à partir d'un échantillon positif seulement. En particulier, aucun exemple positif ne peut éviter une sur-généralisation éventuelle de l'inférence. En d'autres termes, si une solution proposée par un algorithme d'inférence correspond à un langage incluant strictement le langage à identifier, aucun exemple positif ne peut contredire cette solution.

Pour éviter le risque de sur-généralisation, deux approches sont classiquement utilisées. La première possibilité consiste à rechercher la solution dans une sous-classe particulière des langages réguliers[1] ; il s'agit alors d'une méthode caractérisable. La seconde possibilité consiste à utiliser une information a priori pour guider la généralisation recherchée, le propre des méthodes heuristiques.

L'algorithme k-TSSI[2], détaillé au paragraphe 7.1.1, permet d'inférer les langages *k-testables au sens strict* [García et al.90]. Bien que la "méthode du successeur" [Richetin et al.84] soit généralement présentée comme une méthode heuristique, García et Vidal ont démontré qu'elle identifie les *langages locaux*, encore appelés *langages 2-testables au sens strict* [García et al.87]. Elle peut donc être considérée comme un cas particulier de l'algorithme k-TSSI. Un autre exemple, est l'algorithme d'inférence d'un langage appartenant à la classe des langages *k-réversibles,* présenté au paragraphe 7.1.2.

Les deux algorithmes précédents identifient à la limite, pour k fixé, une sous-classe définie des langages réguliers. Ils sont donc caractérisables. Nous présentons ensuite deux techniques d'inférence heuristiques. L'algorithme MGGI[3], décrit au paragraphe 7.1.3, constitue une autre extension de la méthode du successeur. L'identification de la classe entière des langages régu-

[1] Cette sous-classe sera nécessairement non superfinie.

[2] L'abréviation **k-TSSI** provient de la terminologie anglaise pour "k-Testable (languages) in the Strict Sense Inference".

[3] L'abréviation **MGGI** provient de la terminologie anglaise pour "Morphic Generator Grammatical Inference".

liers est possible grâce à l'introduction d'une heuristique d'apprentissage, en l'occurrence le choix d'un *morphisme*. Enfin, l'algorithme ECGI[4], décrit au paragraphe 7.1.4, est une technique d'inférence d'automates sans cycles, c'est-à-dire acceptant la sous-classe des langages finis, qui repose sur une analyse corrective.

7.1.1 L'algorithme k-TSSI

De façon informelle, un *langage k-testable au sens strict* est défini par un ensemble fini de sous-chaînes de longueur k qui ne peuvent apparaître dans les chaînes du langage[5]. La classe k-TSS désigne l'ensemble des langages k-testables au sens strict, pour k fixé.

Définition 7.1 Soit $Z_k = (\Sigma, I_k, F_k, T_k)$ un quadruplet, où Σ est un alphabet fini, $I_k, F_k \subseteq \bigcup_{i=1}^{k-1} \Sigma^i$ deux ensembles, constitués de chaînes de longueur strictement inférieure à k, appelés respectivement, *segments initiaux* et *finaux,* et $T_k \subseteq \Sigma^k$ un ensemble de *segments interdits,* constitués de chaînes de longueur k. Un langage k-testable au sens strict est défini par l'expression régulière suivante :

$$l(Z_k) = (I_k\Sigma^*) \cap (\Sigma^* F_k) - (\Sigma^* T_k \Sigma^*)$$

Les chaînes de $l(Z_k)$ sont donc définies comme commençant par un segment initial, se terminant par un segment final et ne comportant aucun segment interdit de longueur k. L'appellation k-testable provient du fait qu'il suffit d'analyser les chaînes avec une fenêtre de longueur k pour définir l'appartenance ou non au langage $l(Z_k)$. En particulier, la classe 2-TSS est aussi appelée la classe des *langages locaux.* Par ailleurs, la classe LTSS des *langages localement testables au sens strict* désigne la classe de tous les langages k-TSS, pour k quelconque et la classe k-T des langages *k-testables* désigne les langages obtenus par fermeture booléenne de langages k-testables au sens strict. Il n'existe pas, à l'heure actuelle, d'algorithme d'inférence à partir d'échantillon positif ni pour la classe LTSS, ni pour la classe k-T [Vidal et al.93].

Nous présentons ci-dessous l'algorithme k-TSSI qui permet, pour k fixé a priori, d'identifier le plus petit langage appartenant à k-TSS et contenant un échantillon positif I_+ [García et al.90]. La figure 7.1 représente les automates obtenus par l'application de l'algorithme k-TSSI avec $k = 3$ et
$I_+ = \{aa, aba, abba, abbba\}$. Chaque état est identifié par une séquence, comportant au plus $k-1$ symboles, qui correspondent aux derniers symboles lus,

[4]L'abréviation **ECGI** provient de la terminologie anglaise pour "Error Correcting Grammatical Inference".

[5]Nous désignons par Σ^k l'ensemble des chaînes de longueur k définies sur l'alphabet Σ.

jusqu'à cet état, lors de l'acceptation d'une chaîne de I_+. En d'autres termes, lors de l'analyse d'une chaîne les derniers symboles lus sont mémorisés dans une fenêtre glissante de $k-1$ éléments. Chaque contenu de la fenêtre d'analyse identifie un état à créer ou déjà existant. Dans notre exemple, la fenêtre est de longueur 2. Afin d'accepter la chaîne aa, l'on crée les trois premiers états λ, a et aa ; le dernier état étant nécessairement final puisque aa est un élément du langage. L'acceptation de la chaîne aba passe par les états λ et a, déjà existants, et crée les états ab et ba. Ce processus est répété pour toutes les chaînes de I_+. Si nous désignons par $||I_+||$ la somme des longueurs des chaînes de I_+, nous déduisons aisément que cet algorithme est de complexité[6] linéaire, $\mathcal{O}(||I_+||)$.

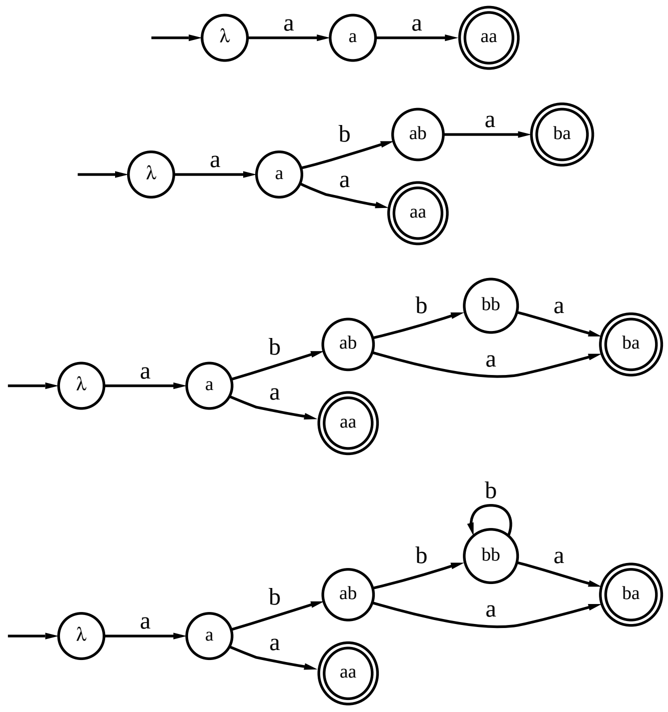


FIG. 7.1: Exemple d'exécution de l'algorithme k-TSSI.

[6] Par défaut, nous appelons complexité d'un algorithme d'inférence sa complexité temporelle asymptotique.

Algorithme k-TSSI

entrée k //L'ordre du modèle
$I_+ = \{x_1, \ldots, x_{|I_+|}\}$ //L'échantillon positif
sortie $A_k = (Q, \Sigma, \delta, q_0, F)$ //Un AFD acceptant le plus petit langage k-TSS incluant I_+

début

$Q \leftarrow q_0$
$\Sigma \leftarrow \phi$
$\delta \leftarrow \phi$
$q_0 \leftarrow \lambda$ // Les états sont représentés par des éléments de Σ^*
$F \leftarrow \phi$
pour $i = 1$ **jusqu'à** $|I_+|$ // Boucle sur les chaînes de I_+
    $q' \leftarrow q_0$
    pour $j = 1$ **jusqu'à** $|x_i|$ // Boucle sur les lettres de x_i
        $\Sigma \leftarrow \Sigma \cup x_{ij}$
        $y \leftarrow q' x_{ij}$
        si $|y| > k - 1$ **alors**
            $y \leftarrow y_{2,\ldots,|y|}$
        fin si
        $q \leftarrow y$
        $Q \leftarrow Q \cup \{q\}$
        $\delta \leftarrow \delta \cup \{q', x_{ij}, q\}$
        si $j = |x_i|$ **alors**
            $F \leftarrow F \cup \{q\}$
        fin si
        $q' \leftarrow q$
    fin pour
fin pour
retourner A_k
fin k-TSSI

Un automate inféré par l'algorithme k-TSSI est, par construction, non ambigu. Une estimation selon le critère du maximum de vraisemblance des probabilités des transitions de l'automate peut être obtenue par le calcul des fréquences relatives d'utilisation des transitions lors de l'analyse d'un ensemble donné de chaînes. Cette estimation peut être faite incrémentalement et simultanément avec la construction de l'automate. Ce modèle stochastique est alors exactement équivalent à un N-gram, avec $N = k$ [Segarra93].

7.1.2 Inférence de langages k-réversibles

Angluin a proposé une méthode d'inférence qui identifie à la limite la classe des langages *k-réversibles*, pour k quelconque mais fixé a priori [Angluin82].

Nous détaillons ci-après les principales notions nécessaires à la définition de cette méthode d'inférence.

Définition 7.2 Etant donné un automate $A = (Q, \Sigma, \delta, I, F)$, son automate inverse $A^r = (Q, \Sigma, \delta^r, F, I)$ est défini comme suit :

$$\forall q \in Q, \quad \forall a \in \Sigma, \quad \delta^r(q, a) = \{q' \in Q | q \in \delta(q', a)\}.$$

Par conséquent, l'automate inverse A^r est obtenu en inversant les états[7] d'entrée et de sortie de l'automate original A, et en inversant également le sens des transitions de A. Nous présentons à la figure 7.2 un exemple d'automate et son inverse.

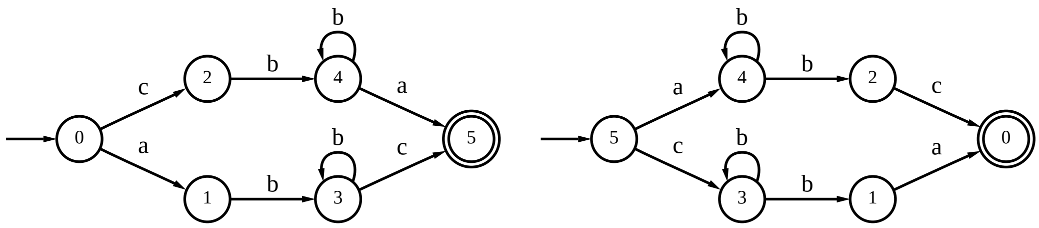


FIG. 7.2: L'automate A et son inverse A^r.

Définition 7.3 Un automate $A = (Q, \Sigma, \delta, q_0, F)$ est *déterministe avec anticipation k* si[8]

$$\forall q \in Q, \forall x \in \Sigma^*, |x| > k, \quad |\delta^*(q, x)| \leq 1.$$

Par conséquent, un automate est déterministe avec anticipation k si, lors de l'acceptation d'une chaîne quelconque, il existe au plus un état auquel cette acceptation peut mener en anticipant de k lettres par rapport à la position courante dans la chaîne. Un AFD est déterministe avec anticipation 0. L'automate A^r de la figure 7.2 est déterministe avec anticipation 1.

Définition 7.4 Un automate A est *k-réversible* s'il est déterministe et si son inverse A^r est déterministe avec anticipation k.

[7]Nous avons défini au paragraphe 6.2, les automates finis comme comprenant un seul état d'entrée et un ensemble d'états de sortie. Ces automates permettent de représenter un langage régulier quelconque. La généralisation aux automates comportant un ensemble d'états d'entrée est triviale.

[8]δ^* désigne l'extension classique à $Q \times \Sigma^* \to 2^Q$, de la fonction de transition δ.

Définition 7.5 Un langage L est *k-réversible* s'il existe un automate k-réversible qui l'accepte.

Une définition équivalente est la suivante.

Définition 7.6 Un langage L est *k-réversible* si et seulement si lorsque deux préfixes de L, (w_1 et $w_2 \in Pr(L)$), contenant un même suffixe v de longueur k, ($w_1 = u_1 v$ et $w_2 = u_2 v$, avec $|v| = k$), ont un même suffixe w dans L, ($u_1 v w$ et $u_2 v w \in L$), alors ils ont même ensemble de finales ($L/w_1 = L/w_2$).

Partant de l'accepteur des préfixes $PTA(I_+)$, l'algorithme k-RI[9] consiste à fusionner des états, c'est-à-dire à définir un nouvel automate quotient $PTA(I_+)/\pi$, tant que cet automate n'est pas k-réversible. Plus précisément, la partition π des états du $PTA(I_+)$ est initialement celle qui associe un bloc à chacun de ses états. Ensuite, on détermine récursivement dans l'automate quotient $PTA(I_+)/\pi$, deux états distincts représentés par les blocs B_1 et B_2 tels qu'au moins une des trois conditions suivantes (C1, C2a ou C2b) soit vérifiée.

1. $\exists a \in \Sigma, \exists B \in \pi$ avec $B_1 \in \delta(B, a)$ et $B_2 \in \delta(B, a)$, (C1 : cas d'indéterminisme sur la lettre a).

2. $\exists v, u_1, u_2 \in \Sigma^*, |v| = k$ avec $B_1 \in \delta^*(q_0, u_1 v)$ et $B_2 \in \delta^*(q_0, u_2 v)$, ($v$ est le suffixe commun des préfixes menant respectivement à B_1 et B_2), et

   (a) B_1 et B_2 sont des états d'acceptation de l'automate quotient (C2a : un suffixe de longueur k menant à deux états d'acceptation).

   (b) $\exists a \in \Sigma, \exists B \in \pi$ avec $B \in \delta(B_1, a)$ et $B \in \delta(B_2, a)$ (C2b : un suffixe de longueur k menant à deux états d'où sont issues des transitions pointant vers le même état B sur la lettre a).

Lorsque deux blocs B_1 et B_2 sont identifiés, un nouvel automate quotient est obtenu par fusion des blocs B_1 et B_2. La partition des états du $PTA(I_+)$ est progressivement mise à jour jusqu'à ce qu'il n'existe plus aucune paire de blocs satisfaisant au moins une des trois conditions décrites. L'automate quotient est alors nécessairement k-réversible. Notons que l'ordre exact des paires de blocs à fusionner est arbitraire.

Nous présentons à la figure 7.3 un exemple d'exécution de l'algorithme k-RI avec $I_+ = \{ab, bb, aab, abb\}$ et $k = 1$. La partition initiale de l'arbre des préfixes est $\{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}\}$. La condition C2a est

[9] L'abréviation **k-RI** provient de la terminologie anglaise pour "k-Reversible Inference".

vérifiée, par exemple, pour les blocs $\{5\}$ et $\{7\}$, avec $v = b$ un suffixe de longueur 1. De même , la condition C2a est vérifiée pour les blocs $\{4\}$ et $\{6\}$, avec également $v = b$. Après deux étapes de l'algorithme k-RI, on obtient le second automate représenté à la figure 7.3. Il correspond à la partition $\{\{0\},\{1\},\{2\},\{3\},\{4,6\},\{5,7\}\}$. La condition C2a est vérifiée pour les blocs $\{4,6\}$ et $\{5,7\}$, avec $v = b$. Après fusion de ces deux blocs, la partition est $\{\{0\},\{1\},\{2\},\{3\},\{4,5,6,7\}\}$. Finalement, la condition C2b est vérifiée pour les blocs $\{1\}$ et $\{3\}$, d'une part, et les blocs $\{2\}$ et $\{4,5,6,7\}$ d'autre part. La partition résultante est $\{\{0\},\{1,3\},\{2,4,5,6,7\}\}$; elle correspond à un automate quotient qui est 1- réversible.

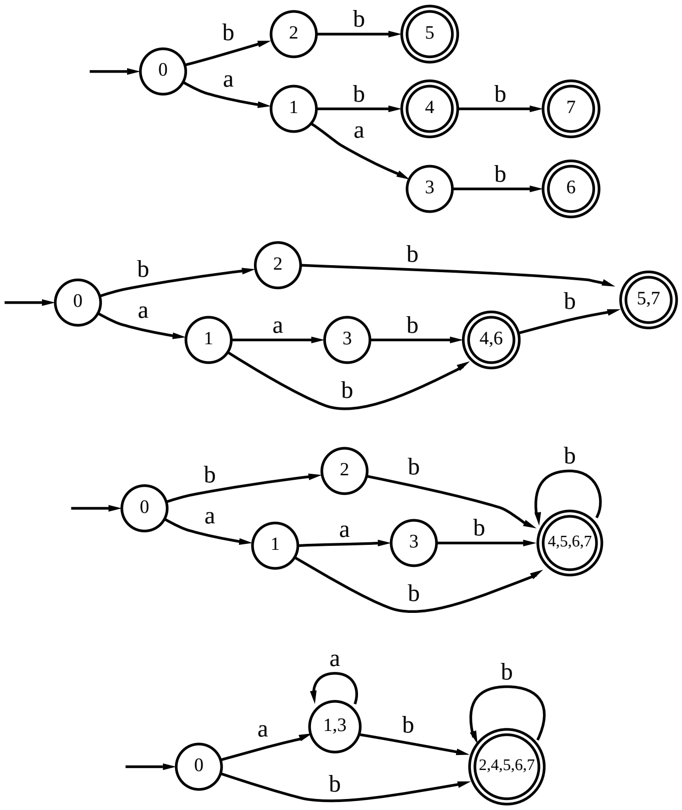


FIG. 7.3: Exemple d'exécution de l'algorithme k-RI.

Algorithme k-RI

entrée k //L'ordre du modèle
I_+ //L'échantillon positif
sortie A_k//Un automate canonique acceptant le plus petit langage k-réversible incluant I_+

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\},\{1\},\ldots,\{N-1\}\}$ // Un bloc par états du $PTA(I_+)$
$A \leftarrow PTA(I_+)$

tant que $((B_1,B_2) \leftarrow$ *non-réversible* $(A/\pi,\pi,k)) \neq (\phi,\phi)$ **faire**
$\pi' \leftarrow \pi\backslash\{B_1,B_2\}U\{B_1UB_2\}$ // Fusion du bloc B_1 et du bloc B_2
$A/\pi' \leftarrow$ *dériver* (A,π')
$\pi \leftarrow \pi'$
fait
retourner A/π
fin k-RI

La fonction *non-réversible* $(A/\pi,\pi,k)$ renvoie une paire de blocs distincts, de la partition π, qui vérifient au moins une des trois conditions citées. Si aucune paire de bloc ne satisfait au moins l'une de ces conditions, elle renvoie (ϕ,ϕ). La fonction *dériver* (A,π') renvoie l'automate quotient de A conformément à la partition π'.

Si l'on pose $n = ||I_+||+1$, remarquons que le $PTA(I_+)$ comporte au plus n états et $n-1$ transitions. L'opération qui domine le coût calculatoire de cet algorithme est la fonction *non-réversible.* Moyennant l'introduction de $k+1$ matrices auxiliaires (voir [Angluin82]), cette fonction est calculable avec une complexité de $\mathcal{O}(k.n^2)$ pour $\mathcal{O}(n^2)$ paires de blocs à tester. Puisqu'il y a au plus $n-1$ fusions à effectuer, la complexité globale de l'algorithme k-RI est $\mathcal{O}(k.n^3)$.

Angluin a démontré que l'algorithme k-RI permet d'identifier la classe des langages k-réversibles et qu'il produisait, pour I_+ et k fixés, l'automate canonique du plus petit langage k-réversible incluant I_+ [Angluin82]. Notons finalement que la classe des langages k-réversibles inclut strictement la classe des langages $k-1$ testables au sens strict [Vidal et al.93].

7.1.3 L'algorithme MGGI

L'algorithme MGGI constitue une extension de l'inférence de langages locaux, qui permet d'introduire, par le biais d'un morphisme, une connaissance a priori sur le langage à inférer [García et al.87, García et al.94]. Il exploite le théorème énoncé ci-après.

Définition 7.7 Si E et H désignent deux alphabets finis et E^*, H^* les monoïdes libres correspondants, un *morphisme* $h : E^* \to H^*$ est une fonction qui satisfait les deux conditions suivantes :

$$\forall x, y \in E^* : h(xy) = h(x)h(y), h(\lambda) = \lambda.$$

De plus, la fonction h est *lettre-à-lettre* si $h(a) \in H$, $\forall a \in E$.

Théorème 7.1 *Soit E, un alphabet fini et soit $L \subset E^+$, un langage régulier. Il existe un alphabet fini E', un morphisme lettre-à-lettre $h : E'^* \to E^*$ et un langage local l de E'^* tel que $L = h(l)$ [Gross et al.70].*

Ce théorème stipule donc que tout langage régulier peut être obtenu à partir d'un langage local et l'application d'un morphisme lettre-à-lettre.

Etant donné un échantillon positif $I_+ \subseteq E^*$, l'algorithme MGGI requiert la définition d'une fonction $g : I_+ \to E'^*$, telle que $g(I_+) = I'_+$ et d'un morphisme lettre-à-lettre $h : E'^* \to E^*$. Si nous désignons par $l_{g(I_+)}$, le langage 2-TSS inféré par l'algorithme présenté au paragraphe 7.1.1, le langage inféré par l'algorithme MGGI est défini par l'expression $L = h(l_{g(I_+)})$. En d'autres termes, les lettres des chaînes de I_+ sont renommées conformément à la fonction g, donnant lieu à un échantillon positif I'_+ défini sur un nouvel alphabet. A partir de l'échantillon I'_+, un langage local est inféré et ensuite une procédure de renommage (généralement inverse) est appliquée par l'intermédiaire du morphisme g lettre-à-lettre.

Exemple :

$$
\begin{aligned}
&E = \{a, b\},\\
&I_+ = \{abba, aaabba, bbaaa, bba\},\\
&E' = \{a^1, a^2, a^3, b^1, b^2\},\\
&I'_+ = g(I_+) = \{a^1b^1b^2a^1, a^1a^2a^3b^1b^2a^1, b^1b^2a^1a^2a^3, b^1b^2a^1\},\\
&h(\alpha^i) = \alpha, \forall \alpha^i \in E'.
\end{aligned}
$$

Les chaînes de l'échantillon I_+ sont faites de séquences de a, de longueur 1 ou 3, et de séquences de b, de longueur 2. Dans la mesure où l'on considère qu'il s'agit d'une information pertinente sur le langage à inférer, il est possible de définir la fonction g qui attribue des indices différents à la même lettre, a ou b, dans une sous-chaîne donnée. Les lettres des chaînes de l'échantillon positif I_+ sont renommées par la fonction g. Sur base du nouvel échantillon positif I'_+, un langage local, ou 2-TSS, est inféré. Il correspond à l'automate représenté à la figure 7.4. Après application du morphisme h qui, dans ce cas, a pour effet de supprimer les indices attachés aux lettres, nous obtenons le langage représenté par l'automate de la figure 7.5. Remarquons qu'il correspond bien au langage acceptant des séquences de a, de longueur 1 ou 3, et des séquences de b, de longueur 2.

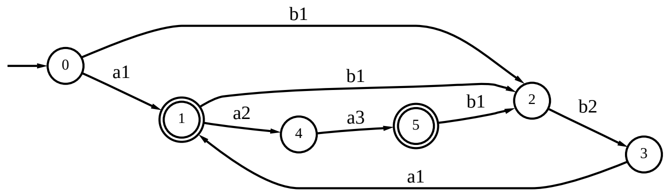


FIG. 7.4: Automate acceptant le langage local sur l'alphabet E'.

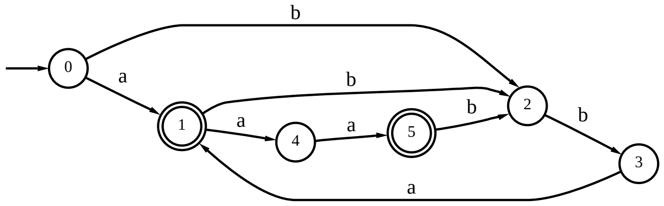


FIG. 7.5: Automate inféré par l'algorithme MGGI.

L'algorithme MGGI offre la possibilité d'introduire un biais d'apprentissage par l'intermédiaire des fonctions g et h. Par conséquent, il est intéressant d'appliquer cet algorithme si nous disposons d'une autre information que celle simplement contenue dans l'échantillon I_+. Notons que lorsque les fonctions g et h consistent à renommer chaque lettre par elle-même, l'algorithme MGGI correspond exactement à l'inférence d'un langage 2-testable au sens strict.

La complexité de l'algorithme MGGI dépend de celle de l'inférence du langage 2-testable qui est $\mathcal{O}(||I'_+||)$ (voir paragraphe 7.1.1). L'application du morphisme h lettre-à-lettre est également linéaire en fonction de $||I'_+||$. Par contre, le lien entre $||I'_+||$ et $||I_+||$, dépend de la fonction g. Généralement, $||I'_+|| = ||I_+||$ et la complexité globale est $\mathcal{O}(||I_+||)$. Ce résultat n'est applicable que si la fonction g ne donne pas lieu à un allongement des chaînes de l'échantillon positif.

7.1.4 L'algorithme ECGI

L'algorithme ECGI est une méthode d'inférence heuristique conçue pour extraire l'information sur la longueur de sous-chaînes des éléments de I_+ et de la concaténation de ces sous-chaînes. Cet algorithme est incrémental, c'est-à-dire qu'il mesure une "distance" entre un nouvel exemple et le modèle existant et adapte le modèle conformément. Le langage accepté par la grammaire inférée inclut I_+ ainsi que d'autres chaînes obtenues par concaténation de sous-chaînes des éléments de I_+.

Nous reprenons la description de l'algorithme ECGI tel qu'il a été formellement présenté dans [Rulot et al.88, Rulot92]. En particulier, la représentation utilisée est directement celle des grammaires régulières. Nous illustrons également son fonctionnement à l'aide de la représentation équivalente en automates.

Propriété 7.1 *[Rulot et al.88] L'algorithme ECGI produit une grammaire $G = (V, \Sigma, P, S)$ régulière, non-déterministe, sans cycle ($\nexists (A \overset{+}{\Rightarrow} \alpha A), A \in V, \alpha \in \Sigma^*$) et qui vérifie la condition suivante :*

$$\forall A, B, C \in V, \quad \forall a, b \in \Sigma \quad si\ (B \rightarrow aA) \in P\ et\ (C \rightarrow bA) \in P\ alors\ a = b.$$

En d'autres termes, 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.

Définition 7.8 A la grammaire $G = (V, \Sigma, P, S)$ sont associés les *règles* *d'erreurs* suivantes :

$$
\begin{array}{rl}
\text{Insertion de } a : & A \to aA, \forall A \in V, \forall a \in \Sigma \\
\text{Substitution de } b \text{ par } a : & A \to aB, \forall (A \to bB) \in P, \forall a \in \Sigma, a \neq b \\
 & A \to a, \forall (A \to b) \in P, \forall a \in \Sigma, a \neq b \\
\text{Suppression de } b : & A \to B, \forall (A \to bB) \in P \\
 & A \to \lambda, \forall (A \to b) \in P
\end{array}
$$

Par conséquent, il y a $\mathcal{O}(|\Sigma|)$ règles d'insertion par non-terminal de la grammaire G (ou par état de l'automate équivalent) et $\mathcal{O}(|\Sigma|)$ règles de substitution ou de suppression par production de la grammaire G (ou par transition de l'automate équivalent).

Définition 7.9 La *grammaire étendue* $G' = (V, \Sigma, P', S)$ de G est la grammaire obtenue en ajoutant les règles d'erreurs à P.

Définition 7.10 La *dérivation corrective optimale* de la chaîne $\beta \in \Sigma^*$ est la dérivation de β, conformément à la grammaire étendue G' et qui utilise un nombre minimal de règles d'erreurs.

L'algorithme ECGI consiste à construire la grammaire canonique acceptant la première chaîne de I_+. Ensuite, pour chaque nouvel exemple β la dérivation corrective optimale est calculée et la grammaire est étendue conformément à cette dérivation. Cette dérivation est obtenue par une procédure d'analyse corrective qui constitue une application de l'algorithme de Viterbi [Forney73]. Elle produit la séquence optimale de règles correctes (celles déjà présentes dans la grammaire) et de règles d'erreurs pour l'acceptation de β.

Nous présentons à la figure 7.6 un exemple d'exécution de l'algorithme ECGI prenant en entrée l'échantillon $I_+ = \{aabb, abbb, abbab, bbb\}$ et en représentant chaque grammaire par un automate associé. Les états et transitions en pointillés correspondent à l'extension de la grammaire à chaque étape. Après la construction de l'automate acceptant la première chaîne $aabb$, une analyse corrective construit la dérivation corrective optimale de la chaîne $abbb$. Dans ce cas, la seule règle d'erreur appliquée est la substitution du second a par b. L'automate résultant est représenté à la figure 7.6. Notons que l'état 3 est créé afin de garantir la propriété 7.1 qui stipule que toutes les transitions pointant sur un état doivent être étiquetées par le même symbole. L'analyse corrective de la chaîne $abbab$ conduit à l'insertion d'un symbole a et à la création correspondante de l'état 5. Finalement, l'analyse corrective de la chaîne bbb détermine une erreur de suppression d'un symbole a en début de chaîne. L'automate final est obtenu en ajoutant une transition étiquetée par b et reliant directement l'état initial à l'état 3.

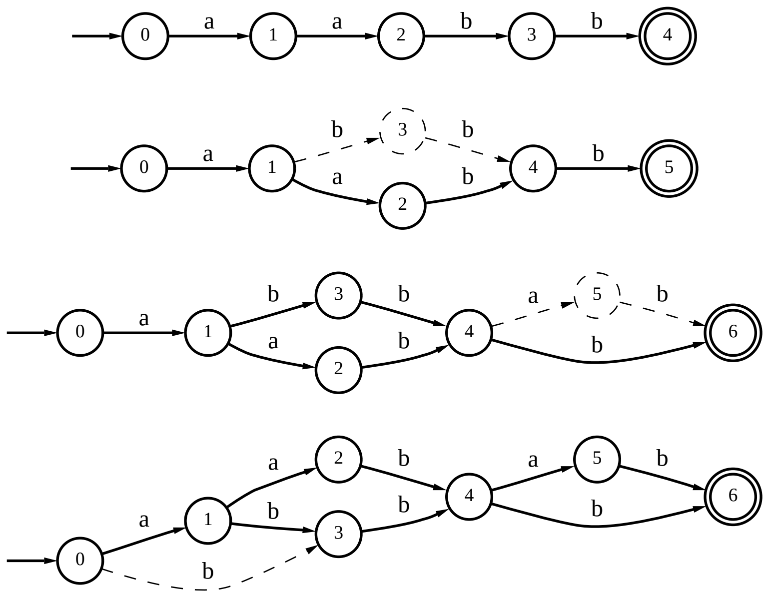


FIG. 7.6: Exemple d'exécution de l'algorithme ECGI.

Algorithme ECGI

entrée I_+
sortie Une grammaire G compatible avec I_+ et vérifiant la propriété 7.1

début

$x \leftarrow I_+^1$
$n \leftarrow |x|$
$V \leftarrow \{A_0, \ldots, A_{n-1}\}$
$\Sigma \leftarrow \{a_1, \ldots, a_n\}$
$P \leftarrow \{(A_{i-1} \rightarrow a_i A_i), i = 1, \ldots, n-1\} \cup \{A_{n-1} \rightarrow a_n\}$
$S \leftarrow A_0$
$G_1 \leftarrow (V, \Sigma, P, S)$
pour $i = 2$ **jusqu'à** $|I_+|$ // Boucle sur les éléments de I_+
  $G \leftarrow G_{i-1}$
  $x \leftarrow I_+^i$
  $D \leftarrow$*dériv_optim* (x, G_{i-1})
  pour $j = 1$ **jusqu'à** $|D|$ // Boucle sur les règles de la dérivation optimale D
    $G \leftarrow$*étendre_gram* (G, p_j)
  fin pour
  $G_i \leftarrow G$
fin pour
retourner G
fin ECGI

La fonction *dériv_optim* (x, G_{i-1}) renvoie la dérivation corrective optimale de la chaîne x conformément à la grammaire G_{i-1}. La fonction *étendre_gram* (G, p_j) renvoie la grammaire étendue conformément à la règle p_j.

La complexité de l'algorithme ECGI est dominée par la fonction d'analyse corrective. Pour chaque chaîne x, l'analyse représente $\mathcal{O}(T.|x|)$ opérations où T est le nombre de transitions de l'automate correspondant à la grammaire étendue. Comme il y a au plus $\mathcal{O}(||I_+||)$ transitions et $\mathcal{O}(||I_+||)$ états dans l'automate inféré, l'automate incluant les règles d'erreurs comporte $\mathcal{O}(|\Sigma|.||I_+||)$ transitions. L'analyse corrective d'une chaîne x ayant une complexité de $\mathcal{O}(|\Sigma|.||I_+||.|x|)$, la complexité globale de l'algorithme est $\mathcal{O}(|\Sigma|.||I_+||^2)$.

Il existe une extension stochastique de l'algorithme ECGI qui permet d'inférer une grammaire régulière stochastique par une procédure incrémentale en une seule passe sur l'échantillon positif [Rulot92]. En effet, les probabilités associées aux productions de la grammaire peuvent être mise à jour par comptage de la fréquence d'utilisation des règles lors de l'analyse corrective de chaque nouvelle chaîne.

Un exemple intéressant d'application de l'algorithme ECGI est présenté à l'annexe C.

7.2 Inférence avec un échantillon négatif

7.2.1 Les algorithmes RIG et BRIG

Etant donné des échantillons positif I_+ et négatif I_-, un algorithme, nommé RIG[10] [Miclet et al.94], de construction complète de l'ensemble frontière $BS_{MCA}(I_+, I_-)$ a été proposé par Miclet. Cet algorithme procède par énumération des automates dérivés du $MCA(I_+)$, c'est-à-dire des partitions dans $Lat(MCA(I_+))$. Il s'agit d'une énumération en largeur à partir de l'élément nul du treillis des partitions, le $MCA(I_+)$. Cette énumération ne conserve que les automates compatibles à chaque profondeur dans le treillis. Par la propriété 6.1 d'inclusion des langages, nous pouvons effectuer un *élagage par héritage*, qui consiste à éliminer tous les automates à la profondeur $i+1$ qui dérivent d'au moins un automate non compatible à la profondeur i. Ensuite, les automates restant à la profondeur $i+1$ subissent un *élagage direct* qui consiste à éliminer les automates non compatibles à cette profondeur. Finalement, l'ensemble frontière est obtenu en mémorisant tous les automates compatibles et qui ne possèdent aucun dérivé compatible. La solution proposée par l'algorithme RIG est, par exemple, le premier automate déterministe rencontré à la profondeur maximale de l'ensemble frontière, mais l'algorithme RIG peut également fournir comme solution tout l'ensemble frontière.

Dans l'algorithme RIG original, des automates identiques peuvent être engendrés à plusieurs reprises. En effet, les partitions à la profondeur $i+1$ sont obtenues en fusionnant, de toutes les façons possibles, deux blocs d'une partition à la profondeur i. Nous présentons ci-dessous une version améliorée de l'algorithme RIG qui évite ce traitement redondant [Miclet et al.95]. Elle consiste à introduire un ordre sur les partitions à une profondeur donnée[11] et à définir un ensemble $G(i)$ de partitions, dites *génératrices*. Il s'agit des partitions de rang minimum à partir desquelles peuvent être engendrées toutes les partitions à la profondeur suivante : chaque nouvelle partition à la profondeur $i+1$ est engendrée à partir de l'union de deux blocs d'une partition génératrice à la profondeur i. L'ensemble $NG(i)$ désigne les autres partitions,

[10] L'abréviation **RIG** provient de la terminologie anglaise pour "Regular Inference of Grammars".

[11] Chaque bloc d'une partition prend pour rang celui de son état de rang minimal dans le MCA. Il en résulte un ordre sur les partitions puisqu'elles ont toutes le même nombre de blocs à une profondeur donnée.

dites *non génératrices.*

Algorithme RIG

entrée I_+, I_-
sortie Un AFD appartenant à $BS_{MCA}(I_+, I_-)$

début

// N désigne le nombre d'états du $MCA(I_+)$

$G(0) \leftarrow \{\{\{0\}, \{1\}, ..., \{N-1\}\}\}$ // L'ensemble des partitions génératrices
$NG(0) \leftarrow \phi$ // L'ensemble des partitions non génératrices
$BS \leftarrow \phi$ // L'ensemble frontière

pour $i = 1$ **jusqu'à** $N-1$ // Boucle sur la profondeur
  $G(i) \leftarrow \phi$
  $NG(i) \leftarrow \phi$
  $BS \leftarrow BS \cup NG(i-1)$
  pour $\pi_j = G(i-1)_1$ **jusqu'à** $G(i-1)_{|G(i-1)|}$ // Boucle sur les partitions génératrices
    $BS\_booléen \leftarrow$ VRAI
    tant que ($\pi_k \leftarrow$ *partition_suivante* $(\pi_j)) \neq \phi$ **faire**
      $A \leftarrow$ *dériver* $(MCA(I_+), \pi_k)$
      si *compatible* (A, I_-) **alors** // Elagage par I_-
        $BS\_booléen \leftarrow$ FAUX
        $BS \leftarrow BS \backslash$*dérivable* $(\pi_k, NG(i-1))$
        si *rang_minimum* (π_k)
          alors $G(i) \leftarrow G(i) \cup \pi_k$
          sinon $NG(i) \leftarrow NG(i) \cup \pi_k$
        fin si
      fin si
    fait
    si $BS\_booléen$ **alors**
      $BS \leftarrow BS \cup \pi_j$
    fin si
  fin pour
fin pour
retourner *extraire_solution* (BS)
fin RIG

La fonction *partition_suivante* (π_j) renvoie, à chaque appel, la partition suivante qui peut être dérivée de la partition génératrice π_j. Lorsque toutes ces partitions ont été générées, elle renvoie ϕ. La fonction *dériver* $(MCA(I_+), \pi_k)$ renvoie l'automate quotient du $MCA(I_+)$ conformément à la partition π_k. La fonction *compatible* (A, I_-) renvoie VRAI si l'automate A n'accepte aucune chaîne de l'échantillon négatif I_-, sinon elle renvoie FAUX. La fonction *dérivable* $(\pi_k, NG(i-1))$ renvoie le sous-ensemble des partitions non génératrices

$NG(i-1)$ dont peut dériver la partition π_k. La fonction *rang_minimum* (π_k) renvoie VRAI si la partition π_k est de rang minimum, sinon elle renvoie FAUX. La fonction *extraire_solution* (BS) renvoie, par exemple, le premier automate déterministe à la profondeur maximale appartenant au BS.

L'algorithme RIG construit par énumération et sans redondance tous les éléments de l'ensemble frontière BS. Il n'existe pas, à l'heure actuelle, de preuve de l'identification à la limite de la classe des langages réguliers par l'algorithme RIG. Cela nécessiterait de caractériser la solution extraite du BS pour un ensemble croissant de données et de démontrer la convergence (voir paragraphe 6.3.3).

La complexité de RIG dépend du nombre de partitions distinctes construites et implique une analyse, en général non-déterministe, pour vérifier la compatibilité de chaque automate quotient par rapport à I_-. La complexité globale étant non-polynomiale[12], un algorithme approximatif, nommé BRIG[13], a été proposé [Miclet et al.94]. Il consiste à ne considérer qu'une proportion finie, générée aléatoirement, des partitions pouvant être dérivées des génératrices. Cette sélection aléatoire permet de garantir, en pratique, une complexité polynomiale et de construire un sous-ensemble du BS. L'algorithme BRIG est clairement de nature heuristique car il n'existe pas de caractérisation de la classe de langages qu'il identifie. Cette caractérisation n'est pas envisageable car son caractère aléatoire implique que différentes exécutions partant des mêmes données ne conduisent pas nécessairement au même résultat.

Remarquons que la construction complète du BS, par l'algorithme RIG, ou d'un sous-ensemble du BS, par l'algorithme BRIG, peut sembler coûteuse dans la mesure où nous ne retenons qu'une solution à la profondeur maximale. Dans ce cas, une simplification évidente de l'algorithme consiste à ne mémoriser le BS qu'à la dernière profondeur où des automates compatibles sont rencontrés. Par ailleurs, l'étude des différents langages acceptés par les automates du BS permettrait probablement de ne retenir qu'un sous-ensemble "pertinent" des automates du BS.

[12]Dans le pire des cas, aucun élagage par I_- n'est effectué. Dans ce cas, l'algorithme RIG explore tout le treillis des partitions d'où sa complexité exponentielle. Malgré un élagage effectif par l'échantillon négatif, il a été observé expérimentalement que la complexité de RIG reste non polynomiale en fonction de N, le nombre d'états du $MCA(I_+)$ [Miclet et al.94, Miclet et al.95].

[13]L'abréviation **BRIG** provient de la terminologie anglaise pour "Boosted beam-search Regular Inference of Grammars".

7.2.2 L'algorithme RPNI

L'algorithme RPNI [Oncina et al.92], développé indépendamment par Lang [Lang92], effectue une recherche en profondeur dans $Lat(PTA(I_+))$ et trouve une solution "localement optimale" au problème du plus petit AFD compatible. Nous savons, par la propriété 6.6 que l'identification d'un langage régulier L par exploration du treillis des partitions du $PTA(I_+)$ n'est envisageable que si l'on suppose la complétude structurelle de I_+ relativement à l'automate canonique $A(L)$. Sous cette hypothèse, l'algorithme RPNI est particulièrement efficace.

Par construction du $PTA(I_+)$ (voir définition 6.18), chacun de ses états correspond à un préfixe unique et les préfixes peuvent être triés conformément à l'ordre standard[14] "$<$". Cet ordre s'applique donc également aux états du $PTA(I_+)$. Une partition de l'ensemble des états du $PTA(I_+)$ est faite d'un ensemble de blocs, chaque bloc recevant l'indice de l'état de rang minimal qu'il contient. L'algorithme RPNI procède en $N-1$ étapes où N est le nombre d'états du $PTA(I_+)$. La partition à l'étape i est obtenue en fusionnant les deux premiers blocs, selon l'ordre standard, de la partition à l'étape $i-1$ et qui, de plus, donne lieu à un automate quotient compatible.

[14] L'ordre standard, parfois appelé *ordre hiérarchique*, sur les chaînes de Σ^*, correspond à un ordre par longueur des chaînes et, pour une longueur donnée, à l'ordre alphabétique. Par exemple, pour l'alphabet $\{a, b\}$, les premières chaînes dans l'ordre standard sont : $\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots$

Notons que les auteurs de l'algorithme RPNI désigne, improprement, cet ordre comme étant l'ordre lexicographique.

Algorithme RPNI

entrée I_+, I_-
sortie Un AFD compatible avec I_+, I_-

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \ldots, \{N-1\}\}$ // Un bloc par préfixe dans l'ordre <
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    $\pi'' \leftarrow$ *fusion_déterm* (A/π')
    si *compatible* $(A/\pi'', I_-)$ **alors** // Analyse déterministe de I_-
      $A \leftarrow A/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI

La fonction *dériver* (A, π') renvoie l'automate quotient de A conformément à la partition π'. L'automate quotient A/π' peut être non déterministe. La fonction *fusion_déterm* (A/π') renvoie la partition π'' obtenue en fusionnant récursivement tous les blocs de π' qui créent le non-déterminisme[15]. Si A/π' est déterministe, la partition π'' est égale à la partition π'.

Sachant que le $PTA(I_+)$ comporte $\mathcal{O}(\|I_+\|)$ états, la complexité de la fonction *fusion_déterm* est $\mathcal{O}(\|I_+\|)$. De plus, comme l'automate quotient A/π'' est nécessairement déterministe, le test de compatibilité avec I_- s'effectue en $\mathcal{O}(\|I_-\|)$. La complexité globale de l'algorithme RPNI est donc $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$.

La solution proposée par l'algorithme RPNI est un automate déterministe appartenant au $BS_{PTA}(I_+, I_-)$. Par la propriété 6.10, nous savons qu'il

[15] L'opération de fusion pour déterminisation d'un AFN ne doit pas être confondue avec l'algorithme classique de déterminisation d'un AFN, qui a pour objet de produire un AFD acceptant le même langage [Aho et al.72]. Conformément à la propriété 6.1, l'AFD obtenu par fusion des états d'un AFN accepte, en général, un sur-langage de l'AFN dont il provient.

s'agit de l'automate canonique pour le langage qu'il accepte. Cependant, il ne s'agit du plus petit AFD compatible que si les données d'apprentissage satisfont une condition supplémentaire, c'est-à-dire qu'elles contiennent un *échantillon* dit *caractéristique,* formellement défini dans [Oncina et al.92]. En d'autres termes, lorsque les données d'apprentissage sont suffisamment représentatives, la découverte de l'automate canonique du langage à identifier, est garantie. De plus, cet automate est également la solution du problème du plus petit AFD compatible, dans ce cas particulier. Remarquons qu'Oncina et García ont démontré que la taille d'un échantillon caractéristique propre à cet algorithme est $\mathcal{O}(n^2)$, où n est le nombre d'états de l'automate cible [Oncina et al.92].

Vu l'intérêt de l'algorithme RPNI et son efficacité calculatoire, nous en avons proposé une version incrémentale qui sera détaillée au chapitre 9. Nous reviendrons alors sur la définition exacte d'un échantillon caractéristique qui garantit l'identification de l'automate cible. Nous étudierons également les résultats de complexité que nous venons d'énoncer.

7.2.3 L'algorithme GIG (*)

Nous avons présenté dans [Dupont94] un algorithme heuristique, dénommé GIG[16], qui optimise par un algorithme génétique la recherche d'une solution optimale, c'est-à-dire à la profondeur maximale. Il fait évoluer dans $Lat(PTA(I_+))$ une population d'automates dérivés du $PTA(I_+)$, par le biais d'opérateurs spécifiques de mutation et de croisement. L'algorithme GIG sera exposé plus complètement au chapitre 8.

7.2.4 Le point de vue connexionniste

Nous terminons ce paragraphe en citant un certain nombre de travaux sur l'utilisation de réseaux connexionnistes en inférence grammaticale, habituellement à partir d'échantillons positif et négatif. Le cadre formel de ces travaux, lorsqu'il existe, ne correspond généralement pas au cadre classique de l'inférence tel que nous l'avons présenté. Cependant une équivalence théorique entre les réseaux de McCullough et Pitts [McCullough et al.43] et les automates finis déterministes a été démontrée par Minsky [Minsky54, Minsky67]. La question générale de déterminer la taille (c'est-à-dire le nombre de neurones) d'un réseau afin que celui-ci reproduise exactement le comportement d'un automate donné, est étudiée dans [Alon et al.91].

[16] L'abréviation **GIG** provient de la terminologie anglaise pour “Grammatical Inference by Genetic search”.

Parmi les nombreuses variantes d'architecture, les *réseaux récurrents,* proposés par Elman [Elman88], ont été utilisés pour approcher le comportement d'automates finis [Cleeremans et al.89]. Des extensions de ces modèles afin d'approximer les *automates à piles* ont été proposées, par exemple, par Pollack [Pollack90] ou par Giles [Giles et al.90].

Des techniques heuristiques ayant pour objet d'approximer la structure d'un langage sont décrites dans [Gorin et al.91, Jain et al.92]. Par ailleurs, Smith et Miller étudient l'apprentissage de grammaires régulières stochastiques selon un critère d'information mutuelle [Smith et al.89] et en proposent une implémentation distribuée. Finalement, Nakamura et Shikano ont développé les NETgrams comme approximation des N-grams. L'avantage potentiel des NETgrams est qu'ils ne requièrent pas un nombre de paramètres libres exponentiellement croissant avec N [Nakamura et al.89].

7.3 Inférence régulière semi-incrémentale (*)

Nous avons vu au paragraphe 6.5.2 que la taille de l'espace de recherche était directement liée au nombre d'états de l'élément nul du treillis. Il en résulte une propriété paradoxale de l'inférence régulière. En effet, plus nous disposons d'information positive sur un langage, c'est-à-dire plus $|Q_{MCA(I_+)}|$ augmente, plus la taille de l'espace de recherche croît, rendant l'identification plus difficile. C'est la raison pour laquelle, nous proposons une procédure d'inférence *semi-incrémentale.* Elle suppose que nous disposions d'une certaine information positive et négative fixée mais que nous définissions un algorithme faisant usage de l'information positive de façon incrémentale [Dupont94, Miclet et al.95].

Nous trions l'échantillon positif I_+ par ordre standard, ce qui nous permet de considérer en premier lieu les chaînes les plus courtes. A partir de la première chaîne I_+^1, nous construisons l'automate $MCA(I_+^1)$ et nous recherchons la partition optimale sous contrôle de l'échantillon négatif complet I_-. Ensuite, nous définissons la procédure semi-incrémentale par la récurrence suivante. Nous désignons par A^i, l'automate quotient relativement à la partition optimale trouvée à l'étape i et par x^{i+1}, la chaîne I_+^{i+1}. Si $x^{i+1} \in L(A^i)$, nous passons à la chaîne suivante. Sinon, nous étendons l'automate A^i afin d'obtenir un nouvel automate $A^i_{étendu}$ tel que $L(A^i_{étendu}) = L(A^i) \cup x^{i+1}$. Il s'agit d'une opération simple de modification d'automate dont nous donnons un exemple à la figure 7.7. Elle peut être effectuée en deux temps : l'ajout d'une nouvelle branche moyennant l'introduction de transitions vides et le calcul de l'automate équivalent sans transitions vides.

Finalement, nous poursuivons la recherche dans $Lat(A^i_{étendu})$ pour trouver

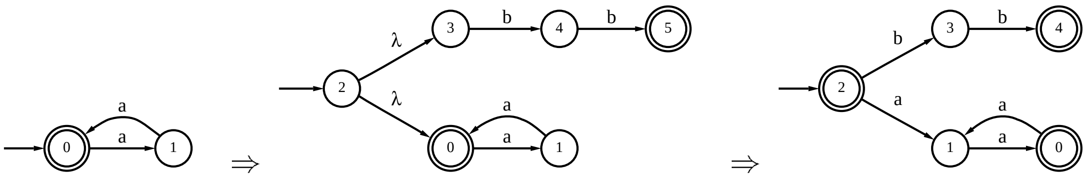


FIG. 7.7: Construction de l'automate acceptant $(aa)^* \cup bb$.

la nouvelle partition optimale sous le contrôle de I_-. Cette procédure semi-incrémentale est poursuivie jusqu'à ce que toutes les chaînes de I_+ aient été considérées.

Procédure semi-incrémentale

entrée I_+, I_-
sortie Un automate compatible avec I_+, I_-

début

$A \leftarrow$*inférer* $(MCA(I_+^1), I_-)$

pour $i = 2$ **jusqu'à** $|I_+|$ // Boucle sur les chaînes de I_+
  $x \leftarrow I_+^i$
  si $(x \notin L(A))$ **alors**
    $A' \leftarrow$*étendre* (A, x)
    $A \leftarrow$*inférer* (A', I_-)
  fin pour
retourner A
fin

Remarquons que la procédure semi-incrémentale constitue une technique heuristique pour limiter le coût calculatoire de l'inférence. Il n'y a aucune garantie que cette réduction permette d'obtenir la solution optimale qui résulterait d'une recherche dans le treillis construit sur l'échantillon I_+ complet. Cette procédure n'est pas seulement utile pour la méthode GIG mais peut s'appliquer à tout méthode de recherche explicite dans le treillis des partitions. Elle présente l'avantage supplémentaire de "filtrer" l'échantillon positif puisque les chaînes de I_+ qui sont déjà acceptées, grâce aux généralisations antérieures, ne doivent plus être considérées. Nous présentons au chapitre 10 des résultats expérimentaux montrant l'intérêt pratique de cette procédure. Notons que l'on peut aussi envisager une procédure semi-incrémentale par sous-ensembles de plusieurs éléments de I_+. Elle consiste à étendre la solution temporaire pour toutes les chaînes non acceptées dans le sous-ensemble courant et à poursuivre l'inférence, sous contrôle de I_-, après ces extensions.

Chapitre 8

Inférence régulière par optimisation génétique (*)

Nous montrons au paragraphe 8.1 que l'inférence régulière peut être vue comme un problème d'optimisation. Cette idée constitue la base d'une nouvelle méthode d'inférence. Elle utilise un algorithme génétique pour faire évoluer un ensemble d'automates dans $Lat(PTA(I+))$.

Après un bref rappel sur les algorithmes génétiques au paragraphe 8.2, nous exposons en détail la méthode proposée au paragraphe 8.3. Le paragraphe 8.4 fournit une première analyse comparative des différentes méthodes concurrentes. Cette analyse sera poursuivie suite à la présentation des résultats expérimentaux au chapitre 10.

8.1 L'inférence régulière : un problème d'optimisation (*)

Nous savons que l'inférence régulière sous contrôle d'échantillons positif et négatif peut être formulée comme un problème de recherche explicite dans un treillis de partitions (voir paragraphes 6.5 et 6.6). Nous pouvons évaluer chaque solution potentielle, c'est-à-dire chaque automate quotient de l'élément nul du treillis, par une fonction f_{I_+,I_-} définie comme suit :

$$f_{I_+,I_-} : P(I_+) \rightarrow \mathbb{R}^+, f_{I_+,I_-}(\pi) = g(r_{I_+}(\pi), k_{I_-}(\pi)),$$

où g est une fonction de pénalité positive dépendant de deux arguments entiers ; $r_{I_+}(\pi)$, désigne le nombre de blocs de la partition π, et $k_{I_-}(\pi)$, désigne le nombre de chaînes de I_-, acceptées par l'automate quotient A/π. Une définition possible de la fonction g est la suivante

$$g(x, 0) = x, \quad g(x, y) = +\infty, \quad y > 0, x \geq 1.$$

Le problème du plus petit AFD compatible devient par conséquent un problème d'optimisation, c'est-à-dire la découverte dans $Lat(PTA(I_+))$ de la partition π pour laquelle f_{I_+,I_-} est minimale et tel que l'automate quotient A/π soit déterministe. En d'autres termes, nous cherchons l'AFD compatible à la profondeur maximale dans le treillis. Remarquons que telle que nous avons défini la fonction f_{I_+,I_-}, il n'y pas de garantie que sa valeur optimale corresponde à un automate déterministe. Nous présentons également au paragraphe 8.3.4 d'autres définitions possibles de la fonction de pénalité g.

Nous avons vu au paragraphe 6.4 que le problème du plus petit AFD compatible est NP-difficile. Dès lors, notre proposition consiste à utiliser une technique d'optimisation pour approximer ce problème. Tomita a développé une méthode heuristique de construction d'un automate compatible par recherche de plus grande pente [Tomita82]. La notion d'optimisation était donc déjà présente dans ce travail, mais la recherche n'était pas restreinte au treillis des partitions. Or, nous avons vu par le théorème 6.1 que la solution se trouve nécessairement dans ce treillis.

8.2 Les algorithmes génétiques

Dans ce paragraphe, nous exposons brièvement les *algorithmes génétiques*, ou AGs, conformément à la présentation de Goldberg [Goldberg89].

Un AG est une technique d'optimisation par exploration parallèle. Un ensemble, appelé *population*, de solutions potentielles, appelées *individus*, est progressivement mis à jour par le biais d'un mécanisme de sélection et par des *opérateurs génétiques*, le *croisement* et la *mutation*. Chaque individu est représenté par un code généralement binaire de longueur fixe, appelé *chromosome*. A la première *génération*, la population initiale est engendrée aléatoirement et la qualité de chaque individu est évaluée par le biais d'une *fonction d'adaptation* qui est la fonction à optimiser. Nous obtenons la population de la génération suivante en trois étapes. Premièrement, nous effectuons un tirage aléatoire avec remise, généralement sur l'ensemble de la population. La probabilité de tirer un individu particulier est liée au rapport entre sa valeur d'adaptation et la valeur d'adaptation moyenne de la population. Il s'agit du mécanisme de reproduction d'individus proportionnelle à l'adaptation. Il favorise la survie voire la duplication des meilleurs individus en maintenant une population de taille constante. Ensuite, nous sélectionnons un certain pourcentage, correspondant au *taux de croisement*, de la population temporaire, c'est-à-dire un certain nombre d'individus que nous regroupons par paires. Chaque paire de *parents* donne naissance à deux *enfants*, dont la représentation codée est obtenue en combinant des parties distinctes de la représentation codée des parents. Après l'opération de croisement, nous remplaçons les parents par leurs enfants dans la population temporaire. Enfin, nous sélectionnons aléatoirement quelques individus selon un certain *taux de mutation*. Nous choisissons aléatoirement une position dans le chromosome de chaque individu sélectionné et nous modifions la valeur à cette position. La population de la génération suivante est constituée de l'ensemble des individus obtenus après reproduction proportionnelle à l'adaptation, croisement et mutation. Nous répétons ce procédé jusqu'à satisfaction d'un certain critère d'arrêt[1]. Finalement, le meilleur individu de la population finale constitue la solution proposée, c'est–à-dire celle trouvée comme étant "la plus adaptée" jusqu'alors.

Nous pouvons résumer un AG par l'algorithme général présenté ci-après. Remarquons que notre objectif n'est pas l'étude générale des algorithmes génétiques ; nous désirons montrer qu'un AG, dans sa formulation classique, peut être utilisé en inférence régulière. La formalisation de ce problème particulier, telle qu'elle a été résumée au chapitre 6, nous permet de définir une représentation codée adéquate et des opérateurs génétiques spécifiques. Ces points seront abordés plus en détail au paragraphe 8.3. Le lecteur intéressé par l'étude des AGs pourra, par exemple, consulter les ouvrages [Goldberg89, Koza91, Michalewicz92, Whitley93].

[1]Habituellement, nous arrêtons la recherche lorsque le nombre d'individus distincts ayant été évalués atteint un certain seuil ou lorsque la population a convergé vers un petit nombre d'individus différents.

Algorithme Génétique

var p_c //Le taux de croisement
p_m //Le taux de mutation

début

//*Pop* désigne la taille de la population

$t \leftarrow 0$
initialiser$(P(t))$ //Initialisation de la population
pour $i = 1$ **jusqu'à** *Pop*
  $x \leftarrow P_i(t)$
  évaluer (x) //Evaluation de l'adaptation de chaque individu
fin pour
tant que $\neg$(*condition d'arrêt*) **faire**
  $t \leftarrow t + 1$
  $P(t) \leftarrow$*sélectionner* $(P(t-1))$
  $P(t) \leftarrow$*croiser* $(P(t), p_c)$
  $P(t) \leftarrow$*muter* $(P(t), p_m)$
  pour $i = 1$ **jusqu'à** *Pop*
    $x \leftarrow P_i(t)$
    évaluer (x) //Evaluation de l'adaptation de chaque individu
  fin pour
fait
retourner *meilleur* $(P(t))$
fin

8.3 La méthode d'inférence GIG (*)

L'utilisation d'un AG pour résoudre le problème de l'inférence grammaticale régulière est motivé par les trois considérations suivantes. Premièrement, étant donné la taille de l'espace de recherche qui croît exponentiellement avec $|I_+|$, nous voudrions tirer parti d'une recherche implicitement parallèle. Ensuite, un AG offre un moyen d'optimiser numériquement des données symboliques. Enfin, puisqu'un AG a pour objectif d'améliorer continuellement un ensemble de solutions potentielles, il pourrait offrir la possibilité de converger rapidement vers une nouvelle solution après mise à jour des données d'apprentissage.

8.3.1 Initialisation de la population (*)

Chaque solution potentielle est un automate quotient défini par une certaine partition π de l'ensemble des états $Q_{MCA(I_+)}$ de l'automate $MCA(I_+)$. Nous construisons une population initiale de partitions en considérant d'abord la partition triviale π_0 qui associe un bloc distinct à chaque élément de $Q_{MCA(I_+)}$. Ensuite, jusqu'à 50% de la population initiale est constituée de partitions qui dérivent directement de la partition π_0. Le reste de la population est fait de partitions tirées aléatoirement dans l'ensemble des partitions de $Q_{MCA(I_+)}$.

Dans la suite de ce paragraphe, nous détaillons la représentation codée, les opérateurs génétiques spécifiques et les définitions possibles de la fonction d'adaptation, ou d'évaluation, de l'algorithme GIG.

8.3.2 Représentation codée des solutions (*)

Chaque individu de la population correspond à une partition de l'ensemble des états du $MCA(I_+)$. Dans la suite de ce paragraphe, N désigne le nombre d'états de cet automate. Nous représentons une partition à l'aide d'*un codage par indices de bloc* [Jones et al.91].

Formellement, chaque partition π comportant $r(\pi)$ blocs, est codée sous la forme de N entiers $(i_1, \ldots, i_N)$ où $i_j \in \{1, \ldots, r(\pi)\}, 1 \leq j \leq N$. Le j–*ème* entier i_j est l'indice du bloc assigné à l'état j et chaque bloc de la partition π est représenté par au moins un entier i_j. A titre d'exemple, le chromosome $(2, 1, 2, 3, 2)$ peut représenter la partition $\{\{1, 3, 5\}, \{2\}, \{4\}\}$ de l'ensemble $\{1, 2, 3, 4, 5\}$.

Le codage par indices de bloc satisfait la propriété de fermeture de la représentation, c'est-à-dire que tout chromosome peut être interprété comme une partition valide. Cependant, cette représentation est fortement redondante puisque il existe $r(\pi)!$ permutations des indices de bloc correspondant à des représentations équivalentes. C'est la raison pour laquelle nous avons recours à un *codage canonique gauche-droite par indices de blocs* [Jones et al.91]. Formellement, les blocs sont numérotés de telle sorte qu'ils apparaissent dans le même ordre gauche droite que leur état de rang minimal. Par exemple, le chromosome $(2, 1, 2, 3, 2)$ sera renuméroté $(1, 2, 1, 3, 1)$.

8.3.3 Opérateurs génétiques (*)

Nous définissons les opérateurs génétiques dans l'espace des partitions de l'ensemble des états du $MCA(I_+)$. Notons que ces opérations sont effectivement réalisées sur la représentation codée associée, sous la contrainte que les chromosomes résultants respectent le codage canonique introduit au paragraphe 8.3.2.

Mutation structurelle (*)

La mutation structurelle utilise une sélection aléatoire d'un état q appartenant à un bloc B de la partition π choisie aléatoirement pour subir une mutation. Elle assigne l'état q à un bloc B' tiré aléatoirement. B' est un bloc de la partition π ou un nouveau bloc.

Considérons, à titre d'exemple, la partition $\pi = \{\{1, \mathbf{3}, 5\}, \{2\}, \{4\}\}$ et supposons que l'état **3**, appartenant au premier bloc, soit sélectionné aléatoirement. Supposons également que le bloc B' ait été aléatoirement choisi égal au second bloc de la partition. La partition π' obtenue après mutation est égale à $\{\{1, 5\}, \{2, \mathbf{3}\}, \{4\}\}$.

Dans l'exemple présenté ci-dessus, le nombre de blocs de la partition π' est identique à celui de la partition π. Cependant, nous pouvons obtenir de façon similaire une partition comportant un bloc en moins (si, par exemple l'état **2** est assigné au premier bloc) ou un bloc en plus (si, par exemple, l'état **1** est assigné à un nouveau bloc). En d'autres termes, la profondeur de l'automate quotient $MCA(I_+)/\pi'$ obtenu après mutation diffère d'une unité, au plus, de celle de l'automate quotient $MCA(I_+)/\pi$. Il s'agit d'une caractéristique propre à notre problème puisque les AGs, appliqués à un problème de partition d'ensembles, utilisent souvent des fonctions de récupération pour garantir que le nombre fixé de blocs reste constant pendant la recherche [Michalewicz92]. Ici, nous cherchons à trouver la partition de cardinalité minimale, c'est-à-dire l'automate dérivé le plus profond, sous le contrôle de l'échantillon négatif. Dans ce sens, il semble intéressant de favoriser une mutation qui diminue le nombre de blocs dans la partition résultante. C'est pourquoi, nous avons introduit un paramètre supplémentaire, appelé *taux de descente*, qui est la probabilité qu'une mutation réduise nécessairement le nombre de blocs de la partition originale.

Croisement structurel (*)

Le croisement structurel consiste en l'union d'un bloc B_1 sélectionné aléatoirement dans le premier parent π_1 avec un autre bloc B_2 sélectionné aléatoirement dans le second parent π_2. Les partitions résultantes π_1' et π_2' sont obtenues à partir des partitions originales et du bloc $\{B1 \cup B2\}$.

Considérons, par exemple, que les deux partitions $\pi_1 = \{\{\mathbf{1}, \mathbf{4}\}, \{2, 3, 5\}\}$ et $\pi_2 = \{\{\mathbf{1}, \mathbf{3}\}, \{2\}, \{4\}, \{5\}\}$ soient sélectionnées pour subir un croisement. Supposons également que le premier bloc de chacune d'elles soit sélectionnés aléatoirement. L'union des blocs sélectionnés est $\{\mathbf{1},\mathbf{4}\} \cup \{\mathbf{1},\mathbf{3}\} = \{\mathbf{1},\mathbf{3},\mathbf{4}\}$ et les deux partitions résultantes sont $\pi'_1 = \{\{\mathbf{1},\mathbf{3},\mathbf{4}\},\{2,5\}\}$ et $\pi'_2 = \{\{\mathbf{1},\mathbf{3},\mathbf{4}\},\{2\},\{5\}\}$.

Les automates quotient $MCA(I_+)/\pi'_1$ et $MCA(I_+)/\pi'_2$ obtenus après croisement ont au moins un bloc commun, c'est-à-dire un ensemble commun d'états fusionnés. De plus, la profondeur de chaque nouvel automate quotient $MCA(I_+)/\pi'_k$ est soit égale à celle de l'automate quotient original $MCA(I_+)/\pi_k$, soit augmentée d'une unité.

8.3.4 Fonctions d'évaluation (*)

La fonction f_{I_+,I_-} présentée au paragraphe 8.1 est la fonction d'adaptation que nous nous proposons d'optimiser par un AG. Nous rappelons sa définition ci-dessous. Si N désigne le nombre d'états de $MCA(I_+)$, toute partition comporte au plus N blocs.

$$f_{I_+,I_-} : P(I_+) \to \mathrm{IR}^+, f_{I_+,I_-}(\pi) = g(r_{I_+}(\pi), k_{I_-}(\pi)),$$

où g est une fonction de pénalité positive dépendant de deux arguments entiers ; $r_{I_+}(\pi)$, désigne le nombre de blocs de la partition π, et $k_{I_-}(\pi)$, désigne le nombre de chaînes de I_-, acceptées par l'automate quotient A/π. La première définition de la fonction g est la suivante

$$g_1(x,0) = x, \quad g_1(x,y) = +\infty, \quad y > 0 \;, x \geq 1.$$

Par définition de l'opération de sélection proportionnelle à l'adaptation (voir paragraphe 8.2), nous savons que la probabilité de sélectionner une partition pour être reproduite à la génération suivante est reliée à la valeur de f_{I_+,I_-}. Plus précisément, la probabilité de sélection d'une partition π est donnée par la formule suivante

$$P(\pi_{\text{sélec}} = \pi) = \frac{(1 + f_{I_+,I_-}(\pi))^{-1}}{\sum_{j=1}^{Pop} (1 + f_{I_+,I_-}(\pi_j))^{-1}},$$

où Pop désigne la taille de la population.

Etant donné la première définition de la fonction g, une partition π telle que l'automate quotient $MCA(I_+)/\pi$ accepte au moins un échantillon de I_- reçoit une probabilité nulle d'être reproduite. Nous avons proposé deux autres définitions de la fonction de pénalité :

$$g_2(x,0) = x, \quad g_2(x,y) = N + 1, \quad y > 0, x \geq 1.$$
$$g_3(x,y) = x + \tfrac{y}{|I_-|} \cdot (N + 1 - x), \quad y \geq 0, x \geq 1.$$

Conformément à la seconde définition de la fonction g, la valeur assignée à n'importe quelle partition pour laquelle l'automate quotient accepte au moins une chaîne de I_- est égale à $N+1$. Par conséquent, une telle partition reçoit une probabilité non nulle d'être reproduite. Cette probabilité est cependant nécessairement inférieure à celle d'une partition correspondant à un automate quotient compatible.

La troisième définition de la fonction g, nous permet de pénaliser les automates non compatibles proportionnellement au nombre d'exemples négatifs qu'ils acceptent. Notons que, conformément à cette troisième définition, il n'y a plus de garantie que la valeur optimale de la fonction $f_{I_+,I_-}(\pi)$ corresponde à une partition pour laquelle l'automate quotient est compatible.

Nous présentons au paragraphe 10.4.2 des résultats expérimentaux comparant les trois définitions possibles de la fonction d'évaluation.

8.3.5 L'algorithme GIG (*)

L'algorithme GIG peut-être présenté de façon synthétique comme suit :

Algorithme GIG

entrée I_+, I_- //Les échantillons positif et négatif
sortie $MCA(I_+)/\pi$ //Un automate quotient du $MCA(I_+)$
var p_c //Le taux de croisement
p_m //Le taux de mutation
p_d //Le taux de descente

Procédure *évaluer* $(P(t))$
début
pour $i = 1$ **jusqu'à** Pop
$\pi \leftarrow P_i(t)$
$A/\pi \leftarrow$*dériver* $(MCA(I_+), \pi)$
évaluer (A/π) //Evaluation de chaque automate quotient
fin pour
fin *évaluer* $(P(t))$

début GIG

// Pop désigne la taille de la population
// N désigne le nombre d'états de $MCA(I_+)$
// $P(t)$ désigne les partitions conservées à la génération t

$t \leftarrow 0$
$\pi_0 \leftarrow \{\{1\}, \ldots, \{N\}\}$
$P(0) \leftarrow$*initialiser* (π_0) //Initialisation de la population
évaluer $(P(0))$
$\pi_{best} \leftarrow$*meilleur* $(P(0))$
tant que $\neg$(*condition d'arrêt*) **faire**
$t \leftarrow t + 1$
$P(t) \leftarrow$*sélectionner* $(P(t-1))$
$P(t) \leftarrow$*croiser* $(P(t), p_c)$
$P(t) \leftarrow$*muter* $(P(t), p_m, p_d)$
évaluer $(P(t))$
fait
$\pi_{best} \leftarrow$*meilleur* $(P(t))$
retourner $MCA(I_+)/\pi_{best}$
fin GIG

Le $MCA(I_+)$ comporte $||I_+|| + 1$ états et $||I_+||$ transitions. L'évaluation de chaque automate quotient distinct requiert une analyse, en général non-déterministe, de I_- avec une complexité de $\mathcal{O}(||I_+||.||I_-||)$. Un critère d'arrêt classique consiste à fixer un nombre maximal e_{max} d'évaluations de solutions distinctes. La complexité globale de l'algorithme GIG est alors $\mathcal{O}(||I_+||.||I_-||.e_{max})$.

8.4 Comparaisons avec d'autres méthodes

L'algorithme GIG est un algorithme heuristique visant à identifier un langage dans la classe entière des langages réguliers. A ce titre, il est à rapprocher de l'algorithme RIG ou de son approximation BRIG. Nous donnerons au paragraphe 10.4.1, des résultats expérimentaux comparatifs. D'un point de vue général, l'algorithme BRIG a pour objet de limiter la croissance exponentielle de l'espace de recherche en élaguant aléatoirement cette espace et en ne poursuivant qu'une fraction des solutions potentielles, c'est-à-dire des suites d'automates compatibles à profondeur croissante dans le treillis des partitions. Par construction de l'algorithme GIG, il n'y a aucune garantie que la recherche s'effectue par profondeur croissante. Par ailleurs, sa complexité calculatoire dépend du nombre maximum, fixé a priori, de solutions envisagées, nombre jusqu'auquel la recherche génétique est poursuivie.

L'algorithme RPNI est caractérisable et, de plus, sa complexité dépend polynomialement de la taille du problème. Il est donc particulièrement intéressant. Remarquons qu'il n'y a, en pratique, pas d'assurance d'identification exacte car pour un échantillon positif et négatif aléatoires quelconques, il n'y a aucune garantie que cet échantillon soit caractéristique. Cependant, même en l'absence d'un échantillon caractéristique, l'algorithme RPNI trouve une solution déterministe dans l'ensemble frontière. Afin d'évaluer, la qualité de cette solution, nous avons également mené des expériences comparatives dont les résultats sont exposés aux paragraphes 10.4.1 et 10.5.2.

Un travail très préliminaire sur l'utilisation d'un AG pour l'inférence de grammaires hors-contexte a été proposé récemment [Wyard94]. Cependant, ce travail manque de toute formulation théorique du problème et, en particulier, l'espace de recherche n'y est nullement caractérisé. Cette méthode utilise un grand nombre d'informations a priori (nombre de règles, longueur maximale des règles, ensembles des terminaux et non-terminaux de la grammaire à identifier supposés connus). De plus, les résultats expérimentaux présentés ne reposent sur aucun protocole d'évaluation systématique, tel que celui que nous décrivons au paragraphe 10.2.

Chapitre 9

Inférence caractérisable et incrémentale (*)

L'algorithme RPNI, présenté au paragraphe 7.2.2, offre un certain nombre de propriétés intéressantes. Premièrement, il est caractérisable (voir chapitre 7) dans la mesure où il a été démontré que cet algorithme identifie à la limite n'importe quel langage régulier [Oncina et al.92]. De plus, sa complexité est polynomiale en fonction de la taille des échantillons positif et négatif. Néanmoins, cet algorithme doit recevoir en entrée toute l'information positive et négative disponible, c'est-à-dire qu'il utilise une *présentation à données fixées* (voir paragraphe 9.1.1). Si de nouvelles données d'apprentissage sont présentées, la recherche du nouvel automate solution doit être recommencée sans pouvoir tirer parti de la solution trouvée antérieurement. Nous nous proposons de lever cette limitation et de définir une version incrémentale de l'algorithme RPNI qui soit, elle aussi, caractérisable. Les données d'apprentissage seront supposées être reçues séquentiellement c'est-à-dire une à une, dans un ordre quelconque, et chacune étiquetée comme exemple positif ou négatif du langage à identifier.

Nous allons détailler au paragraphe 9.1 la notion d'échantillon caractéristique d'un langage régulier et étudier un certain nombre de propriétés importantes d'un tel échantillon. Poursuivant l'analyse de l'algorithme RPNI au paragraphe 9.2, nous en proposons une nouvelle variante avec *suppression des finales*. Cette variante garantit également l'identification à la limite mais peut offrir une moins bonne généralisation des données lorsque celles-ci ne contiennent pas d'échantillon caractéristique. Elle permettra néanmoins une analyse de la complexité théorique qui correspond mieux aux observations expérimentales (voir paragraphe 9.2.2). Ensuite, nous poursuivons au paragraphe 9.3 la caractérisation de l'espace de recherche présentée aux paragraphes 6.5 et 6.6. Nous nous intéressons ici à la structure de l'espace de recherche lorsque les données d'apprentissage sont présentées séquentiellement. Enfin, nous exposons au paragraphe 9.4 une version incrémentale de l'algorithme RPNI.

9.1 Caractérisation des données d'apprentissage (*)

9.1.1 Identification à la limite pour une présentation à données fixées (*)

Nous avons vu au paragraphe 6.3.3 que, pour une présentation séquentielle des données, l'identification à la limite est garantie lorsqu'une méthode d'inférence converge avec certitude vers une représentation correcte du langage à identifier. Bien que nous nous intéressions particulièrement aux présentations séquentielles aux paragraphes 9.3 et 9.4, il est utile d'étudier l'identification dans le cadre d'une présentation à *données fixées*. Dans ce cas, l'ensemble d'apprentissage D est fixe et constitué d'une paire d'échantillons positif et négatif, $D = (I_+, I_-)$.

Gold a défini l'identification d'un langage pour une présentation à données fixées [Gold78]. Nous nous inspirons de sa définition pour introduire la notion d'échantillon caractéristique $D^c = (I^c_+, I^c_-)$.

Définition 9.1 Une méthode d'inférence M *identifie à la limite* un langage L *pour une présentation à données fixées* s'il existe un échantillon $D^c = (I^c_+, I^c_-)$ tel que pour tout ensemble de données $D = (I_+, I_-)$, avec $I_+ \supseteq I^c_+$ et $I_- \supseteq I^c_-$, l'hypothèse $H(D)$ fournie par la méthode M est correcte, c'est-à-dire $L(H(D)) = L$. L'échantillon D^c est appelé *caractéristique* pour le langage L et la méthode d'inférence M.

Rappelons que l'hypothèse fournie par une méthode d'inférence est un élément dans l'espace des hypothèses, dans notre cas celui des automates finis.

Cette définition est une caractérisation générale d'une méthode d'inférence. Elle ne précise pas, pour une méthode M et un langage L, comment spécifier D^c. Comme l'automate canonique $A(L)$ d'un langage régulier est unique, la spécification d'un échantillon caractéristique se fait plus naturellement sur base de cet automate. Néanmoins, une méthode d'inférence qui aurait pour objet l'identification d'un langage régulier représenté par un automate non déterministe pourrait requérir une autre spécification.

Nous avons utilisé le terme d'*identification à la limite* dans la définition 9.1 pour insister sur sa similitude avec la définition 6.26 reposant sur une présentation séquentielle.

Propriété 9.1 *Une méthode M identifie un langage L pour une présentation à données fixées si et seulement si elle identifie le langage L pour une présentation séquentielle.*

Preuve :
Si une méthode identifie un langage à partir d'une présentation à données fixées, elle identifiera nécessairement ce langage sur base d'une présentation séquentielle de ces données. Il suffit qu'elle mémorise les données reçues une à une et propose, à chaque étape de la présentation séquentielle, une hypothèse sur base de l'ensemble des données mémorisées. Réciproquement, si une méthode identifie un langage sur base d'une présentation séquentielle, cela veut dire qu'il existe un point de convergence à partir duquel, pour toute nouvelle donnée reçue, l'hypothèse proposée restera identique et correcte. Par la définition 9.1, l'ensemble des données reçues jusqu'au point de convergence inclut donc un échantillon caractéristique. □

9.1.2 Spécification d'un échantillon caractéristique

Partant d'un langage L, représenté par son automate canonique $A(L)$, nous allons définir les conditions que doivent vérifier les données d'apprentissage afin d'être caractéristique relativement à L pour l'algorithme RPNI. A cette fin, Oncina et García ont défini les notions de *préfixes courts* et *noyau* [Oncina et al.92]. Nous y associons les notions d'*états fixes*, *variables* et *auxiliaires.* Dans la suite, le symbole "$<$ " désigne l'ordre standard sur les chaînes d'un alphabet Σ.

L'ensemble des préfixes courts et le noyau

Définition 9.2 L'ensemble des *préfixes courts* $Sp(L)$ d'un langage L est défini par

$$Sp(L) = \{x \in \Pr(L) \ \ | \nexists u \in \Sigma^* \ avec \ L/u = L/x \ et \ u < x\}$$

Nous savons que dans l'automate canonique $A(L)$ du langage L, il y a autant d'états que d'ensembles de finales L/x des chaînes x appartenant aux préfixes de L (voir définition 6.12). Par conséquent, l'ensemble des préfixes courts est l'ensemble des premières chaînes dans l'ordre standard qui chacune mène à un état de l'automate canonique. Il y a donc également autant de préfixes courts que d'états dans $A(L)$.

A titre d'exemple, l'automate canonique du langage défini sur l'alphabet $\{a, b\}$ et acceptant un nombre pair de b est représenté à la figure 9.1. L'ensemble des préfixes courts de ce langage est $Sp(L) = \{\lambda, b\}$. En effet, λ et b sont les deux premières chaînes dans l'ordre standard permettant de mener respectivement à l'état 0 et à l'état 1.

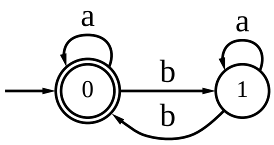


FIG. 9.1: Automate acceptant un nombre pair de b sur l'alphabet $\{a, b\}$.

Définition 9.3 Le *noyau* $N(L)$ d'un langage L est défini par

$$N(L) = \{\lambda\} \cup \{xa \mid x \in Sp(L), a \in \Sigma, xa \in \Pr(L)\}$$

Le noyau est constitué, outre de la chaîne λ, des préfixes du langage correspondant à des préfixes courts allongés d'une lettre.

Par construction $Sp(L) \subseteq N(L)$, c'est-à-dire que tous les éléments des préfixes courts appartiennent au noyau. En effet, on peut obtenir chaque préfixe court en allongeant d'une lettre un autre préfixe court du langage.

Si $|Q|$ désigne le nombre d'états de l'automate canonique $A(L)$, le noyau comporte au plus $1 + |Q| \cdot |\Sigma|$ éléments. Les éléments du noyau identifient les transitions de l'automate canonique $A(L)$ puisqu'ils sont obtenus en ajoutant une lettre aux préfixes courts identifiant les états de $A(L)$. De plus, tous les éléments du noyau qui appartiennent également au langage identifient les états d'acceptation dans $A(L)$.

Le noyau du langage de notre exemple est $N(L) = \{\lambda, a, b, ba, bb\}$. Les trois chaînes λ, a et bb appartiennent à L ; elles mènent à l'état d'acceptation de l'automate de la figure 9.1.

Définition 9.4 Un échantillon $D^c = (I_+^c, I_-^c)$ est *caractéristique* relativement à un langage L et pour l'algorithme RPNI s'il vérifie les conditions suivantes

1. *$\forall x \in N(L)$,**si** $x \in L$ **alors** $x \in I_+^c$ **sinon** $\exists u \in \Sigma^*$ tel que $xu \in I_+^c$.*
2. *$\forall x \in Sp(L), \forall y \in N(L)$ **si** $L/x \neq L/y$ **alors** $\exists u \in \Sigma^*$ tel que ($xu \in I_+^c$* et *$yu \in I_-^c$) ou ($xu \in I_-^c$* et *$yu \in I_+^c$).*

La condition 1 assure que tous les éléments du noyau soient présents comme chaîne de I_+^c lorsqu'ils appartiennent au langage ou sinon comme préfixe d'une chaîne de I_+^c. Les éléments du noyau représentant les transitions et les états d'acceptation de l'automate canonique, on vérifie aisément que cette condition assure la complétude structurelle de l'échantillon I_+^c relativement à $A(L)$ (voir définition 6.16). Dans ce cas, nous savons par le théorème 6.2 que l'automate $A(L)$ appartient au treillis $Lat(PTA(I_+^c))$. Lorsqu'un élément x des préfixes courts et y du noyau n'ont pas même ensemble de finales ($L/x \neq L/y$), cela signifie qu'il ne correspondent pas au même état dans l'automate canonique. Dans ce cas, la condition 2 assure qu'un suffixe u les distingue[1]. En d'autres termes, la fusion d'un état du $PTA(I_+^c)$ correspondant à un préfixe court x avec un autre état correspondant à un élément y du noyau (incluant les préfixes courts) est rendue incompatible par l'existence de yu dans I_-^c avec xu dans I_+^c ou l'inverse.

Notons que pour un langage L fixé, il peut exister plusieurs échantillons caractéristiques distincts[2]. En effet, plusieurs suffixes u peuvent satisfaire la première condition ou la seconde. Comme l'ensemble des préfixes courts comporte $|Q|$ éléments et le noyau $\mathcal{O}(|Q| \cdot |\Sigma|)$ éléments, la taille d'un échantillon caractéristique est donnée par

$$|I_+^c| = \mathcal{O}(|Q|^2 \cdot |\Sigma|) \text{ et } |I_-^c| = \mathcal{O}(|Q|^2 \cdot |\Sigma|).$$

Elle ne dépend donc, pour un alphabet Σ fixé, que du nombre d'états de l'automate canonique du langage à identifier.

Un exemple d'échantillon caractéristique

Pour le langage accepté par l'automate $A(L)$ de la figure 9.1, nous avons $Sp(L) = \{\lambda, b\}$ et $N(L) = \{\lambda, a, b, ba, bb\}$. Un échantillon caractéristique peut être construit de la manière suivante. Les chaînes λ, a et bb sont des éléments de $N(L) \cap L$. Elles appartiendront donc nécessairement à I_+^c. Afin de satisfaire complètement la condition 1, il suffit de trouver des suffixes de b et de ba pour obtenir des chaînes de L. Par exemple, la chaîne bb, déjà présente dans I_+^c, contient un suffixe de b et, par exemple, la chaîne $baaaba$ contient un suffixe de ba. La condition 1 est donc satisfaite avec $I_+^c = \{\lambda, a, bb, baaaba\}$. Cet échantillon est structurellement complet relativement à l'automate $A(L)$.

Afin de compléter l'échantillon caractéristique, nous parcourons les paires constituées d'un élément x de $Sp(L)$ et d'un élément y de $N(L)$. Lorsque $x =$

[1] Puisque, par hypothèse $L/x \neq L/y$, l'existence d'au moins un suffixe u satisfaisant la condition 2 est garantie.

[2] Il peut même exister une infinité d'échantillons caractéristiques différents si le langage est de cardinalité infinie.

(x,y)	u	I_+^c	I_-^c
(λ, a)	-	-	-
(λ, b)	b	bb	b
(λ, ba)	ab	$baab$	ab
(λ, bb)	-	-	-
(b, λ)	b	bb	b
(b, a)	ba	bba	aba
(b, ba)	-	-	-
(b, bb)	λ	bb	b

TAB. 9.1: Construction d'un échantillon caractéristique.

y, les ensembles de finales sont nécessairement égaux. Nous ne considérons donc que les paires d'éléments distincts. Pour chaque paire (x, y), si x et y ont des ensembles de finales différents, c'est-à-dire s'ils mènent à des états distincts dans l'automate canonique, nous trouvons un suffixe u satisfaisant la condition 2. Nous ajoutons les éléments xu et yu dans les échantillons positif ou négatif selon leur appartenance au langage. Ces opérations sont résumées à la table 9.1. Notons que u n'est pas défini lorsque x et y ont un même ensemble de finales.
En unifiant les conditions 1 et 2, un échantillon caractéristique pour le langage de notre exemple est donné par

$$I_+^c = \{\lambda, a, bb, bba, baab, baaaba\} \text{ et } I_-^c = \{b, ab, aba\}.$$

Etats fixes, variables et auxiliaires (*)

Nous présentons à la figure 9.2 l'arbre accepteur des préfixes de l'échantillon positif. Nous appelons *états fixes*, les états correspondant aux préfixes courts dans l'arbre des préfixes. Dans notre exemple, les états 0 et 2 sont fixes. Nous appelons *états variables* les états correspondant aux éléments de $N(L) \setminus Sp(L)$ dans l'arbre des préfixes. Dans notre exemple, les états 1, 3 et 4 sont variables. Les autres états de l'arbre des préfixes sont appelés *états auxiliaires*.

Le parcours depuis la racine jusqu'à un état d'acceptation du $PTA(I_+^c)$ est nécessairement composé d'un ou plusieurs états fixes, suivi éventuellement d'un état variable et suivi éventuellement d'un ou plusieurs états auxiliaires. En outre, le long de toute branche du $PTA(I_+^c)$ les préfixes sont ordonnés par l'ordre standard. En d'autres mots, le long de toute branche comportant au moins un état fixe q_{fixe}, un état variable q_{var} et au moins un état auxiliaire q_{aux}, nous avons $q_{fixe} < q_{var} < q_{aux}$. Dans notre exemple, l'acceptation de la chaîne *bba* passe par les états 0, 2, 4 et 6 ; nous vérifions l'ordre 0 (fixe) < 2 (fixe) < 4 (variable) < 6 (auxiliaire).

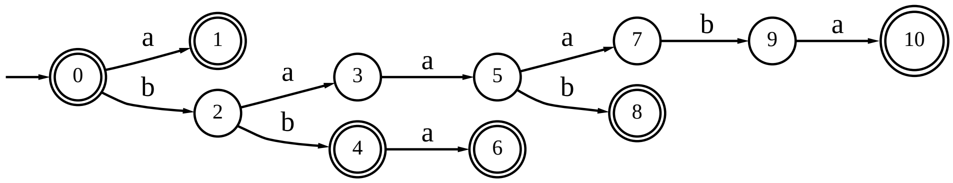


FIG. 9.2: Arbre accepteur des préfixes de $\{\lambda, a, bb, bba, baab, baaaba\}$.

Nous avons vu qu'un échantillon caractéristique pour un langage L n'est pas nécessairement unique. Cependant, tout échantillon caractéristique $D^c = (I_+^c, I_-^c)$ est tel que, par la condition 1, les **mêmes** états fixes et variables soient présents dans le $PTA(I_+^c)$ puisque ces états constituent les éléments de $N(L)$. De plus, puisque les états fixes correspondent aux préfixes courts, c'est-à-dire aux états de $A(L)$, ils ont nécessairement des ensembles de finales distincts. Tout échantillon caractéristique est, par la condition 2, tel que la fusion de deux états fixes est rendue incompatible par une chaîne de I_-^c. Finalement, comme il y a autant d'ensembles de finales dans le langage L que d'états fixes, cela implique que chaque état variable partage nécessairement l'ensemble de finales d'un état fixe. La condition 2 assure également que tout fusion d'un état variable avec un état fixe ayant un ensemble de finales différent est rendue incompatible par une chaîne de I_-^c. En d'autres termes, la fusion d'un état variable avec tout état fixe est niée par un contre-exemple sauf pour le "bon" état fixe, c'est-à-dire celui ayant même ensemble de finales.

Toutes les propriétés évoquées au paragraphe précédent restent valables si les données d'apprentissage $D = (I_+, I_-)$ incluent strictement un échantillon caractéristique, c'est-à-dire $I_+ \supset I_+^c$ et $I_- \supset I_-^c$ puisque l'ajout de nouveaux exemples correspond à l'ajout d'états auxiliaires, sans modifier les contraintes sur les fusions possibles entre états fixes et variables. De même, de nouveaux contre-exemples peuvent définir comme incompatibles des fusions entre états fixes et variables mais ces fusions étaient déjà nécessairement niées par des éléments de I_-^c.

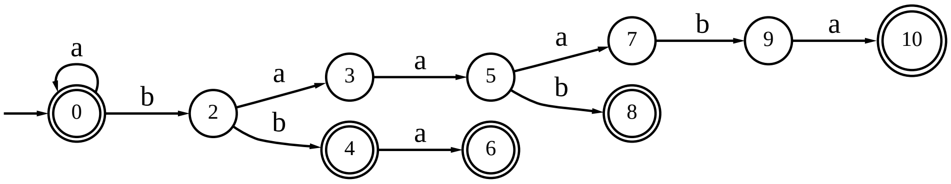
Etats auxiliaires, les "responsables" du non-déterminisme (*)

Les états d'un automate canonique sont représentés par les états fixes de l'arbre des préfixes et ses transitions et états d'acceptation par les éléments du noyau incluant états fixes et variables. L'automate canonique étant déterministe, le non-déterminisme d'un automate quotient du $PTA(I_+^c)$ est nécessairement introduit par des transitions pointant vers un état auxiliaire. Nous allons illustrer cette propriété par un exemple.

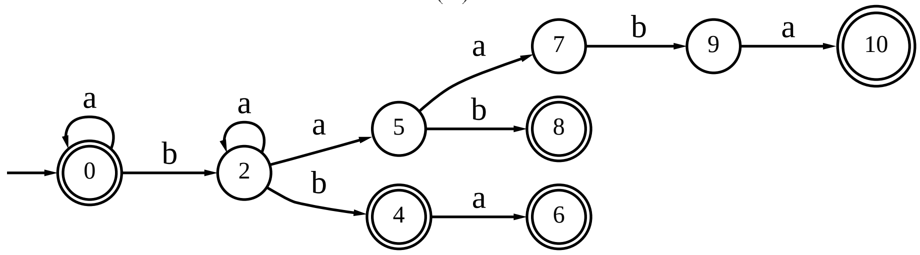
Reprenant l'arbre des préfixes de la figure 9.2, nous avons défini un automate quotient par la partition

$$\pi = \{\{0,1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\},\{10\}\}.$$

Cet automate est représenté[3] à la figure 9.3 (a). Comme nous le verrons, il peut être obtenu au cours de l'application de l'algorithme RPNI avec l'échantillon caractéristique défini précédemment. Rappelons que les états 0 et 2 sont fixes, les états 1, 3 et 4 variables et les autres états auxiliaires. L'automate de la figure 9.3 (a) est déterministe. Si l'on fusionne l'état variable 3 et l'état fixe 2, on obtient l'automate représenté à la figure 9.3 (b). Si l'on n'y considère que les états fixes 0 et 2 et les transitions pointant vers ces états, le sous-automate obtenu est déterministe. Par contre, dans l'automate complet, la transition pointant vers l'état auxiliaire 5 introduit un non-déterminisme sur la lettre a. Puisque l'on cherche à construire un automate déterministe, il faut "replier" cette transition sur celle qui est déjà issue de l'état 2. Cela consiste à fusionner les états 2 et 5. Cette opération donne lieu à la *fusion pour déterminisation*, une des procédures de l'algorithme RPNI. Notons que la fusion de l'état 5 avec un autre état que l'état 2 ne permettrait pas de réduire le non-déterminisme. Par contre, cette fusion introduit un nouveau non-déterminisme que l'on peut supprimer en appliquant de nouveau la même opération.



(a)



(b)

FIG. 9.3: Automates quotient déterministe (a) et non-déterministe (b).

[3]Chaque bloc est représenté par son état de rang minimal.

Nous définissons formellement la fusion pour déterminisation comme suit. Chaque bloc B_i de la partition π a pour rang celui de son état de rang minimal dans l'ordre standard.

> **Si** B_i et B_j sont deux blocs de la partition π, avec $B_i < B_j$, tels qu'il existe un bloc $B < B_j$ et $a \in \Sigma$ avec $B_i \in \delta(B, a)$ et $B_j \in \delta(B, a)$ **alors** la nouvelle partition π' est définie par
>
> $$\pi' = \{B_i \cup B_j\} \cup (\pi \setminus \{B_i, B_j\})$$

En d'autres termes, l'automate quotient relativement à la partition π' est obtenu en fusionnant les blocs B_i et B_j. La fusion pour déterminisation peut entraîner un nouveau non-déterminisme introduit par un autre état. Ce nouvel état est également auxiliaire puisque nous avons, le long d'une branche du $PTA(I_+^c)$, la relation $q_{fixe} < q_{var} < q_{aux}$. Par conséquent, la fusion par déterminisation est appliquée à nouveau jusqu'à l'obtention d'un automate déterministe.

La figure 9.4 illustre la séquence d'automates obtenus lors de la fusion pour déterminisation de l'automate de la figure 9.3 (b).

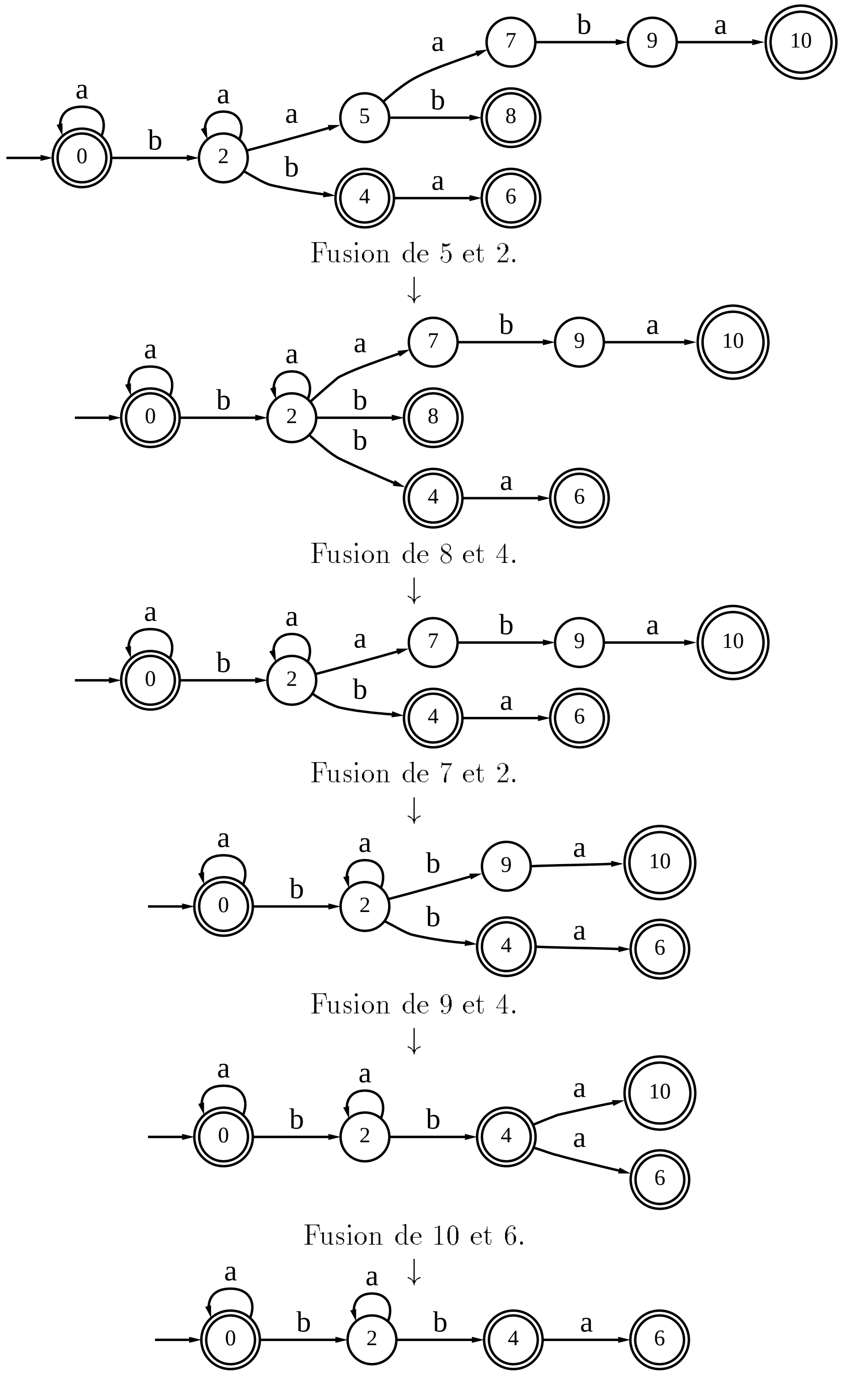


FIG. 9.4: Fusion pour déterminisation.

9.1.3 Identification à la limite de l'algorithme RPNI (*)

Nous reprenons la présentation de l'algorithme RPNI du paragraphe 7.2.2 pour en donner une preuve de l'identification à la limite[4].

Algorithme RPNI

entrée I_+, I_-
sortie Un AFD compatible avec I_+, I_-

début

// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \dots, \{N-1\}\}$ // Un bloc par préfixe dans l'ordre <
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    $\pi'' \leftarrow$ *fusion_déterm* (A/π')
    si *compatible* $(A/\pi'', I_-)$ **alors** // Analyse déterministe de I_-
      $A \leftarrow A/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI

Théorème 9.1 *L'algorithme RPNI identifie à la limite la classe des langages réguliers.*

Preuve :
Les états de l'arbre du $PTA(I_+)$ sont parcourus dans l'ordre standard (boucle sur l'indice i). Pour chaque état, on teste la fusion avec les états de rang inférieur (boucle sur l'indice j). Remarquons que si l'automate A/π' est incompatible avec I_-, c'est-à-dire $L(A/\pi') \cap I_- \neq \phi$, alors l'automate A/π'' est également incompatible avec I_- puisque la fusion pour déterminisation

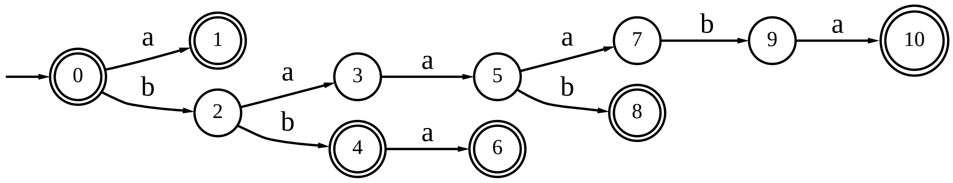
[4] L'identification à la limite de l'algorithme RPNI a été démontrée dans [Oncina et al.92]. Nous en donnons une démonstration reformulée qui, en particulier, inclut explicitement la fusion pour déterminisation.

implique $L(A/\pi') \subseteq L(A/\pi'')$. La fusion pour déterminisation permet d'effectuer une analyse déterministe afin de vérifier cette compatibilité.

Nous supposons que les données d'apprentissage contiennent un échantillon caractéristique pour le langage régulier L. Par conséquent, si les états i et j sont des états fixes, leur fusion sera nécessairement niée par un contre-exemple et l'automate A/π'' sera incompatible avec I_-. Cela conduit à incrémenter j et à tester une autre fusion. La première fusion, dans l'ordre d'exploration, qui donnera lieu à un automate A/π' compatible correspond nécessairement à un état j fixe et un état i variable, avec $j < i$. Tous les états du sous-arbre dont l'état i est la racine sont alors des états auxiliaires. Ils introduisent un non-déterminisme qui sera éliminé par la fusion pour déterminisation. Nous avons vu que cette opération consiste à replier les transitions pointant vers les états auxiliaires sur des transitions pointant vers des états fixes. Dans ce cas, si l'automate A/π' est compatible avec I_-, l'automate A/π'' l'est également.

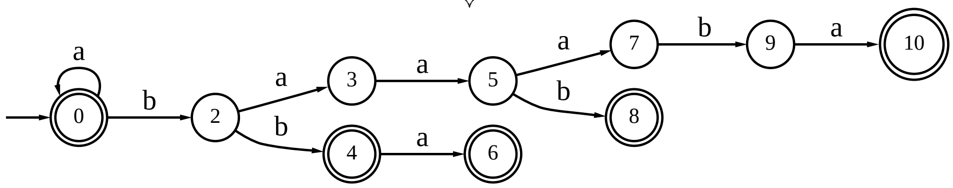
En résumé, si les données d'apprentissage contiennent un échantillon caractéristique, aucune fusion entre états fixes ne donne lieu à un automate compatible avec I_-. De plus, seule la fusion de chaque état variable avec l'état fixe partageant le même ensemble de finales, donne lieu à un automate compatible. Enfin, les états auxiliaires sont fusionnés nécessairement après la fusion d'un état variable sur un état fixe. Comme la seule manière de réduire le non-déterminisme est de replier les transitions pointant vers les états auxiliaires sur des transitions pointant déjà vers des états fixes, cette opération ne modifie pas, dans ce cas particulier, la compatibilité avec I_-. Finalement, comme tous les états fixes et variables sont nécessairement présents, les ensembles d'états (éventuellement d'acceptation) et transitions de $A(L)$ sont donc obtenus et l'identification du langage L est garantie. $\Box$

La figure 9.5 présente un exemple d'exécution complète de l'algorithme RPNI avec un échantillon caractéristique ; $I_+^c = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_-^c = \{b, ab, aba\}$. Chaque état y est identifié par l'état de rang minimal du bloc qu'il représente. Le symbole "$\downarrow$" relie les automates obtenus par fusion pour déterminisation. Le symbole "$\Downarrow$" relie les automates dérivés par une autre fusion. Lorsque l'automate obtenu est incompatible, une autre fusion est effectuée dans l'ordre défini.

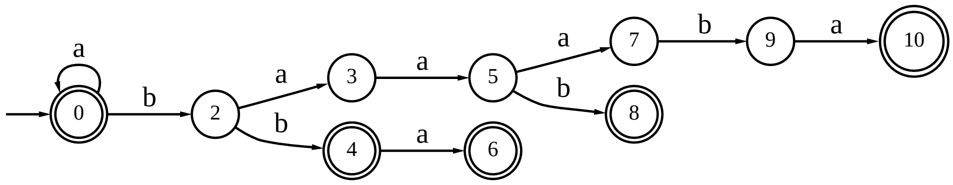


Fusion de 1 et 0.

⇓

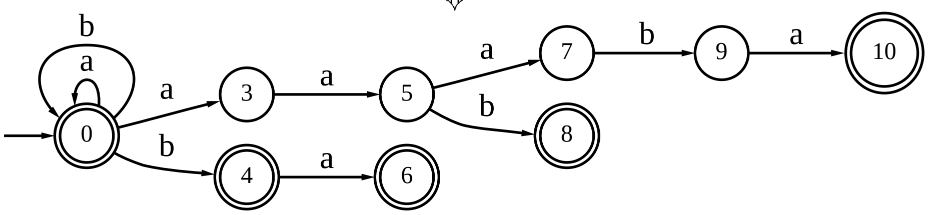


Compatibilité avec I_-



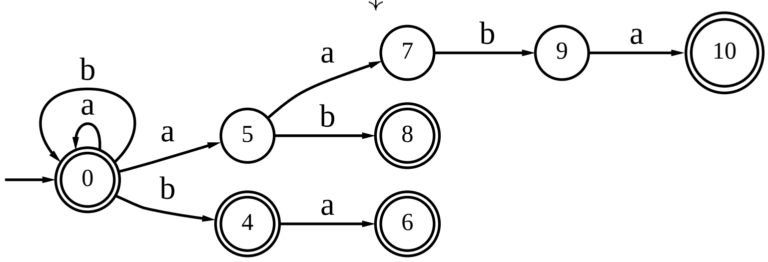
Fusion de 2 et 0.

⇓



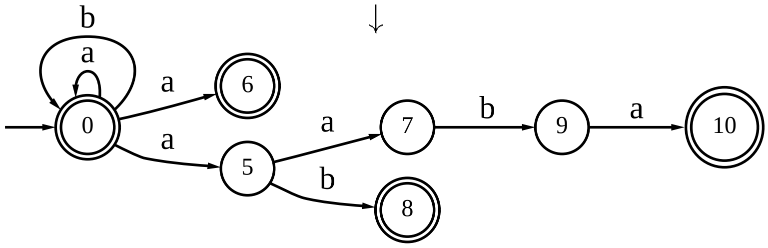
Fusion de 3 et 0.

↓



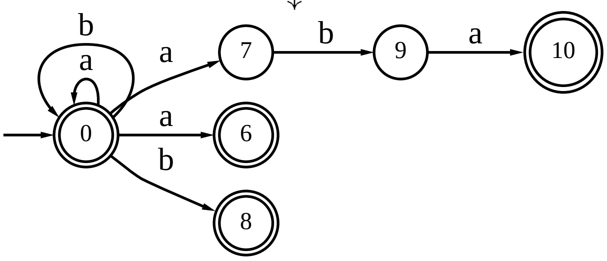
Fusion de 4 et 0.

↓



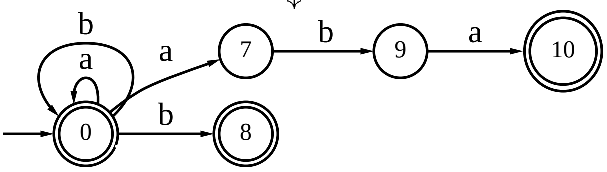
Fusion de 5 et 0.

$\downarrow$



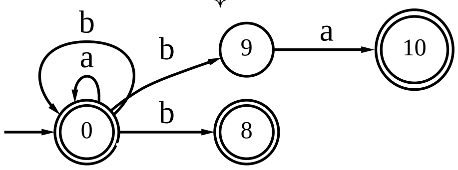
Fusion de 6 et 0.

$\downarrow$



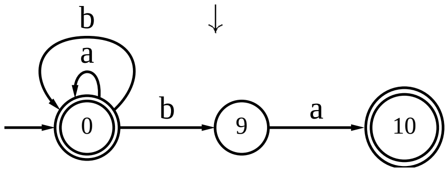
Fusion de 7 et 0.

$\downarrow$



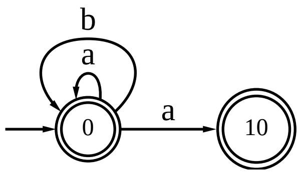
Fusion de 8 et 0.

$\downarrow$



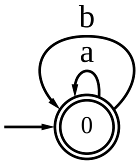
Fusion de 9 et 0.

$\downarrow$

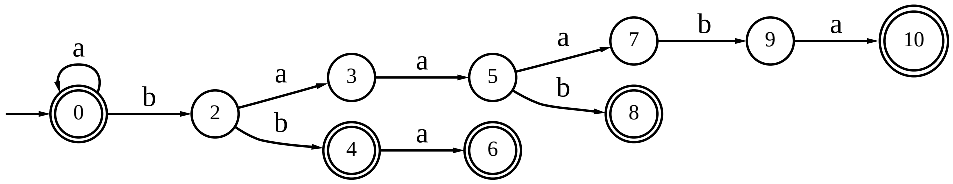


Fusion de 10 et 0.

$\downarrow$

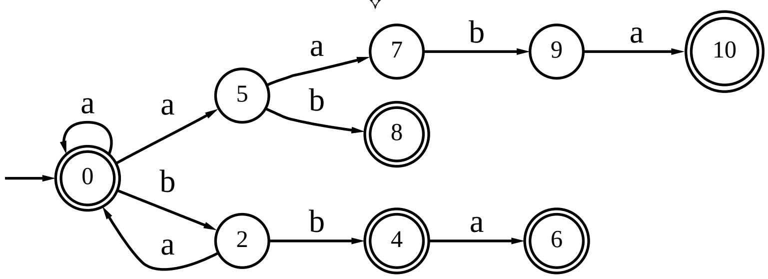


Incompatibilité avec I_-



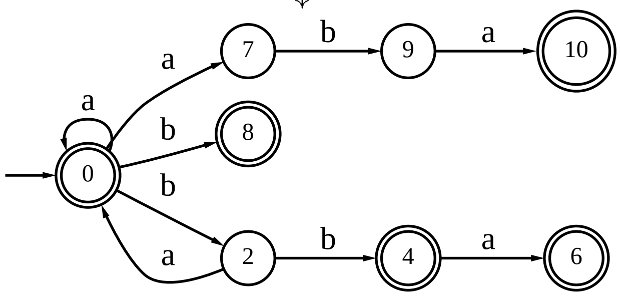
Fusion de 3 et 0.

$\Downarrow$



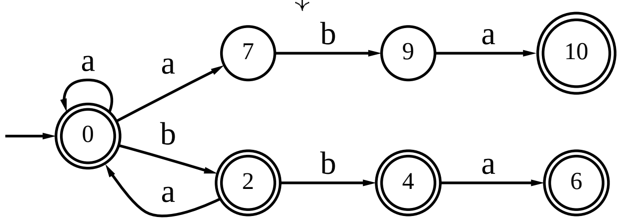
Fusion de 5 et 0.

$\downarrow$



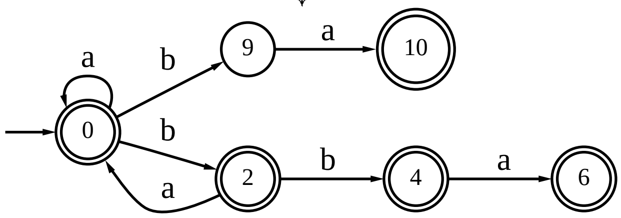
Fusion de 8 et 2.

$\downarrow$



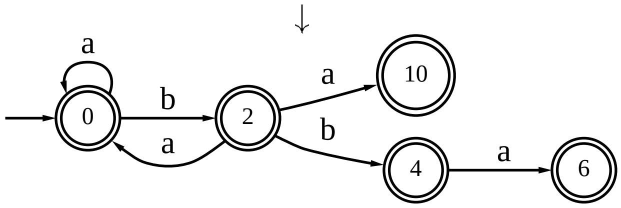
Fusion de 7 et 0.

$\downarrow$



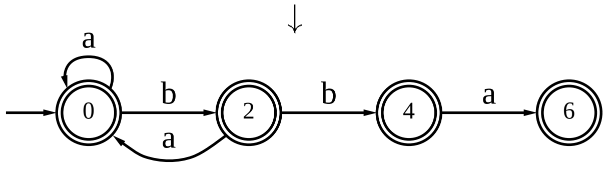
Fusion de 9 et 2.

$\downarrow$

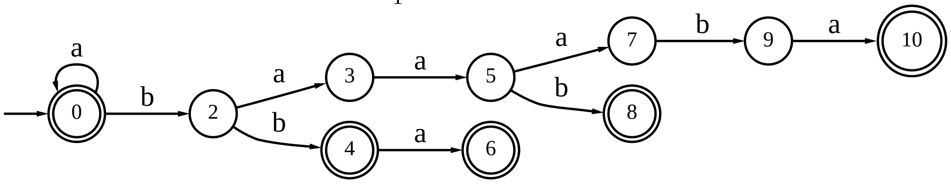


Fusion de 10 et 0.

↓

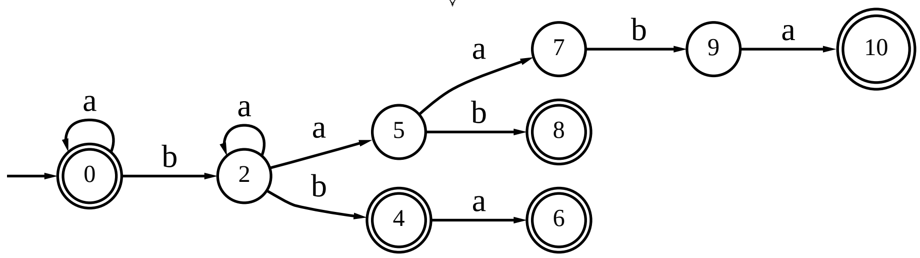


Incompatibilité avec I_-



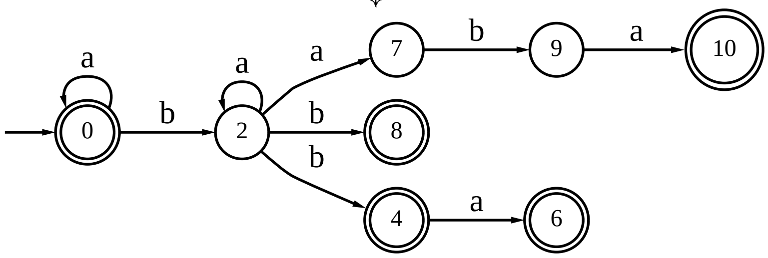
Fusion de 3 et 2.

⇓



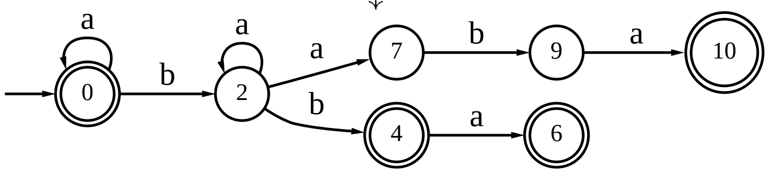
Fusion de 5 et 2.

↓



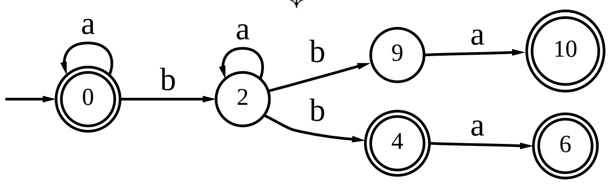
Fusion de 8 et 4.

↓



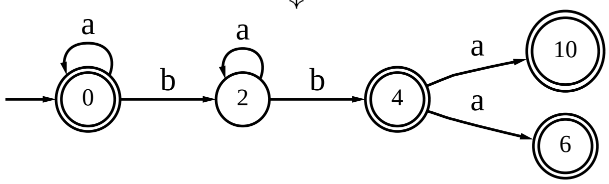
Fusion de 7 et 2.

↓



Fusion de 9 et 4.

↓



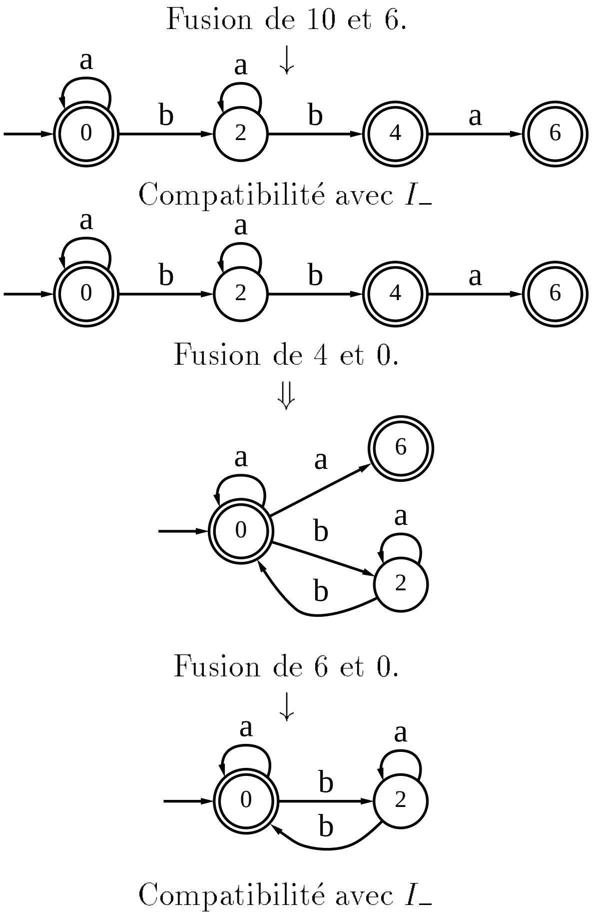


FIG. 9.5: Exemple d'exécution de l'algorithme RPNI.

9.1.4 Un élément remarquable de l'ensemble frontière

Théorème 9.2 *Lorsque les données d'apprentissage contiennent un échantillon caractéristique pour le langage L, l'automate $A(L)$ est l'automate déterministe comportant le moins d'états et appartenant à $BS_{PTA}(I_+, I_-)$. Par conséquent, $A(L)$ est la solution au problème du plus petit AFD compatible.*

Preuve[5] *:*
Comme les données contiennent un échantillon caractéristique, tous les états fixes et variables sont présents dans le $PTA(I_+)$. Aucun automate comportant moins d'états que le nombre d'états fixes, c'est-à-dire le nombre d'états

[5] La preuve de ce théorème est, semble-t-il, incluse dans le document de thèse de José Oncina. Il a eu l'amabilité de m'en tracer les grandes lignes, un jour, entre la poire et le fromage... Qu'il en soit vivement remercié.

de $A(L)$, ne peut être compatible avec I_- car cela nécessiterait la fusion d'au moins deux états fixes. De plus, tout automate A déterministe, comportant le même nombre d'états que $A(L)$ et compatible, est nécessairement isomorphe à $A(L)$. En effet, pour assurer la compatibilité de A, il faut que chaque état variable soit fusionné avec l'état fixe partageant le même ensemble de finales. Si tous les états variables sont fusionnés avec leur état fixe respectif, la seule manière d'obtenir un automate déterministe est de replier les états auxiliaires conformément à la fusion pour déterminisation.□

Le lecteur aura sans doute remarqué la similitude entre les preuves des théorèmes 9.1 et 9.2. Nous avons distingué ces deux résultats pour insister sur le fait que le second complète la caractérisation de l'espace de recherche, indépendamment de l'algorithme d'inférence utilisé. En ce sens, la définition d'un échantillon caractéristique devient plus générale. En résumé, le théorème 9.2 garantit l'existence d'un élément remarquable de l'ensemble frontière alors que le théorème 9.1 garantit que l'on trouve cet élément par application de l'algorithme RPNI.

Bien que l'algorithme RPNI soit de complexité polynomiale, le théorème 9.2 ne contredit pas le caractère NP-difficile du problème du plus petit AFD compatible. En effet, ce résultat négatif suppose des données quelconques alors que les données qui assurent l'identification contiennent un échantillon caractéristique.

Le théorème 9.2 est également cohérent avec le résultat de Gold qui, dans le cas des langages réguliers, consiste à obtenir l'identification en énumérant des automates déterministes de taille croissante à partir d'un automate à un seul état [Gold78]. En appliquant une stratégie conservatrice, qui consiste à ne poursuivre l'énumération que si le dernier automate construit n'est pas compatible, l'énumération s'arrêtera nécessairement sur le plus petit AFD compatible. Cet algorithme d'énumération a évidemment une complexité non-polynomiale.

Finalement, le théorème 9.2 est une justification théorique des algorithmes heuristiques BRIG et GIG puisqu'il démontre l'utilité de rechercher un élément compatible à une profondeur maximale dans le treillis $Lat(PTA(I_+))$. Néanmoins, ceci illustre également la limite des algorithmes BRIG et GIG car si l'on suppose que les données contiennent un échantillon caractéristique, l'algorithme qui s'impose est RPNI.

9.2 L'algorithme RPNI avec suppression des finales (*)

9.2.1 Présentation de l'algorithme (*)

Nous savons que les états auxiliaires sont ceux qui introduisent le non-déterminisme lors des fusions dans l'arbre des préfixes. La fusion pour déterminisation a été introduite dans l'algorithme RPNI afin de supprimer le non-déterminisme. Elle a pour effet de replier les transitions pointant sur des états auxiliaires sur celles pointant vers des états fixes. Dans la mesure où l'on replie des transitions sur des transitions déjà existantes, les états auxiliaires et les transitions associées sont superflus. Nous proposons une variante de l'algorithme RPNI avec *suppression des finales*. Elle consiste à éliminer tous les états qui sont considérés comme états auxiliaires (voir paragraphe 9.1.2).

Le problème consiste à déterminer quels sont les états fixes, variables et auxiliaires dans le $PTA(I_+)$ sans pour autant connaître l'automate canonique du langage cible. Nous savons que, si les données contiennent un échantillon caractéristique, les fusions entre états fixes sont interdites car elles donnent lieu à des automates incompatibles avec I_-. Par contre, si une fusion entre un état j et un état i, avec $j < i$, est permise et si les fusions sont effectuées dans l'ordre propre à l'algorithme RPNI alors l'état i est nécessairement variable et l'état j est fixe. Le sous-arbre dont l'état i est la racine est constitué d'états auxiliaires à l'exception de l'état i. Dès lors, avant fusion avec l'état j, toutes les finales de l'état i correspondent aux finales de l'état i dans le $PTA(I_+)$ et peuvent être supprimés du nouvel automate quotient.

L'algorithme RPNI avec suppression des finales peut être résumé de la manière suivante.

Algorithme RPNI avec suppression des finales

entrée I_+, I_-
sortie Un AFD compatible avec I_-

début

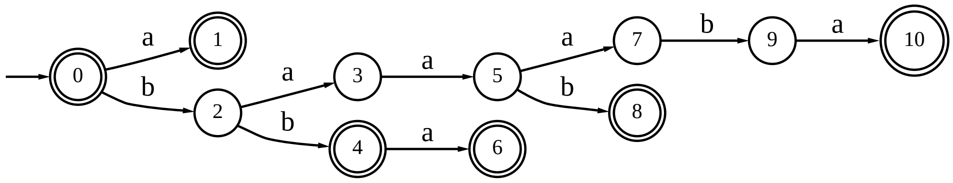
// N désigne le nombre d'états du $PTA(I_+)$

$\pi \leftarrow \{\{0\}, \{1\}, \ldots, \{N-1\}\}$ // Un bloc par préfixe dans l'ordre <
$A \leftarrow PTA(I_+)$

pour $i = 1$ **jusqu'à** $|\pi| - 1$ // Boucle sur les blocs de la partition π
  pour $j = 0$ **jusqu'à** $i - 1$ // Boucle sur les blocs de rang inférieur
    $\pi' \leftarrow \pi \backslash \{B_j, B_i\} U \{B_i U B_j\}$ // Fusion du bloc B_i et du bloc B_j
    $A/\pi' \leftarrow$ *dériver* (A, π')
    si *compatible* $(A/\pi', I_-)$ **alors** // Analyse non-déterministe de I_-
      $A'/\pi'' \leftarrow$ *suppression_finales* $(A/\pi', i)$
      $A \leftarrow A'/\pi''$
      $\pi \leftarrow \pi''$
      sortir boucle // Quitter la boucle sur j
    fin si
  fin pour // Fin boucle sur j
fin pour // Fin boucle sur i
retourner A
fin RPNI avec suppression des finales

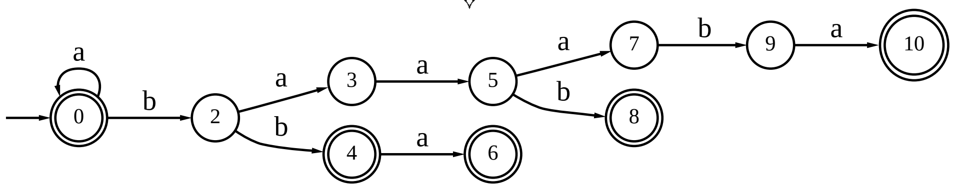
La fonction *compatible* $(A/\pi', I_-)$ renvoie VRAI si l'automate quotient est compatible avec I_-, sinon elle renvoie FAUX. La fonction *suppression_finales* $(A/\pi', i)$ renvoie l'automate obtenu en supprimant de l'automate A/π' les finales de l'état i dans le $PTA(I_+)$. Elle adapte la partition π' et renvoie l'automate A'/π''.

La figure 9.6 présente un exemple d'exécution de l'algorithme RPNI avec suppression des finales. Les données sont celles de l'échantillon caractéristique de l'exemple déjà cité, c'est-à-dire $I_+^c = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_-^c = \{b, ab, aba\}$. Le symbole "$\Downarrow$" relie les automates dérivés par une fusion. Lorsque l'automate obtenu est incompatible, une autre fusion est effectuée dans l'ordre défini. Le symbole "|" relie les automates obtenus par suppression des finales.

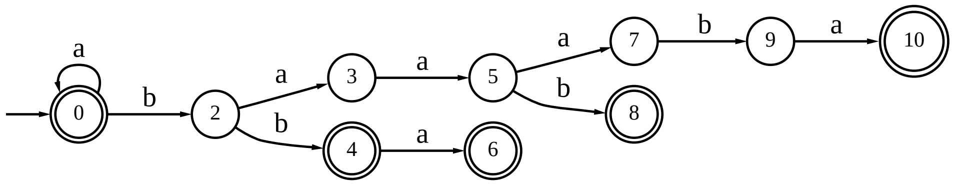


Fusion de 1 et 0.

$\Downarrow$

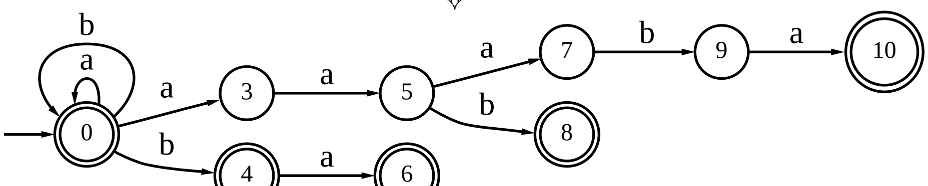


Compatibilité avec I_-

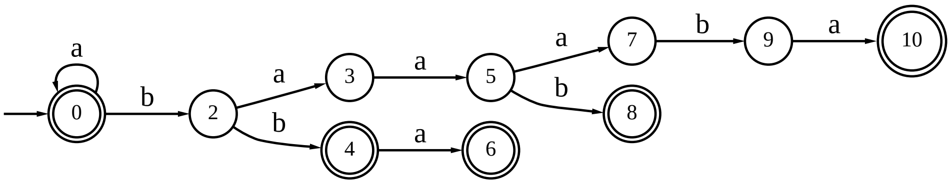


Fusion de 2 et 0.

$\Downarrow$

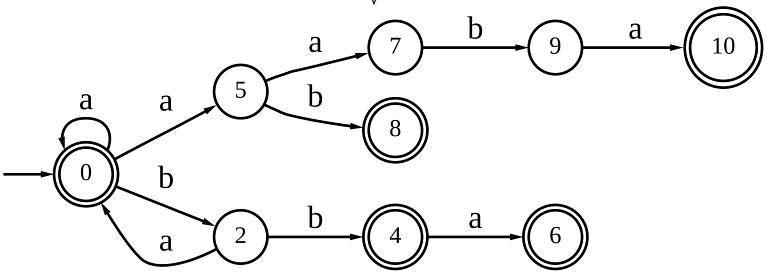


Incompatibilité avec I_-

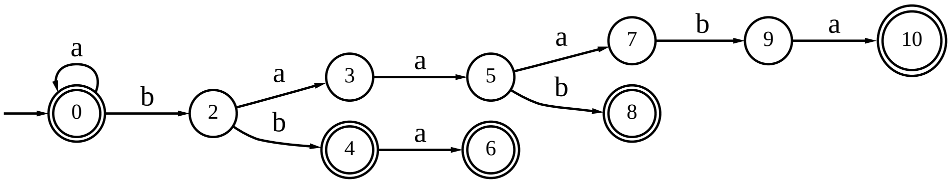


Fusion de 3 et 0.

$\Downarrow$

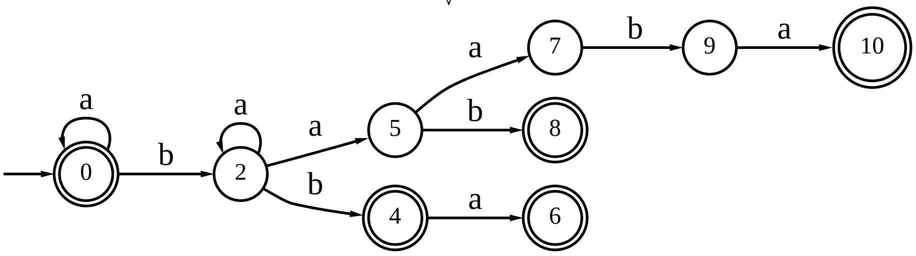


Incompatibilité avec I_-



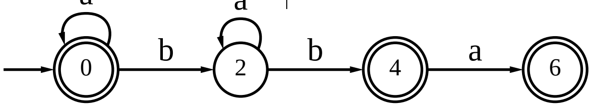
Fusion de 3 et 2.

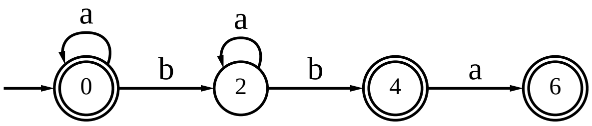
$\Downarrow$



Compatibilité avec I_-

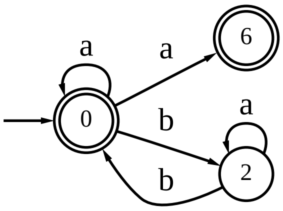
Suppression des finales de 3 avant sa fusion avec 2.





Fusion de 4 et 0.

$\Downarrow$



Compatibilité avec I_-

Suppression des finales de 4 avant sa fusion avec 0.

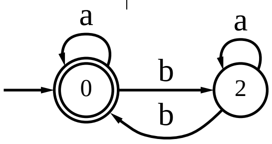


FIG. 9.6: Exécution de l'algorithme RPNI avec suppression des finales.

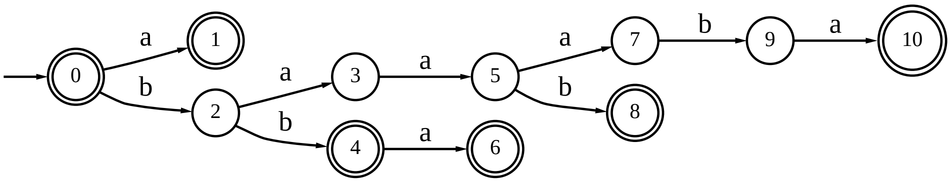
9.2.2 Propriétés de l'algorithme RPNI avec suppression des finales (*)

Cette variante de l'algorithme RPNI offre une autre analyse de sa complexité. Nous avions vu au paragraphe 7.2.2 qu'elle est estimée à $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$. Or, nous constatons que les fusions importantes sont celles qui concernent les état fixes et les états variables puisque la compatibilité avec I_- ne doit être testée que pour ces fusions. Dans l'algorithme RPNI original, la compatibilité est testée après la fusion pour déterminisation. Elle nécessite donc une analyse déterministe de I_- dont la complexité globale est $\mathcal{O}(||I_-||)$. Dans notre variante, la compatibilité est testée sur un automate qui peut être non-déterministe. Ceci est dû aux états auxiliaires qui ne sont éliminés que lorsque la compatibilité est assurée. Cependant, le non-déterminisme est limité car, par hypothèse de récurrence de l'algorithme, chaque fois que l'on sort de la boucle intérieure (la boucle sur j) l'automate courant A est déterministe. Seul l'état i, une fois fusionné avec l'état j, peut présenter un non-déterminisme, éventuellement sur plusieurs symboles. Chaque chaîne de I_- présente au plus deux acceptations par l'automate quotient. L'analyse non-déterministe peut donc être calculée par deux analyses déterministes, dont la complexité globale est $\mathcal{O}(||I_-||)$. Finalement, en négligeant la complexité de la suppression des finales, sachant qu'il y a $|Q|$ états fixes et que l'union des états variables et fixes représentent $\mathcal{O}(|Q|\cdot|\Sigma|)$ états, la complexité globale de l'algorithme peut être estimée à $\mathcal{O}(|Q|^2 \cdot |\Sigma| \cdot ||I_-||)$. Cette complexité dépend donc *linéairement* de la taille des données d'apprentissage. Bien que la complexité calculée soit optimiste, puisque nous avons négligé le coût de la suppression des finales, elle correspond mieux à l'évolution, observée expérimentalement, du temps de calcul en fonction du volume de données d'apprentissage.

Dans la mesure où le traitement n'est modifié, par rapport à l'algorithme original, que pour les états auxiliaires, l'algorithme RPNI avec suppression des finales identifie à la limite la classe des langages réguliers.

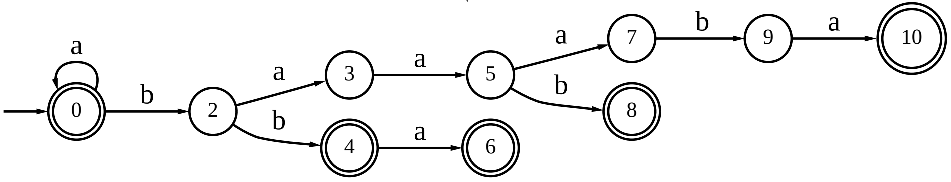
Cet algorithme souffre néanmoins d'une limitation importante car faute d'un échantillon caractéristique, la suppression des finales peut conduire à une solution qui ne soit pas compatible avec l'échantillon positif I_+. Cette propriété est illustrée par l'exemple suivant.

Nous considérons le même échantillon positif mais l'échantillon négatif est réduit à $I_- = \{b\}$. L'exécution de l'algorithme est présenté à la figure 9.7. La fusion de l'état 3 et de l'état 0 engendre un automate qui est compatible avec l'échantillon négatif. Par conséquent l'état 3 est considéré comme un état variable et ses finales, dans le $PTA(I_+)$, sont supprimées. La solution obtenue n'est pas compatible avec I_+, car la chaîne $baab$ n'est pas acceptée par la solution. Au contraire, dans l'algorithme original la solution proposée est toujours compatible avec I_+ et I_-, car les états considérés comme auxiliaires sont fusionnés avec d'autres au lieu d'être supprimés.

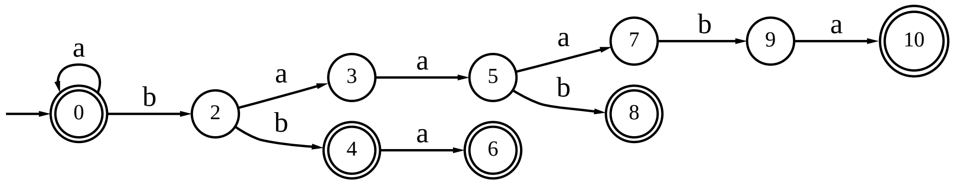


Fusion de 1 et 0.

$\Downarrow$

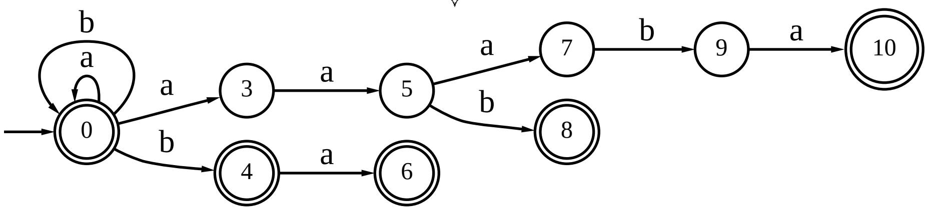


Compatibilité avec I_-

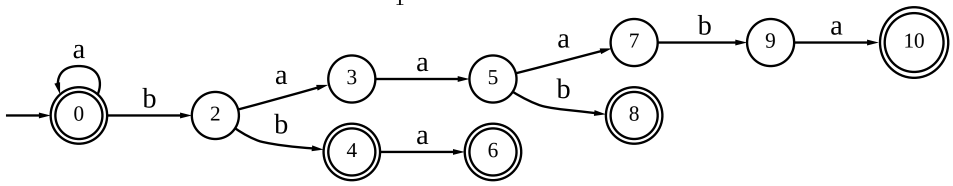


Fusion de 2 et 0.

$\Downarrow$

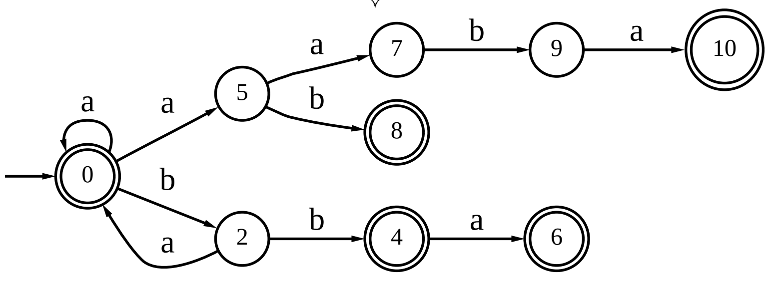


Incompatibilité avec I_-



Fusion de 3 et 0.

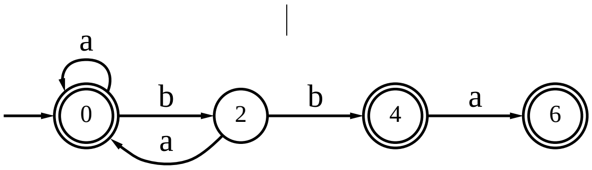
$\Downarrow$



Compatibilité avec I_-

Suppression des finales de 3 avant sa fusion avec 0.

|



Incompatibilité avec I_+ : rejet de la chaîne *baab*.

FIG. 9.7: Incompatibilité vis-à-vis de l'échantillon positif par l'exécution de l'algorithme RPNI avec suppression des finales.

9.3 Construction incrémentale de l'espace de recherche (*)

Dans ce paragraphe, nous étudions la structure de l'espace de recherche pour une présentation séquentielle des données d'apprentissage. Dans ce cas, les données sont reçues une par une dans un ordre quelconque et elles sont étiquetées comme exemple ou contre-exemple. L'alphabet du langage à identifier est également construit au fur et à mesure en fonction des lettres présentes dans les données reçues.

Hypothèses de récurrence

Nous supposons, par hypothèse de récurrence, que la solution courante à l'étape k est un automate $A(k)$ déterministe et compatible avec l'ensemble des données reçues jusque là. Plus précisément, nous mémorisons l'ensemble des p exemples constituant l'échantillon positif courant $I_+(p)$, sous la forme de l'arbre des préfixes $PTA(I_+(p))$, et nous mémorisons l'ensemble des q contre-exemples constituant l'échantillon négatif courant $I_-(q)$, avec $k = p$ (exemples) $+\ q$ (contre-exemples). De plus, la solution $A(k)$ est soit l'automate vide soit supposée appartenir à l'ensemble frontière $BS_{PTA}(I_+(p), I_-(q))$.

Tous les exemples et contre-exemples reçus sont supposés distincts. Cette hypothèse sera justifiée par les propriétés 9.4 et 9.5 qui impliquent notamment qu'une donnée déjà présentée ne peut modifier la solution courante.

Initialisation de la récurrence

Nous pouvons aisément initialiser la récurrence en appliquant l'algorithme RPNI avec la première chaîne. S'il s'agit d'un exemple, la solution sera l'automate universel et s'il s'agit d'un contre-exemple la solution sera l'automate vide. Si la solution $A(k)$ est supposée déterministe et appartenant à l'ensemble frontière, nous savons qu'il s'agit de l'automate canonique pour le langage qu'il accepte (voir propriété 6.10).

Construction des treillis successifs

La propriété 9.2 caractérise la relation entre les treillis successifs lorsqu'un nouvel exemple est reçu.

Propriété 9.2 $Lat(PTA(I_+(p))) \nsubseteq Lat(PTA(I_+(p+1)))$.

Preuve :
Les deux treillis peuvent être complètement distincts, c'est-à-dire $Lat(PTA(I_+(p))) \cap Lat(PTA(I_+(p+1))) = \phi$. Lorsque l'exemple d'indice $p+1$ introduit un nouveau symbole de l'alphabet, l'automate universel sur le nouvel alphabet n'appartient pas à $Lat(PTA(I_+(p)))$. Comme l'automate universel est quotient de chacun des automates du treillis $Lat(PTA(I_+(p+1)))$, aucun de ces automates ne peut appartenir à $Lat(PTA(I_+(p)))$.

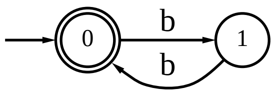
Par ailleurs, lorsque l'exemple d'indice $p+1$ n'introduit pas de nouvel élément de l'alphabet, l'automate universel est commun aux deux treillis, mais l'inclusion des treillis est impossible car $PTA(I_+(p)) \notin Lat(PTA(I_+(p+1)))$. En effet, le $PTA(I_+(p+1))$ est obtenu en étendant le $PTA(I_+(p))$ pour accepter le nouvel exemple. Ces deux automates diffèrent par au-moins un état d'acceptation et éventuellement plusieurs états et transitions. En particulier, $I_+(p+1)$ n'est pas inclus dans le langage accepté par le $PTA(I_+(p))$. Comme, le langage accepté par les automates de $Lat(PTA(I_+(p+1)))$ comprend $I_+(p+1)$, on en déduit que le $PTA(I_+(p))$ n'appartient pas à ce treillis. □

Acceptation d'un nouvel exemple

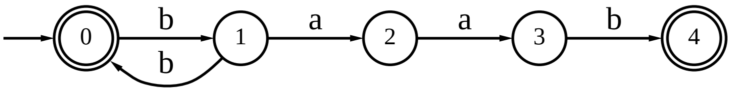
Partant d'une solution $A(k)$ nous vérifions si le nouvel exemple x appartient à $L(A(k))$. Sinon, nous pouvons étendre $A(k)$ par *le plus court suffixe*. Cette opération consiste à chercher le plus long préfixe u de x accepté par l'automate $A(k)$. L'acceptation du préfixe u suppose qu'il existe un état q tel que $q \in \delta^*(q_0, u)$[6]. Comme $A(k)$ est déterministe, l'état q est nécessairement unique. Depuis l'état q, l'automate $A(k)$ est étendu par le suffixe v pour accepter $x = uv$.

Nous notons $A^x(k)$ l'automate obtenu par extension du plus court suffixe de x à partir de l'automate $A(k)$. $A^x(k)$ est nécessairement déterministe. Si x est déjà accepté par $A(k)$, le plus court suffixe v est la chaîne vide et $A^x(k) = A(k)$. Un autre cas particulier se présente si l'automate $A(k)$ est l'automate vide. L'extension correspond alors à la construction de l'arbre accepteur des préfixes de la chaîne x.

[6]Au cas où le plus long préfixe accepté est la chaîne vide, l'état q correspond à l'état initial q_0.



(a)



(b)

FIG. 9.8: Extension par le plus court suffixe.

A titre d'exemple, nous supposons que l'automate $A(k)$ est celui présenté à la figure 9.8 (a) et nous supposons que le nouvel exemple est $bbbaab$. Le plus long préfixe accepté est bbb et mène à l'état 1. Depuis cet état, l'automate est étendu par le suffixe aab. L'automate résultant est présenté à la figure 9.8 (b).

L'extension par le plus court suffixe peut créer un automate incompatible avec l'échantillon négatif courant même si $A(k)$ est compatible. Dans notre exemple, $L(A(k)) = (bb)^*$. L'extension a permis d'accepter la chaîne $bbbaab$ mais également toute chaîne de la forme $b(bb)^*aab$. Par exemple si la chaîne $baab$ appartient à l'échantillon négatif courant, la solution étendue est incompatible. Cet inconvénient sera résolu au paragraphe 9.4.1.

La propriété 9.3 stipule que l'extension par le plus court suffixe garantit que, si la solution courante à l'étape k appartient à l'espace de recherche construit sur les p premiers exemples, l'automate étendu appartient nécessairement à l'espace de recherche construit par l'ajout du nouvel exemple.

Propriété 9.3 *$A(k) \in Lat(PTA(I_+(p))) \Rightarrow A^x(k) \in Lat(PTA(I_+(p+1)))$, avec x l'exemple d'indice $p+1$.*

Preuve :
Par hypothèse, $A(k)$ est un automate quotient de $PTA(I_+(p))$. Donc, $I_+(p)$ est structurellement complet relativement à $A(k)$. Nous désignons par v le plus court suffixe correspondant à l'extension de $A(k)$ pour obtenir $A^x(k)$. Nous désignons par w le plus court suffixe correspondant à l'extension du $PTA(I_+(p))$ pour obtenir $PTA(I_+(p+1))$. Par construction, v est un suffixe de w et donc $I_+(p+1)$ est structurellement complet relativement à $A^x(k)$. Par conséquent, $A^x(k) \in Lat(MCA(I_+(p+1)))$. Finalement, puisque $A^x(k)$ est déterministe, il appartient à $Lat(PTA(I_+(p+1)))$. □

La propriété 9.4 stipule que si l'automate $A(k)$, qui, par hypothèse de récurrence, appartient à l'ensemble frontière du treillis à l'étape k, accepte le nouvel exemple alors il appartient également à l'ensemble frontière dans le nouveau treillis.

Propriété 9.4 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *et* $A(k) = A^x(k) \Rightarrow A^x(k) \in BS_{PTA}(I_+(p+1), I_-(q))$, *avec* x *l'exemple d'indice* $p+1$.

Preuve :
Nous avons $A(k) = A^x(k)$ si la chaîne x appartient au langage accepté par $A(k)$. Par hypothèse $A(k)$ appartient à $BS_{PTA}(I_+(p), I_-(q))$, c'est-à-dire qu'il est à une profondeur maximale dans $Lat(PTA(I_+(p)))$. Comme l'échantillon négatif n'est pas modifié $A(k)$ est également à une profondeur maximale dans le nouveau treillis $Lat(PTA(I_+(p+1)))$. □

Cette propriété justifie l'hypothèse que tous les exemples reçus soient distincts car un exemple x déjà rencontré impliquerait nécessairement $A^x(k) = A(k)$. Par ailleurs, l'égalité $A^x(k) = A(k)$ peut être obtenue lorsqu'un exemple x n'a pas encore été rencontré mais que les généralisations déjà effectuées jusqu'à l'étape k permettent l'acceptation de x par $A(k)$.

Réception d'un nouveau contre-exemple

La réception d'un nouveau contre-exemple ne modifie pas l'espace de recherche puisque, en vertu du théorème 6.2, la solution est à rechercher dans le treillis relatif à l'arbre accepteur des préfixes de l'échantillon positif. Par contre, l'ensemble frontière est généralement modifié par l'ajout d'un nouveau contre-exemple. En d'autres termes, la solution courante $A(k)$ appartenant à l'ensemble frontière $BS_{PTA}(I_+(p), I_-(q))$ peut ne pas appartenir à $BS_{PTA}(I_+(p), I_-(q+1))$. La propriété 9.5 stipule que si l'automate $A(k)$, qui, par hypothèse de récurrence appartient à l'ensemble frontière du treillis à l'étape k, n'accepte pas le nouveau contre-exemple alors il appartient également à l'ensemble frontière dans le même treillis avec un échantillon négatif étendu.

Propriété 9.5 $A(k) \in BS_{PTA}(I_+(p), I_-(q))$ *et* $x \notin L(A(k)) \Rightarrow A(k) \in BS_{PTA}(I_+(p), I_-(q+1))$, *avec* x *le contre-exemple d'indice* $q+1$.

Preuve :
Puisque la chaîne x n'appartient pas au langage accepté par $A(k)$ elle ne modifie pas la compatibilité de $A(k)$ relativement à l'échantillon négatif. L'automate $A(k)$ étant par hypothèse à une profondeur maximale conformément aux q premiers contre-exemples, il reste à une profondeur maximale par l'ajout du nouveau contre-exemple. Par conséquent, $A(k)$ appartient à $BS_{PTA}(I_+(p), I_-(q+1))$.□

Cette propriété justifie également l'hypothèse que tous les contre-exemples reçus soient distincts car dans le cas de la réception d'un contre-exemple déjà rencontré l'ensemble frontière ne serait pas modifié et la solution courante resterait forcément compatible.

9.4 L'algorithme RPNI2 (*)

9.4.1 Comment garantir la compatibilité avec I_- (*)

Les différentes propriétés démontrées dans le paragraphe 9.3 caractérisent l'espace de recherche et son ensemble frontière lors de l'ajout d'un nouvel exemple ou d'un nouveau contre-exemple. De plus, nous avons vu qu'il y a deux cas possibles d'incompatibilité avec l'échantillon négatif. Le premier cas résulte de l'extension par le plus court suffixe. Elle permet d'étendre la solution courante et d'obtenir un élément qui appartient nécessairement au nouvel espace de recherche. Cet élément peut néanmoins être incompatible avec l'échantillon négatif. Le second cas concerne l'ajout d'un nouveau contre-exemple qui rend la solution courante incompatible.

Deux cas équivalents d'incompatibilité

Les deux cas d'incompatibilité avec I_- peuvent être traités de façon identique. Nous avons un automate A déterministe qui appartient à $Lat(PTA(I_+))$ et tel que $L(A) \cap I_- \neq \phi$. De plus, nous connaissons la partition π telle que $A = PTA(I_+)/\pi$. En effet, cette partition peut être mémorisée d'une étape à la suivante lorsque un nouveau contre-exemple est reçu ou facilement adaptée lors de l'extension par le plus court suffixe[7].

Un exemple d'incompatibilité avec I_-

A titre d'exemple, considérons l'arbre des préfixes et un automate quotient A déterministe représentés à la figure 9.9. Les données déjà reçues sont supposées être $I_+ = \{\lambda, a, bb, bba, baab, baaaba\}$ et $I_- = \{abbb\}$.

[7]Chaque état créé par l'extension est associé à un nouveau bloc dont il est le seul élement.

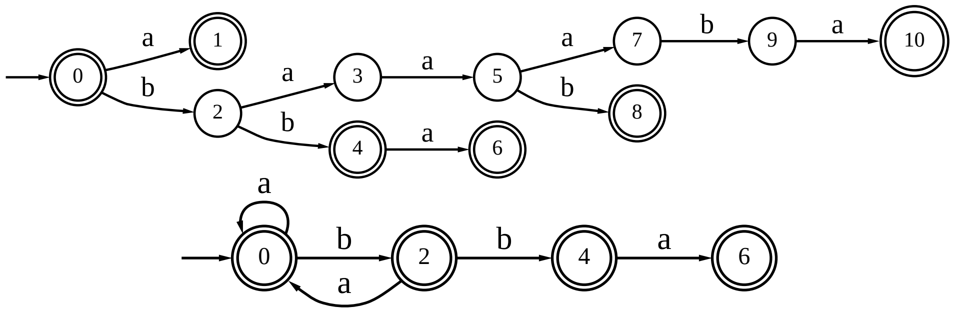


FIG. 9.9: L'arbre des préfixes et un automate quotient déterministe.

On vérifie aisément que la partition permettant d'obtenir A à partir du $PTA(I_+)$ est donnée par

$$\pi = \{\{0, 1, 3, 5, 7, 10\}, \{2, 8, 9\}, \{4\}, \{6\}\}.$$

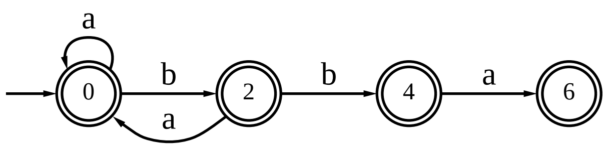
Supposons que nous recevions b comme nouveau contre-exemple. L'automate A accepte la chaîne b et devient donc incompatible avec $I_- = \{b, abbb\}$. L'automate A est incompatible car il y a au moins une fusion d'états qui permet l'acceptation de la chaîne b. Nous cherchons une partition π' plus fine que π et tel que $PTA(I_+)/\pi'$ soit compatible avec I_-.

Traitement d'un contre-exemple qui crée l'incompatibilité avec I_-

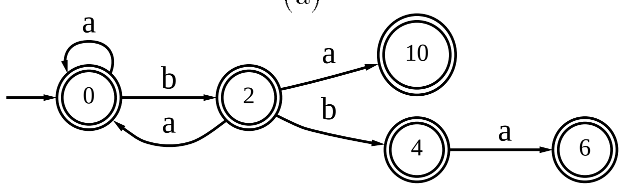
Nous savons que le rôle des contre-exemples est de distinguer les éléments du noyau et des préfixes courts qui n'ont pas même ensemble de finales. Plus précisément, nous supposons que le contre-exemple w participe à la définition d'un échantillon caractéristique. Dans ce cas, il existe un préfixe $u \in \mathrm{Pr}(L)$ et une chaîne v tels que $w = uv$ et $uv \in I_-^c$. Connaissant le contre-exemple w et l'arbre des préfixes, il est aisé de déterminer les chaînes u et v. Dans notre cas, le contre-exemple b mène à l'état 2 du $PTA(I_+)$. Nous avons $u = b$ et $v = \lambda$. Puisque le contre-exemple est accepté par l'automate quotient, il y a nécessairement un état q_v dont le suffixe est v et qui appartient au bloc du préfixe u. Dans notre exemple, l'état q_v a λ pour suffixe et correspond à l'état 8 du $PTA(I_+)$. Il appartient dans la partition π au bloc de l'état 2. L'opération que nous venons de décrire constitue la *définition du préfixe d'acceptation.* Il détermine en particulier le *rang* de l'état q_v. Nous dirons que le bloc de l'état q_v est *fissionné* ou que l'on sort l'état q_v de son bloc pour créer un nouveau bloc $\{q_v\}$.

Respect de l'ordre standard, considéré en sens inverse

Nous cherchons à trouver une partition π' plus fine que la partition π tel que l'automate quotient soit compatible tout en garantissant l'identification à la limite. Or nous avons vu, par la démonstration du théorème 9.1, que l'ordre des fusions est important. Nous cherchons, afin de garantir l'identification, à retrouver l'automate quotient que l'algorithme RPNI construirait à partir du $PTA(I_+)$ sous contrôle du même échantillon négatif I_-. Une des caractéristiques de l'algorithme RPNI est d'essayer les fusions dans l'ordre standard. Par conséquent, si l'on nie la fusion d'un état q_v avec un autre état, il est nécessaire de nier la fusion de tous les états de rang supérieur à q_v. Dans notre exemple, q_v étant l'état 8, nous devons sortir les états 8, 9 et 10 de leurs blocs respectifs. Nous effectuons ces fissions dans l'ordre inverse c'est-à-dire que nous commençons par l'état 10. La figure 9.10 (a) représente l'automate original et la figure 9.10 (b) représente l'automate obtenu après avoir sorti l'état 10.



(a)



(b)

FIG. 9.10: Fission d'un automate quotient déterministe.

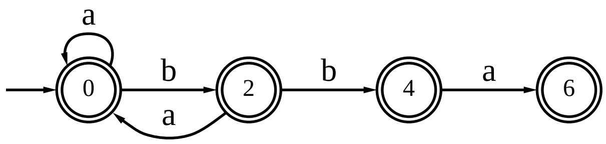
La fission pour déterminisation

Alors que l'automate original est déterministe, la fission du bloc de l'état 10 introduit un non-déterminisme. Cela signifie que l'état 10 de notre exemple est un état auxiliaire. Nous pouvons appliquer une opération de *fission pour déterminisation,* réciproque de la fusion pour déterminisation. Elle est formellement définie comme suit.

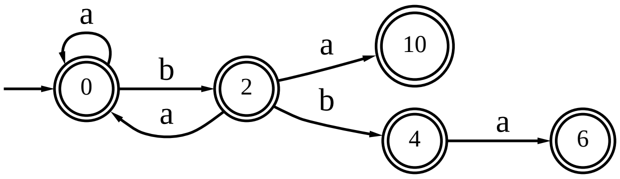
> Si $A = PTA(I_+)/\pi$ et si Bq désigne le bloc de l'état q dans la partition π, la *fission* de π *relativement* à q consiste à définir une nouvelle partition
>
> $$\pi' = (\pi \setminus Bq) \cup \{Bq \setminus q\} \cup \{q\}$$
>
> **Si** dans le $PTA(I_+)$ l'état q a un prédécesseur q_p sur la lettre a, c'est-à-dire $q \in \delta_{PTA}(q_p, a)$, et **si** $\exists Bq_p, B' \in \pi'$ avec $B' \neq \{q\}$ et $\delta(Bq_p, a) \supseteq \{\{q\}, B'\}$ dans $PTA(I_+)/\pi'$ **alors** effectuer la fission de π' relativement à q_p.

En d'autres termes, la fission de l'état q introduit un non déterminisme sur la lettre a puisqu'il existe un état q_p appartenant au bloc Bq_p, prédécesseur de q, duquel est issu, dans l'automate quotient $PTA(I_+)/\pi'$, une autre transition sur la lettre a. Dans ce cas, on effectue la fission de π' relativement à q_p.

L'opération de fission pour déterminisation est récursive, c'est-à-dire qu'elle implique une fission qui peut elle même induire un non-déterminisme qui sera réduit par une nouvelle fission. Pour les raisons déjà évoquées, nous désirons respecter un ordre des fissions qui soit l'ordre inverse des fusions dans l'algorithme RPNI. Chaque fois que l'on effectue la fission de la partition π relativement à l'état q, nous sortons également de leurs blocs respectifs les états du sous-arbre dont l'état q est la racine dans le $PTA(I_+)$. Ces états correspondent aux finales de q dans le $PTA(I_+)$. Dans notre exemple, ces opérations sont résumées à la figure 9.11. Le symbole "$\downarrow$" relient les automates obtenus par fission pour déterminisation.

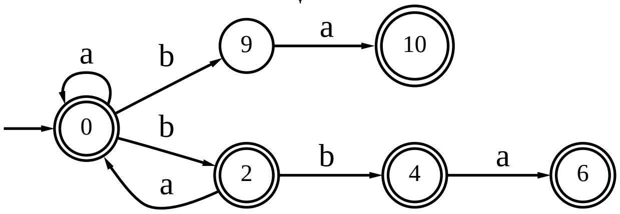


10 est sorti du bloc de 0.



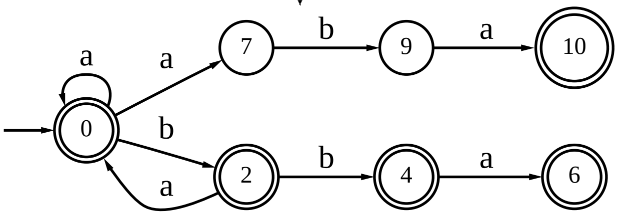
9 est sorti du bloc de 2.

↓



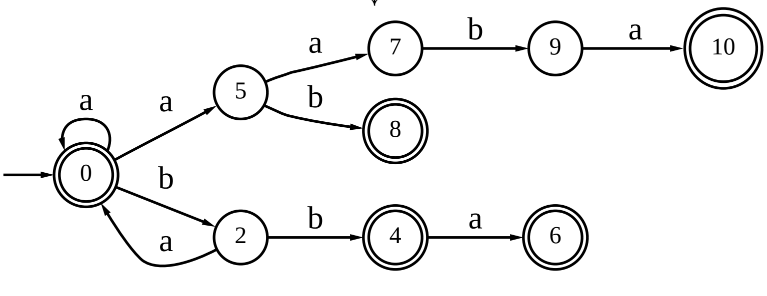
7 est sorti du bloc de 0.

↓



5 est sorti du bloc de 0 et 8 (finale de 5) est sorti.

↓



3 est sorti du bloc de 0.

↓

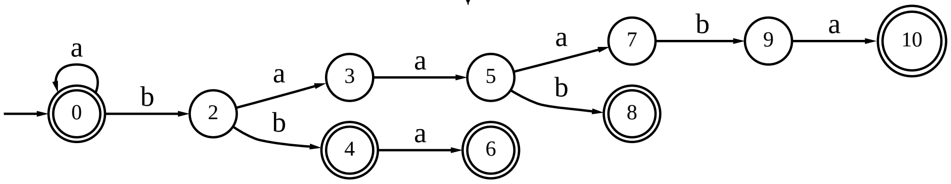


FIG. 9.11: Fission pour déterminisation.

Propriété 9.6 *L'opération de fission pour déterminisation s'arrête nécessairement sur un automate déterministe. Si les données contiennent un échantilllon caractéristique, elle s'arrête par la fission relativement à un état variable.*

Preuve :
Par construction, la fission pour déterminisation garantit l'obtention d'un automate déterministe. Lorsque les données incluent un échantillon caractéristique, les états auxiliaires responsables du non-déterminisme ont été sortis de leurs blocs respectifs. De plus, la définition du préfixe d'acceptation détermine le rang minimal de tous les états qui doivent participer à une fission. Lorsque tous ces états ont été traités, l'automate résultant est compatible. Puisque tous les états auxiliaires ont été sortis, le dernier état sorti est nécessairement un état variable. □

9.4.2 Identification à la limite de l'algorithme RPNI2 (*)

L'algorithme RPNI2[8] constitue l'extension incrémentale de l'algorithme RPNI. Il peut être résumé de la manière suivante.

A la réception d'un nouvel exemple x, l'arbre des préfixes est étendu par le biais de la fonction *étendre* $(PTA(I_+(p)), x)$. La fonction *compatible* (A, I) renvoie VRAI si l'automate A est compatible avec l'échantillon I, sinon elle renvoie FAUX. Lorsque I est un échantillon positif, la compatibilité requiert l'acceptation de toutes les phrases de I. Réciproquement, lorsque I est un échantillon négatif, la compatibilité requiert le rejet de toutes les phrases de I. Dès lors, si l'automate courant reste compatible après la réception d'un nouvel exemple ou d'un nouveau contre-exemple, il constitue la nouvelle solution qui est retournée. Insistons sur le fait que ceci n'implique pas que la nouvelle donnée ait été vue auparavant. Par contre, lorsqu'il y a incompatibilité avec un nouvel exemple x, l'automate courant est étendu par le plus court suffixe de x. La fonction *extension_suffixe* $(A/\pi, x)$ réalise cette opération. A ce stade, l'automate courant A appartient nécessairement à l'espace de recherche défini par l'ensemble des données reçues jusqu'alors. Cet automate est éventuellement incompatible avec I_-. La fonction *préfixe_acceptation* (A, I_-) renvoie le rang minimal de l'état relativement auquel la partition π sera fissionnée. Si, plusieurs éléments de I_- sont acceptés, c'est le rang minimal défini sur l'ensemble des acceptations qui sera considéré.

[8] L'abréviation **RPNI2** provient de la terminologie anglaise pour "Regular Positive and Negative Incremental Inference".

La fonction *fission_déterminisation* $(A/\pi, j)$ effectue la fission de l'automate A/π relativement à l'état j et recherche, par fissions successives, un automate déterministe. Dans le pire des cas, l'automate obtenu après fission correspond à l'arbre des préfixes. Finalement, l'algorithme RPNI original est appliqué en partant de l'automate obtenu après fission. Dès lors, dans le pire des cas, l'algorithme RPNI est appliqué sans tirer parti de la solution temporaire qui avait été obtenue à l'étape précédente de la présentation séquentielle. Cependant, en général, l'algorithme RPNI sera appliqué sur un automate comportant moins d'états que l'arbre des préfixes. En d'autres termes, la recherche s'effectue dans un sous-treillis de $Lat(PTA(I_+))$.

Algorithme RPNI2

```
entrée PTA(I+(p))            // L'arbre accepteur des préfixes de p exemples
       I-(q)                 // L'échantillon négatif comportant q contre-exemples
       A/π                   // L'automate solution courante
       x                     // La nouvelle donnée à traiter
sortie Un AFD compatible avec (I+(p), I-(q)) et x

début

si x est un exemple alors
  PTA(I+(p+1)) ← étendre (PTA(I+(p)), x)     // Extension de l'arbre des préfixes
  si compatible (A/π,{x}) alors     // Vérification de l'acceptation du nouvel exemple
    retourner A/π
  sinon
    A/π' ← extension_suffixe (A/π, x)        // Extension par le plus court suffixe
    A ← A/π'
    π ← π'
  fin si
fin si

si x est un contre-exemple alors
  si compatible (A/π,{x}) alors     // Vérification du rejet du nouveau contre-exemple
    retourner A/π
  sinon
    A ← A/π'
    π ← π'
  fin si
fin si

i ← préfixe_acceptation (A, I-)              // Rang minimal des états à fissioner
// N désigne le nombre d'états du PTA
pour j = N jusqu'à i              // Fission des états dans l'ordre < inverse
    A/π' ← fission_déterminisation (A/π, j)
    π ← π'
fin pour
A ← A/π

// Application de l'algorithme RPNI à partir de l'automate A
pour i = 1 jusqu'à |π| - 1           // Boucle sur les blocs de la partition π
    pour j = 0 jusqu'à i - 1         // Boucle sur les blocs de rang inférieur
        π' ← π\{Bj, Bi}U{BiUBj}            // Fusion du bloc Bi et du bloc Bj
        A/π' ← dériver (A, π')
        π'' ← fusion_déterm (A/π')
        si compatible (A/π'',I-) alors          // Analyse déterministe de I-
          A ← A/π''
          π ← π''
          sortir boucle                         // Quitter la boucle sur j
        fin si
    fin pour                                    // Fin boucle sur j
fin pour                                        // Fin boucle sur i
retourner A
fin RPNI2
```

Théorème 9.3 *L'algorithme RPNI2 identifie à la limite la classe des langages réguliers.*

Preuve :
Par hypothèse de récurrence et par la propriété 9.3, nous avons la garantie que l'automate obtenu, éventuellement après son extension par le plus court suffixe, est déterministe et appartient au nouvel espace de recherche. En vertu des propriétés 9.4 et 9.5, nous savons que s'il est compatible avec la nouvelle donnée alors il appartient à l'ensemble frontière. Si, par contre, il est incompatible avec I_- l'automate sera fissionné. La propriété 9.6 nous garantit que, lorsque les données contiennent un échantillon caractéristique, le résultat de la fission pour déterminisation est un automate compatible et que le dernier état sorti de son bloc est un état variable. En d'autres termes, un état variable est soit déjà fusionné avec l'état fixe partageant les mêmes finales, soit il est sorti de son bloc ainsi que toutes les finales de cet état dans le $PTA(I_+)$. Par conséquent, l'ordre des fusions ultérieures est cohérent avec celui de l'algorithme RPNI et, en vertu du théorème 9.1, l'identification à la limite est garantie.□

Le théorème 9.3 garantit l'identification de l'automate cible lorsque les données, reçues séquentiellement, contiennent un échantillon caractéristique. L'identification peut également être obtenue avec les mêmes données par l'algorithme RPNI appliqué sur une présentation à données fixées. Cela revient à mémoriser toutes les données reçues séquentiellement et à appliquer l'algorithme RPNI sur cet ensemble de données. Dans ce cas particulier, la solution proposée par les deux algorithmes est identique. Par contre, lorsque les données n'incluent pas d'échantillon caractéristique il n'y a pas d'assurance que les solutions proposées soient identiques. Cependant, les évaluations expérimentales du chapitre 10 montreront que ces deux algorithmes fournissent en pratique des résultats équivalents. L'intérêt de l'algorithme RPNI2 sur l'algorithme RPNI est qu'il permet de réduire la complexité globale de l'inférence. En effet, en cas d'incompatibilité de la solution trouvée à l'étape précédente, il cherche à construire une nouvelle solution dans un espace de recherche qui est généralement un sous-espace de celui de l'algorithme RPNI. Ce point sera également étudié expérimentalement au chapitre 10.

9.4.3 Complexité de l'algorithme RPNI2 (*)

Dans ce paragraphe, nous allons comparer la complexité théorique de l'algorithme RPNI2 par rapport à celle de l'algorithme RPNI. Si l'on désire appliquer l'algorithme RPNI lorsque les données sont présentées de façon séquentielle, il suffit de mémoriser les échantillons positif I_+ et négatif I_-

reçus jusqu'à l'étape en cours. Dans la mesure où ces informations sont également mémorisées pour l'application de l'algorithme RPNI2, leur complexité spatiale est identique[9] et vaut $\mathcal{O}(||I_+|| + ||I_-||)$.

Nous désirons évaluer le coût du traitement lors de la réception d'un nouvel exemple dans le cas où la solution de l'étape précédente est incompatible avec la nouvelle donnée. Dans ce cas, l'application de l'algorithme RPNI est effectuée dans le nouvel espace avec une complexité globale de $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$ (voir paragraphe 7.2.2). La dernière partie de l'algorithme RPNI2 consiste à appliquer l'algorithme RPNI à partir d'un automate quotient de l'arbre des préfixes. Dans le pire des cas, cet automate est l'arbre des préfixes lui-même et la complexité de l'algorithme RPNI2 est donc également $\mathcal{O}((\|I_+\| + \|I_-\|) \cdot \|I_+\|^2)$. Les autres opérations propre à l'algorithme RPNI2 ont une complexité négligeable asymptotiquement. En effet, l'extension par le plus court suffixe de la chaîne x a une complexité linéaire $\mathcal{O}(|x|)$. La définition du préfixe d'acceptation nécessite une analyse de tout l'échantillon négatif, car plusieurs contre-exemples peuvent être acceptés après l'extension par le plus court suffixe. Comme à ce stade l'automate courant est nécessairement déterministe, cette analyse présente une complexité linéaire $\mathcal{O}(||I_-||)$. Pour chaque contre-exemple accepté, il faut déterminer le préfixe d'acceptation dans le $PTA(I_+)$. Ceci représente une deuxième analyse déterministe des contre-exemples acceptés qui est également de complexité linéaire $\mathcal{O}(||I_-||)$. Finalement, la fission pour déterminisation consiste à sortir au plus autant d'éléments de la partition courante qu'il y a d'états dans le $PTA(I_+)$. Elle a donc une complexité qui dépend de la taille de l'arbre des préfixes, c'est-à-dire $\mathcal{O}(||I_+||)$.

En résumé, les algorithmes RPNI et RPNI2 ont une complexité spatiale et temporelle identique asymptotiquement. L'apport de l'inférence incrémentale, outre son intérêt conceptuel, permet de réduire la complexité pratique. En effet, en général, l'algorithme RPNI2 applique des fusions dans un sous-ensemble de $Lat(PTA(I_+))$. Nous verrons au chapitre 10 que la réduction apportée est intéressante.

[9]En toute rigueur, la complexité spatiale de l'algorithme RPNI2 est légèrement supérieure puisque l'automate solution à l'étape précédente est également mémorisé. Néanmoins, la complexité spatiale asymptotique est effectivement $\mathcal{O}(||I_+|| + ||I_-||)$.

Chapitre 10

Evaluation expérimentale de l'inférence (*)

10.1 Historique et motivations

Depuis le développement des travaux en inférence grammaticale, en particulier depuis les travaux fondateurs de Gold [Gold67, Gold78], l'évaluation expérimentale comparant différentes méthodes d'inférence a été relativement limitée. Miclet compare trois méthodes d'inférence sur une dizaines de langages types et sur base d'échantillons positifs construits manuellement [Miclet79]. Par ailleurs, la méthode d'inférence des langages k-réversibles [Angluin82] n'est illustrée que sur quelques langages et, généralement, un seul échantillon d'apprentissage pour chacun d'eux. Dans un certain sens, une évaluation expérimentale peut sembler inutile dans la mesure où il est démontré que cette méthode identifie exactement n'importe quel langage k-réversible, pour k fixé. Néanmoins, étant donné un échantillon positif, nous ne connaissons pas a priori la valeur de k et, de plus, nous n'avons aucune garantie que le langage à identifier est effectivement réversible. Par conséquent, il est intéressant de tester les méthodes d'inférence sur des langages considérés comme "représentatifs" d'une classe générale, dans notre cas la classe des langages réguliers. Lorsque ces langages n'appartiennent pas à la sous-classe propre à une méthode d'inférence, nous pouvons évaluer la qualité de l'approximation qui en résulte. Cette idée a déjà été mise en œuvre dans [García et al.90] où l'objectif était de mesurer la qualité d'approximation de l'extension stochastique de l'algorithme k-TSSI sur des données générées à partir d'une grammaire engendrant un langage non nécessairement k-testable.

Une part originale de notre travail est de proposer une méthode d'évaluation de n'importe quel méthode d'inférence régulière, à partir d'échantillons positif et négatif, et de présenter des résultats comparatifs entre les trois méthodes concurrentes RPNI, BRIG, GIG. Pour ce faire, nous avons défini un ensemble de langages réguliers que nous considérons comme typiques. Cette idée a déjà été proposée par Tomita [Tomita82]. Mais, dans ce cas, les échantillons d'apprentissage avaient été définis manuellement. Nous avons repris la base d'évaluation de Tomita et nous l'avons étendue pour obtenir un ensemble de 15 langages types. Ces langages seront présentés au paragraphe 10.3. De plus, nous proposons un protocole expérimental afin de tester chaque méthode sur plusieurs échantillons générés aléatoirement à partir des langages cibles. Ce protocole est détaillé au paragraphe 10.2.

Notons que les travaux récents de Lang [Lang92] constituent une évaluation expérimentale très approfondie d'un algorithme apparemment identique à l'algorithme RPNI. Cette étude présente l'avantage d'avoir été menée sur des automates cibles comportant jusqu'à quelques milliers d'états, générés aléatoirement. Cependant, un certain a priori est également présent dans cette étude, celui qui définit le protocole de génération aléatoire des automates cibles. De plus, les résultats présentés ne sont pas comparés avec ceux obtenus par d'autres méthodes.

Malgré l'intérêt comparatif de la base d'évaluation que nous proposons, elle souffre d'une limitation. En effet, bien que les langages étudiés puissent être considérés comme représentatifs de la classe des langages réguliers, ils sont définis sur un alphabet comportant au plus 3 lettres. C'est pourquoi nous avons également utilisé le langage de la tâche de Feldman (présentée au paragraphe 5.1.1) pour mener d'autres expérimentations. Elles nous ont permis de comparer les algorithmes GIG, RPNI et RPNI2. Des adaptations du protocole expérimental pour les tests sur ce langage sont justifiées au paragraphe 10.5.1 et les résultats comparatifs sont présentés au paragraphe 10.5.2.

10.2 Le protocole expérimental (*)

Nous avons défini un ensemble de langages considérés comme représentatifs de la classe des langages réguliers. Ces langages seront présentés au paragraphe 10.3. A partir de chaque langage L, nous construisons l'automate canonique $A(L)$ et l'automate complémentaire[1] $\overline{A(L)}$, acceptant le langage

[1] L'automate complémentaire $\overline{A(L)}$ d'un automate canonique $A(L)$ peut être obtenu aisément en deux étapes. Premièrement, l'automate $A(L) = (Q, \Sigma, \delta, q_0, F)$ est *complété*, c'est-à-dire qu'une transition par symbole de l'alphabet Σ est obligatoirement issue de chaque état. Cela peut nécessiter l'introduction d'un nouvel état (inutile pour l'acceptation du langage L) dans l'ensemble Q. Toutes les transitions ajoutées afin d'obtenir un automate

$\Sigma^* - L$. Les échantillons d'apprentissage sont générés comme suit :
(1) L'échantillon positif I_+ est fait d'un ensemble de chaînes générées à partir de $A(L)$, avec la propriété que I_+ soit structurellement complet. Nous désignons par $|I_+|_c$ le nombre de chaînes requis pour obtenir aléatoirement un premier échantillon structurellement complet. Selon la valeur d'un paramètre désigné par M, nous poursuivons la génération aléatoire jusqu'à obtenir un échantillon positif dont la taille $|I_+|$ est égale à $M.|I_+|_c$.
(2) L'échantillon négatif I_- est obtenu de façon similaire à partir de l'automate complémentaire $\overline{A(L)}$.

L'hypothèse de complétude structurelle de l'échantillon I_+ est justifiée dans la mesure où, sans cette hypothèse, nous devrions "inventer" des transitions dans l'automate inféré alors qu'elles n'auraient pas été exercées dans l'automate cible inconnu, modèle du processus que l'on tente d'identifier. Dans la mesure où nous définissons, à des fins d'évaluation, un certain nombre d'automates cibles, il devient légitime de s'assurer de la complétude structurelle des échantillons positifs relativement à ces automates.

Les échantillons négatifs sont générés à partir de l'automate complémentaire. Ceci nous permet d'obtenir par le biais d'une procédure automatique un ensemble d'exemples négatifs. Il n'y a aucune garantie théorique que l'ensemble ainsi généré constitue un "bon guide" pour l'apprentissage, ni qu'il soit plus approprié pour une méthode plutôt qu'une autre.

L'objectif de l'inférence est de retrouver l'automate cible, voire un automate acceptant le même langage, pour chacune des 10 paires d'échantillons (I_+, I_-) que nous générons pour chacun des 15 langages cibles. Remarquons qu'il n'existe pas de garantie théorique que la solution au problème du plus petit AFD compatible corresponde précisément à l'automate cible.

De plus, il nous semble intéressant de ne pas nous limiter à évaluer les possibilités d'identification exacte de l'automate cible mais également d'évaluer la qualité de l'automate inféré lorsque celui-ci ne correspond pas exactement à l'automate cible. Il s'agit d'une mesure expérimentale d'une identification approchée. Nous proposons donc de mesurer les taux de classification correcte, respectivement des chaînes positives et négatives, sur l'ensemble $T = \bigcup_{i=0}^{l} \Sigma^i$ de toutes les chaînes, définies sur l'alphabet Σ, de longueur inférieure ou égale à l. Cette longueur dépend des caractéristiques de l'automate cible[2].

complet pointent vers ce nouvel état. Ensuite, l'automate complété est *complémenté*, c'est-à-dire que les états non finaux deviennent des états finaux et réciproquement.

[2]Il est possible de caractériser un langage accepté par un automate canonique à partir du sous-langage fini constitué de toutes les chaînes du langage jusqu'à une longueur $l \leq 2N-1$, où N est le nombre d'états de l'automate canonique. Cette propriété découle du "Lemme de l'étoile" ou "Pumping lemma" [Aho et al.72].

Il nous semble également important de distinguer les taux de classification des chaînes respectivement positives et négatives. Sinon, l'automate vide pourrait avoir un taux de classification quasi parfait dans le cas où seulement une très faible proportion de chaînes positives apparaîtrait dans T. De même, l'automate universel aurait un taux de classification quasi parfait dans le cas d'une très faible proportion de chaînes négatives dans T.

10.3 La base d'évaluation

L'ensemble des 15 langages réguliers cibles est défini ci-dessous. Les langages L_1 à L_7 correspondent à ceux définis[3] par Tomita [Tomita82] :
L_1 : a^*
L_2 : $(ab)^*$
L_3 : toute chaîne définie sur l'alphabet $\{a, b\}$ sans un nombre impair de a consécutifs après un nombre impair de b consécutifs.
L_4 : toute chaîne définie sur l'alphabet $\{a, b\}$ sans plus de deux a consécutifs.
L_5 : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a et un nombre pair de b.
L_6 : toute chaîne définie sur l'alphabet $\{a, b\}$ telle que la différence entre le nombre de a et celui de b est divisible par 3.
L_7 : $a^*b^*a^*b^*$
L_8 : a^*b
L_9 : $(a^* + c^*)b$
L_{10} : $(aa)^*(bbb)^*$
L_{11} : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a et un nombre impair de b.
L_{12} : $a(aa)^*b$
L_{13} : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a.
L_{14} : $(aa)^*ba^*$
L_{15} : $bc^*b + ac^*a$

Les automates canoniques de ces 15 langages cibles sont présentés à l'annexe B.

Nous résumons dans le tableau 10.1 le nombre d'états et de transitions de l'automate canonique de chaque langage cible, ainsi que la longueur maximale des chaînes de test. Rappelons que l'ensemble de test est constitué de toutes

[3] Rappelons que les échantillons d'apprentissage sont différents de ceux proposés par Tomita et qu'ils ont été obtenus conformément au protocole décrit au paragraphe 10.2.

Langage	états	transitions	Longueur
L_1	1	1	9
L_2	2	2	9
L_3	4	7	9
L_4	3	5	9
L_5	4	8	9
L_6	3	6	9
L_7	4	7	9
L_8	2	2	9
L_9	4	7	7
L_{10}	5	6	9
L_{11}	4	8	9
L_{12}	3	3	9
L_{13}	2	4	9
L_{14}	3	4	9
L_{15}	4	6	7

TAB. 10.1: Caractéristiques des automates cibles et de l'ensemble de test.

les chaînes sur l'alphabet défini jusqu'à cette longueur[4], ensemble dont nous avons retiré les données d'apprentissage.

10.4 Résultats comparatifs de RPNI, GIG et BRIG (*)

10.4.1 Les trois algorithmes concurrents (*)

Nous présentons dans ce paragraphe les résultats comparatifs des trois algorithmes RPNI, GIG et BRIG. Ces résultats ont été obtenus conformément au même protocole expérimental et sur des jeux identiques de données d'apprentissage et de test. La valeur du paramètre M du processus de génération aléatoire des échantillons a été fixée à 3 (voir paragraphe 10.2).

Algorithme RPNI

Les résultats obtenus par l'algorithme RPNI sont présentés à la table 10.2. La première colonne correspond au taux de classification correcte des chaînes

[4]Dans le cas des langages L_9 et L_{15}, cette longueur est réduite à 7. Cela permet d'obtenir des ensembles de test de taille comparable avec celle des autres cas, malgré le fait que l'alphabet comporte ici 3 lettres au lieu de 2.

	+class	**-class**	**class**
L_1	100,0	100,0	100,0
L_2	93,3	99,9	96,6
L_3	100,0	100,0	100,0
L_4	90,7	90,4	90,5
L_5	52,5	89,5	71,0
L_6	82,4	97,3	89,8
L_7	89,7	94,4	92,0
L_8	100,0	100,0	100,0
L_9	100,0	98,1	99,0
L_{10}	94,4	99,9	97,2
L_{11}	85,3	92,4	88,8
L_{12}	100,0	99,6	99,8
L_{13}	100,0	100,0	100,0
L_{14}	90,2	98,9	94,6
L_{15}	100,0	99,0	99,5
Moyenne	91,9	97,3	94,6

TAB. 10.2: Résulats de l'algorithme RPNI.

positives et la deuxième colonne, celui des chaînes négatives. La dernière colonne correspond à la moyenne arithmétique des deux précédentes. Ces résultats sont moyennés, pour chaque langage cible, sur 10 paires (I_+, I_-) générées aléatoirement et indépendamment. La dernière ligne du tableau correspond à la moyenne arithmétique de ces différents résultats.

Les résultats obtenus avec l'algorithme RPNI correspondent à une inférence sur base des échantillons d'apprentissage pris globalement. Vu l'efficacité calculatoire de la méthode RPNI et la qualité des résultats obtenus, il n'a pas été nécessaire de recourir à une procédure d'inférence semi-incrémentale.

Algorithme GIG

Les algorithmes GIG et BRIG ayant une composante aléatoire, nous avons choisi d'effectuer 10 exécutions indépendantes pour chaque paire (I_+, I_-). Par conséquent, les résultats présentés dans les tables suivantes sont moyennés également sur ces 10 exécutions[5]. De plus, nous avons ajouté deux mesures de performance. La colonne "max" désigne le meilleure taux de classification (positif et négatif moyenné) obtenu sur les 10 exécutions indépendantes. Re-

[5]Chaque ligne correspond donc à 100 tests, c'est-à-dire 10 exécutions indépendantes pour chacune des 10 paires (I_+, I_-).

	+class	-class	class	max	sol
L_1	100,0	100,0	100,0	100,0	100,0
L_2	100,0	99,8	99,9	100,0	100,0
L_3	53,1	92,4	72,8	85,3	76,3
L_4	92,1	97,5	94,8	100,0	96,5
L_5	42,6	58,2	50,4	56,1	50,0
L_6	61,3	60,3	60,8	70,2	67,1
L_7	65,1	67,8	66,4	81,9	69,1
L_8	100,0	99,9	99,9	100,0	100,0
L_9	98,2	97,1	97,6	99,1	99,0
L_{10}	56,9	97,8	77,4	97,8	82,1
L_{11}	56,3	44,2	50,2	51,8	50,4
L_{12}	99,0	98,9	98,9	99,9	99,8
L_{13}	79,4	75,4	77,4	96,6	95,2
L_{14}	90,7	96,1	93,4	99,1	97,1
L_{15}	91,5	98,4	94,9	99,8	99,1
Moyenne	79,1	85,6	82,3	89,2	85,4

TAB. 10.3: Résultats de l'algorithme GIG.

marquons que nous devons connaître la solution recherchée pour pouvoir sélectionner la meilleure des 10 solutions proposées. Il s'agit donc d'une mesure seulement illustrative. La colonne "sol" correspond à la solution comportant le moins d'états, sur les 10 exécutions indépendantes. Nous pouvons sélectionner cette solution sans connaître l'automate cible. Par conséquent, c'est la mesure de performance que nous conservons pour évaluer la qualité d'une méthode incluant une composante aléatoire[6].

Les paramètres internes de l'algorithme GIG n'ont pas été ajustés avant chaque expérience mais ils ont été fixés après quelques essais et gardés constants pour toutes les expériences. De plus, ces essais nous ont incité à retenir la deuxième définition de la fonction d'évaluation (voir paragraphe 8.3.4). Il est possible qu'un réglage fin des paramètres internes permette d'améliorer les résultats obtenus. Nous pensons qu'il y a peu d'intérêt à rechercher le réglage optimal pour chaque nouveau jeu de données. Nous donnons ci-dessous la valeur des paramètres internes :

- Taille de la population = 100

[6]Il y a ici un compromis temps-performance. En effet, nous pouvons augmenter le nombre d'exécutions afin d'améliorer la solution proposée. Cette propriété résulte du fait que nous pouvons rechercher a priori la solution de taille minimale sur l'ensemble des exécutions et que nous supposons que cette solution est généralement la meilleure.

	+class	**-class**	**class**	**max**	**sol**
L_1	100,0	100,0	100,0	100,0	100,0
L_2	99,7	100,0	99,8	100,0	100,0
L_3	91,9	98,8	95,3	99,4	94,6
L_4	68,0	79,7	73,9	90,6	81,2
L_5	74,6	63,4	69,0	83,5	80,5
L_6	92,0	86,3	89,2	95,0	95,0
L_7	96,3	93,3	94,8	100,0	99,2
L_8	100,0	100,0	100,0	100,0	100,0
L_9	99,4	98,5	99,0	99,6	99,2
L_{10}	93,9	99,1	96,5	99,9	96,6
L_{11}	78,6	42,7	60,7	72,2	70,8
L_{12}	100,0	99,6	99,7	99,8	99,8
L_{13}	98,8	97,7	98,2	100,0	100,0
L_{14}	89,2	96,4	92,8	99,9	99,8
L_{15}	94,2	98,7	96,4	99,5	99,4
Moyenne	91,8	90,3	91,0	96,0	94,4

TAB. 10.4: Résultats de l'algorithme GIG semi-incrémental.

- Nombre maximal d'évaluations = 2000
- Taux de croisement = $0,2$
- Taux de mutation/bit = $0,01$
- Taux de descente = $0,5$

Nous effectuons 10 exécutions indépendantes pour chaque paire (I_+, I_-). Les résultats obtenus sont présentés dans les tables 10.3 et 10.4, respectivement sans et avec la procédure semi-incrémentale décrite au paragraphe 7.3.

Algorithme BRIG

Les résultats obtenus avec l'algorithme BRIG sont résumés aux tables 10.5 et 10.6, respectivement sans et avec la procédure d'inférence semi-incrémentale. Ils correspondent à une restriction aléatoire de l'espace de recherche contrôlée par un faisceau réglé de façon à garantir, en pratique, une complexité acceptable [Miclet et al.95, Henaff95].

Tout comme pour l'algorithme GIG, nous effectuons 10 exécutions indépendantes pour chaque paire (I_+, I_-) et les différentes colonnes des tables 10.5 et 10.6 ont la même signification.

	+class	**-class**	**class**	**max**	**sol**
Moyenne	51,1	95,4	73,6	84,8	79,5

TAB. 10.5: Résultats de l'algorithme BRIG.

	+class	**-class**	**class**	**max**	**sol**
Moyenne	82,9	92,6	87,8	93,8	91,8

TAB. 10.6: Résultats de l'algorithme BRIG semi-incrémental.

Comparaison des résultats

Nous avons résumé les résultats expérimentaux obtenus avec les différentes méthodes à la table 10.7. Nous avons également donné les résultats obtenus avec deux valeurs du paramètre M. Dans le cas $M = 3$, les résultats présentés correspondent à la dernière ligne des tables précédentes (colonne “class” pour l'algorithme RPNI et colonne “sol” pour les algorithmes GIG et BRIG).

M	**RPNI**	**GIG**	**GIG**$_{inc}$	**BRIG**	**BRIG**$_{inc}$
1	80,5	82,3	82,2	-	-
3	94,6	85,4	94,4	79,5	93,9

TAB. 10.7: Résultats comparatifs.

Remarquons que, dans le cas $M = 1$, nous avons observé qu'il existait des automates de tailles inférieures à celle de l'automate cible. Ceci explique les résultats plus mauvais obtenus avec ces jeux de données. Par ailleurs, il apparaît clairement que la procédure semi-incrémentale permet d'obtenir, avec l'algorithme GIG, des résultats comparables à ceux obtenus par l'algorithme RPNI. Le recours a une procédure semi-incrémentale s'avère donc particulièrement utile lorsque la taille des échantillons augmente.

Les performances obtenues avec l'algorithme GIG peuvent être interprétées comme suit. Sur la base d'évaluation définie, elles illustrent que l'inférence régulière peut être résolue par optimisation génétique et que les résultats obtenus sont comparables avec ceux du meilleur algorithme d'inférence connu jusqu'alors. Par ailleurs, la complexité polynomiale de l'algorithme RPNI étant garantie, cet algorithme d'inférence reste très avantageux[7].

Notons que nous aurions pu construire artificiellement des échantillons d'apprentissage garantissant l'impossibilité d'identification exacte par l'algorithme RPNI. En effet, son espace de recherche est $Lat(PTA(I_+))$ et nous savons par la propriété 6.6, que l'ensemble des langages que l'on peut identifier dans cet espace est strictement inclus dans celui que l'on peut identifier dans $Lat(MCA(I_+))$[8]. De plus, l'algorithme RPNI produit nécessairement un automate déterministe. Il peut donc également être mis en défaut si les échantillons positifs ne sont pas structurellement complets relativement à l'automate canonique du langage cible, tout en étant structurellement complets relativement à un AFN acceptant le même langage (voir propriété 6.5).

10.4.2 Sensibilité de la méthode GIG au réglage des paramètres (*)

Les résultats des expériences présentées au paragraphe 10.4.1 permettent de vérifier que la méthode GIG est capable d'inférer des langages réguliers sans nécessiter un réglage fin de ses paramètres internes pour chacun des problèmes envisagés. Nous considérons qu'il s'agit d'un avantage de cette méthode par comparaison avec d'autres travaux, par exemple ceux de Wyard [Wyard94]. Cependant, une étude plus approfondie de la sensibilité de la méthode GIG au réglage des différents paramètres internes mérite d'être effectuée.

La corrélation éventuelle entre les différents paramètres internes rend

[7] A titre indicatif, le temps de traitement d'une paire d'échantillons est en moyenne de 0,003 seconde avec l'algorithme RPNI et de l'ordre de la seconde avec l'algorithme GIG. Les temps de calcul obtenus avec BRIG dépendent du réglage du faisceau. Pour les résultats présentés, ils sont également de l'ordre de la seconde par paire d'échantillons.

[8] En réalité, afin de comparer des méthodes utilisant le même espace de recherche nous avons appliqué l'algorithme GIG également dans $Lat(PTA(I_+))$ et non dans $Lat(MCA(I_+))$. Mais, il ne s'agit pas ici d'une limitation intrinsèque à l'algorithme GIG.

g_1	g_2	g_3
84,0	85,4	81,8

TAB. 10.8: Influence de la fonction d'évaluation.

cette étude délicate. A titre d'exemple, la diminution du taux de mutation implique que moins d'individus nouveaux soient introduits à chaque génération. Comme la recherche génétique est arrêtée après un nombre fixé d'évaluations de solutions distinctes, le nombre de générations calculées sera naturellement augmenté lorsque l'on diminue le taux de mutation. Par conséquent, les performances obtenues ne seront pas nécessairement affectées par la modification du taux de mutation. Par ailleurs, la procédure d'inférence semi-incrémentale ne s'arrête que lorsque toutes les données ont été traitées ou lorsque toutes les données restant à traiter sont déjà compatibles avec la solution trouvée. Une solution non compatible avec les données restantes deviendra nécessairement compatible par la suite. Par conséquent, la modification d'un paramètre interne peut modifier le nombre d'étapes de la procédure semi-incrémentale sans modifier la qualité de la solution obtenue à la dernière étape.

Afin de limiter les inconvénients évoqués, nous évaluons la sensibilité de l'algorithme GIG sans avoir recours à la procédure semi-incrémentale. De plus, nous ne modifions qu'un paramètre par essai. Sauf mention contraire, les autres paramètres sont maintenus à leur valeur de référence. La mesure de qualité est le taux de classification moyen sur l'ensemble des langages et des échantillons. Rappelons que, sur notre base d'évaluation, il est de 85,4 % pour l'algorithme GIG sans procédure semi-incrémentale (voir table 10.7).

Différentes fonctions d'évaluations ont été présentées au paragraphe 8.3.4. La table 10.8 montre que les meilleurs résultats sont obtenus avec la deuxième définition de la fonction d'évaluation. Nous conservons cette définition pour les tests ultérieurs.

Les valeurs de référence des paramètres internes de la méthode GIG sont les suivantes :

- Taille de la population = 100
- Nombre maximal d'évaluations = 2000
- Taux de croisement $= 0,2$
- Taux de mutation/bit $= 0,01$
- Taux de descente $= 0,5$

La figure 10.1 présente l'évolution des performances lorsque la taille de la population est modifiée. Les meilleurs résultats sont obtenus avec une population de 20 individus. Une faible décroissance des performances est observée lorsque la taille de la population augmente. Insistons sur la corrélation probable de ce paramètre avec le nombre maximal d'évaluations. En effet, lorsque nous augmentons la taille de la population en maintenant fixe le nombre maximal d'évaluations, la recherche génétique a tendance à s'arrêter après un nombre plus limité de générations.

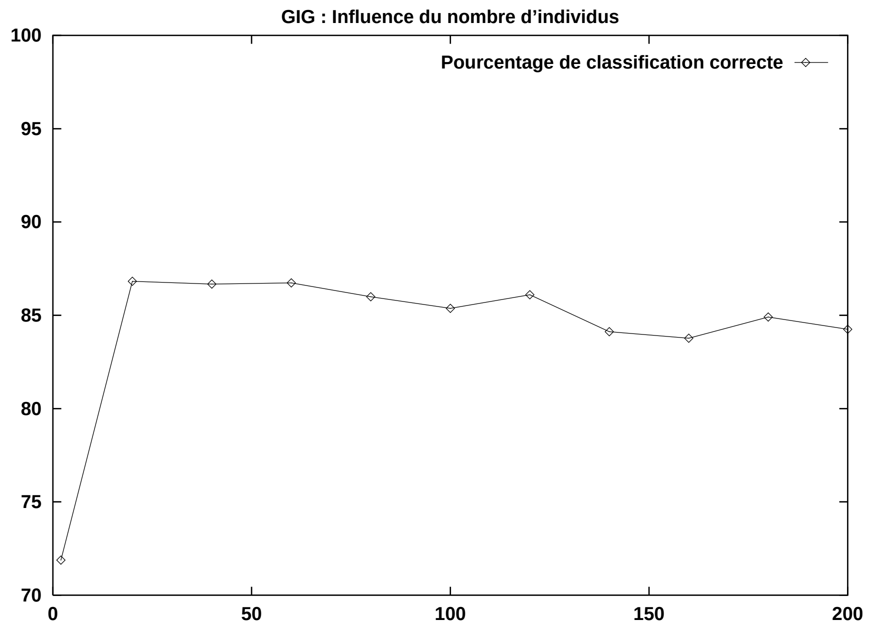
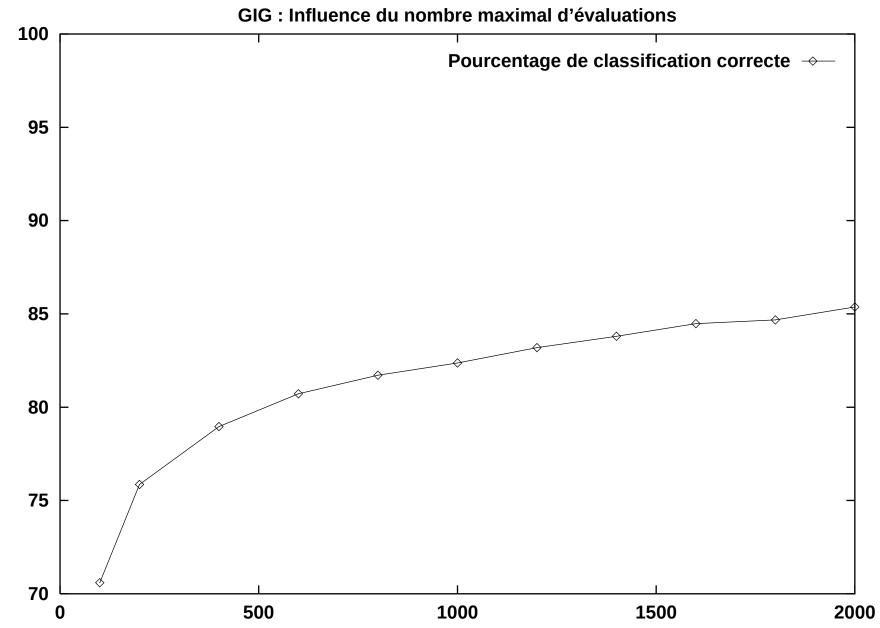
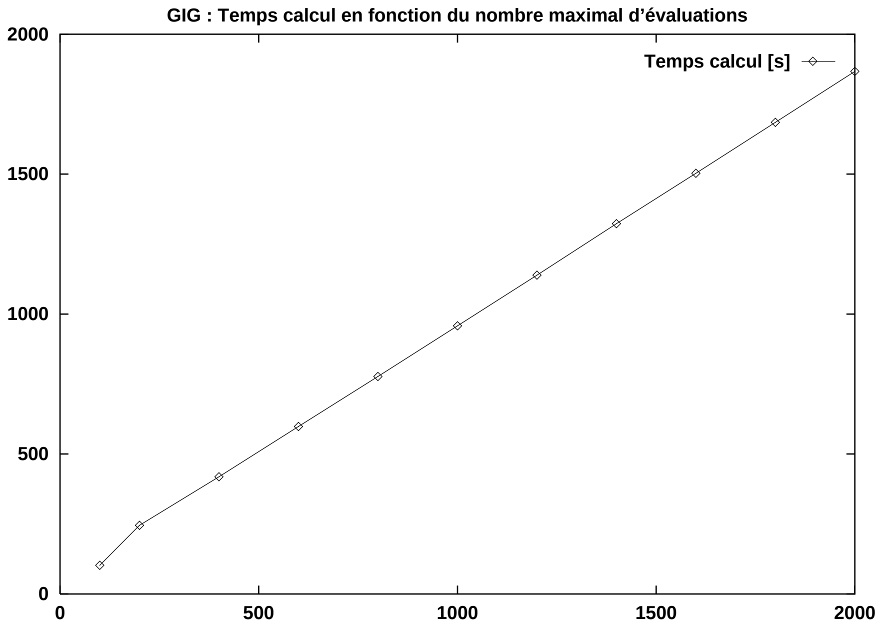


FIG. 10.1: Performance de GIG en fonction de la taille de la population.

La figure 10.2 (a) présente l'évolution des performances lorsque le nombre maximal d'évaluations est modifié. Nous observons une croissance monotone des résultats qui montre qu'il serait intéressant d'augmenter la valeur de ce paramètre par rapport à sa valeur de référence. Remarquons cependant que ce paramètre conditionne directement le temps d'exécution de l'algorithme GIG puisqu'il constitue le principal critère d'arrêt de la recherche génétique. La figure 10.2 (b) illustre cette propriété.



(a)



(b)

FIG. 10.2: Performance et temps calcul de GIG en fonction du nombre maximal d'évaluations.

Nous constatons que le temps[9] d'exécution dépend linéairement du nombre maximal d'évaluations. Le réglage fin de ce paramètre permet dès lors de modifier le compromis temps-performance de l'algorithme GIG.

La figure 10.3 présente l'évolution des performances lorsque le taux de croisement est modifié. Afin d'éviter la corrélation possible entre mutation et croisement, nous avons effectué ces expériences à taux de mutation nul. Dans ce cas, seul le croisement permet d'introduire de nouvelles solutions au cours de la recherche génétique. Nous constatons que l'opération de croisement apporte un gain de performance mais que la valeur effective du taux de croisement n'est pas critique. De plus, le croisement seul ne permet pas d'obtenir des performances proches des valeurs de référence. Cette observation n'est pas surprenante car l'opération de croisement a pour effet de brasser la population sans favoriser fortement la descente dans le treillis.

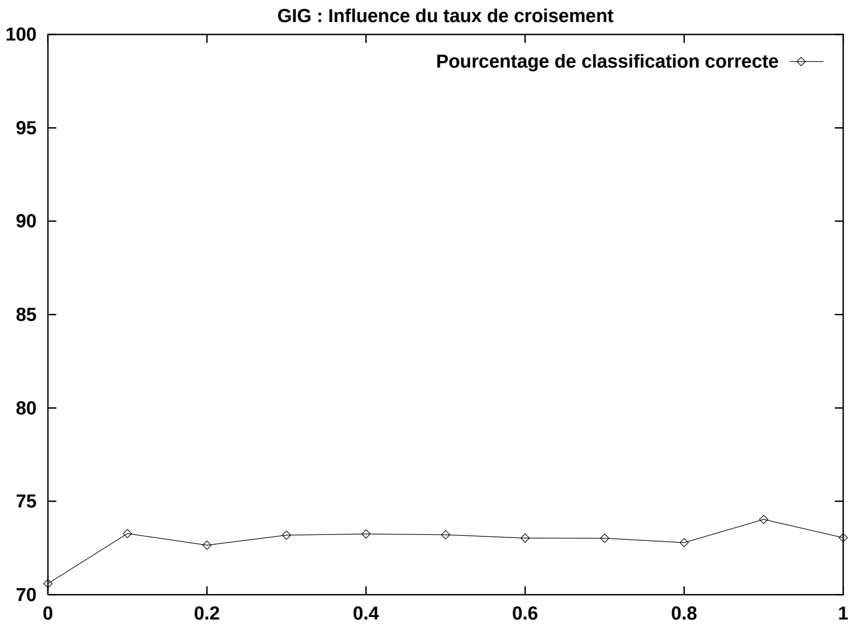


FIG. 10.3: Performance de GIG en fonction du taux de croisement.

La figure 10.4 présente l'évolution des performances lorsque le taux de mutation est modifié. Afin d'éviter la corrélation possible entre mutation et croisement, nous avons effectué ces expériences à taux de croisement nul. Nous constatons que les résultats se dégradent lorsque le taux de mutation

[9] Le temps calcul est mesuré en secondes pour le traitement de l'ensemble de la base d'évaluation. Rappelons que nous considérons 10 exécutions indépendantes par jeu de données d'apprentissage. Par conséquent, le temps total représente 1500 inférences.

par bit est inférieur à 0,003 alors qu'ils restent relativement stables pour des valeurs supérieures. La mutation combinée avec un taux de descente non nul permet d'obtenir des performances proches des valeurs de référence.

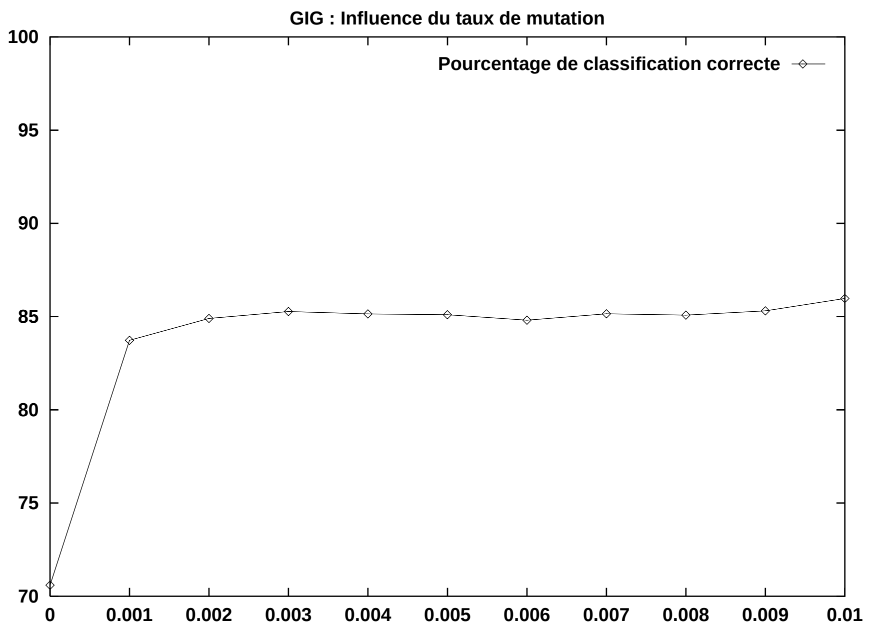


FIG. 10.4: Performance de GIG en fonction du taux de mutation par bit.

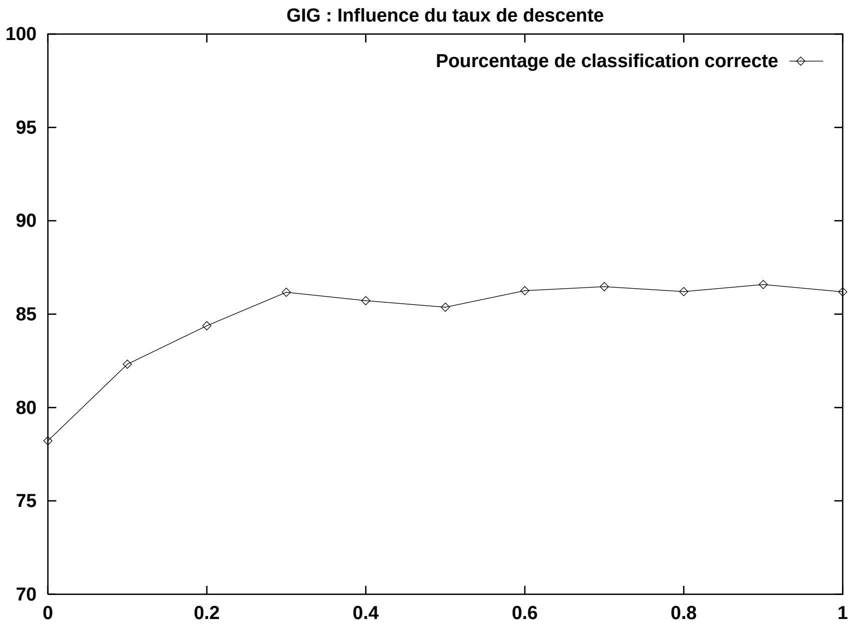


FIG. 10.5: Performance de GIG en fonction du taux de descente.

La figure 10.5 présente l'évolution des performances lorsque le taux de descente est modifié. Nous constatons que les résultats se dégradent sensiblement lorsque le taux de descente est inférieur à 0,3 alors qu'ils restent relativement stables pour des valeurs supérieures.

En résumé, l'algorithme GIG repose essentiellement sur l'opération de mutation avec un taux de descente dont la valeur n'est pas critique pour autant qu'elle ne soit pas inférieure à 0,3. L'augmentation du nombre maximal d'évaluations permet d'améliorer les performances au prix d'un temps calculatoire accru.

10.4.3 Apport de l'inférence semi-incrémentale (*)

Les résultats présentés au paragraphe 10.4.1 montrent clairement l'avantage de recourir à une procédure d'inférence semi-incrémentale (voir paragraphe 7.3). Nous illustrons à présent cette propriété sur un exemple particulier. Les échantillons d'apprentissage sont ici :

$$
\begin{aligned}
I_+ = \{ \; & aa, bb, abab, baab, bbbaba, abaabbba, aaabbbbbabbb, \\
& bbbaabaabaabbabbabaa, \\
& aaaababbbbbabaaabbaaaaaaabaabbaaabbaabbaaa\} \\
I_- = \{ \; & a, b, ab, ba, aaa, aab, aba, abb, bab, bba, bbb, aaab, abbb, \\
& aaaaa, aabbb, baaba, bbaaa, bbaab, baaabb, aabaabb, abbbbba, \\
& baabbab, aabbabaa, bbbbabaa, bababaabb, bbabaabab, aabbbbbabbb, \\
& bbbaabaababbabbabaa, \\
& aaaababbbbbabaaabbbaaaaaaabaabbaaabbaabbaaa\}
\end{aligned}
$$

La figure 10.6 présente l'automate cible et les solutions obtenues par application de l'algorithme GIG en bornant le nombre maximal d'évaluations respectivement à 2000 et 20000. Dans ce cas particulier, il est évident que les solutions trouvées sont loin de correspondre à l'automate cible.

La figure 10.7 illustre l'exécution semi-incrémentale de l'algorithme GIG. La partie gauche de la figure représente l'automate initial et la partie droite l'automate solution à chaque étape. Nous obtenons l'identification exacte de l'automate cible en 5 étapes, correspondant dans ce cas particulier aux 5 premiers éléments de I_+. Cet exemple illustre donc que l'échantillon positif peut être automatiquement filtré grâce aux généralisations des étapes antérieures.

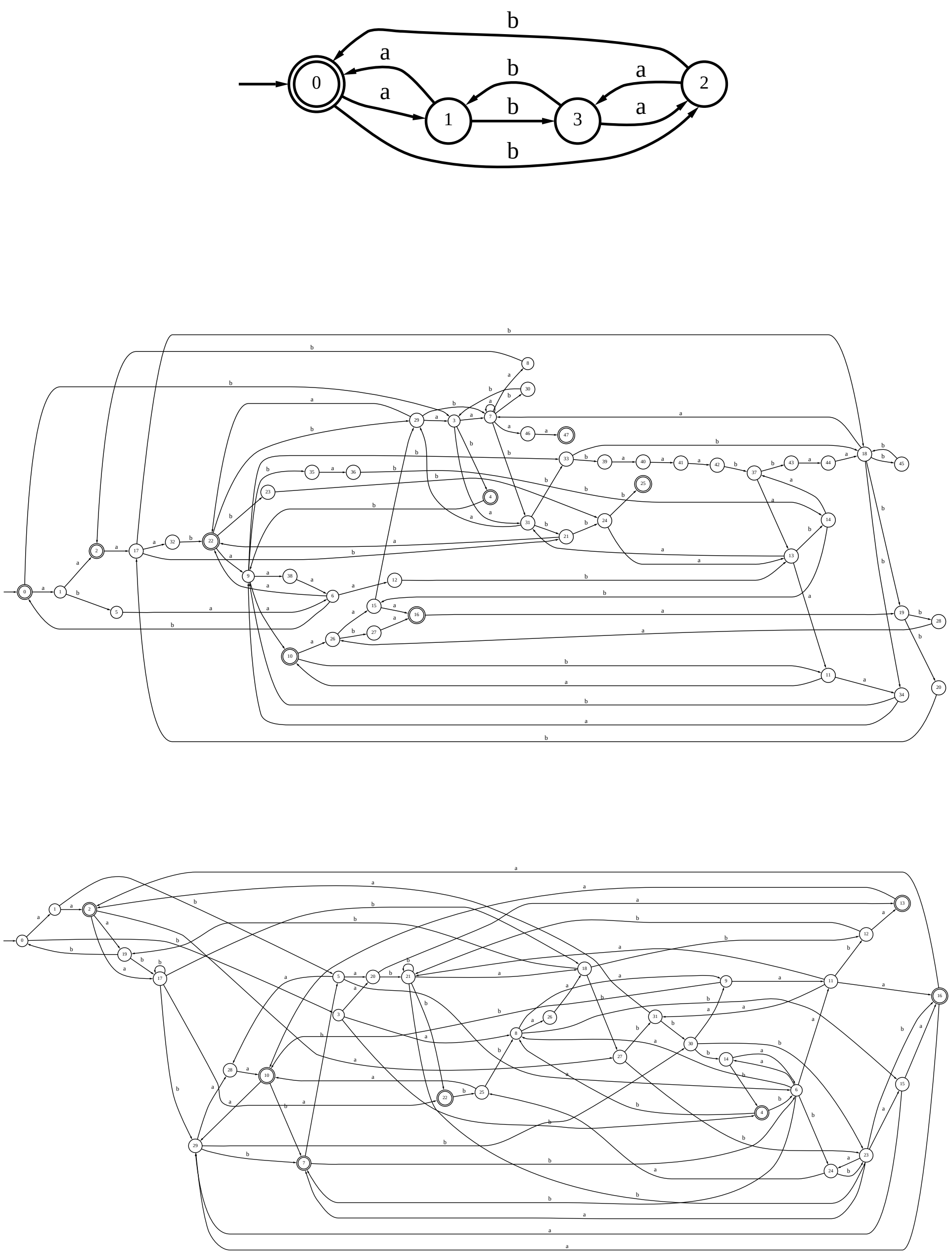


FIG. 10.6: L'automate cible et les deux solutions proposées par l'algorithme GIG.

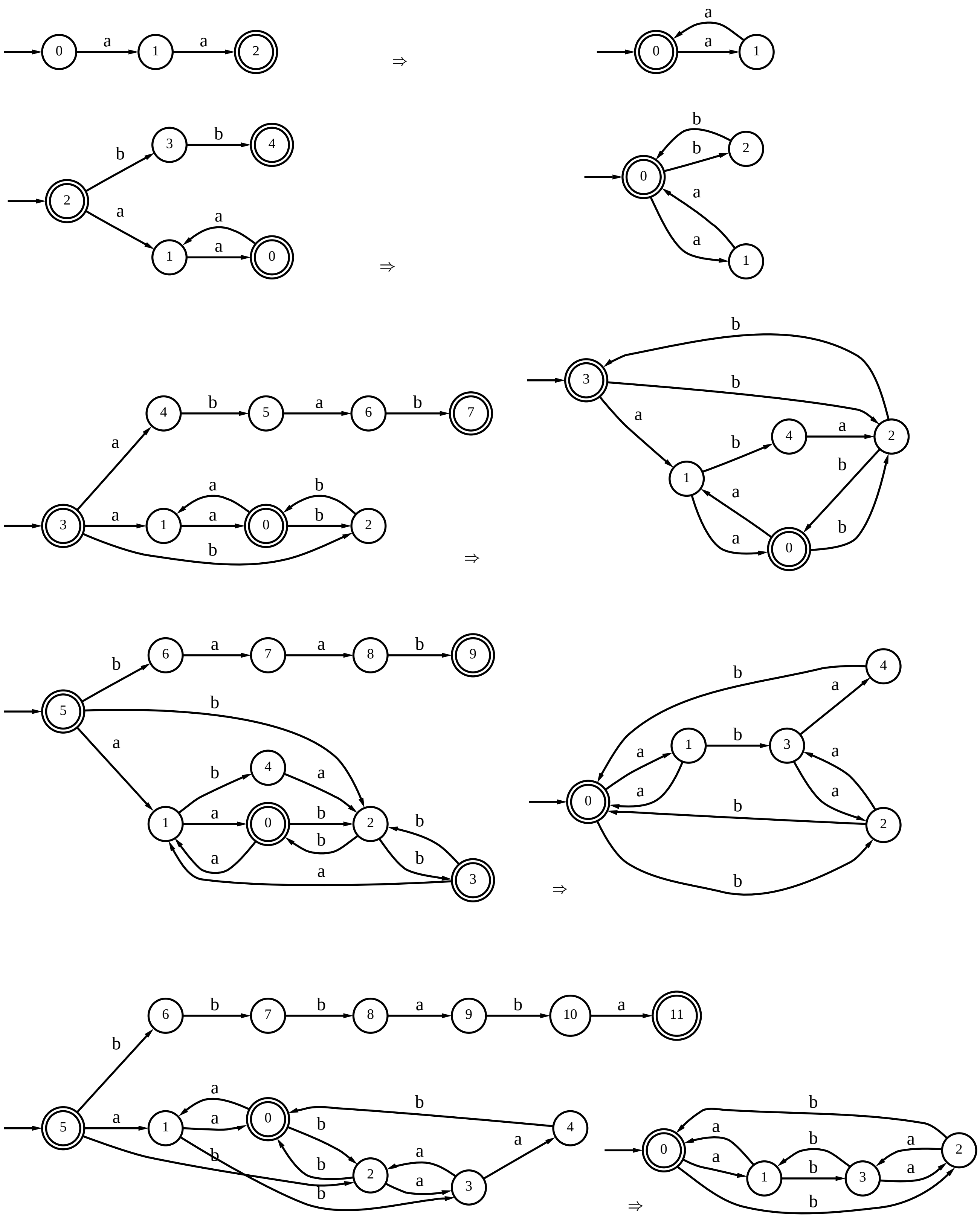


FIG. 10.7: Exécution semi-incrémentale de l'algorithme GIG.

10.5 Evaluations sur la tâche de Feldman (*)

Les expériences menées jusqu'ici se sont appuyées sur le protocole expérimental décrit au paragraphe 10.2 et sur la base d'évaluation détaillée au paragraphe 10.3. Bien que cette démarche nous semble justifiée dans la mesure où elle définit un cadre précis d'évaluation de l'inférence de langages formels, sa pertinence par rapport à des données réelles peut être critiquée. Dans ce souci, nous avons menés de nouvelles expériences sur la tâche de Feldman. Cette tâche a été décrite au paragraphe 5.1.1 où nous avons justifié son intérêt. En outre, nous avons présenté des résultats de reconnaissance de la parole continue sur cette tâche au paragraphe 5.1.2. Il devient dès lors particulièrement intéressant de mesurer la qualité d'apprentissage d'un modèle de langage sur cette tâche.

10.5.1 Adaptation du protocole expérimental (*)

Afin d'évaluer la qualité de l'apprentissage sur des données "réalistes", nous avons modifié le protocole expérimental de la manière suivante. L'échantillon positif est généré aléatoirement sans garantie de sa complétude structurelle vis-à-vis de l'automate cible. Afin d'obtenir un échantillon négatif, nous générons indépendamment et aléatoirement un nouvel échantillon positif. Chacune de ses chaînes est alors perturbée aléatoirement en permettant l'insertion, la substitution ou l'omission de symboles dans la chaîne (avec une certaine probabilité fixée a priori). Cette procédure de distorsion stochastique constitue une application de modèles de Markov cachés décrite dans [Prat95]. Les données perturbées sont testées par rapport à l'automate cible afin d'en éliminer les chaînes positives. En général, 95 % des chaînes générées et perturbées constituent des contre-exemples du langage à inférer.

Conformément à la procédure décrite, nous générons 10 paires indépendantes d'échantillons positif et négatif. La taille $|I_+|$ de chaque échantillon positif est fixée à 2.500 exemples. Chaque exemple comportant en moyenne 10 symboles, le nombre total $||I_+||$ de symboles présents est de l'ordre de 25.000 sur un alphabet de 29 symboles. La taille de l'arbre accepteur des préfixes sur l'ensemble d'un échantillon positif est de l'ordre de 10.000 états et 10.000 transitions. Elle est supérieure par un facteur 100 à la taille moyenne observée dans notre précédente base d'évaluation. Nous obtenons en moyenne 2.250 contre-exemples par échantillon négatif comportant un nombre total $||I_-||$ de symboles de l'ordre de 20.000.

Finalement, nous générons par la même procédure une nouvelle paire d'échantillons positif et négatif comportant chacun 500 chaînes. Cette paire d'échantillons est l'ensemble de test sur lequel nous calculons les taux de classification correcte des chaînes positives et négatives. Pour la tâche de Feldman la proportion de chaînes positives est négligeable par rapport au nombre de chaînes définies sur l'alphabet de 29 symboles et pour une longueur fixée. Il est donc essentiel de distinguer les taux de classification des chaînes positives et négatives (voir paragraphe 10.2).

Pour chaque paire d'échantillons positif et négatif, nous présentons les données de façon séquentielle. Plus précisément, nous tirons aléatoirement la nouvelle donnée à fournir à l'algorithme d'inférence. Il la mémorise dans l'échantillon positif ou négatif reçu jusqu'alors selon qu'il s'agisse d'un nouvel exemple ou d'un nouveau contre-exemple. Sur base de ces données, l'algorithme d'inférence propose une solution qui sera évaluée sur la base de test. Lorsque l'algorithme est incrémental, nous utilisons la solution trouvée à l'étape précédente pour construire la nouvelle solution. C'est la cas de l'algorithme RPNI2. Sinon, nous testons simplement si la solution précédente est compatible avec la nouvelle donnée et, dans le cas contraire, l'algorithme d'inférence est appliqué à partir de l'arbre des préfixes construit sur l'ensemble des exemples reçus.

Ce mode de présentation permet d'évaluer la qualité des algorithmes d'inférence non-incrémentaux en fonction du volume de données d'apprentissage. Il permet également de mesurer le gain de complexité apporté par une méthode incrémentale qui utilise la solution construite à l'étape précédente.

10.5.2 Résultats comparatifs des algorithmes GIG, RPNI et RPNI2 (*)

Une première observation concerne la comparaison des algorithmes GIG et RPNI. Lorsque nous utilisons l'algorithme GIG sur des échantillons d'apprentissage de taille croissante, les taux de classification observés sur la base de test sont similaires à ceux obtenus par l'algorithme RPNI. Cependant, dès que le nombre d'exemples dépasse plusieurs centaines, c'est-à-dire dès que l'arbre accepteur des préfixes comporte plus d'un millier d'états, la complexité calculatoire de GIG devient pénalisante. Il s'agit d'une limitation propre aux algorithmes GIG et BRIG qui explorent un espace de recherche qui explose combinatoirement avec la taille du $PTA(I_+)$. Il est certes possible de contrôler la complexité de GIG en modifiant ses paramètres internes mais cela induit une dégradation de ses performances (voir paragraphe 10.4.2). Conformément à l'ordre des fusions qui lui est propre, l'algorithme RPNI n'explore qu'une partie très réduite de l'espace de recherche ce qui limite fortement sa complexité calculatoire.

Nous nous intéressons à présent à la comparaison des algorithmes RPNI et

RPNI2. La figure 10.8 présente les taux de classification des exemples et celui des contre-exemples de la base de test. Les deux courbes sont tracées en fonction de $||I_+|| + ||I_-||$, le nombre de symboles des chaînes positives et négatives reçues au cours de la présentation séquentielle des données d'apprentissage. Ces courbes sont moyennées sur 10 paires indépendantes d'échantillons positif et négatif. Les algorithmes RPNI et RPNI2 ont des taux de classification identiques (leurs courbes de classification respectives se superposent) conformément aux tests effectués. Bien que l'identification de l'automate cible soit garantie lorsque ces algorithmes reçoivent en entrée des données contenant un échantillon caractéristique (voir chapitre 9), il n'y a pas de garantie théorique qu'ils produisent le même résultat sur un jeu de données quelconques. Nous constatons néanmoins que, sur les données utilisées, c'est bien le cas.

Nous pouvons également remarquer que le taux de classification de chaînes négatives restent toujours excellent. Cela n'est pas surprenant vu la proportion très majoritaire de chaînes négatives pour le cas étudié. La qualité de l'apprentissage est donc mesurée essentiellement par le taux de classification d'exemples générés indépendamment par rapport aux données d'apprentissage.

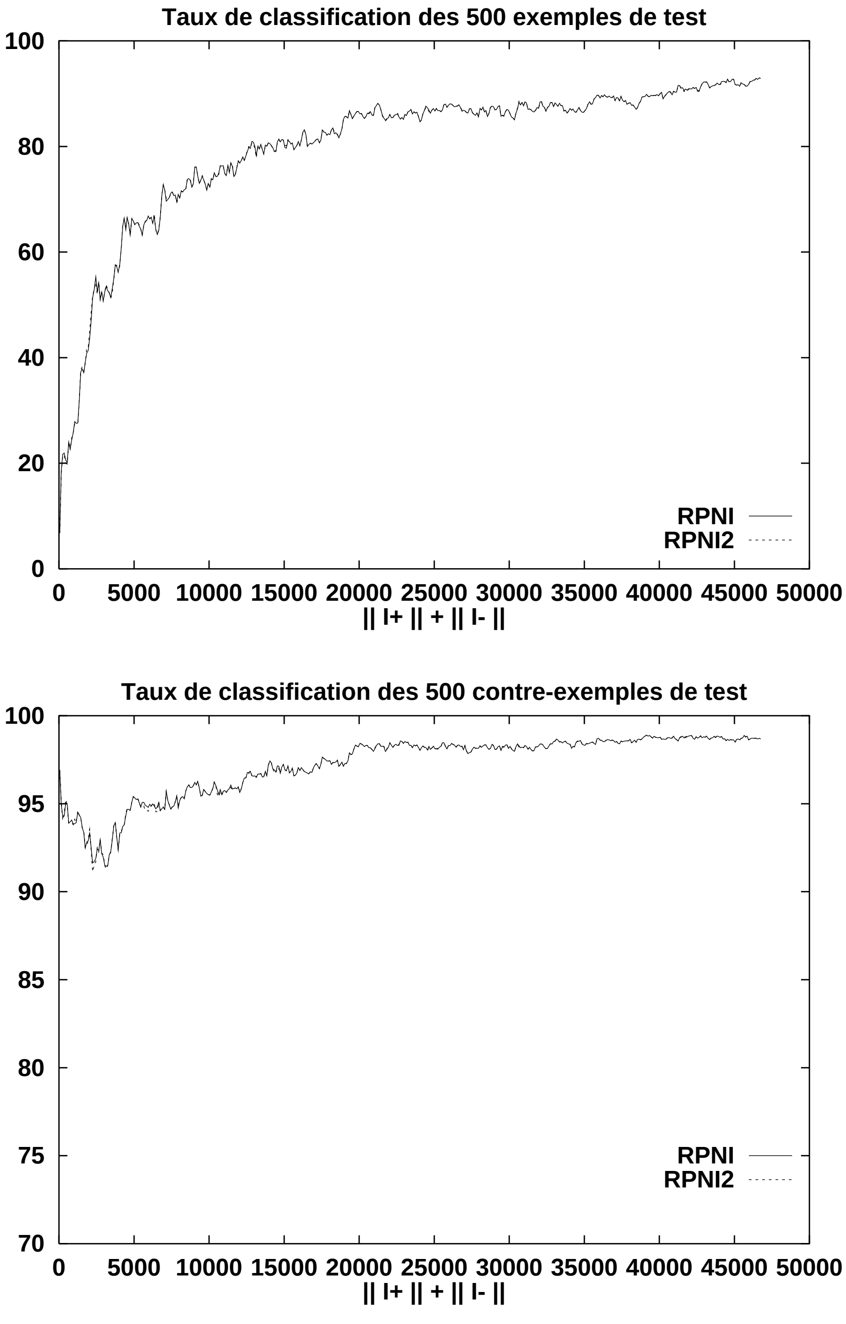


FIG. 10.8: RPNI et RPNI2 : taux de classification sur les données de test.

La figure 10.9 présente le nombre cumulé d'analyses de l'échantillon I_- effectuées sur l'ensemble de l'apprentissage incrémental. A chaque étape de la présentation séquentielle, nous mesurons le nombre d'appels à la fonction qui teste la compatibilité avec l'échantillon négatif. En effet, cette opération domine le coût observé en pratique. Finalement, nous cumulons cette mesure sur l'ensemble des données reçues et traitées séquentiellement.

Remarquons que l'algorithme RPNI2 pourrait dans le pire de cas nécessiter un nombre supérieur d'analyses de I_-, car l'appel à cette fonction s'effectue non seulement pour contrôler les fusions mais également lors des opérations de fission décrites au paragraphe 9.4. D'autre part, puisqu'il tire parti de la solution obtenue précédemment pour construire la nouvelle solution, il effectue généralement sa recherche dans un sous-espace de celui exploré par l'algorithme RPNI. Nous constatons en pratique que cet effet positif domine.

A la fin de la présentation séquentielle, l'algorithme RPNI a effectué en moyenne 10,2 % d'opérations supplémentaires par rapport à l'algorithme RPNI2. Le gain observé peut dépendre de l'ordre de présentation des exemples qui est ici complètement aléatoire. Indépendamment du gain réellement observé, la propriété intéressante est qu'il croît de façon monotone avec le volume des données d'apprentissage.

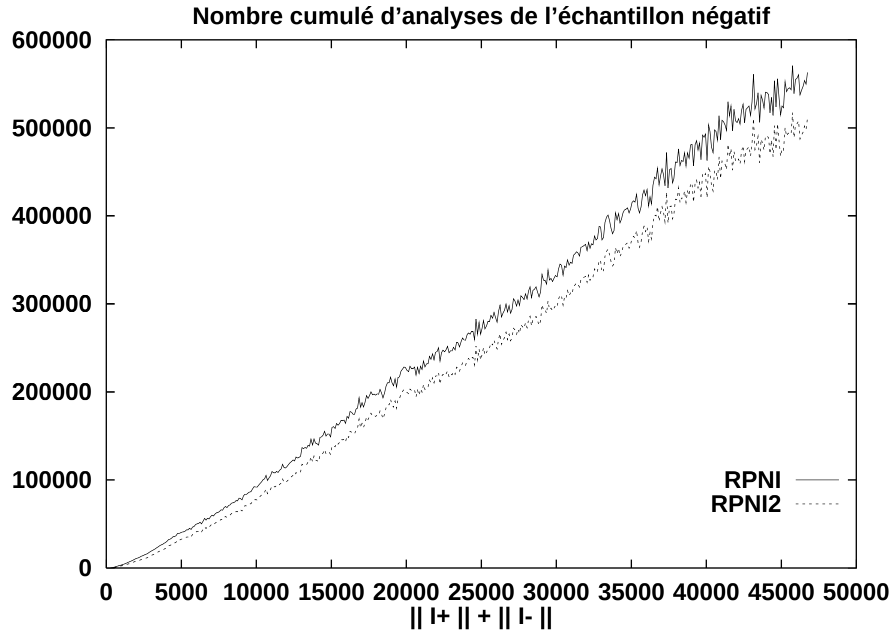


FIG. 10.9: Complexité comparative de RPNI2 et RPNI.

Chapitre 11

Conclusions et perspectives

11.1 Bilan de notre étude

La première partie originale de notre travail a concerné l'utilisation d'un modèle de langage hors-contexte pour la reconnaissance de la parole continue. Elle a montré que le coût de traitement du modèle de langage est négligeable devant celui du décodage acoustique. Ce résultat déjà vérifié pour les langages réguliers se généralise donc à une nouvelle classe de langages. Notre approche offre également des perspectives intéressantes quant à l'utilisation de contraintes fournies par l'état courant du dialogue. Elle justifie aussi l'intérêt de l'apprentissage automatique de modèles de langage reposant sur des grammaires formelles.

Dans cette optique, nous nous sommes concentrés sur l'inférence de grammaires régulières à partir d'échantillons positif et négatif. Un certain nombre de développements théoriques ont permis de définir la structure de l'espace de recherche de ce problème. Nous avons alors proposé de considérer l'inférence régulière comme un problème d'optimisation combinatoire. Un algorithme basé sur une optimisation génétique a été conçu à cet effet. Il s'est avéré performant lorsque les données d'apprentissage sont de taille limitée. Néanmoins, son coût calculatoire est pénalisant lorsque la taille des données devient importante.

Finalement, nous nous sommes intéressés à l'inférence régulière lorsque les données d'apprentissage sont reçues une à une dans un ordre quelconque. Nous avons développé une extension incrémentale d'un algorithme connu. Nous en avons démontré la convergence et étudié sa complexité théorique et pratique par rapport à celle de l'algorithme original. Des évaluations pratiques ont montré que le gain calculatoire croît de manière monotone avec la taille des données. Bien que nous nous soyons concentrés sur la conception d'un algorithme incrémental dont la convergence puisse être démontrée, de nombreuses variantes heuristiques sont envisageables. A titre illustratif, la mémorisation des données négatives reçues n'est sans doute pas indispensable. Dans la mesure où nous pouvons toujours reconstruire une solution compatible, nous pourrions introduire une notion d'oubli des contre-exemples déjà vus. Cet oubli aurait probablement pour effet de retarder la convergence. De façon similaire, comme nous pouvons toujours étendre une solution temporaire afin qu'elle accepte de nouveaux exemples, nous pourrions ne mémoriser qu'une fraction de l'arbre accepteur des préfixes. Une question intéressante est de savoir comment définir cette fraction afin de s'assurer que la solution temporaire puisse toujours être définie comme automate quotient du sous-arbre retenu.

11.2 Inférence de grammaires régulières stochastiques

Une limitation évidente des techniques d'inférence à partir d'échantillons positif et négatif d'un langage est la nécessité de disposer, en pratique, d'un échantillon négatif. Sans cette information, nous savons que nous ne pouvons identifier la classe entière des langages réguliers (voir paragraphe 6.3.5). Cependant, les travaux de Horning nous indiquent que l'identification est possible à partir d'une présentation stochastique [Horning69], où les exemples négatifs ne sont pas explicitement présents dans les données d'apprentissage.

Deux méthodes d'inférence stochastique de grammaires régulières ont été récemment proposées. La première utilise un critère bayésien pour contrôler le compromis entre adéquation du modèle aux données observées et généralisation à partir de ces données [Stolcke et al.94a]. Cette approche nous semble particulièrement intéressante dans la mesure où elle s'inscrit dans un cadre statistique classique. Elle nécessite cependant de définir une probabilité a priori des modèles, généralement en attribuant une probabilité plus élevée aux modèles plus simples. Cet inconvénient apparent peut être considéré comme un avantage dans la mesure où cela permet d'introduire une connaissance a priori sur le problème. La seconde méthode constitue une extension stochastique de l'algorithme RPNI [Carrasco et al.94]. La fusion des états du $PTA(I_+)$ est effectuée dans l'ordre standard sous la contrainte supplémentaire que les suffixes des états fusionnés aient une vraisemblance “proche”. Cette notion de proximité repose sur une mesure de distance entre les estimations des probabilités des différents suffixes. La question principale soulevée par cette méthode est la pertinence du critère optimisé. En particulier, nous savons que l'arbre des préfixes stochastique donne une estimation de vraisemblance maximale des données d'apprentissage. Quel autre critère est optimisé en fusionnant les états de cet arbre ? S'agit-il d'une mise en oeuvre d'un apprentissage bayésien ? Dans la mesure où des réponses satisfaisantes puissent être obtenues, il serait intéressant d'étudier une extension stochastique similaire de l'algorithme RPNI2.

Une extension aisée de l'algorithme GIG devrait permettre d'inférer des grammaires régulières stochastiques. Il suffit, en principe, de modifier la fonction d'évaluation de chaque solution en remplaçant la mesure de l'acceptation de chaînes négatives par une mesure de la perplexité sur un échantillon positif indépendant. Nous recherchons alors la solution de perplexité minimale sur cet échantillon indépendant. De nombreuses questions restent cependant ouvertes. A quel critère statistique, cette estimation correspond-elle ? Comment traiter le cas où la perplexité devient infinie si un exemple du nouvel échantillon n'est pas accepté par la solution proposée ? Existe-t-il de nouvelles propriétés de l'espace de recherche qui permettent de guider l'inférence ?

11.3 Inférence de grammaires hors-contexte

L'extension des techniques d'inférence à la classe entière des grammaires hors-contexte n'est pas encore résolue. Un certain nombre de techniques d'identification de sous-classes existent [Takada88, Radhakrishnan et al.88]. Giordano étudie la structure de l'espace de recherche de l'inférence hors-contexte et propose une méthode d'inférence par énumération [Giordano94, Giordano95]. Afin de limiter la complexité calculatoire de l'inférence, une identification approchée est recherchée. Par ailleurs, l'inférence à partir d'échantillons positifs structurés, c'est-à-dire munis de parenthèses marquant par exemple les syntagmes, est possible. En particulier, Sakakibara a démontré que ce problème se réduit à l'inférence de grammaires régulières d'arbres et a proposé un algorithme d'inférence [Sakakibara92]. Dans le même esprit, García et Oncina ont démontré que n'importe quelle grammaire hors-contexte peut être efficacement inférée à partir d'un échantillon positif structuré et un échantillon négatif [García et al.93].

En ce qui concerne les grammaires hors-contexte stochastiques, une technique d'inférence par recherche de plus grande pente a été proposée par Cook et Rozenfeld [Cook et al.74]. Dans ce travail, l'inférence est également vue comme un problème d'optimisation explicite. Finalement, l'estimation localement optimale d'une grammaire hors-contexte stochastique est possible par le biais de l'algorithme "Inside-Outside", proposé originellement par Baker [Baker79] et reformulé par Lari et Young [Lari et al.90]. L'utilisation d'échantillons structurés, introduisant des contraintes supplémentaires, a été présentée dans [Pereira et al.92] et d'autres améliorations ont été récemment proposées par Young [Young et al.94].

Annexe A

Grammaires de la tâche de Feldman

Nous présentons ici les différentes grammaires utilisées pour la tâche de Feldman. A ce propos, nous utilisons la notation abrégée $A \rightarrow (\alpha_1 \mid \ldots \mid \alpha_n)$ pour un ensemble de productions $\{A \rightarrow \alpha_1, \ldots, A \rightarrow \alpha_n\}$.

A.1 Grammaire régulière française (*)

S	$\longrightarrow$ (NPS | NPP | NPS VPS | NPP VPP)
S	$\longrightarrow$ (PPA NPS | PPB NPSB | PPA NPS VP)
NPS	$\longrightarrow$ DET NP1
NPP	$\longrightarrow$ DET NP1 AND DET NP1
NPPR	$\longrightarrow$ DET NP1 AND DET4 NP1
VPS	$\longrightarrow$ (VTS OD | VIS OI)
VPP	$\longrightarrow$ (VTP OD | VIP OI)
VP	$\longrightarrow$ (RELB NPS2 | RELB NPP2)
NPSB	$\longrightarrow$ (NPS0 | NPS0 RELATIVE)
NPS0	$\longrightarrow$ DET2 NP1
RELATIVE	$\longrightarrow$ WHICH VIS VP
NP1	$\longrightarrow$ (OBJ | OBJ SHADE | OBJ SIZE2)
NP1	$\longrightarrow$ (OBJ SIZE2 AND SHADE | SIZE OBJ | SIZE OBJ SHADE)
REL	$\longrightarrow$ (REL1 | REL2 | FAR1 REL1 | FAR1 REL2)
RELB	$\longrightarrow$ (REL1B | REL2B | FAR1 REL1B | FAR1 REL2B)
NPS2	$\longrightarrow$ DET3 NP1
NPP2	$\longrightarrow$ DET3 NP1 AND DET3 NP1
OI	$\longrightarrow$ (REL NPS | REL NPPR)
OD	$\longrightarrow$ (NPS | NPP)
PPA	$\longrightarrow$ ajouter

PPB	$\longrightarrow$ éliminer
VIS	$\longrightarrow$ est
VIP	$\longrightarrow$ sont
AND	$\longrightarrow$ et
DET	$\longrightarrow$ un
DET2	$\longrightarrow$ le
VTS	$\longrightarrow$ touche
VTP	$\longrightarrow$ touchent
OBJ	$\longrightarrow$ (triangle | carré | cercle)
SHADE	$\longrightarrow$ (clair | sombre)
SIZE	$\longrightarrow$ (petit | grand)
SIZE2	$\longrightarrow$ de taille moyenne
FAR1	$\longrightarrow$ très
REL1	$\longrightarrow$ (en dessous d' | au dessus d')
REL2	$\longrightarrow$ (à gauche d' | à droite d')
REL1B	$\longrightarrow$ (en dessous | au dessus)
REL2B	$\longrightarrow$ (à gauche | à droite)
DET3	$\longrightarrow$ du
DET4	$\longrightarrow$ d' un
WHICH	$\longrightarrow$ qui

A.2 Grammaire régulière espagnole

S	$\longrightarrow$ (NPS | NPP | NPS VPS | NPP VPP)
S	$\longrightarrow$ (VR PPA NPSA | VR PPB NPSB | VR PPA NPSA VP)
NPSB	$\longrightarrow$ (NPS0 | NPS0 RELATIVE)
NPSA	$\longrightarrow$ DET NP1
VP	$\longrightarrow$ (RELB NPS2 | RELB NPP2)
NPP	$\longrightarrow$ DET NP1 AND DET NP1
VPP	$\longrightarrow$ (VTP OD | VIP OI)
NPS	$\longrightarrow$ DET NP1
NPS0	$\longrightarrow$ DET2 NP1
VPS	$\longrightarrow$ (VTS OD | VIS OI)
NP1	$\longrightarrow$ (OBJ | OBJ SHADE | OBJ SIZE | OBJ SIZE AND SHADE)
OD	$\longrightarrow$ (NPS | NPP)
OI	$\longrightarrow$ (REL NPS | REL NPP)
REL	$\longrightarrow$ (REL1 | REL2 | FAR1 REL1 | FAR2 REL2)
RELB	$\longrightarrow$ (REL1B | REL2B | FAR1 REL1B | FAR2 REL2B)
NPS2	$\longrightarrow$ DET3 NP1
NPP2	$\longrightarrow$ DET3 NP1 AND DET3 NP1
RELATIVE	$\longrightarrow$ WHICH VIS VP
VR	$\longrightarrow$ se

PPA ⟶ añade
PPB ⟶ elimina
VIS ⟶ esta
VIP ⟶ estan
AND ⟶ y
DET ⟶ un
DET2 ⟶ el
VTS ⟶ toca a
VTP ⟶ tocan a
OBJ ⟶ (triangulo | cuadrado | circulo)
SHADE ⟶ (claro | oscuro)
SIZE ⟶ (pequeño | mediano | grande)
FAR1 ⟶ muy por
REL1 ⟶ (encima de | debajo de)
REL1B ⟶ (encima | debajo)
FAR2 ⟶ muy
REL2 ⟶ (a la izquierda de | a la derecha de)
REL2B ⟶ (a la izquierda | a la derecha)
DET3 ⟶ del
WHICH ⟶ que

A.3 Grammaire hors-contexte française (*)

Les règles précédées d'un commentaire ont été ajoutées par rapport à celles de la grammaire régulière du paragraphe A.1.

S ⟶ (NPS | NPP | NPS VPS | NPP VPP)
S ⟶ (PPA NPS | PPB NPSB | PPA NPS VP)

// Inversion : ajouter à gauche . . . un triangle
S ⟶ PPA VP NPS

NPS ⟶ DET NP1R
NPP ⟶ DET NP1R AND DET NP1R

// Relatives plurielles : . . . qui sont . . .
NPP ⟶ DET NP1 AND DET NP1 RELATIVEP

NPPR ⟶ DET NP1R AND DET4 NP1R
VPS ⟶ (VTS OD | VIS OI)
VPP ⟶ (VTP OD | VIP OI)
VP ⟶ (RELB NPS2 | RELB NPP2)
NPSB ⟶ (NPS0 | NPS0 RELATIVE)

```
NPS0        —→ DET2 NP1R
RELATIVE    —→ WHICH VIS VP

// Relative : qui touche ...
RELATIVE    —→ WHICH VTS OD
// Relative : qui sont ...
RELATIVEP   —→ WHICH VIP VP
// Relative : qui touchent ...
RELATIVEP   —→ WHICH VTP OD

NP1         —→ (OBJ | OBJ SHADE | OBJ SIZE2)
NP1         —→ (OBJ SIZE2 AND SHADE | SIZE OBJ | SIZE OBJ SHADE)

// Relative liée à un nom : un triangle, qui ..., ...
NP1R        —→ (NP1 | NP1R RELATIVE | NP1R VP)

REL         —→ (REL1 | REL2 | FAR1 REL1 | FAR1 REL2)
RELB        —→ (REL1B | REL2B | FAR1 REL1B | FAR1 REL2B)
NPS2        —→ DET3 NP1R
NPP2        —→ DET3 NP1R AND DET3 NP1R
OI          —→ (REL NPS | REL NPPR)
OD          —→ (NPS | NPP)
PPA         —→ ajouter
PPB         —→ éliminer
VIS         —→ est
VIP         —→ sont
AND         —→ et
DET         —→ un
DET2        —→ le
VTS         —→ touche
VTP         —→ touchent
OBJ         —→ (triangle | carré | cercle)
SHADE       —→ (clair | sombre)
SIZE        —→ (petit | grand)
SIZE2       —→ de taille moyenne
FAR1        —→ très
REL1        —→ (en dessous d' | au dessus d')
REL2        —→ (à gauche d' | à droite d')
REL1B       —→ (en dessous | au dessus)
REL2B       —→ (à gauche | à droite)
DET3        —→ du
DET4        —→ d' un
WHICH       —→ qui
```

A.4 Grammaire hors-contexte espagnole (*)

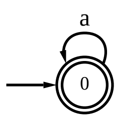
S	$\longrightarrow$	(NPS | NPP | NPS VPS | NPP VPP)
S	$\longrightarrow$	(VR PPA NPS | VR PPB NPSB | VR PPA NPS VP)
S	$\longrightarrow$	VR PPA VP NPS
NPSB	$\longrightarrow$	(NPS0 | NPS0 RELATIVE)
NPS	$\longrightarrow$	DET NP1R
VP	$\longrightarrow$	(RELB NPS2 | RELB NPP2)
NPP	$\longrightarrow$	DET NP1R AND DET NP1R
NPP	$\longrightarrow$	DET NP1 AND DET NP1 RELATIVEP
VPP	$\longrightarrow$	(VTP OD | VIP OI)
NPS0	$\longrightarrow$	DET2 NP1R
VPS	$\longrightarrow$	(VTS OD | VIS OI)
NP1	$\longrightarrow$	(OBJ | OBJ SHADE | OBJ SIZE | OBJ SIZE AND SHADE)
NP1R	$\longrightarrow$	NP1
NP1R	$\longrightarrow$	NP1R RELATIVE
NP1R	$\longrightarrow$	NP1R VP
OD	$\longrightarrow$	(NPS | NPP)
OI	$\longrightarrow$	(REL NPS | REL NPP)
REL	$\longrightarrow$	(REL1 | REL2 | FAR1 REL1 | FAR2 REL2)
RELB	$\longrightarrow$	(REL1B | REL2B | FAR1 REL1B | FAR2 REL2B)
NPS2	$\longrightarrow$	DET3 NP1R
NPP2	$\longrightarrow$	DET3 NP1R AND DET3 NP1R
RELATIVE	$\longrightarrow$	WHICH VIS VP
RELATIVE	$\longrightarrow$	WHICH VTS OD
RELATIVEP	$\longrightarrow$	WHICH VIP VP
RELATIVEP	$\longrightarrow$	WHICH VTP OD
VR	$\longrightarrow$	se
PPA	$\longrightarrow$	añade
PPB	$\longrightarrow$	elimina
VIS	$\longrightarrow$	esta
VIP	$\longrightarrow$	estan
AND	$\longrightarrow$	y
DET	$\longrightarrow$	un
DET2	$\longrightarrow$	el
VTS	$\longrightarrow$	toca a
VTP	$\longrightarrow$	tocan a
OBJ	$\longrightarrow$	(triangulo | cuadrado | circulo)
SHADE	$\longrightarrow$	(claro | oscuro)
SIZE	$\longrightarrow$	(pequeño | mediano | grande)

FAR1	$\longrightarrow$ muy por
REL1	$\longrightarrow$ (encima de | debajo de)
REL1B	$\longrightarrow$ (encima | debajo)
FAR2	$\longrightarrow$ muy
REL2	$\longrightarrow$ (a la izquierda de | a la derecha de)
REL2B	$\longrightarrow$ (a la izquierda | a la derecha)
DET3	$\longrightarrow$ del
WHICH	$\longrightarrow$ que

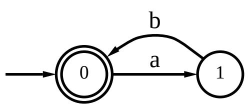
Annexe B

Automates de la base d'évaluation

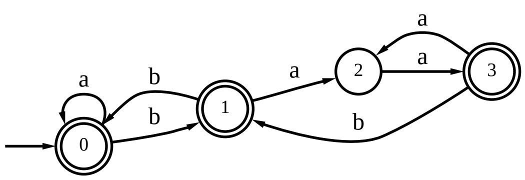
$L_1 : a^*$



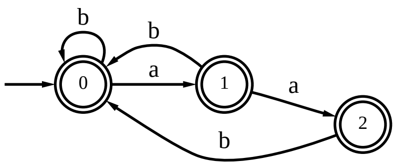
$L_2 : (ab)^*$



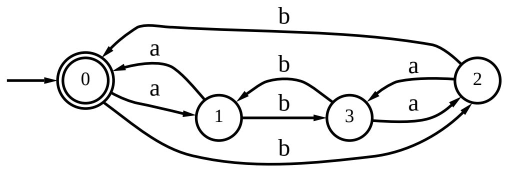
L_3 : toute chaîne définie sur l'alphabet $\{a, b\}$ sans un nombre impair de a consécutifs après un nombre impair de b consécutifs.



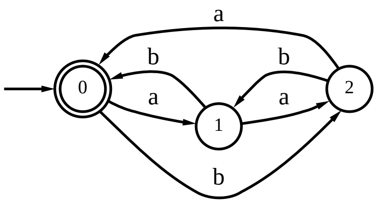
L_4 : toute chaîne définie sur l'alphabet $\{a, b\}$ sans plus de deux a consécutifs.



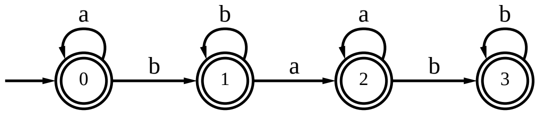
L_5 : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a et un nombre pair de b.



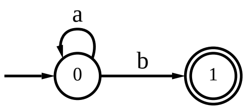
L_6 : toute chaîne définie sur l'alphabet $\{a, b\}$ telle que la différence entre le nombre de a et celui de b est divisible par 3.



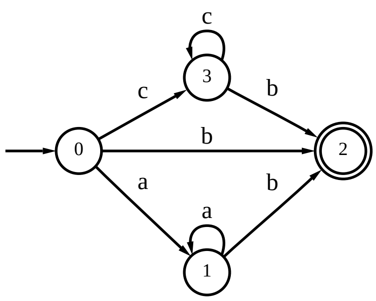
L_7 : $a^*b^*a^*b^*$



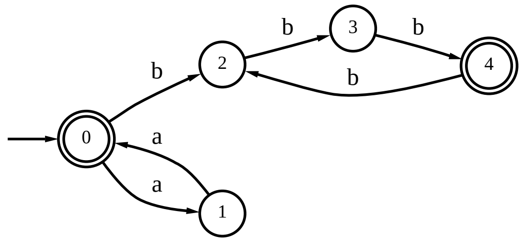
L_8 : a^*b



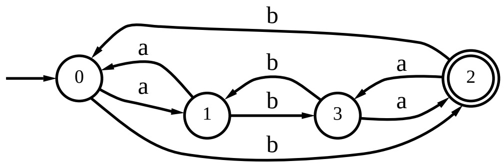
L_9 : $(a^* + c^*)b$



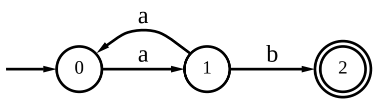
L_{10} : $(aa)^*(bbb)^*$



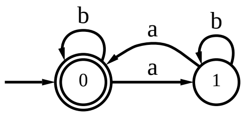
L_{11} : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a et un nombre impair de b.



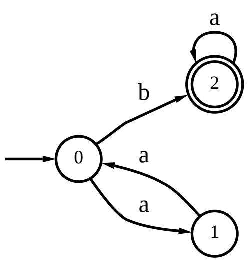
L_{12} : $a(aa)^*b$



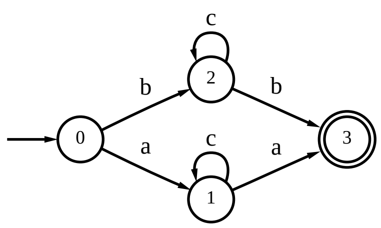
L_{13} : toute chaîne définie sur l'alphabet $\{a, b\}$ comportant un nombre pair de a.



L_{14} : $(aa)^*ba^*$



L_{15} : $bc^*b + ac^*a$



Annexe C

La grammaire du cidre

L'algorithme ECGI a été appliqué sur base de l'échantillon positif suivant.

Una botella de sidra.
(Une bouteille de cidre).
Una botella de sidra vieja.
(Une bouteille de vieux cidre).
Una botella de agua de vida vieja.
(Une bouteille d'eau de vieille vie).
Una botella de agua vieja de sidra.
(Une bouteille de vieille eau de cidre).
Una botella de agua antigua de sidra vieja.
(Une bouteille d'ancienne eau de vieux cidre).
Una botella de agua de vida de sidra antigua.
(Une bouteille d'eau de vie de cidre ancien).
Una botella de agua de vida antigua de sidra vieja.
(Une bouteille d'ancienne eau de vie de vieux cidre).
Una botella de vida corta de agua con mucha sidra.
(Une bouteille de vie coupée d'eau avec beaucoup de cidre).
Una botella de agua corta de vida con sidra antigua.
(Une bouteille d'eau coupée de vie avec du cidre ancien).
Una botella de vida vieja corta de agua antigua con mucha sidra.
(Une bouteille de vieille vie coupée d'ancienne eau avec beaucoup de cidre).
Una botella de vida corta de agua antigua con mucha sidra vieja.
(Une bouteille de vie coupée d'ancienne eau avec beaucoup de vieux cidre).
Una botella de calva, jovencita.
(Une bouteille de calva, jeunette).
Una vieja botella de sidra antigua sin agua y llena de vida joven.

(Une vieille bouteille d'ancien cidre sans eau et pleine de jeune vie).

Rappelons que l'algorithme ECGI permet d'inférer des grammaires sans cycles et offre des généralisations construites à partir de sous-chaînes de l'ensemble d'apprentissage. En vertu de la propriété 7.1 toutes les transitions pointant vers une bouteille de cidre sont nécessairement étiquetées par le même symbole. Dès lors, une représentation équivalente consiste à associer les symboles aux états. La figure C.1 présente l'automate inféré à partir de l'échantillon positif décrit.

Parmi les généralisations obtenues citons à titre d'exemple :

Una vieja botella de agua de vida de calva, jovencita.

(Une vieille bouteille d'eau de vie de calva, jeunette).

Una vieja botella de sidra antigua corta de agua de calva, jovencita.

(Une vieille bouteille de cidre ancien coupé d'eau de calva, jeunette).

Una botella de sidra antigua sin agua y llena de sidra vieja.

(Une bouteille de cidre ancien sans eau et pleine de vieux cidre).

L'auteur remercie instamment Enrique Vidal pour sa contribution à cette application d'une technique d'inférence éprouvée à partir de données réelles.

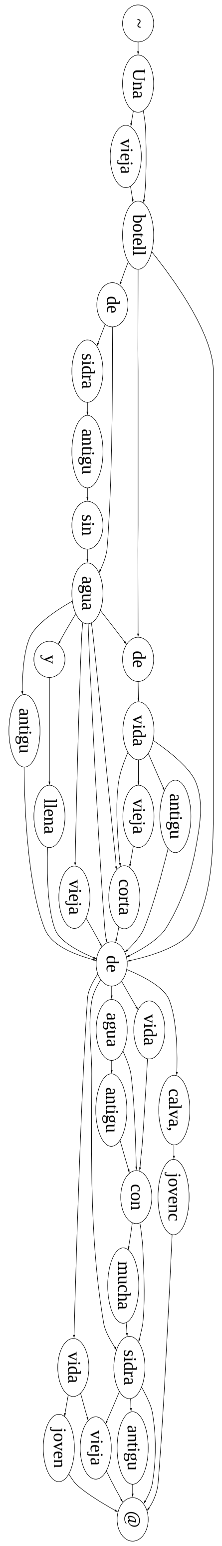


FIG. C.1: La grammaire du cidre.

Bibliographie

[Aho et al.72] Aho (A.) et Ullman (J.). – *The Theory of Parsing, Translation and Compiling, Vol. 1 : Parsing.* – Englewood Cliffs, Prentice-Hall, 1972.

[Alon et al.91] Alon (N.), Dewdey (A.K.) et Ott (T.J.). – Efficient Simulation of Finite Automata by Neural Nets. *Journal of the Association for Computing Machinery*, vol. 38, n° 2, 1991, pp. 495–514.

[Angluin et al.83] Angluin (D.) et Smith (C.H.). – Inductive Inference : Theory and Methods. *Computing Surveys*, vol. 15, n° 3, 1983, pp. 237–269.

[Angluin78] Angluin (D.). – On the Complexity of Minimum Inference of Regular Sets. *Information and Control*, 1978, pp. 337–350.

[Angluin82] Angluin (D.). – Inference of Reversible Languages. *Journal of the Association for Computing Machinery*, vol. 29, n° 3, 1982, pp. 741–765.

[Angluin87] Angluin (D.). – Queries and Concept Learning. *Machine Learning*, vol. 2, 1987, pp. 319–342.

[Angluin88] Angluin (D.). – *Identifying Languages from Stochastic Examples.* – Rapport technique n° YALEU/DCS/RR-614, Yale University, March 1988.

[Bahl et al.75] Bahl (L.) et Jelinek (F.). – Decoding for Channels with Insertions, Deletions and Substitutions with Application to Speech Recognition. *IEEE Transactions on Information Theory*, vol. IT-21, 1975, pp. 404–411.

[Bahl et al.78] Bahl (L.), Baker (J.), Cohen (P.), Cole (A.), Jelinek (F.), Lewis (B.) et Mercer (R.). – Automatic Recognition of Continuously Spoken Sentences from a Finite State Grammar. *International Conference on Acoustic, Speech and Signal Processing*, pp. 418–421. – Tulsa, 1978.

[Bahl et al.83] Bahl (L.), Jelinek (F.) et Mercer (R.). – A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, n° 2, 1983, pp. 179–190.

[Bahl et al.89] Bahl (L.), Brown (P.), de Souza (P.) et Mercer (R.). – A Tree-Based Statistical Language Model for Natural Language Speech Recognition. *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 37, n° 7, 1989, pp. 1001–1008.

[Baker75] Baker (J.K.). – Stochastic Modeling for Automatic Speech Understanding. *Speech Recognition*, éd. par Reddy, pp. 512–542. – New York, Academic Press, 1975.

[Baker79] Baker (J.K.). – Trainable Grammars for Speech Recognition. *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, éd. par Klatt (D.) et Wolf (J.), pp. 547–550. – 1979.

[Baker93] Baker (J.M.). – Dictation, Directories, and Data Bases; Emerging PC applications for Large Vocabulary Speech Recognition. *European Conference on Speech Communication and Technology*, pp. 3–10. – Berlin, 1993.

[Barzdin74] Barzdin (J. M.). – *Two Theorems on the Limiting Synthesis of Functions.* – Zapiski n° 210, Latv. Gos. Univ., 1974.

[Baum72] Baum (L.). – An Inequality and Associated Maximization Technique in Statistical Estimation for Probalistic Functions of Markov Processes. *Inequalities*, vol. 3, 1972, pp. 1–8.

[Bellman57] Bellman (R.). – *Dynamic Programming.* – Princeton, Princeton University Press, 1957.

[Biermann et al.72] Biermann (A.W.) et Feldman (J.A.). – A Survey of Results in Grammatical Inference. *Frontiers of Pattern Recognition*, éd. par Watanabe (S.), pp. 31–54. – New York, Academic Press, 1972.

[Bourlard et al.85a] Bourlard (H.), Kamp (Y.), Ney (H.) et Wellekens (C.). – Speaker Dependent Connected Speech Recognition via Dynamic Programming and Statistical Methods. *Speech and Speaker Recognition*, éd. par Schroeder (M.), pp. 115–148. – Bâle, Karger, 1985.

[Bourlard et al.85b] Bourlard (H.), Kamp (Y.) et Wellekens (C.). – Speaker Dependent Connected Speech Recognition via Phonemic Markov Models. *International Conference on Acoustic, Speech and Signal Processing*, pp. 1213–1216. – Tampa, 1985.

[Breiman et al.84] Breiman (L.), Friedman (J.), Olshen (R.) et Stone (C.). – *Classification and Regression Trees.* – Wadsworth Inc., 1984.

[Bresnan82] Bresnan (J.). – *The Mental Representation of Grammatical Relations.* – Cambridge, Massachusetts, MIT Press, 1982.

[Bridle et al.82] Bridle (J.), Brown (M.) et Chamberlain (R.). – An Algorithm for Connected Word Recognition. *International Conference on Acoustic, Speech and Signal Processing*, pp. 899–902. – Paris, 1982.

[Brown et al.83] Brown (P.) et Lee (C.). – Bayesian Adaptation in Speech Recognition. *International Conference on Acoustic, Speech and Signal Processing*, pp. 761–764. – Boston, 1983.

[Brown et al.92a] Brown (P.), Della Pietra (S.), Della Pietra (V.), Lai (J.) et Mercer (R.). – An Estimate of an Upper Bound for the Entropy of English. *Computational Linguistics*, vol. 18, 1992, pp. 31–40.

[Brown et al.92b] Brown (P.), Della Pietra (V.), de Souza (P.), Lai (J.) et Mercer (R.). – Class-based N-gram Models of Natural Language. *Computational Linguistics*, vol. 18, 1992, pp. 467–479.

[Carrasco et al.94] Carrasco (R.) et Oncina (J.). – Learning Stochastic Regular Grammars by Means of a State Merging Method. *Grammatical Inference and Applications, ICGI'94*. pp. 139–150. – Springer Verlag, 1994.

[Case et al.83] Case (J.) et Smith (C.). – Comparison of Identification Criteria for Machine Inductive Inference. *Theor. Comput. Science*, vol. 25, 1983, pp. 193–220.

[Castellanos et al.94] Castellanos (A.), Galiano (I.) et Vidal (E.). – Application of OSTIA to Machine Translation Tasks. *Grammatical Inference and Applications, ICGI'94*. pp. 93–105. – Springer Verlag, 1994.

[Chomsky57] Chomsky (N.). – *Syntactic Structures*. – La Haye, Mouton, 1957.

[Chow et al.89] Chow (Y.) et Roukos (S.). – Speech Understanding using a Unification Grammar. *International Conference on Acoustic, Speech and Signal Processing*, pp. 727–730. – Glasgow, 1989.

[Cleeremans et al.89] Cleeremans (A.), Servan-Schreiber (D.) et McClelland (J.). – Finite State Automata and Simple Recurrent Networks. *Neural Computation*, vol. 1, n° 3, 1989, pp. 372–381.

[Cook et al.74] Cook (C.M.) et Rozenfeld (A.). – Some Experiments in Grammatical Inference. *Computer Oriented Learning Process*. Nato Advanced Study Institute, pp. 157–171. – Bonas, France, 1974.

[Cormen et al.90] Cormen (T.H.), Leiserson (C.E.) et Rivest (R.L.). – *Introduction to Algorithms*. – Cambridge, Massachusetts, MIT Press, 1990.

[David et al.62] David (F.N.) et Barton (D.E.). – *Combinatorial Chance.* – London, Charles Griffin & Co. Ltd., 1962.

[Della Pietra et al.94] Della Pietra (S.), Della Pietra (V.), Gillett (J.), Lafferty (J.), Printz (H.) et Ureš (L.). – Inference and Estimation of a Long-Range Trigram Model. *Grammatical Inference and Applications, ICGI'94.* pp. 78–92. – Springer Verlag, 1994.

[Derouault85] Derouault (A.-M.). – *Modélisation de la Langue Naturelle pour la Désembiguation des Chaînes Phonétiques.* – Thèse de Doctorat d'Etat ès Sciences, Université Paris VII, 1985.

[Dupont et al.89] Dupont (P.) et Snyers (D.). – *Efficient Dynamic Expansion of Context-Free Grammar in Speech Recognition.* – Overview of Research in Speech Recognition at PRLB in 1988, Rapport technique n° R 534, Philips Research Laboratory Brussels, February 1989.

[Dupont et al.90] Dupont (P.) et Kamp (Y.). – Reconnaissance de la parole pilotée par un modèle linguistique. *Approche logique de l'Intelligence Artificielle*, chap. 1, pp. 1–61. – Paris, France, Dunod informatique, 1990.

[Dupont et al.91] Dupont (P.) et Kamp (Y.). – Guiding Speech Recognition by a Language Model. *From Natural Language Processing to Logic for Expert Systems*, éd. par Thayse (A.), chap. 1, pp. 1–48. – Chichester, Wiley, 1991.

[Dupont et al.94] Dupont (P.), Miclet (L.) et Vidal (E.). – What is the Search Space of the Regular Inference ? *Grammatical Inference and Applications, ICGI'94.* pp. 25–37. – Springer Verlag, 1994.

[Dupont93a] Dupont (P.). – Dynamic Use of Syntactical Knowledge in Continuous Speech Recognition. *European Conference on Speech Communication and Technology*, pp. 1959–1962. – Berlin, 1993.

[Dupont93b] Dupont (P.). – Efficient Integration of Context-Free Based Language Models in Continuous Speech Recognition. *New Advances and Trends in Speech Recognition and Coding*. NATO ASI, pp. 179–182. – Granada, Spain, 1993.

[Dupont94] Dupont (P.). – Regular Grammatical Inference from Positive and Negative Samples by Genetic Search : the GIG method. *Grammatical Inference and Applications, ICGI'94*. pp. 236–245. – Springer Verlag, 1994.

[Earley70] Earley (J.). – An Efficient Context-Free Parsing Algorithm. *Communications of the Association for Computing Machinery*, vol. 13, n° 2, 1970, pp. 94–102.

[Elman88] Elman (J.L.). – *Finding Structure in Time*. – Rapport technique n° CRL Tech. Report 9901, San Diego, California, University of California, 1988.

[Feldman et al.90] Feldman (J.A.), Lakoff (G.), Stolcke (A.) et Weber (S.H.). – *Miniature Language Acquisition : A touchstone for cognitive science*. – Rapport technique n° TR-90-009, Berkeley, California, International Computer Science Institute, 1990.

[Fissore et al.88] Fissore (L.), Giachin (E.), Laface (P.), Micca (G.), Pieraccini (R.) et Rullent (C.). – Experimental Results on Large Vocabulary Continuous Speech Recognition and Understanding. *International Conference on Acoustic, Speech and Signal Processing*, pp. 414–417. – New-York, 1988.

[Forney73] Forney (G.D.). – The Viterbi Algorithm. *IEEE Proceedings*, vol. 3, 1973, pp. 268–278.

[Fu et al.71] Fu (K.S.) et Huang (H.). – On Stochastic Context-Free Languages. *Information Sciences*, vol. 3, 1971, pp. 201–224.

[Fu et al.75a] Fu (K.S.) et Booth (T.L.). – Grammatical Inference : Introduction and Survey, Part 1. *IEEE*

Transactions on Systems, Man and Cybernetics, vol. 5, 1975, pp. 85–111.

[Fu et al.75b] Fu (K.S.) et Booth (T.L.). – Grammatical Inference : Introduction and Survey, Part 2. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 5, 1975, pp. 409–423.

[Furui86] Furui (S.). – Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum. *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 34, n° 1, 1986, pp. 52–59.

[García et al.87] García (P.), Vidal (E.) et Casacuberta (F.). – Local Languages, the Successor Method, and a Step Towards a General Methodology for the Inference of Regular Grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, n° 6, 1987, pp. 841–845.

[García et al.90] García (P.) et Vidal (E.). – Inference of K-testable Languages in the Strict Sense and Applications to Syntactic Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, n° 9, 1990, pp. 920–925.

[García et al.93] García (P.) et Oncina (J.). – *Learning General Context-Free Grammars from Positive Structural Samples and Negative Strings.* – Research report, Universidad Politécnica de Valencia, 1993.

[García et al.94] García (P.), Segarra (E.), Vidal (E.) et Galiano (I.). – On the use of the Morphic Generator Grammatical Inference (MGGI) Methodology in Automatic Speech Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 4, n° 4, 1994, pp. 667–685.

[Gauvain et al.93] Gauvain (J.L.), Lamel (L.F.), Adda (G.) et Adda-Decker (M.). – Speaker-Independent Continuous Speech Dictation. *European Conference on Speech*

Communication and Technology, pp. 125–128. – Berlin, 1993.

[Gazdar et al.85] Gazdar (G.), Klein (E.), Pullum (G.) et Sag (I.). – *Generalized Phrase Structure Grammar.* – Oxford, Basil Blackwell, 1985.

[Gazdar et al.89] Gazdar (G.) et Mellish (C.). – *Natural Language Processing in Prolog : An Introduction to Computational Linguistics.* – Reading, Massachusetts, Addison-Wesley, 1989.

[Giles et al.90] Giles (C.L.), Sun (G.Z.), Chen (H.H.), Lee (Y.C.) et Chen (D.). – High Order Recurrent Networks and Grammatical Inference. *Advances in Neural Information Processing System*, éd. par Touretzky (D.S.), pp. 380–387. – San Mateo, Morgan Kaufman, 1990.

[Giordano94] Giordano (J.-Y.). – Inference of Context-Free Grammars by Enumeration : Structural Containment as an Ordering Bias. *Grammatical Inference and Applications, ICGI'94.* pp. 212–221. – Springer Verlag, 1994.

[Giordano95] Giordano (J.-Y.). – *Inférence de Grammaires Algébriques.* – Thèse de Doctorat, Université de Rennes, Janvier 1995.

[Godin et al.89] Godin (C.) et Lockwood (P.). – DTW schemes for Continuous Speech Recognition : a Unified View. *Computer Speech and Language*, vol. 3, n° 2, 1989, pp. 169–198.

[Godin90] Godin (C.). – *Proposition d'un Cadre Algorithmique Unifié pour la Compréhension de Parole Continue.* – Thèse de Doctorat, Université de Compiègne, 1990.

[Gold67] Gold (E.M.). – Language Identification in the Limit. *Information and Control*, vol. 10, n° 5, 1967, pp. 447–474.

[Gold78] Gold (E.M.). – Complexity of Automaton Identification from Given Data. *Information and Control*, vol. 37, 1978, pp. 302–320.

[Goldberg89] Goldberg (D.E.). – *Genetic Algorithms in Search, Optimization and Machine Learning.* – Reading, Massachusetts, Addison-Wesley, 1989.

[Gonzalez et al.78] Gonzalez (R.C.) et Thomason (M. G.). – *Syntacic Pattern Recognition, An Introduction.* – Reading, Massachusetts, Addition-Wesley, 1978.

[Gorin et al.91] Gorin (A.L.), Levinson (S.E.), Gertner (A.N.) et Goldman (E.). – Adaptive Acquisition of Language. *Computer Speech and Language*, vol. 5, 1991, pp. 101–132.

[Gregor94] Gregor (J.). – Data-driven Inductive Inference of Finite-state Automata. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 8, n° 1, 1994, pp. 305–322.

[Gross et al.70] Gross (M.) et Lentin (A.). – *Notion sur les Grammaires Formelles.* – Paris, Gauthier-Villars, 1970, 2-ème édition.

[Harrison78] Harrison (M.A.). – *Introduction to Formal Language Theory.* – Reading, Massachusetts, Addison-Wesley, 1978.

[Hartmanis et al.66] Hartmanis (J.) et Stearns (R. E.). – *Algebraic Structure Theory of Sequential Machines.* – Englewood Cliffs, New Jersey, Prentice-Hall, 1966.

[Haton et al.78] Haton (J.-P.) et Pierrel (J.-M.). – Data Structures and Architecture of Myrtille II Speech Understanding System. *4th International Joint Conference on Pattern Recognition*, pp. 1033–1038. – Kyoto, Japon, 1978.

[Hemphill et al.89] Hemphill (C.) et Picone (J.). – Speech Recognition in a Unification Grammar Framework. *International Conference on Acoustic, Speech and Signal Processing*, pp. 723–726. – Glasgow, 1989.

[Henaff95] Henaff (M.-M.). – *Inférence de grammaires régulières à partir d'échantillons positifs et négatifs.* – Rapport de stage, DEA Informatique, IFSIC, ENSSAT, 1995.

[Hopcroft et al.79] Hopcroft (J.E.) et Ullman (J.D.). – *Introduction to Automata Theory, Languages and Computation.* – Reading, Massachusetts, Addison Wesley, 1979.

[Horning69] Horning (J.J.). – *A Study of Grammatical Inference.* – Stanford, California, Ph. D. dissertation, Computer Science Department, Stanford University, 1969.

[Jain et al.92] Jain (A.N.), Waibel (A.) et Touredsky (D.S.). – PARSEC : a Structured Connectionist Parsing System for Spoken Language. *International Conference on Acoustic, Speech and Signal Processing*, pp. 205–208. – 1992.

[Jelinek et al.80] Jelinek (F.) et Mercer (R.). – Interpolated Estimation of Markov Source Parameters from Sparse Data. *Pattern Recognition in Practice*, éd. par Gelsema (E.) et Kanal (L.), pp. 381–397. – Amsterdam, North-Holland, 1980.

[Jelinek et al.92] Jelinek (F.), Lafferty (J.D.) et Mercer (R.L.). – Basic Methods of Probabilistic Context Free Grammars. *Speech Recognition and Understanding. Recent Advances, Trends and Applications*, éd. par Laface (P.) et De Mori (R.). NATO ASI, pp. 345–360. – Springer Verlag, 1992.

[Jelinek76] Jelinek (F.). – Continuous Speech Recognition by Statistical Methods. *Proceedings of the IEEE*, vol. 64, n° 4, 1976, pp. 532–556.

[Jelinek85] Jelinek (F.). – The Development of an Experimental Discrete Dictation Recognizer. *Proceedings of the IEEE*, vol. 73, n° 11, 1985, pp. 1616–1624.

[Jiménez et al.94] Jiménez (V.), Vidal (E.), Oncina (J.), Castellanos (A.), Rulot (H.) et Sánchez (J.). – Spoken-Language Machine Translation in Limited-Domain

Tasks. *CRIM/FORWISS Workshop on Progress and Prospects of Speech Research and Technology*, éd. par Niemann (H.), De Mori (R.) et Hanrieder (G.), pp. 262–265. – 1994.

[Jones et al.91] Jones (D.R.) et Beltrano (M.A.). – Solving Partitioning Problems with Genetic Algorithms. *4th International Conference on Genetic Algorithms*. pp. 442–449. – California, 1991.

[Jouvet et al.91] Jouvet (D.), Bartkova (K.) et Monné (J.). – On the Modelization of Allophones in an HMM based Speech Recognition System. *European Conference on Speech Communication and Technology*, pp. 923–926. – Genova, 1991.

[Jouvet et al.93a] Jouvet (D.), Lainé (A.), Monné (J.) et Gagnoulet (C.). – Speaker-Independent Spelling over the Telephone. *International Conference on Acoustic, Speech and Signal Processing*, pp. II 235–238. – Minneapolis, 1993.

[Jouvet et al.93b] Jouvet (D.), Lokbani (M.) et Monné (J.). – Application of the N-Best Solutions Algorithm to Speaker-Independent Spelling Recognition over the Telephone. *European Conference on Speech Communication and Technology*, pp. 2081–2084. – Berlin, 1993.

[Jouvet88] Jouvet (D.). – *Reconnaissance de Mots Connectés Indépendamment du Locuteur par des Méthodes Statistiques*. – Paris, France, Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, 1988.

[Kai et al.92] Kai (A.) et Nakagawa (S.). – A Frame-Synchronous Continuous Speech Recognition Algorithm using a Top-Down Parsing of Context-Free Grammar. *ICSLP*, pp. 257–260. – Banff, Canada, 1992.

[Katz87] Katz (S.M.). – Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 35, n° 3, 1987, pp. 400–401.

[Kearns et al.89] Kearns (M.) et Valiant (L.). – Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *21st ACM Symposium on Theory of Computing*, pp. 433–444. – 1989.

[Kita et al.89] Kita (K.), Kawabata (T.) et Saito (H.). – HMM Continuous Speech Recognition using Predictive LR Parsing. *International Conference on Acoustic, Speech and Signal Processing*, pp. 703–706. – 1989.

[Kochman et al.92] Kochman (F.) et Kupin (J.J.). – *Removing Left Recursion in Stochastic Grammars.* – Working Paper n° 1103, IDA-Center for Communications Research, December 1992.

[Koza91] Koza (J. R.). – *Genetic Programming.* – Cambridge, Massachusetts, MIT Press, 1991.

[Kuhn et al.90] Kuhn (R.) et De Mori (R.). – A Cache-Based Natural Language Model for Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, n° 6, 1990, pp. 570–583.

[Lang92] Lang (K.J.). – Random DFA's can be Approximately Learned from Sparse Uniform Examples. *5th ACM workshop on Computational Learning Theory*, pp. 45–52. – 1992.

[Lari et al.90] Lari (K.) et Young (S.J.). – The Estimation of Stochastic Context-Free Grammars using the Inside-Outside Algorithm. *Computer, Speech and Language*, vol. 4, 1990, pp. 35–56.

[Lee et al.91] Lee (C.H.), Lin (C.H.) et Juang (B.H.). – A Study on Speaker Adpatation of Continuous Density Hidden Markov Models. *IEEE Transactions on Signal Processing*, vol. 39, n° 4, 1991, pp. 806–814.

[Lee88] Lee (K.F.). – *Large Vocabulary Speaker Independent Continuous Speech Recognition : The*

SPHINX System. – Ph. D. dissertation, Computer Science Department, Carnegie Mellon University, 1988.

[Lee93] Lee (K.F.). – The Conversational Computer : An Apple Perspective. *European Conference on Speech Communication and Technology*, pp. 1377–1384. – Berlin, 1993.

[Leggetter et al.94] Leggetter (C.J.) et Woodland (P.C.). – Speaker Adaptation of Continuous Density HMMs using Multivariate Linear Regression. *International Conference on Spoken Language Processing*, pp. 451–454. – Yokohama, 1994.

[Lesser et al.75] Lesser (V.R.), Fennel (R.D.), Erman (L.D.) et Reddy (D.R.). – Organization of the Hearsey II Speech Understanding System. *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 23, n° 1, 1975, pp. 11–23.

[Levinson et al.83] Levinson (S.), Rabiner (R.) et Sondhi (M.). – An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. *Bell Systems Technical Journal*, vol. 62, 1983, pp. 1035–1074.

[Liporace82] Liporace (L.). – Maximum Likelihood Estimation for Multivariate Observation of Markov Sources. *IEEE Transactions on Information Theory*, vol. IT-28, 1982, pp. 729–734.

[Lowerre76] Lowerre (B.). – *The HARPY Speech Recognition System.* – Ph. D. dissertation, Computer Science Department, Carnegie Mellon University, 1976.

[McCullough et al.43] McCullough (W.S.) et Pitts (W.). – A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bull. Math. Biophysics*, vol. 5, 1943, pp. 115–133.

[Méloni83] Méloni (H.). – Traitement des Contraintes Linguistiques en Reconnaissance de la Parole. *Techniques et Sciences Informatiques*, vol. 2, n° 5, 1983, pp. 349–363.

[Mercier et al.78] Mercier (G.), Quinton (P.) et Vivès (R.). – KEAL : Un Système pour un Dialogue Oral avec une Machine. *Congrès AFCET*, pp. 304–314. – Gif-sur-Yvette, France, 1978.

[Mercier et al.89] Mercier (G.), Bigorgne (D.), Miclet (L.), Le Guennec (L.) et Querre (M.). – Recognition of Speaker-Dependent Continuous Speech with KEAL. *IEE Proceedings*, vol. 136, Part 1, n° 2, 1989, pp. 145–154.

[Michalewicz92] Michalewicz (Z.). – *Genetic Algorithms + Data Structures = Evolution Programs.* – Berlin, Springer-Verlag, 1992.

[Miclet et al.94] Miclet (L.) et de Gentile (C.). – Inférence Grammaticale à partir d'Exemples et de Contre-Exemples : deux Algorithmes Optimaux (BIG et RIG) et une Version Heuristique (BRIG). *Neuvièmes Journées Francophones sur l'Apprentissage*, pp. F1–F13. – Strasbourg, France, 1994.

[Miclet et al.95] Miclet (L.), Dupont (P.) et Vial (S.). – Inférence Grammaticale Régulière : Méthodes Semi-itératives et Mesure de Performance. *Journées Acquisition, Validation, Apprentissage*, pp. 103–116. – Grenoble, France, 1995.

[Miclet79] Miclet (L.). – *Inférence de Grammaires Régulières.* – Paris, France, Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, 1979.

[Miclet80] Miclet (L.). – Regular Inference with a Tail-Clustering Method. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 10, 1980, pp. 737–743.

[Miclet90] Miclet (L.). – Grammatical Inference. *Syntactic and Structural Pattern Recognition : Theory and Applications*, éd. par Bunke (H.) et Sanfeliu (A.), pp. 237–290. – Singapore, World Scientific, 1990.

[Miller et al.90] Miller (P.) et Torris (T.). – *Formalismes Syntaxiques pour le Traitement Automatique du Langage Naturel.* – Paris, Hermès, 1990.

[Minsky54] Minsky (M.L.). – *Neural Nets and the Brain Model problem.* – Princeton, New Jersey, Ph. D. dissertation, Princeton University, 1954.

[Minsky67] Minsky (M.L.). – *Computation : Finite and Infinite Machines.* – New York, Prentice-Hall, 1967.

[Mitchell78] Mitchell (T.). – *Version Spaces : an approach to Concept Learning.* – Ph. D. dissertation, Standford University, 1978.

[Muggleton84] Muggleton (S.). – *Induction of Regular Languages from Positive Examples.* – Rapport technique n° TIRM-84-009, Turing Institute, 1984.

[Murveit et al.88] Murveit (H.) et Weintraub (M.). – 1000-word Speaker-Independent Continuous-Speech Recognition using Hidden Markov Models. *International Conference on Acoustic, Speech and Signal Processing*, pp. 115–118. – New-York, 1988.

[Myers et al.81] Myers (C.) et Rabiner (L.). – A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition. *IEEE Transactions on Acoustic, Speech and Signal Processing*, vol. 29, n° 2, 1981, pp. 284–296.

[Nakamura et al.89] Nakamura (M.) et Shikano (K.). – A Study of English Word Category Prediction Based on Neural Networks. *International Conference on Acoustic, Speech and Signal Processing*, pp. 731–734. – Glasgow, 1989.

[Natarajan91] Natarajan (B.L.). – *Machine Learning : A Theoretical Approach.* – San Mateo, CA, Morgan Kauffman Pub., 1991.

[Ney et al.87] Ney (H.), Mergel (D.), Noll (A.) et Paeseler (A.). – A Data-driven Organization of the Dynamic Programming Beam Search for Continuous Speech

Recognition. *International Conference on Acoustic, Speech and Signal Processing*, pp. 833–836. – Dallas, 1987.

[Ney et al.94] Ney (H.), Essen (U.) et Kneser (R.). – On Structuring Probabilistic Dependences in Stochastic Language Modelling. *Computer Speech and Language*, vol. 8, 1994, pp. 1–38.

[Ney84] Ney (H.). – The use of a One-stage Dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustic Speech and Signal Processing*, vol. 32, n° 2, 1984, pp. 263–271.

[Ney91] Ney (H.). – Dynamic Programming Parsing for Context-Free Grammars in Continuous Speech Recognition. *IEEE Transactions on Signal Processing*, vol. 39, n° 2, 1991, pp. 336–340.

[Nicolas93] Nicolas (J.). – L'espace des versions. *Ecole sur l'Apprentissage automatique*, pp. 19–32. – Saint-Raphaël, 1993.

[Oncina et al.92] Oncina (J.) et García (P.). – Inferring Regular Languages in Polynomial Update Time. *Pattern Recognition and Image Analysis*, éd. par Pérez de la Blanca (N.), Sanfeliu (A.) et E.Vidal, pp. 49–61. – World Scientific, 1992.

[Pereira et al.80] Pereira (F.) et Warren (D.). – Definite Clause Grammar for Language Analysis – a survey of the formalism and a comparison with ATN. *Artificial Intelligence*, vol. 13, 1980, pp. 231–278.

[Pereira et al.92] Pereira (F.) et Schabes (Y.). – Inside-Outside Reestimation from Partially Bracketed Corpora. *30th Annual Meeting of the ACL*, pp. 128–135. – 1992.

[Pierrel82] Pierrel (J.-M.). – Utilisation de Contraintes Linguistiques en Compréhension Automatique de la Parole Continue. *Techniques et Sciences Informatiques*, vol. 1, n° 5, 1982, pp. 403–419.

[Pitt et al.89] Pitt (L.) et Warmuth (M.). – *The Minimum Consistent DFA Problem Cannot be Approximated Within any Polynomial.* – Rapport technique n° UIUCDCS-R-89-1499, University of Illinois, 1989.

[Pitt89] Pitt (L.). – Inductive Inference, DFA's, and Computational Complexity. *Lecture Notes in Artificial Intelligence*, éd. par Jantke (K.P.), pp. 18–44. – Berlin, Springer-Verlag, 1989.

[Pollack90] Pollack (J.B.). – Recursive Distribute Representation. *Artificial Intelligence*, vol. 46, 1990, pp. 77–105.

[Prat95] Prat (F.). – *Distorsión Estocástica de Cadenas de Símbolos mediante dos Métodos basados en Modelos de Markov Ocultos.* – Research report, Universidad Politécnica de Valencia, 1995.

[Quinton82] Quinton (P.). – Utilisation de Contraintes Syntaxiques pour la Reconnaissance de la Parole Continue. *Techniques et Sciences Informatiques*, vol. 1, n° 3, 1982, pp. 233–248.

[Rabiner et al.78] Rabiner (L.) et Schafer (R.). – *Digital Processing of Speech Signals.* – Englewood Cliffs, Prentice-Hall, 1978.

[Rabiner et al.83] Rabiner (L.), Levinson (S.) et Sondhi (M.). – On the application of Vector Quantization and Hidden Markov Models to Speaker Independent, Isolated Word Recognition. *Bell Systems Technical Journal*, vol. 62, 1983, pp. 1075–1105.

[Rabiner et al.87] Rabiner (L.) et Wilpon (J.). – Some Performance Benchmarks for Isolated Word Speech Recognition Systems. *Computer Speech and Language*, vol. 2, n° 3/4, 1987, pp. 343–358.

[Radhakrishnan et al.88] Radhakrishnan (V.) et Nagaraja (G.). – Inference of Even Linear Grammars and its Applications to Picture Description Language. *Pattern Recognition*, vol. 21, n° 1, 1988, pp. 55–62.

[Richetin et al.84] Richetin (M.) et Vernadat (F.). – Efficient Regular Grammatical Inference for Pattern Recognition. *Pattern Recognition*, vol. 17, n° 2, 1984, pp. 245–250.

[Rosenkrantz67] Rosenkrantz (D.J.). – Matrix Equations and Normal Forms for Context-Free Grammars. *Journal of the Association for Computing Machinery*, vol. 14, n° 3, 1967, pp. 501–507.

[Rulot et al.88] Rulot (H.) et Vidal (E.). – An Efficient Algorithm for the Inference of Circuit-free Automata. *Advances in Structural and Syntactic Pattern Recognition*, éd. par Ferratè (G.), Pavlidis (T.), Sanfeliu (A.) et Bunke (H.). NATO ASI, pp. 173–184. – Springer-Verlag, 1988.

[Rulot92] Rulot (H.). – *ECGI : Un Algoritmo de Inferencia Gramatical Mediante Corrección de Errores.* – Ph. D. dissertation, Universitát de València, 1992.

[Sadek94] Sadek (D.). – Communication Theory = Rationality Principles + Communicative Act Models. *AAAI Workshop on Planning for Interagent Communication*, pp. 100–106. – Seattle, USA, 1994.

[Sakakibara92] Sakakibara (Y.). – Efficient Learning of Context-Free Grammars from Positive Structural Examples. *Information and Computation*, vol. 97, 1992, pp. 23–60.

[Sakoe79] Sakoe (H.). – Two-level DP matching - a Dynamic Programming based Pattern Matching Algorithm for Connected Word Recognition. *IEEE Transactions on Acoustic Speech and Signal Processing*, vol. 27, n° 6, 1979, pp. 588–595.

[Schwartz et al.87] Schwartz (R.), Chow (Y.) et Kubala (F.). – Rapid Speaker Adaptation using a Probabilistic Spectral Mapping. *International Conference on Acoustic, Speech and Signal Processing*, pp. 633–636. – Dallas, 1987.

[Segarra93] Segarra (E.). – *Una Aproximacion Inductiva a la Comprension del Discurso Continuo.* – Ph. D. dissertation, Universidad Politécnica de Valencia, 1993.

[Smith et al.89] Smith (K.R.) et Miller (M.I.). – Learning Regular Grammars on Connection Architectures. *International Conference on Acoustic, Speech and Signal Processing*, pp. 2501–2504. – Glasgow, 1989.

[Steinbiss et al.90] Steinbiss (V.), Noll (A.), Paeseler (A.), Ney (H.), Bergmann (H.), Dugast (C.), Hamer (H.-H.), Piotrowski (H.), Tomaschewski (H.) et Zielinski (A.). – A 10,000-word Continuous-speech Recognition System. *International Conference on Acoustic, Speech and Signal Processing*, pp. 57–60. – Albuquerque, 1990.

[Stolcke et al.94a] Stolcke (A.) et Omohundro (S.). – Inducing Probabilistic Grammars by Bayesian Model Merging. *Grammatical Inference and Applications, ICGI'94*. pp. 106–118. – Springer Verlag, 1994.

[Stolcke et al.94b] Stolcke (A.) et Segal (J.). – *Precise N-gram Probabilities from Stochastic Context-free Grammars.* – Rapport technique n° TR-94-007, ICSI, January 1994.

[Stolcke94] Stolcke (A.). – *Bayesian Learning of Probabilistic Language Models.* – Ph. D. dissertation, Univerity of California, 1994.

[Takada88] Takada (Y.). – Grammatical Inference for Even Linear Languages based on Control Sets. *Information Processing Letters*, vol. 28, n° 4, 1988, pp. 193–199.

[Tomita82] Tomita (M.). – Dynamic Construction of Finite-Automata From Examples Using Hill-Climbing. *4th Annual Cognitive Science Conference*, pp. 105–108. – 1982.

[Tomita86] Tomita (M.). – An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. *In-*

ternational Conference on Acoustic, Speech and Signal Processing, pp. 1569–1572. – Tokyo, 1986.

[Trakhtenbrot et al.73] Trakhtenbrot (B.) et Barzdin (Ya.). – *Finite Automata : Behavior and Synthesis*. – Amsterdam, North Holland Pub. Comp., 1973.

[Valiant84] Valiant (L.G.). – A theory of the learnable. *Communications of the Association for Computing Machinery*, vol. 27, n° 11, 1984, pp. 1134–1142.

[Vidal et al.93] Vidal (E.), Casacuberta (P.) et García (P.). – *Grammatical Inference and Applications to Automatic Speech Recognition*. – Rapport technique n° DSIC II/41/93, Universidad Politécnica de Valencia, 1993.

[Wetherell80] Wetherell (C.). – Probabilistic Languages : A Review and Some Open Questions. *Computing Surveys*, vol. 12, n° 4, 1980, pp. 361–379.

[Wharton74] Wharton (R.M.). – Approximate Language Identification. *Information and Control*, vol. 26, 1974, pp. 236–255.

[Wharton77] Wharton (R.M.). – Grammar Enumeration and Inference. *Information and Control*, vol. 33, 1977, pp. 253–272.

[Whitley93] Whitley (L.D.) (édité par). – *Foundations of Genetic Algorithms-2*. – San Mateo, California, Morgan Kaufmann, 1993.

[Woods70] Woods (W.). – Transition Network Grammars for Natural Language Analysis. *Communications of the Association for Computing Machinery*, vol. 13, 1970, pp. 591–606.

[Wyard94] Wyard (P.). – Representational Issues for Context Free Grammars Induction Using Genetic Algorithms. *Grammatical Inference and Applications, ICGI'94*. pp. 222–235. – Springer Verlag, 1994.

[Young et al.88] Young (S.), Russel (N.) et Thornton (J.). – Speech Recognition in Vodis II. *International Conference on Acoustic, Speech and Signal Processing*, pp. 441–444. – 1988.

[Young et al.89] Young (S.), Hauptmann (A.), Ward (W.), Smith (E.) et Werner (P.). – High Level Knowledge Sources in Usable Speech Recognition Systems. *Communications of the Association for Computing Machinery*, vol. 32, n° 2, 1989, pp. 183–194.

[Young et al.94] Young (S.J.) et Shih (H.-H.). – Computer Assisted Grammar Construction. *Grammatical Inference and Applications, ICGI'94*. pp. 282–290. – Springer Verlag, 1994.

[Younger67] Younger (D.). – Recognition and Parsing of Context-free Languages in time n^3. *Information and Control*, vol. 10, n° 2, 1967, pp. 189–208.