Sous-automates à nombre fini d'états. Application à la compression de dictionnaires électroniques

Lamia Tounsi

▶ To cite this version:

Lamia Tounsi. Sous-automates à nombre fini d'états. Application à la compression de dictionnaires électroniques. Informatique [cs]. Université François - Rabelais de Tours, 2007. Français. NNT : . tel-04631579

HAL Id: tel-04631579
https://hal.science/tel-04631579v1

Submitted on 2 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ FRANCOIS RABELAIS TOURS
École Doctorale "Santé, Sciences, Technologies"
Année Universitaire 2006-2007

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE TOURS

Discipline : Informatique
Présentée et soutenue publiquement

par

Lamia TOUNSI
le 12 Décembre 2007

SOUS-AUTOMATES À NOMBRE FINI D'ÉTATS APPLICATION À LA COMPRESSION DE DICTIONNAIRES ÉLECTRONIQUES

Directeur de thèse :

MAUREL Denis	Professeur
BOUCHOU Béatrice	Co-encadrante, Maître de Conférence

Jury :

BOUCHOU Béatrice	Maître de Conférence, Université de Tours
CHAMPARNAUD Jean Marc	Professeur, Université de Rouen
LAPORTE Éric	Professeur, Université Marne-la-Vallée
LEVRAT Bernard	Professeur, Université d'Angers
MAUREL Denis	Professeur, Université de Tours

Remerciements

Je remercie d'abord les professeurs Jean-Marc Champarnaud et Éric Laporte pour leur travail de rapporteur, ainsi que le professeur Bernard Levra pour son rôle d'examinateur lors de mon jury de soutenance. Je remercie également Denis Maurel et Béatrice Bouchou pour leur travail, leur conseils et pour leurs encouragements.

Je souhaite remercier également les professeurs Franz Guenthner, Éric Laporte, Jacques Savoy et Duško Vitas, mais aussi Pierre-Emmanuel Gros pour la libre utilisation des dictionnaires présentés dans ma thèse.

Mes remerciements s'adressent aussi à tous les chercheurs et personnels de Polytech'Tours pour leur accueil et leur sympathie.

Un grand MERCI à Christophe Lenté pour sa disponibilité, sa collaboration scientifique, son aide précieuse dans la relecture de mon manuscrit et son soutien moral tout au long de ma thèse. Il m'a permis de mener à bien ce travail. Grâce à lui mon esprit de recherche et mon esprit critique se sont développés. Je lui adresse mes plus vifs remerciements, ma profonde gratitude et toute mon amitié.

Je tiens à remercier également mes parents, mon frère, ma famille, mes amis et Mehdi pour leur fidélité, leur soutient constant et leur patience.

Table des matières

1 Introduction **16**

2 État de l'art **20**

2.1 Les automates à nombre fini d'états pour le traitement automatique des langues . . . 20
2.1.1 Automates et dictionnaires . . . 22
2.1.2 Minimisation des automates . . . 24
2.1.3 Représentation et compression des automates . . . 26
2.1.4 Compression et automates représentant des langues naturelles . . . 34
2.2 Recherche de motifs (fréquents) . . . 36
2.2.1 Indexation de textes par des automates *DAWG* . . . 37
2.2.2 Notations de base . . . 38
2.2.3 Directed Acyclic Word Graphs . . . 38
2.2.4 Algorithmes de recherche de composants dans les graphes . . . 41

3 Recherche de sous-automates **43**

3.1 Préliminaires . . . 44
3.1.1 Définitions et notations . . . 44
3.1.2 Hauteur et cardinalité . . . 45
3.2 Sous-automates recherchés . . . 45
3.2.1 Sous-automate . . . 45
3.2.2 Propriétés . . . 47
3.2.3 Sous-automates fermés "*SAF*" . . . 48
3.2.4 Plus petits sous-automates fermés ("*PPSAF*") . . . 50
3.2.5 Plus petit sous-automate ("PPSA") . . . 52
3.3 Recherche des plus petits sous-automates fermés ("PPSAF") . . . 54
3.3.1 Analyse du problème . . . 55
3.3.2 Algorithme PPSAF . . . 59
3.3.3 Résultats expérimentaux . . . 62
3.3.4 Preuve de l'algorithme . . . 63
3.3.5 Complexité . . . 66
3.4 Recherche des plus petits sous-automates "*PPSA*" . . . 67
3.4.1 Définitions et notations . . . 67
3.4.2 Algorithmes de recherche du plus petit sous-automate associé à un ensemble d'états . . . 71
3.4.3 Recherche de tous les plus petits sous-automates associé à un état . . . 75
3.4.4 Complexité de l'algorithme CalculPPSA . . . 80

3.5 Algorithmes de recherche des sous-Automates séries et sous-automates parallèles . . . 81
3.5.1 Exemple . . . 83
3.5.2 Algorithme général de recherche de sous-automates . . . 85
3.6 Conclusion . . . 87

4 Structure interne d'automates représentant des dictionnaires 89
4.1 Généralités sur les automates . . . 89
4.1.1 Représentation d'un dictionnaire par un automate . . . 89
4.1.2 Représentation d'un automate . . . 90
4.2 Présentation des automates étudiés . . . 94
4.2.1 Les dictionnaires . . . 94
4.2.2 Les automates associés . . . 96
4.3 Application de l'algorithme de recherche des sous-automates *Recherche SA* 108
4.3.1 Expérimentations numériques . . . 108
4.3.2 Parallèles purs . . . 112
4.3.3 Séries pures . . . 115
4.4 Conclusion . . . 123

5 Indexation des automates acycliques à nombre fini d'états et application à la compression 126
5.1 Représentation et stockage d'un automate . . . 127
5.1.1 Optimisation de l'occupation mémoire d'un automate . . . 127
5.1.2 Résultats expérimentaux . . . 128
5.1.3 Décompression . . . 129
5.1.4 Représentation étendue et stockage d'un automate . . . 130
5.2 Lecture d'un mot à partir de l'automate réduit . . . 132
5.3 Indexation automatique des automates acycliques à nombre fini d'états . . 136
5.3.1 Indexation des séries et parallèles . . . 137
5.3.2 Indexation des sous-séries et sous-parallèles . . . 137
5.3.3 Automate des suffixes "$DAWG$" . . . 137
5.3.4 Comment adapter le $DAWG$? . . . 138
5.4 Factorisation et compression d'automates acycliques à nombre fini d'états 141
5.4.1 Comment choisir les sous-structures à factoriser ? . . . 142
5.4.2 Calcul des fréquences à l'aide d'un $DAWG$ . . . 143
5.4.3 Comment factoriser ? . . . 143
5.4.4 Comment adapter le $DAWG$ à la factorisation ? . . . 144
5.4.5 Algorithme glouton de factorisation . . . 144
5.5 Mise en oeuvre de la factorisation et de la compression . . . 146
5.5.1 Factorisation et compression d'un automate minimal "FCM" . . . 146
5.5.2 Factorisation et compression d'un automate minimal des mots inversés "$FCM_inverse$" . . . 149
5.5.3 Factorisation et compression d'un automate non minimal "FCNM" 151
5.5.4 Factorisation et compression d'un automate non minimal des mots inversés "$FNCM_{inverse}$" . . . 155
5.5.5 Factorisations et compression des mots d'un dictionnaire "$FCDic$" 157
5.6 Discussion et Conclusion . . . 159

A Statistiques des automates étudiés **174**
A.1 Caractéristique des automates après application de *Recherche SA-SP* . . . 174
A.2 Parallèles Purs 174
A.3 Similarité et dissimilarité des automates étudiés 176
A.4 Étude des fréquences 185

B Factorisation et compression d'un automate **187**
B.1 Description de l'en-tête utilisé pour le codage et le décodage d'un automate 187
B.2 Caractéristiques des factorisations d'un automate minimal (Factorisations FCM) 187
B.3 Caractéristiques des factorisations d'un automate minimal des mots inversés (Factorisations $FCM_{inverse}$) 197

Table des figures

2.1 Automate des mots se terminant par 0 21
2.2 Automate des phrases simples 21
2.3 Automate possédant un état inaccessible 24
2.4 Arbre binaire du mot "anticonstitutionnellement" 30
2.5 DAWG du mot "corolle" 39

3.1 Sous-automate parallèle et sous-automate série 46
3.2 Exemple de sous-automates 47
3.3 Automate de flexion 49
3.4 Sous-automate parallèle fermé et sous-automate série fermé 49
3.5 $PPSAF$ emboîtés 51
3.6 Recherche $PPSA$ 54
3.7 États source et puits associés à différents ensembles d'états 56
3.8 Recherche de sous-automates 62
3.9 Cas d'une mise à jour de $p.efmax$ 65
3.10 Exemple d'automate traité en temps quadratique par l'algorithme $PPSAF$ 66
3.11 Recherche de sous-automates minimaux en largeur 68
3.12 Recherche du plus petit sous-automate associé aux états gris de l'automate (a) 74
3.13 Recherche du $PPSA$ associé à l'état 5 79
3.14 Illustration de la propriété 3.19 81
3.15 Recherche de sous-automates séries et sous-automates parallèles 84
3.16 Réduction de sous-automates 86

4.1 Automates représentant la liste de mots : ℓ, ℓa, $\ell ampe$ 90
4.2 Expression rationnelle finale 94
4.3 Comparaison à l'automate DELAF français 105
4.4 Corrélation 107
4.5 Exemples de sous-automate 111
4.6 Répartition des parallèles purs en fonction des fréquences et des largeurs . 114
4.7 Répartition des séries pures en fonction de leurs longueurs 117
4.8 Répartition des séries pures en fonction de leurs fréquences 119

5.1 Indexation texte vs automate 136
5.2 DAWG de "$aabba$" 138
5.3 $DAWG(S_1, S_2, S_3)$ 140
5.4 Diagramme de factorisation d'un automate minimal 146
5.5 Diagramme de factorisation d'un automate non minimal 151
5.6 Factorisation d'un automate non minimal 152
5.7 Diagramme de la factorisation des mots d'un dictionnaire 157
5.8 Représentation d'un automate vs arbre lexicographique 162

Liste des tableaux

2.1 Exemple de codage 29
2.2 Fréquence des caractères du mot "anticonstitutionnellement" 30
2.3 Codage des caractères du mot "anticonstitutionnellement" 30
2.4 Compression LZ "abracadabra" 33

3.1 Catégories de plus petits sous-automates fermés 52
3.2 Caractérisation de sous-automates particuliers 61
3.3 Dictionnaires 62
3.4 Résultats expérimentaux 62

4.1 Représentation d'un automate 91
4.2 Algorithme *Recherche SA-SP* et expressions rationnelles 93
4.3 Dictionnaires électroniques de la catégorie 1 95
4.4 Dictionnaires électroniques de la catégorie 2 96
4.5 Dictionnaire électronique de la catégorie 3 96
4.6 Statistiques générales 97
4.7 Normalisation de l'automate DELAF En relativement à l'automate DELAF Fr 98
4.8 Étude de la similarité des automates 104
4.9 Calcul des coefficients de corrélation 106
4.10 Similarité décroissante des automates du plus foncé au plus clair 107
4.11 Caractéristiques des automates après application de *Recherche SA-SP* . . . 110
4.12 Structure interne des automates représentant des dictionnaires 110
4.13 Parallèles purs 112
4.14 Répartition des parallèles purs 115
4.15 Séries pures 116
4.16 Répartition des séries pures en fonction de leurs fréquences 121
4.17 Répartition des séries pures en fonction de leurs longueurs 122

5.1 Occupation mémoire des automates 129
5.2 Représentation étendue d'un automate 131
5.3 Factorisation et compression de toutes les séries et tous les parallèles . . . 141
5.4 Factorisation et compression d'un automate minimal 148
5.5 Factorisation et compression d'un automate minimal de mots inversés . . 150
5.6 Factorisation, minimisation et compression d'un automate non minimal . . 154
5.7 Factorisation, minimisation et compression d'un automate non minimal de mots inversés 156
5.8 Factorisation des mot d'un dictionnaire 158
5.9 Récapitulatif des meilleurs résultats des différentes méthodes de factorisation 160

A.1 Gain du nombre de transitions et du nombre d'états 174
A.2 Répartition des parallèles purs en fonction des fréquences 175
A.3 Répartition des parallèles purs en fonction des largeurs 176
A.4 Bigrammes et trigrammes fréquents des séries pures 185

B.1 En-tête utilisé pour le codage et le décodage d'un automate 187
B.2 Caractéristiques de la factorisation à $Taille_{alphabet} = 128$ 189
B.3 Caractéristiques de la factorisation à $Taille_{alphabet} = 256$ 190
B.4 Caractéristiques de la factorisation à $Taille_{alphabet} = 512$ 191
B.5 Caractéristiques de la factorisation à $Taille_{alphabet} = 1024$ 192
B.6 Caractéristiques de la factorisation à $Taille_{alphabet} = 2048$ 193
B.7 Caractéristiques de la factorisation à $Taille_{alphabet} = 4096$ 194
B.8 Caractéristiques de la factorisation à $Taille_{alphabet} = 8192$ 195
B.9 Caractéristiques de la factorisation à $Taille_{alphabet} = 16384$ 196
B.10 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 128$ 198
B.11 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 256$ 199
B.12 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 512$ 200
B.13 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 1024$ 201
B.14 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 2048$ 202
B.15 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 4096$ 203
B.16 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 8192$ 204
B.17 Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 16384$ 205

// Résumé

Les automates acycliques à nombre finis d'états se sont très largement répandus en traitement automatique des langues pour la représentation et le stockage de gros volumes de données, comme des dictionnaires électroniques.
Dans ce travail, nous nous intéressons à l'étude de la structure interne de tels automates ; plus précisément, nous souhaitons détecter des structures présentes à l'intérieur d'un automate à nombre fini d'états, que nous appelons sous-automates. Ainsi, nous proposons un algorithme en $O(n^3)$ pour calculer l'ensemble des sous-automates associés à un automate donné.
Cette étude ouvre les portes à des applications diverses telles que la décomposition des automates, la recherche de sous-automates identiques, la factorisation des sous-automates redondants et peut aussi contribuer à une compression de ces données.
La seconde partie du travail est consacrée à l'application de notre algorithme pour la compression et l'indexation des automates représentant des dictionnaires. Aussi, nous proposons un algorithme de compression qui permet de réduire l'espace mémoire nécessaire au stockage des automates et de conserver un accès efficace aux données. Dans ce contexte, nous avons été amené à développer diverses applications utilisant, d'une part, l'automate des suffixes initialement dédié pour l'indexation de texte, pour indexer les sous-automates, et, d'autre part, des heuristiques permettant de sélectionner les sous-structures les plus intéressantes à factoriser ; celles qui maximisent le gain de mémoire et réduisent la taille de l'automate initial.

Mots clés : Automate, Sous-automate, Indexation, DAWG, Compression, Factorisation, Dictionnaire, Graphe, Algorithme Glouton.

Abstract

Acyclic finite state automata are widely used in Natural Language Processing in order to represent and store huge data such as dictionaries.
This work deals with the study of internal structure of acyclic automaton ; more precisely we are interested in finding structures inside a finite state automaton, which we call sub-automata. Thus, we propose a $O(n^3)$ algorithm to compute all subautomata of a given automaton. This study can be used in applications whose aim is to decompose a very large FSA into smaller ones, to discover frequently occurring data and to reduce memory consumption.
The second part of our work is devoted to the application of our algorithm for compression and indexing of automata that represent electronic dictionaries. Also, we propose a compression algorithm to reduce the memory required to store the automata and to preserve an effective access to data.
The main propositions are, on the one hand, the application of the direct acyclic word graph, initially dedicated for indexing text, to index the subautomata, and, on the other hand, heuristic to select the most interesting substructure to factorize. The best candidates to be factorized are those which increase memory storage efficiency and reduce the size of the initial automaton.

Key Words : Automaton, Subautomaton, Indexing, DAWG, Compression, Factorization, Dictionary, Graph, Greedy Algorithm.

Chapitre 1

Introduction

Cette thèse se situe dans le domaine de la théorie des automates à nombre fini d'états avec des préoccupations liées au traitement automatique des langues (TAL). En effet, les automates acycliques à nombre finis d'états se sont très largement répandus en TAL pour la représentation et le stockage de gros volumes de données, comme des dictionnaires électroniques [Gross et Perrin, 1987], [Revuz, 1991], etc.
Dans ce travail nous nous intéressons à l'étude de la structure interne de tels automates [Tounsi *et al.*, 2007], plus précisément, nous souhaitons détecter des structures présentes à l'intérieur d'un automate à nombre fini d'états, que nous appelons sous-automates.
Ces travaux ouvrent les portes à des applications diverses telles que la décomposition des automates, la recherche de sous-automates identiques, la factorisation des sous-automates redondants et peut aussi contribuer à une compression de ces données.
Le travail est présenté comme suit dans ce document :

Dans un premier temps, à travers l'état de l'art présenté au chapitre 1, nous aborderons les différents thèmes de cette thèse, à savoir, le traitement des automates à nombre finis d'états, l'indexation et la compression des données.

Au chapitre 2, nous définissons ce que nous appelons *sous-automate*. Ensuite, nous proposons un algorithme pour calculer l'ensemble des sous-automates associés à un automate donné.
Les automates sur lesquels nous travaillons sont déterministes et acycliques, ils possèdent un ensemble d'états dont deux états distincts, l'état initial et l'état final.
Intuitivement, un sous-automate est une sous-structure d'un automate composée d'un ensemble d'états et d'un ensemble de transitions. Ici, nous définissons un sous-automate comme un automate composé d'un unique état initial, d'un unique état final et un ensemble d'états internes qui communiquent exclusivement entre eux. L'intuition qui se cache derrière cette idée est de pouvoir isoler et extraire ces sous-parties de l'automate.
Dans le but de calculer tous les sous-automates d'un automate à nombre fini d'états, sachant qu'un sous-automate peut être inclus dans un autre sous-automate, nous proposons de chercher d'abord les sous-automates minimaux en largeur et en longueur.
L'idée est de remplacer chaque sous-automate détecté par une seule transition, pour localiser itérativement tous les autres.
Notre méthode considère trois types de sous-automates, i) un sous-automate parallèle, qui se compose de deux états connectés par au moins deux transitions, (ii) un sous-automate série, qui se compose au moins de trois états en séquence reliés par une

seule transition, et (iii) un sous-automate minimal qui ne contient ni parallèle, ni série, appelé, "plus petit sous-automate".
Nous avons développé un algorithme qui calcule et factorise d'abord les parallèles et les séries jusqu'à ce que l'automate courant en soit totalement dénué, ensuite, il calcule et factorise les plus petits sous-automates, et ainsi de suite, jusqu'à ce que l'automate donné soit réduit à deux états reliés par une seule transition. Ainsi, on calcule itérativement toute sous-structure minimale et on la remplace par une transition.

Dans le chapitre 3, nous avons appliqué notre méthode à plusieurs automates représentant des dictionnaires électroniques de différentes langues : allemand, anglais, bulgare, français, hongrois et portugais. Les résultats obtenus montrent que ces automates contiennent un nombre considérable de sous-automates. De plus, certains de ces sous-automates possèdent plusieurs niveaux d'imbrications [Tounsi *et al.*, 2006a], [Tounsi *et al.*, 2006b].
La complexité théorique de recherche des sous-automates à partir des séries et des parallèles est de $O(m^2)$, où m représente le nombre de transitions dans A, et, dans le pire des cas, la complexité de l'algorithme de recherche des plus petits sous-automates est de $O(n^3)$, où n représente le nombre d'états de A. Les résultats expérimentaux démontrent que ces algorithmes ont un bon comportement pour les automates représentant des dictionnaires de langue.

Le chapitre 4 est consacré à la présentation d'une palette d'outils pour compresser et indexer des automates représentant des dictionnaires. Nous proposons un algorithme de compression qui permet, d'une part, de réduire l'espace mémoire nécessaire au stockage des automates et, d'autre part, de conserver un accès efficace aux données. Dans ce contexte, nous avons été amenés à développer divers modèles utilisant à la fois l'automate des suffixes pour indexer les sous-automates et diverses heuristiques permettant de sélectionner les sous-structures les plus intéressantes à factoriser (celles qui maximisent le gain de mémoire et réduisent la taille de l'automate initial).
Par extrapolation de l'indexation automatique d'un document texte qui recense toutes les occurrences des mots de ce document, l'indexation automatique d'un automate liste tous ses sous-automates accompagnés de leurs adresses (positions dans l'automate) pour former son index. Ainsi, les sous-automates constituent une information accessible et représentative du contenu de l'automate.
Étant donné que la recherche de sous-automates a révélé la présence massive de sous-automates de type série ou parallèle, nous nous sommes intéressés spécifiquement à ces deux sous-structures et à leur indexation.
Pour indexer ces parties internes d'un automate, nous avons étudié la possibilité d'utiliser des systèmes dédiés initialement à l'indexation des textes. En l'adaptant à nos besoins, l'automate des suffixes (Directed Acyclic Word Graph "DAWG") permet d'avoir accès, non seulement, aux positions des séries et des parallèles dans l'automate, mais aussi, à l'ensemble des positions des sous-séries et des sous-parallèles dans ce même automate.
Pour alléger l'automate, il est nécessaire de diminuer son nombre d'états et son nombre de transitions sous la condition stricte de ne pas modifier le langage initialement reconnu. Ainsi, nous proposons un algorithme glouton pour factoriser l'automate. Sans la possibilité d'un retour arrière, il sélectionne à chaque étape un sous-automate à

factoriser en utilisant une fonction appelée "fonction gain". Cette fonction calcule le gain apporté lors de la factorisation de chaque sous-structure, elle permet ainsi de comparer les plus intéressantes et de sélectionner la sous-structure qui sera factorisée. Ensuite, cet algorithme a été mis en œuvre pour traiter plusieurs compressions possibles : la première s'applique directement sur des automates minimisés, la deuxième considère les automates non minimisés et la troisième traite directement du texte et s'applique aux dictionnaires avant même de générer l'automate associé. Pour pousser plus loin l'expérimentation, nous avons considéré le problème à l'envers en inversant les mots des dictionnaires. Ainsi, les trois méthodes de factorisation ont été, à nouveau, testées sur l'ensemble des données.
Nous terminerons ce document par une discussion autour des résultats de nos méthodes et détaillant les conclusions que nous avons pu tirer de nos recherches et expérimentations.

Chapitre 2

État de l'art

L'objectif initial de cette thèse est, d'une part, d'étudier la structure interne des automates représentant des dictionnaires électroniques et, d'autre part, de rattacher le travail à la problématique de la compression de tels volumes de données.

2.1 Les automates à nombre fini d'états pour le traitement automatique des langues

Les automates sont intrinsèquement liés à l'évolution de l'informatique. En effet, ils sont en étroite connexion avec une grande variété de domaines tels que l'algorithmique, la logique, la vérification de systèmes et des protocoles, les systèmes dynamiques, la combinatoire, la théorie des nombres, le traitement de l'information, le traitement des langues naturelles, etc. C'est ce dernier point qui constitue notre sujet d'application.

Les activités du monde contemporain sont de plus en plus liées à l'information. Le développement des nouvelles technologies de l'information et de la communication, ainsi que l'ouverture des marchés internationaux place l'acquisition, la gestion, l'analyse et l'exploitation des informations au centre des grands débats du monde de l'économie et de la recherche.

Le traitement automatique des langues (TAL) est une discipline qui propose une large gamme d'activités permettant de traiter cette information. Il repose sur la linguistique et l'informatique pour créer des applications capables de traiter des données linguistiques. Différentes approches existent ; parmi celles-ci, un courant privilégie l'utilisation de modèles linguistiques qui s'appuient sur la théorie des automates. Citons par exemple, les Centres de Recherche de Xerox et Bell Labs Research Center. C'est également le cas des travaux du Laboratoire d'Automatique Documentaire et Linguistique (LADL, laboratoire du CNRS) fondé par Maurice Gross en 1969, ou encore, de l'Institut d'électronique et d'informatique Gaspard-Monge (IGM) et du LAboratoire de SEmio Linguistique Didactique et Informatique (LASELDI).

Le TAL permet d'aborder une vaste palette d'applications, allant de la linguistique fondamentale ou appliquée au développement de produits multimédias, en passant par la lexicographie, la terminologie, la traduction automatique ou assistée par ordinateur, etc. Cette dernière est d'ailleurs à l'origine du TAL. Plus loin encore, il y a la reconnaissance

et la synthèse de la parole, le filtrage d'informations et la veille technologique. Toutes ces applications sont utilisées par des industriels et des particuliers.

Quelque soit le domaine informatique considéré, nous rencontrons et utilisons des langages. Un langage est vu comme un ensemble de mots construits sur un alphabet. De manière intuitive, un système de reconnaissance de langages est une machine qui permet de lire des mots. Un **automate** est un modèle abstrait de machine qui peut être vu comme une représentation graphique des données à modéliser. Il possède un alphabet, un ensemble d'états et une relation de transition. Une transition (p, a, q) indique que l'état q est atteint à partir de p en lisant le symbole a de l'alphabet.

L'automate de la figure 2.1 possède deux états p et q, son alphabet est binaire et il reconnaît les mots se terminant par un 0. p représente l'état initial et q l'état final.

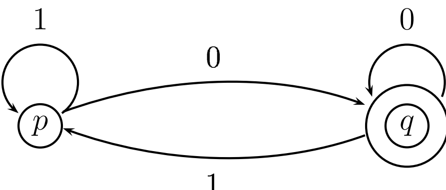


FIG. 2.1 – Automate des mots se terminant par 0

La paternité des automates peut être attribuée à Shannon qui, en 1948, a utilisé un modèle très proche des automates finis [Shannon, 1948]. Étudiés par Kleene dans les années 50 [Kleene, 1956] cette idée a été ensuite reprise par Chomsky en 1956 qui l'applique aux langues naturelles [Chomsky, 1956], [Chomsky, 1957]. Il donne des exemples comme celui de la figure 2.2 pour décrire des phrases simples.

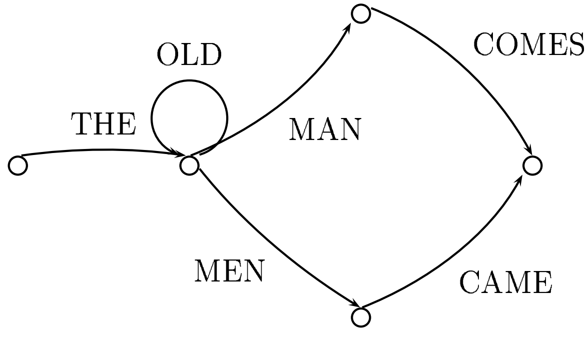


FIG. 2.2 – Automate des phrases simples

Les automates finis ayant l'avantage de se prêter à un nombre considérable de manipulations automatisables, Maurice Gross et Dominique Perrin considèrent qu'ils

fournissent l'essentiel pour la description d'une langue [Gross et Perrin, 1987], [Perrin, 1995].

2.1.1 Automates et dictionnaires

Les dictionnaires électroniques des langues naturelles sont essentiels pour l'analyse et la génération automatique de textes. Ils sont aussi utilisés pour la correction orthographique ou l'extraction d'informations des dictionnaires multilingues et ils permettent également de constituer des programmes d'aide à la traduction. La représentation des dictionnaires par des automates acycliques est très répandue car ils rendent le temps d'accès à un mot linéaire en fonction de sa taille (il ne dépend pas de la taille du dictionnaire, seulement de la longueur du mot).
"Les automates acycliques sont une structure de données bien adaptée à la représentation de lexiques de langues naturelles, $\cdots$ La représentation par automates acycliques est économique en espace, elle permet de réaliser une compression importante des données. Elle fournit également des fonctions d'accès et de recherche de motifs, des plus rapides" [Revuz, 1991].
"Les unités élémentaires des langues sont décrites dans des dictionnaires électroniques $\cdots$ En effet, du point de vue de la description linguistique, les automates finis, même lorsqu'ils sont utilisés pour abréger l'énumération d'ensembles finis, permettent une mise en facteur des objets sans laquelle toute description précise serait impossible. On peut ainsi décrire des faits de langue de manière à la fois systématique et cumulative. D'autre part, les données étant particulièrement volumineuses, les réalisations informatiques nécessitent des formalismes et des algorithmes économes en temps. La représentation des données par automates et transducteurs est remarquablement adaptée à cette contrainte." E.Laporte[1].

On formalise les notions liées aux automates par les définitions suivantes :

Définition 2.1 (Alphabet, mot, langage)
Soit Σ un ensemble fini non vide de symboles appelé alphabet. Les symboles de Σ sont dénommés lettres ou caractères.
Un mot w sur Σ est une suite finie $x_1, ..., x_n$ d'éléments de Σ. w est aussi appelé séquence et sa longueur est notée $|w|$. On note Σ^* l'ensemble des mots définis sur Σ.
Un langage est un sous-ensemble de Σ^*.

Définition 2.2 (Automate fini non déterministe)
Un automate fini non déterministe est un quintuplet $A =< \Sigma, Q, \delta, I, F >$ tel que :
- *Σ est l'alphabet.*
- *Q est un ensemble fini non vide d'états.*
- *δ est une fonction de transition: $\delta : Q \times \Sigma \rightarrow 2^Q$.*
- *I est l'ensemble non vide des états initiaux ($I \subset Q$).*
- *F est l'ensemble non vide des états finaux ($F \subset Q$).*

[1]http ://ladl.univ-mlv.fr/Presentation/ThemesDeRecherche.html, consulté le 19-09-2007.

La fonction δ est étendue aux mots et aux ensembles d'états, $2^Q \times \Sigma^* \rightarrow 2^Q$ telle que :
- $\forall q \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma : \underline{\delta}(q, \epsilon) = q, \underline{\delta}(q, aw) = \bigcup_{q' \in \delta(q,a)} \underline{\delta}(q', w)$
- $\forall Q' \subseteq Q, \underline{\delta}(Q', w) = \bigcup_{q \in Q'} \underline{\delta}(q, w)$

Le langage associé à cet automate, noté $L(A)$, représente l'ensemble des mots w de Σ^* tels que $\underline{\delta}(I, w) \cap F \neq \varnothing$.

Pour un même langage, un automate non déterministe peut être nettement plus petit qu'un automate déterministe. Cependant, le non déterminisme est coûteux en mémoire et en temps de calcul pour conserver les chemins suivis et toutes les alternatives rencontrées.

Définition 2.3 (Automate fini déterministe)
L'automate A est déterministe ssi $|I| = 1, I = \{q_0\}$ et $\delta : Q \times \Sigma \rightarrow Q$ (à chaque couple (q, a) correspond au plus un état q').

Le déterminisme d'un automate assure la séquentialité de l'analyse d'un mot par l'automate car la définition de δ permet de constater qu'un état s'obtient à partir d'un autre état à travers la lecture d'une seule lettre : $\delta(q, a)$ est un singleton, $\forall q \in Q, a \in \Sigma$

Théorème 2.1 ([Myhill, 1957])
Étant donné un automate A, il existe un automate déterministe qui reconnaît le même langage.

Définition 2.4 (Chemin et acceptation dans un automate déterministe A)

- *Soit un mot $w = x_1 ... x_{|w|}$, le chemin de w dans A est une séquence de $|w| + 1$ états $q_0, ..., q_{|w|}$ tel que :*
$$q_{i+1} = \delta(q_i, x_{i+1}), \forall i \in [0, |w| - 1]$$

- *Un mot $w = x_1 ... x_{|w|}$ est accepté par l'automate A si le chemin de w dans A relie l'état initial q_0 à un état final $q_{|w|}$.*

2.1.2 Minimisation des automates

Étant donné un automate déterministe A, la minimisation consiste à calculer un autre automate A' équivalent à A dont le nombre d'états est minimal. Cet automate est unique à un isomorphisme d'automate près. A et A' reconnaissent le même langage.

Définition 2.5
Deux automates A_1 et A_2 sont équivalents s'ils acceptent le même langage, c'est-à-dire si $L(A_1) = L(A_2)$.

Théorème 2.2 ([Moore, 1956])
Étant donné un automate déterministe A, il existe un et un seul automate déterministe minimal A' qui reconnaît le même langage à une renumérotation des états près.

Corollaire 2.1 (Théorème 2.1, Théorème 2.2)
Étant donné un automate A, il existe un et un seul automate déterministe minimal A' qui reconnaît le même langage à une renumérotation des états près.

2.1.2.1 Principe de minimisation

Soit $A =< \Sigma, Q, \delta, q_0, F >$ un automate déterministe à nombre fini d'états. On propose de réduire A et de construire l'automate minimal équivalent à l'aide des définitions suivantes :

Définition 2.6
*Un état $q \in Q$ est **accessible** si il existe un chemin (un mot) partant de l'état initial et menant à q.*

$$q \text{ est accessible si } \exists w \in \Sigma^* : \underline{\delta}(q_0, w) = q$$

*Un état $q \in Q$ est **coaccessible** si il existe un chemin partant de l'état q et allant jusqu'à un état final.*

$$q \text{ est coaccessible si } \exists w \in \Sigma^* : \underline{\delta}(q, w) = q', q' \in F.$$

Un état est inaccessible lorsque aucune transition n'arrive dessus, ainsi, la figure 2.3 représente un automate possédant un état inaccessible q.

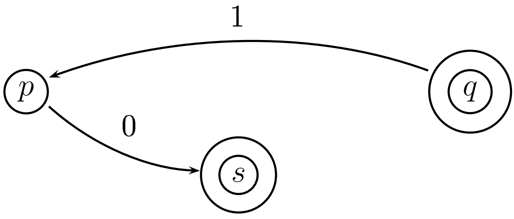


FIG. 2.3 – Automate possédant un état inaccessible

Définition 2.7 *(β équivalence)*
Deux états p et q sont β ***équivalents****, noté $p\beta q$, si on peut atteindre un état final en partant de p ou en partant de q en utilisant le même chemin.*

$$p\beta q \text{ ssi } \forall w \in \Sigma^* : \underline{\delta}(p, w) \in F \Leftrightarrow \underline{\delta}(q, w) \in F$$

Définition 2.8 *(Congruence)*
Soit R une relation d'équivalence sur Q, on appelle R une congruence sur A (automate déterministe complet), si elle vérifie les conditions suivantes :
- $p\ R\ q \Leftrightarrow \forall x \in \Sigma : \delta(p, x)\ R\ \delta(q, x)$
- $p\ R\ q \Rightarrow p \in F \Leftrightarrow q \in F$

Théorème 2.3
La relation d'équivalence β est une congruence d'automates.

Minimiser un automate consiste à diminuer le nombre d'états de telle sorte qu'il n'existe pas deux états β équivalents.
La minimisation d'un automate déterministe A se fait en deux temps, la première étape consiste à supprimer tous les états inaccessibles dans A. La seconde étape repose sur le regroupement des états congruents suivant des classes d'états en utilisant la relation de congruence d'automates β. La relation β permet de rassembler les états en constituant des ensembles de congruences β_i,$i \in \mathbb{N}$.
Initialement, deux classes sont constituées, celle des états finaux et celle des états non finaux. La construction d'un β_i est déterminée à partir de l'étape $i - 1$ et lorsque deux classes successives sont identiques on arrête la construction.
La minimalité de l'automate est liée à la propriété de saturation[2] de la relation β sur l'ensemble des états finaux F.
L'automate déterministe minimal obtenu est $A' =< \Sigma', Q', \delta', q_0', F' >$ tel que **l'alphabet** $\Sigma' = \Sigma$, **l'ensemble des états** Q' représente toutes les classes distinctes calculées, la **fonction de transition** $\delta' : Q' \times \Sigma' \rightarrow Q'$. Le nouvel **état initial** q_0' représente la classe qui contient l'état initial q_0 et les nouveaux **états finaux** représentent les classes qui contiennent les états finaux de départ.

Initialement, les premières recherches sur la minimisation ont été développées par Huffman [Huffman, 1954] et Moore [Moore, 1956] à la fin des années cinquante, ils ont permis le calcul d'un automate minimal. Le principe de leur algorithme de minimisation est de partir d'une partition grossière de l'ensemble des états de l'automate puis, de raffiner cette partition jusqu'à obtenir une partition stable par la fonction de transition.
L'algorithme de minimisation de Hopcroft a l'avantage, par rapport à celui de Moore d'éviter des insertions d'éléments lorsqu'elles sont inutiles et améliore donc les temps de calcul [Hopcroft, 1971], [Gries, 1973], [Blum, 1996].
La complexité de l'algorithme de Hopcroft est de $O(k.nlogn)$ où k représente la taille de l'alphabet de l'automate d'entrée à minimiser A et n son nombre d'états.

Ces travaux ont été repris et plusieurs algorithmes ont vu le jour depuis. L'algorithme de Aho, Sethi et Ullman [Aho *et al.*, 1988], l'algorithme de Hopcroft et Ullman

[2] Une relation R sur un ensemble A **sature** un sous-ensemble E de A : Si $x\ R\ y$ et $x \in E$ alors $y \in E$. La congruence d'automate β sature l'ensemble des états finaux F.

[Hopcroft et Ullman, 1979] ainsi que l'algorithme de Hopcroft [Hopcroft, 1971] utilisent la stratégie décrite plus haut. L'algorithme de Brzozowski [Brzozowski, 1962] utilise une autre stratégie qui permet de calculer, en temps exponentiel, l'automate minimal d'un langage à partir d'un automate non déterministe spécifiant ce langage. En 1995, Watson propose une taxonomie des différents algorithmes existants sur la minimisation dans [Watson, 1995]. Il observe que la majorité des algorithmes de minimisation ont une complexité en $O(|\Sigma| . |Q|^2)$ et emploient une stratégie de point fixe. L'algorithme de Hopcroft est l'algorithme de minimisation ayant la meilleure complexité théorique $O(|\Sigma| . |Q| .log\,|Q|)$.
En 1992, Revuz optimise l'algorithme de Hopcroft pour les automates acycliques [Revuz, 1992] et présente un algorithme de minimisation linéaire des automates acycliques qui procède en trois étapes : i) il construit un arbre lexicographique, ensuite, ii) il trie les états de l'arbre par hauteur et enfin, iii) itérativement sur la hauteur, il fusionne les états de même hauteur, même contenu et mêmes transitions sortantes. Cette méthode de minimisation est rapide mais gourmande en mémoire.

En 2000, a été proposé un algorithme incrémental de minimisation utilisant une stratégie différente [Daciuk *et al.*, 2000]. Le principe de cet algorithme est d'ajouter les mots les uns après les autres à l'automate. Ainsi, avant d'insérer un mot, la méthode recherche les plus longs préfixes et suffixes communs entre le mot et l'automate qui ne se chevauchent pas, ensuite, elle ajoute ce qui reste du mot entre la fin du préfixe et le début du suffixe. La différence majeure avec ce qui a été proposé avant est qu'il est possible d'obtenir un automate partiellement minimisé en interrompant l'algorithme. Cet algorithme est économe en mémoire, mais plus lent en temps de calcul.

La taille d'un automate déterministe peut être réduite à celle de l'automate déterministe minimal, et ce, même lors de sa construction, mais celle-ci reste importante. Prenons un exemple simple. En français, un dictionnaire de mots monolexicaux contient environ huit cent mille mots et on dépasse le million avec les mots polylexicaux. L'automate déterministe minimal du dictionnaire français DELAF [Courtois et Silberztein, 1990] comprend 67 995 états et 177 465 transitions.
Une possibilité pour réduire la taille de stockage des données est d'utiliser certaines techniques de compression.

2.1.3 Représentation et compression des automates

2.1.3.1 Généralités autour de la compression

Un fichier compréssé permet de minimiser les coûts de son stockage en mémoire, ses temps de transfert et de récupération [Salomon, 2000], [Sayood, 2000] et [Barlaud et Labit, 2002].

Il existe un grand nombre d'algorithmes de compression de données, on y distingue deux catégories. Les algorithmes avec perte (*lossy* en anglais) et les algorithmes sans perte (*lossless* en anglais). Cette différence se retrouve lors de la décompression, opération inverse qui permet de retrouver le fichier originel.

La compression avec perte s'effectue généralement dans le domaine du traitement des images et du son car la perte d'information sur ce type de données n'influe pas trop sur les informations récupérées après décompression. Partant du constat que l'ouïe et la vision humaines sont limitées et distinguent difficilement des nuances de couleurs proches ou des tonalités de son voisines, le but de cette compression est de réduire la taille des données en conservant les informations les plus pertinentes pour leur reconstitution. Ainsi, les formats **jpeg** pour les images, **mp3**, **ogg** ou **wma** pour le son ont été développés pour compresser efficacement les données en autorisant des pertes qui n'altèrent que faiblement l'image et le son pour l'œil et l'oreille humaine.

La compression sans perte est nécessaire pour traiter des données des langues naturelles. Une fois décompressé, un texte compressé doit pouvoir être reconstitué à l'identique. C'est une compression qui garantit l'intégrité des données lors de la décompression.
Les algorithmes adaptés pour traiter les textes en langue naturelle fonctionnent sur deux niveaux. Le premier analyse l'ensemble des éléments qui constituent un texte, mots et séparateurs, en les découpant en caractères (unité atomique de traitement). Le second, considère et traite le texte en entier en le découpant en mots.

Modèle général de compression de texte

En mémoire virtuelle, un texte représente une donnée numérique constituée d'une suite de caractères de longueur fixe, codés en binaire (Alphabet=$\{0,1\}$). La compression de texte passe par les trois étapes suivantes :

1. **Modélisation :** Cette étape consiste à choisir une unité de traitement du texte (caractère, syllabe, etc.) puis procède à son découpage.

2. **Factorisation :** Cette étape recherche et rassemble les éléments redondants.

3. **Codage :** Cette phase repose sur le choix d'un codage adéquat pour représenter chaque élément.

Le taux de compression représente la performance d'un algorithme de compression, il s'obtient en divisant la taille du fichier initial par rapport à la taille du fichier compressé.

$$Taux\ compression = Taille\ fichier\ compressé\ /\ Taille\ fichier\ initial \tag{2.1}$$

Les travaux de Martineau dans [Martineau, 2001] présentent et comparent de nombreux algorithmes de compression de textes en langue naturelle. Il s'intéresse particulièrement aux algorithmes utilisant les mots comme unité de base. Il y décrit aussi les résultats d'applications sur des entrées réelles.

Codage des données

Shannon [Shannon et Weaver, 1949] a mis au point la théorie du codage et de la compression. Il a associé à une séquence de mots aléatoires son entropie. Guiasu a défini

la notion de probabilité de l'information par une classe d'algèbres booléennes, d'où il est possible de construire une axiomatique des événements élémentaires, en attachant à chaque élément une classe de probabilité [Guiasu et Theodorescu, 1971]. Cette approche a fait naître un théorème majeur en théorie de la décision[3], "le maximum d'entropie[4]" [Réfrégier, 1993], [Brémaud, 1984]. Les livres [Williams et Sloane, 1978] et [van Lint, 1982] détaillent la théorie des codes.

Pour trouver le meilleur code pour une source donnée, l'inégalité de **Kraft-McMillan** impose une contrainte sur la longueur des mots des codes uniquement décodables[5]. Dans le cas binaire (utilisé pour les textes), une séquence codée de longueur ℓ peut se décomposer en k mots de codes ($\ell = \ell_1 + \ell_2 + .. + \ell_k$) lorsque la condition suivante est vérifiée :

$$\sum_{i=1}^{k} 2^{-\ell_i} \leq 1$$

Cette condition est nécessaire et suffisante pour la décomposition ; des exemples explicites sont données dans [Martineau, 2001].

Il y a deux types de codage standards selon les fichiers à traiter.

1. **Codage de longueur fixe :**
   Après avoir segmenté le texte d'entrée en caractères et défini q caractères distincts, le codage à longueur fixe associe le même nombre de bits à chacun des q caractères détectés. Le codage global utilisera $|Log_2(q)|$ bits.
   Udi Manber utilise cette technique de codage dans son algorithme de compression [Manber, 1993]. Il va même plus loin en utilisant les codes ASCII non utilisés, dans le texte d'entrée, pour coder des digrammes[6] redondants. Il divise ainsi en deux le coût de leur stockage.
2. **Codage de longueur variable :**
   Le codage à longueur variable examine les q caractères distincts détectés dans un texte et étudie leur fréquence d'apparition. Un code est affecté à un caractère en fonction de sa fréquence.
   Un codage de ce type a été établi par Samuel F.B Morse en 1838 ("code Morse"). Il utilise uniquement des points et des tirets pour représenter 42 caractères.
   L'algorithme de Huffman utilise le même principe en associant les codes les plus courts aux lettres les plus fréquentes. En anglais, les caractères "e" et "t" sont les plus redondants dans un texte, ainsi, il leur sera associé le code le plus court possible.

[3]La théorie de la décision est une théorie de mathématiques appliquées ayant pour objet la prise de décision en univers risqué (http ://fr.wikipedia.org, consulté le 19-09-2007.)

[4]L'entropie d'une source est la quantité moyenne d'information contenue dans cette source. En pratique, elle représente une mesure de l'incertitude liée à la probabilité d'occurrence des événements [Shannon, 1948].

[5]Un code permettant un décodage sans ambiguïté est qualifié d'uniquement décodable.

[6]Un digramme se compose d'une suite de deux caractères.

Algorithme de compression de type statistique

Le code Shannon-Fano [Shannon et Weaver, 1949] a été le premier algorithme statistique à connaître un succès important dans le domaine de la compression. Il reprend l'idée du code morse mais utilise un codage différent pour chaque texte, selon le nombre d'occurrence de chaque caractère. Le but est donc de trouver un codage d'un fichier qui tienne compte de la fréquence d'occurrence de chaque élément du fichier. Cependant, il a rapidement été concurrencé par la méthode de Huffman pour les traitements informatiques.

1. **Algorithme de Huffman**
   Cet algorithme a été développé en 1952 par David Huffman. Le codage employé est "entropique à taille variable" [Huffman, 1952], il se base sur les statistiques d'apparitions des différents octets du fichier et chacun de ces octets est codé selon une suite de bits, dont la taille diffère selon l'octet. Ainsi, une valeur fréquente est représentée par un petit nombre de bits, et une valeur peu fréquente par un grand nombre de bits.
   L'algorithme de Huffman construit un arbre de codes à partir d'une table d'occurrences (ou fréquences) de chaque caractère. La construction de l'arbre est itérative : initialement il crée un nœud terminal pour chaque entrée de la table des occurrences, et, à chaque étape, il regroupe les deux arbres dont le poids est minimal. Le codage préfixe est obtenu en numérotant chaque paire de branches par 0 et 1.

Codage préfixe

Le codage à taille variable ne peut être arbitraire car un mauvais choix de codage peut engendrer des ambiguïtés lors du décodage.
L'exemple de codage donné par la table 2.1 associe un code unique aux nombres 0, 2, 4 et 6. Supposons que la séquence à décoder soit "11011110". Il est impossible de retranscrire le code initial car deux interprétations sont possibles (442 et 406).
Le problème est que certains codes correspondent au début d'autres codes. Le "01" est le code du nombre 0 et le début du code du nombre 4.
Pour résoudre ce problème, il faut absolument un codage dans lequel aucun code n'est le début d'un autre. Cette technique est appelée **code-préfixe**.

	0	2	4	6
Codage	01	110	011	1110

TAB. 2.1 – Exemple de codage

L'algorithme de Huffman génère un code-préfixe à taille variable, c'est une méthode classique de construction de code optimal, qui utilise des arbres binaires[7].

Proposition 2.1 ([Huffman, 1952])
Un code optimal pour un fichier est toujours représenté par un arbre binaire complet, c'est-à-dire un arbre dont tous les noeuds ont, soit deux fils différents de l'arbre vide, soit deux fils vides.

La figure 2.4 représente l'arbre binaire de codage du mot "anticonstitutionnellement". Ce mot est constitué de 25 caractères, chaque

[7] Un arbre binaire est, soit l'arbre vide, soit un arbre dont chaque noeud contient exactement deux fils, eux mêmes arbre binaires.

caractère est codé par un octet de 8 bits (codage ASCII), ce qui représente 25 octets (200 bits) au total. La table 2.2 donne la fréquence de chaque caractère du mot.

	a	c	s	u	m	o	ℓ	i	e	n	t
Fréquence	1	1	1	1	1	2	2	3	3	5	5

Tab. 2.2 – Fréquence des caractères du mot "anticonstitutionnellement"

La construction de l'arbre de codage 2.4 est itérative ; initialement il faut créer un noeud terminal pour chaque entrée du tableau 2.2, ce qui correspond à 11 arbres contenant un seul noeud chacun. Ensuite, à chaque itération on supprime les deux arbres de plus petite fréquence et on les remplace par un "arbre somme". La construction s'arrête lorsqu'il ne reste qu'un seul arbre, celui de la figure 2.4.

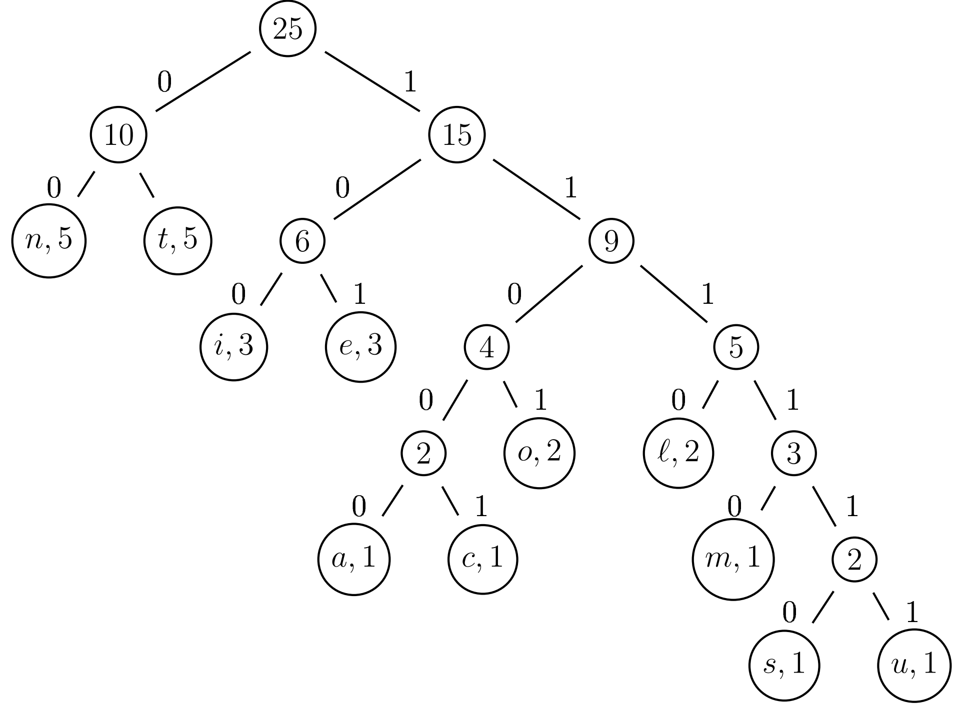


Fig. 2.4 – Arbre binaire du mot "anticonstitutionnellement"

La table 2.3 représente le code associé à chaque caractère, il correspond au chemin parcouru de la racine au noeud terminal correspondant, en notant les successions de "0" et de "1". Le codage du mot "anticonstitutionnellement" est :
"110000001100110011101001111100110001111111011001101000010111101110101111101010001"

Ce codage utilise 81 bits au lieu des 200 de départ, ainsi, le taux de compression est de 60 %.

	n	t	i	e	a	c	o	ℓ	m	s	u
Code	00	01	100	101	11000	11001	1101	1110	11110	111110	111111

Tab. 2.3 – Codage des caractères du mot "anticonstitutionnellement"

Le codage représenté par l'arbre binaire assure qu'aucun code de caractère ne peut

être le préfixe d'un autre. Ce codage est incorporé au fichier compressé, de manière à ce que le décryptage soit possible (sa taille est négligeable par rapport à l'ensemble du fichier). Le coût de la construction du code est en $O(n\ log\ n)$ pour un alphabet à n symboles.

2. **Algorithme de Shannon-Fano**
   Cet algorithme procède par dichotomie en divisant récursivement la table de fréquences, la partition est effectuée de manière à ce que deux noeuds fils d'un même parent aient la plus proche valeur possible. Le codage est le même que celui de Huffman.

   La mise en oeuvre de Shannon-Fano est différente de celle de Huffman parce que l'algorithme de Shannon-Fano est récursif. À chaque étape, il faut découper la table des fréquences en deux, alors que Huffman réitère en ajoutant des éléments à la table. D'autre part, dans l'algorithme de Shannon-Fano on peut hésiter entre plusieurs partitions pour une même table lorsqu'elles sont équivalentes. La sélection concerne alors les deux groupes le plus proche de 1 en termes de fréquence.
   Du point de vue efficacité, l'algorithme de Shannon Fano donne de meilleurs résultats quand la dispersion des fréquences est importante.

   Ces deux algorithmes ne sont plus utilisés directement, mais en combinaison avec d'autres algorithmes pour créer de nouveaux codeurs et atteindre de meilleurs résultats ; on peut citer les travaux publiés dans [Nelson, 1992], [Burrows et Wheeler, 1994] et [Balkenhol *et al.*, 1999].
   Pour corriger les inconvénients majeurs de l'algorithme de Huffman, c'est-à-dire la nécessité de lire le texte entièrement avant de lancer la compression et transmettre le code trouvé, une version adaptative a été proposée et développée dans [Faller, 1973], [Gallager, 1978] et [Knuth, 1985].
   Une autre évolution de l'algorithme de Huffman est d'utiliser les automates comme structure de donnée pour éviter des problèmes d'espace mémoire et temps de traitement, les travaux de Tanaka [Tanaka, 1987] et Sieminski [Sieminski, 1988] font appel aux structures d'automate.

   Il a été prouvé que le codage de Huffman est la meilleure méthode de codage à taille variable. Mais les codes de Huffman doivent avoir un nombre entier de bits, ce qui ne permet pas toujours de réaliser une compression optimale. Le codage arithmétique est alors une méthode plus performante. Il a été suggéré par Rissanen, Langdon et Guazzo [Rissanen et Langdon, 1979], [Guazzo, 1980] et mis en pratique par Witten, Neal et Clarly [Witten *et al.*, 1987]
   Dans cette méthode, chaque symbole du texte est entré avec sa probabilité d'occurrence comprise entre 0 et 1 (en commençant par celui qui a la probabilité la plus élevée), et le codage se traduit par l'affectation d'un nombre unique, à virgule flottante, compris entre 0 et 1, à l'ensemble des symboles. De nombreux modèles de probabilité ont été proposés : [Moffat, 1990], [Cleary et Teahan, 1997], etc.

Algorithme de compression par facteurs

En 1977-1978, Jacob Ziv et Abraham Lempel développent un algorithme de compression basé sur la redondance de séquences de caractères. Il s'agit là, d'une compression sans perte dans la mesure où les données ne sont pas altérées, ni par la compression, ni par la décompression. En 1984, Terry Welch [Welch, 1984] améliore l'algorithme précédent et dépose un brevet : c'est la naissance de l'algorithme LZW.

Cet algorithme est à la base de tous les algorithmes par dictionnaire. Le principe d'un **algorithme par dictionnaire** est de remplacer des séquences qui apparaissent plusieurs fois par un symbole qui sera stocké dans une table appelée dictionnaire. Ainsi, les valeurs des symboles rencontrés dans la partie décompression sont récupérés directement dans le dictionnaire.

L'algorithme de Lempel Ziv Welch repose sur une analyse par dictionnaire. Initialement, ce dictionnaire contient les 255 éléments du code ASCII, ensuite, le parcours des données permet d'enregistrer chaque nouveau symbole en lui affectant une nouvelle position dans le dictionnaire.
Comparé à l'algorithme de Huffman, le gros avantage de cet algorithme est qu'il peut travailler sur des données comme elles arrivent : pas besoin d'analyser toutes les données à l'avance. De plus, il n'est pas nécessaire de transférer le dictionnaire pour pouvoir décompresser le texte transmis car le dictionnaire se reconstruit en fonction des nouveaux symboles rencontrés. Le décodage de données compressées lit donc un code et l'ajoute au dictionnaire s'il n'est pas encore présent. Le code est ensuite traduit par la chaîne de caractère qu'il représente et est écrit dans le fichier de sortie non compressé.
D'autres avantages de cette méthode sont qu'elle est facile à programmer, qu'elle compacte différents types de données et que la taille des dictionnaires est modifiable (elle dépend de la mémoire disponible).
La première version LZ77 [Ziv et Lempel, 1977] était utilisée pour l'archivage, son principe est de garder en mémoire les données déjà rencontrées, et quand on rencontre une phrase déjà vue, on la supprime pour ne garder que la position de sa première occurrence (les formats ZIP, ARJ et LHA l'utilisent). La version suivante LZ78 [Ziv et Lempel, 1978] est spécialisée dans la compression d'images et de tout fichier de type binaire, cet algorithme de compression s'appuie aussi sur l'utilisation d'un dictionnaire. Lorsqu'une phrase du dictionnaire est rencontrée, on la remplace dans le fichier compressé par son index dans le dictionnaire. Lors de la décompression, l'algorithme reconstruit le dictionnaire dans le sens inverse, ce dernier n'a donc pas besoin d'être stocké.

Principe de fonctionnement :
Étant donné une source de donnée représentant un ensemble d'octets S (le plus souvent des caractères), la compression passe par les quatre étapes suivantes :
- **Étape 1 :** Séparer la source en mots tel que chaque mot soit la plus petite chaîne de caractères non observée jusqu'à présent. Le premier mot est la chaîne vide ϵ.
- **Étape 2 :** Indexer tous les mot de S. ϵ possède le numéro 0.
- **Étape 3 :** Numéroter les sous-chaînes par les indices correspondants.
- **Étape 4 :** Remplacer les index par la sous chaîne correspondante.

Un arbre de recherche des index est alors construit de manière itérative en étiquetant la racine par 0, chaque noeud par un index et chaque branche par une lettre.

Pour trouver le contenu d'un index, il faut parcourir le chemin depuis le noeud correspondant jusqu'à la racine.
Cette méthode a plusieurs points forts, car elle ne nécessite aucune connaissance de la source, elle transforme des suites variables de symboles en des codes de longueur fixe et le dictionnaire est reconstruit à partir du fichier compressé.

Exemple 2.1
Soit S=abracadabra, les étapes 1 et 2 décomposent S en 7 sous-séquences, la première sous-séquence de S est le mot vide ϵ, ensuite, chaque sous-séquence est identifiée par une unique référence (création de l'index). L'étape 3 permet de numéroter les sous-séquences par les indices correspondants et l'étape 4 produit le codage. Il n'utilise aucun bit pour coder la première sous-séquence, 1 bit pour coder la deuxième sous-séquence, deux bits pour coder la troisième et quatrième sous-séquence et enfin, trois bits pour coder les trois dernières sous-séquences. (voir le tableau 2.4).

Étape 1	Étape 2	Étape 3	Étape 4
ϵ	ϵ →0		
a	a→1	0a→1	a→1
b	b→2	0b→2	1b→2
r	r→3	0r→3	00r→3
ac	ac→4	1c→4	01c→4
ad	ad→5	1d→5	001d→5
ab	ab→6	1b→6	001b→6
ra	ra→7	3a→7	011a→7

TAB. 2.4 – Compression LZ "abracadabra"

Théorème 2.4 ([Welch, 1984])
Pour toute source texte avec des symboles 1,.., n indépendants, aléatoirement répartis, avec des probabilités d'occurrence p_1,..., p_n, le nombre de bits nécessaires pour coder la source tend vers son entropie.

D'autre part, on peut combiner l'algorithme LZW et un algorithme de codage statistique. C'est le cas des programmes **ARJ** ou **PKZIP** qui utilisent LZW puis Shanon-Fano.

2.1.4 Compression et automates représentant des langues naturelles

[Hamrouni, 1996] L'étude proposée par Hamrouni affirme l'utilité des approches par dictionnaires de formes pour représenter et comprimer de gros volumes de données lors de la construction de certaines applications multilingues en TALN (détection/correction orthographique, reconnaissance de la parole, etc.). En effet, partant du constat que, à part les méthodes utilisant les automates d'états finis déterministes, la plupart des méthodes classiques sont peu efficaces pour comprimer de grands dictionnaires, il définit une approche de compression de dictionnaire de formes qui procède par factorisation d'un ensemble d'affixes appelé "paradigme" . Cette méthode n'utilise que des connaissances morphologiques élémentaires pour être efficace. Les expérimentation ont permis de réduire, pour certaines langues, le dictionnaire de formes initial à un dictionnaire pratiquement de même taille que le dictionnaire de lemmes. Cependant, en moyenne, le gain apporté n'est pas très important lorsqu'il s'agit d'automate déterministe représentant des dictionnaires de formes ; de plus, l'accès aux données perd en performance. Toujours est-il que cette approche se justifie pour d'autres types de structures : hachage, arborescence, etc. Et permet de combler les lacunes de ces méthodes pour comprimer de grands dictionnaires de formes. Cette étude a aussi traité le cas des automates non minimisés.

Un autre exemple de combinaison de méthodes pour de la compression est donné dans [Ristov et Laporte, 1999] : les auteurs représentent de grandes banques de données en langues naturelles par un arbre lexicographique mis sous forme de liste chaînée et compressé suivant la méthode LZW, sachant que la recherche de répétitions au sein de l'arbre se fait en utilisant les tableaux de suffixes. La structure de données utilisée rend la méthode très efficace en espace et en temps d'accès.

[Ristov et Laporte, 1999] : L'algorithme de compression proposé dans cette étude représente les données sous forme hiérarchique en utilisant un "Trie[8]" T, c'est-à-dire arbre lexicographique déterministe, et associe à chaque élément (état) de T les quatre informations suivantes :

1. Un caractère noté "a" représentant l'étiquette portée par l'état ($a \in \Sigma$).
2. Un booléen noté "f" indiquant si l'état est final ou non.
3. Un booléen noté "c" indiquant que l'état est final mais pas terminal (il est inclus dans une plus grande séquence que celle qu'il reconnaît).
4. Un pointeur (valeur relative) qui indique s'il y a un choix de chemins et pointe donc vers la branche suivante.

L'application de la compression passe par les étapes suivantes :

- Codage de chaque élément de T en prenant soin de remplacer les éléments identiques par le même symbole, cette représentation construit une séquence notée "LLT[9]".
- Création de la table des suffixes de la séquence LLT.
- détection des répétitions en utilisant une table des suffixes[10].

[8]Un Trie T est un automate déterministe à multi états finaux.

[9]$LLT = u_0 u_1 ... u_n$, $u_i \in U(alphabet)$, $n = |LLT|$, $u_i = a_i f_i c_i l_i$ telque : $a_i \in A$, $f_i \in \{0,1\}$, $c_i \in \{0,1\}$ et $0 \prec l_i \prec n$.

[10]La table des suffixes contient tous les suffixes d'un mot placés selon l'ordre lexicographique.

- Compression de la séquence LLT.
- Application de la compression à T pour générer une nouvelle structure composée des parties non factorisées de T auxquelles s'ajoutent deux types de pointeurs. Un premier pointeur qui remplace une branche complètement factorisée et un second pointeur qui remplace une partie d'une branche munie de sa longueur. Cependant, cette compression s'applique lorsque les trois règles suivantes sont satisfaites :
  - Règle 1 : Les sous-structure factorisées ne peuvent pas se recouvrir.
  - Règle 2 : Une sous-structure incluant un choix de branche ne peut pas être compressée.
  - Règle 3 : Une sous-structure incluant l'arrivée d'un choix de branche ne peut pas être compressée.

Les résultats obtenus démontrent que cette méthode de compression est efficace pour réduire de gros volumes de données langagières et surtout lorsqu'elle est appliquée pour traiter des langages flexionnels comme le français. Dans [Ristov, 2005] Ristov démontre la supériorité de cette méthode de compression de Trie par rapport à plusieurs autres méthodes de compression de lexiques tel que [Ciura et Deorowicz, 2001] . Cependant, on peut signaler que le taux de compression dépend du stockage des données, en effet, compresser la même liste de mots dans un ordre différent ne donne pas forcément le même résultat.

[Daciuk, 2000] Les travaux de Daciuk présentent plusieurs expérimentations de diverses méthodes de compression pour réduire des automates à transitions finales, minimaux, déterministes, et acycliques représentant des dictionnaires de langue.
Initialement, il décrit un automate par l'ensemble des transitions sortantes de chaque état telle que chaque transition comporte quatre informations : 1) une étiquette, 2) un pointeur vers son état d'arrivée, 3) le nombre de transitions sortantes de son état d'arrivé et 4) un booléen pour indiquer la nature de la transition (final ou pas).
Ces méthodes de compression sont les suivantes :

- **Compression 1 :** Pour gagner en espace mémoire, cette méthode élimine le compteur de transitions (2) et le remplace par un booléen pour indiquer si la transition est la dernière de l'état courant.

- **Compression 2 :** Pour économiser de l'espace, cette méthode représente une seule fois les transitions portant les mêmes étiquettes et aboutissant au même état d'arrivée. Cette méthode permet donc de réduire l'ensemble des transitions de l'automate.

- **Compression 3 :** Pour réduire la taille occupée par l'automate Daciuk s'appuie sur les recherches de Kowaltowski dans [Kowaltowski *et al.*, 1993] pour réorganiser l'automate et stocker les transitions formant une série les unes derrière les autres. Cette réorganisation permet de remplacer le pointeur (4) associé à chaque transition appartenant à une série par un booléen (le pointeur qui référence l'état d'arrivée de la transition).

- **Compression 4 :** Démarrant de la représentation obtenue par la compression 1 qui permet de réduire une partie des états de l'automate (les états de départ des transitions aboutissant au même état d'arrivée et portant les mêmes étiquettes), la compression 4 traite à nouveau ces états lorsqu'ils possèdent encore des transitions sortantes qu'ils ne partagent pas. Elle réduit les transitions restantes en utilisant un booléen et un pointeur.

- **Compression 5 :** La compression 1 et la compression 4 allègent l'automate en supprimant certaines transitions. Cependant, cette surpression est conditionnée par la structure de l'automate. En effet, l'ordre dans lequel sont stockées les transitions peut influencer sur la consommation mémoire. D'après l'étude menée par Kowaltowski [Kowaltowski *et al.*, 1993], disposer les transitions sur chaque état par ordre décroissant selon la fréquence de leurs étiquettes permet une meilleure compression.

Ces méthodes ont toutes été testées sur les automates (à transitions finales) représentant différentes langues naturelles et les résultats obtenus montrent qu'il n'y a pas une compression optimale pour traiter tous les dictionnaires. Selon le dictionnaire, certaines méthodes peuvent être plus efficaces que d'autres. Les compressions les moins performantes sont les compressions 3 et 4.

Discussion

Bien que naturellement non exhaustif, le panorama des différents travaux présentés portant sur la minimisation et la compression des automates a permis d'établir quelles sont les différentes méthodes existantes. L'intérêt d'une telle démarche étant de pouvoir positionner nos travaux par rapport à ceux-ci.
Néanmoins, cet état de l'art soulève certaines interrogations quant à l'adaptabilité des méthodes décrites à des structures comparables, par exemple, aux automates sur lesquels se porte notre travail. Aussi, serait-il possible d'identifier des sous-automates correspondant à notre définition en utilisant ces procédés ?
Une analyse sera développée lorsque toutes les définitions seront présentées.

L'étude de la structure interne d'un automate nous a conduit à l'indexation, mais aussi à l'étude des redondances dans l'automate. Ainsi, nous étudions, d'une part, la possibilité d'indexer des automates et, d'autre part, la possibilité d'utiliser des algorithmes dédiés initialement à l'indexation des textes pour le faire. Aussi, nous nous sommes dirigés vers la recherche des solutions existantes pour le calcul de redondance dans un texte comme dans l'étude menée par Crochemore sur la recherche de motifs (*pattern matching*) [Crochemore et Hancart, 1997]. Par ailleurs, la recherche de sous-structures dans un automate que nous avons mise en œuvre rappelle le problème de recherche de composants dans un graphe : nous rappelons ce cadre général et montrons que notre problème ne s'y réduit pas.

2.2 Recherche de motifs (fréquents)

Soit un alphabet fini Σ constitué de caractères. Un langage L est une partie de Σ^* et un mot w est une suite finie de ses caractères. $w = w_1...w_n \in \Sigma^*$. le problème consiste à découvrir tous les sous-mots v de w qui appartiennent à L.
Ce problème possède $O(n^2)$ solutions en faisant référence à a^* à partir du mot a^n.
L'approche consiste à chercher toutes les positions dans le mot initial terminant les sous-mots reconnus.

$$Pos(L,w) = \{j \in [1,n] \,/ \exists i \in [1,j]\,, w_i w_{i+1}...w_j \in L\}.$$

Lorsque L est un singleton, la recherche de motifs devient un problème de recherche au sein d'un mot de longueur n. Une solution algorithmique primitive est d'énumérer la séquence. Les temps de calcul des algorithmes de Knuth-Morris-Pratt ou de Boyer-Moore sont d'au moins $O(n)$.

Pour un ensemble fini de mots tel qu'un dictionnaire, une approche indexée est plus appropriée. Les arbres des suffixes et les DAWG (directed acyclic word graphs) permettent d'effectuer des pré-traitements sur le texte en $O(n)$ dans le but de simplifier les requêtes ultérieures [Holub et Melichar, 2000].

2.2.1 Indexation de textes par des automates $DAWG$

Aujourd'hui la plupart des textes ou documents anciens ou actuels sont numérisés, la difficulté à présent est de créer des outils performants pour rechercher et extraire des informations dans ces documents. Rechercher une séquence de caractères dans un corpus textuel exige un temps linéaire en la taille du corpus. Dans le cas de multiples recherches, il est plus avantageux de construire une structure de données, nommée index, capable d'accélérer la recherche de n'importe quel motif dans le texte.

L'index d'un document est une liste organisée de descripteurs représentatifs de son contenu, accompagnés de références aux zones d'apparition [Curcio et Chauvin, 1987] et [Nazarenko et Mekki, 2005]. Il figure généralement à la fin du document et le représente de manière synthétique [Bellot, 2000].
Le but de l'index est de permettre un accès rapide à l'information mais, le coût de sa construction est rédhibitoire. Le travail présenté dans [Gros et Assadi, 1997] permet de constituer l'index d'un document technique, la méthode impose un bonne connaissance du domaine et ne permet pas d'envisager un outil générique. Une démarche plus globale est présentée dans [Bourigault et Charlet, 1999]. Elle permet de former un index thématique à partir du document à indexer en utilisant des outils terminologiques. Cette idée à été reprise dans [Mekki et Nazarenko, 2001] pour développer un outil de construction et de consultation d'index devant faciliter la navigation dans les documents.

Plusieurs applications tiennent leur efficacité d'index de textes : la recherche dans les dictionnaires, l'exploration de bases de données, la compression, la fouille de données, etc. Ces études ont réalisé des variantes d'index adaptés au type de recherches : le trie [Aho *et al.*, 1974], la table des suffixes [Manber et Myers, 1990], les arbres [Ukkonen, 1995], le Directed Acyclic Word Graph (DAWG)

[Crochemore et Rytter, 1994], [Crochemore *et al.*, 2001] et le String B-arbre (SB tree) [Ferragina et Grossi, 1999]. L'objectif de ces structures est le développement d'un index *dense* contenant un maximum d'informations, facile à mettre à jour et peu gourmand en mémoire.

Soit T un texte de longueur n sur l'alphabet Σ, un "bon index" de T doit satisfaire des critères de temps de calcul et d'accès mémoire. Il doit pouvoir être construit en $O(n)$, occuper une mémoire de l'ordre de $O(n)$ et permettre la recherche de motifs de taille m en temps $O(m)$.
Une question se pose sur le stockage de tous les "facteurs" du texte T.

Définition 2.9
Un facteur f de T est un mot qui apparaît dans T tel que :

$$\exists 1 \leq i \leq j \leq n : T_{(i,j)} = f$$

Pour une indexation efficace, nous nous sommes intéressés aux méthodes de recherche exactes utilisant des structures de données les mieux adaptées et les plus simples, tant du point de vue de l'espace mémoire occupé, que du point de vue de rapidité de traitement qu'elles procurent, ces structures sont appelées arbres ou automates de suffixes. En effet, elles ont l'avantage primordial d'avoir une taille linéaire par rapport au texte de base. De plus, le temps d'accès à un facteur du texte se fait en temps linéaire par rapport à la taille de ce facteur.
Cependant la construction d'un arbre des suffixes en temps linéaire n'est pas triviale et l'implémentation d'un algorithme qui le réalise proprement non plus.

A côté du développement des arbres de suffixes, qui sont une compression de l'arbre de base, les automates de suffixes ont fait leur apparition en tant que minimisation de cet arbre de base. Ils ont été introduits par Blumer [Blumer *et al.*, 1985], [Blumer *et al.*, 1987] et appelés DAWG d'après leur nom anglais (Directed Acyclic Word Graphs). Les auteurs ont proposé à cette occasion un algorithme linéaire pour le construire. Contrairement aux arbres des suffixes[11] qui sont de taille quadratique, ces automates sont construits de façon totalement séquentielle [Vérin, 1998].

Le DAWG est un automate des suffixes qui représente une structure efficace pour le traitement et l'analyse de répétitions dans un texte. L'algorithme repose sur une lecture de gauche à droite du texte pour construire l'automate minimal des suffixes en temps linéaire. Il représente ainsi le texte sous forme d'un graphe.

2.2.2 Notations de base

Soit Σ un alphabet fini non vide, composé de caractères ou lettres. Soit u et v deux mots de Σ^*. u_i représente la i^{eme} lettre de u.
On appelle concaténation de u et v le mot x tel que : $x = uv = u_{1 \ldots |u|} v_{1 \ldots |v|}$
Soit $x = wyz$ avec $w, y, z \in \Sigma^*$, on définit alors les notions suivantes :

- w est un préfixe de x.

[11] L'inconvénient majeur de l'arbre des suffixes d'un mot est sa taille.

– z est un suffixe de x.
– y est un facteur de x.
On notera que w et z sont aussi des facteurs de x.
On note S(x) l'ensemble des suffixes de x.

2.2.3 Directed Acyclic Word Graphs

L'automate des suffixes d'un mot x, noté $DAWG(x)$, est le plus petit automate déterministe acceptant pour états terminaux l'ensemble $S(x)$ des suffixes du mot x. Il est construit sur la congruence syntaxique droite de l'ensemble S(x) et permet de vérifier si un mot y est un facteur de x en un temps $O(|y|)$.
Un algorithme séquentiel permet de construire l'automate des suffixes pour un et plusieurs mots, cet automate est linéaire en nombre d'états et de transitions. [Blumer *et al.*, 1989] présentent une analyse du nombre moyen d'états et de transitions utilisant un modèle d'équiprobabilité et d'indépendance des lettres et Raffinot développe une analyse du nombre moyen d'états terminaux [Raffinot, 1997].

Théorème 2.5 ([Crochemore et Vérin, 1997a])
La taille du DAWG d'un mot x est en $O(|x|)$, son temps de calcul est aussi en $O(|x|)$. Le nombre maximum des états de cet automate est $2 \times O\,|w| - 1$ et le nombre maximum de ses transitions est $3 \times O\,|w| - 4$.

Le DAWG est une structure de données efficace pour la représentation du lexique [Aho *et al.*, 1974], [Revuz, 1992], la recherche rapide de séquences [Aho *et al.*, 1974], [Blumer *et al.*, 1984], [Crochemore et Vérin, 1997a] et possède une grande variété d'applications s'étendant du traitement de la parole [Lacouture et Mori, 1991] à l'analyse d'ADN et la visualisation des séquences répétées dans les génome, [Durand *et al.*, 2005], [Vérin, 1998], [Crochemore et Vérin, 1997a]. L'apport principal de [Crochemore et Vérin, 1997a] et [Crochemore et Vérin, 1997b] est une construction directe de l'automate compact qui ne nécessite pas de construire l'automate des suffixes, ni l'arbre des suffixes, au préalable.

L'automate de la figure 2.5 représente le DAWG associé au mot "corolle", il se compose d'un ensemble d'états et de transitions étiquetées. Le mot et ses sous-mots associés sont stockés sur les chemins de l'automate. Ils peuvent être reconstruits en parcourant le DAWG de l'état initial à l'état final en sauvegardant les étiquettes. L'algorithme de construction du DAWG est détaillé au chapitre 5.

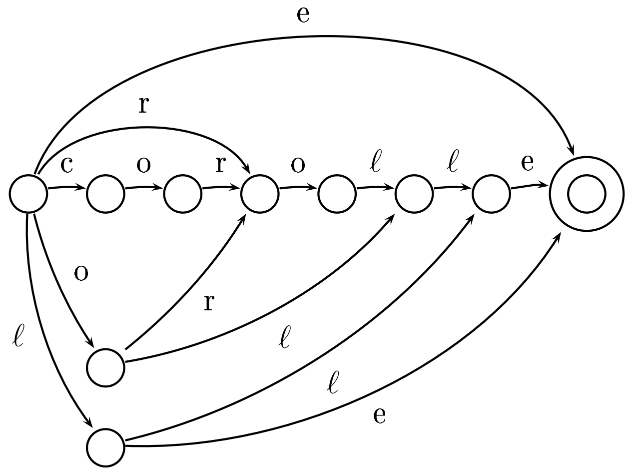


FIG. 2.5 – DAWG du mot "corolle"

Le DAWG associé à un mot est un automate : il est déterministe, accessible et coaccessible, il retourne tous les suffixes de ce mot, il est minimal et rarement complet. Dans le but de gagner de l'espace mémoire, le DAWG peut être compacté en fusionnant les états de l'automate des suffixes qui possèdent une unique transition sortante avec l'image de cette transition. Ainsi, une transition n'est plus étiquetée par une lettre mais elle peut être étiquetée par un mot. Le temps de parcours des transitions devient évidemment plus lent et les structures de données sont adaptées [Crochemore et Vérin, 1997a], [Blumer *et al.*, 1984].

Le DAWG peut être aussi dynamique, ainsi, l'insertion et la suppression d'un mot w de l'automate est possible, elle se fait de manière linéaire par rapport à la taille de w. Ce développement est donné par Kucherov et Rusinowitch dans [Kucherov et Rusinowitch, 1997]. Les auteurs de [Sgarbas *et al.*, 2003] décrivent un algorithme qui permet d'insérer plusieurs mots aux DAWGs. Cette insertion est optimale et lorsqu'elle est appliquée à un DAWG minimal, le DAWG obtenu après l'insertion reste minimal. Une application itérative de cet algorithme d'insertion pour n mots permet de construire un DAWG déterministe minimal de manière incrémentale en $O(n^2)$, ce qui n'est pas très efficace. Par contre, il trouve son utilité dans le cas des mises à jours rapides du DAWG minimal parce que chaque insertion de mot n'explore qu'une partie de l'automate sans avoir à réaliser une nouvelle minimisation.

Dans le but d'indexer des parties internes d'un automate, nous avons étudié la possibilité d'utiliser l'automate des suffixes (Directed Acyclic Word Graph "DAWG") dédié initialement à l'indexation des textes pour le faire. En l'adaptant à nos besoins, il permet d'avoir accès rapidement à l'ensemble des positions des sous-structures dans l'automate (cf. chapitre 5). Cette recherche rejoint implicitement l'étude de la similarité entre des sous-structures et la fouille de données. Voici un éventail des algorithmes classiques d'appariement :

- Les algorithmes d'appariement exacts qui étudient l'isomorphisme de graphe, l'isomorphisme de sous graphe, et la recherche de sous graphe commun maximum [Babai *et al.*, 1980], [McKay, 1981]. Tous ces problèmes sont des problème NP-Complets[12] et ils proposent des solutions optimales même si elles ne sont pas robustes au bruit et distorsions.
- Les algorithmes d'appariement inexacts sont plus robustes aux distorsions.
- Le concept de sondage de graphes "Graph Probing" qui calcule une mesure de similarité entre deux graphes mais ne donne pas d'information sur la meilleure mise en correspondance entre les deux graphes [Lopresti et Wilfong, 2001].

Le principal avantage attendu des différentes approches présentées pour la comparaison de structure étant d'être auto-adaptative aux données présentées en entrée.

2.2.4 Algorithmes de recherche de composants dans les graphes

A l'instar des automates, les graphes modélisent des objets constitués d'éléments en relations. Étant donné que le cadre des graphes de celui des automates sont proches, il est naturel de se demander si les algorithmes de recherche de composantes dans un graphe peuvent s'appliquer à la recherche de sous-automates.

- Un graphe non orienté G est un couple (S, A), où S représente un l'ensemble des sommets et A représente l'ensemble des arrêtes.
  Dans le cas ou le graphe est orienté, les arrêtes sont appelées arcs.

- Un chemin (resp. chaîne) reliant un sommet s_0 à un sommet s_n est une suite de sommets où deux éléments successifs sont reliés par un arc (resp. une arête)
- Un graphe non orienté est connexe si toute paire de sommets distincts est reliée par une chaîne. Dans le cas d'un graphe orienté, on ne parle plus de connexité, mais de forte connexité.
- Un graphe orienté est fortement connexe si tout couple de sommets distincts est relié par un chemin.
- Composante fortement connexe (resp. connexe)
  Un sous-graphe d'un graphe orienté (non orienté) est une composante fortement connexe (resp. connexe), s'il est fortement connexe (resp. connexe) et s'il est maximal.

Les algorithmes de recherche de ces composantes connexes ont l'avantage d'être linéaires (algorithme de Kosaraju) mais ils ne s'appliquent pas à la recherche de sous-automates pour les raisons suivantes :

1. Un sous-automate est une composante connexe, mais il n'est pas fortement connexe. En effet, ses états ne sont pas obligatoirement connectés tous ensemble.

[12]De manière intuitive, dire qu'un problème peut être décidé à l'aide d'un algorithme non-déterministe polynomial signifie qu'il est facile, pour une solution donnée, de vérifier en un temps polynomial si celle-ci répond au problème pour une instance donnée (à l'aide d'un certificat) ; mais que le nombre de solutions à tester pour résoudre le problème est exponentiel par rapport à la taille de l'instance.

2. Les automates sur lesquels nous travaillons sont acycliques.

3. Les sous-automates qui nous intéressent sont des sous-structures possédant une seule entrée et une seule sortie qui peuvent être remplacés par une transition dans l'automate sans modifier son langage alors que les composantes fortement connexe ne sont pas obligatoirement isolées des autres sommets et si elles sont extraites et remplacées par une transitions, cela modifie le langage de l'automate.

Dans ce qui va suivre, nous allons définir ce que nous appelons *sous-automate* et nous proposons un algorithme pour calculer l'ensemble des sous-automates associés à un automate donné en $O(n^3)$.

Chapitre 3

Recherche de sous-automates

Nous nous intéressons dans ce chapitre à la recherche de sous-automates. Leur détection systématique serait un premier pas dans l'analyse de la structure des automates. Elle pourrait à terme permettre d'identifier les redondances et contribuer à une compression de ces automates.
Nous ne nous intéressons pas dans ce chapitre aux étiquettes portées par les transitions, de ce fait il s'agit plus d'une recherche de sous-graphes (ou plutôt de faisceaux [Gondran et Minoux, 1985]) que de sous-automates en tant que tels. Ce sont des parties du graphe de l'automate qui peuvent être isolées, par exemple pour être remplacées par une transition. Pour cela nous cherchons des portions qui n'ont qu'une seule entrée et qu'une seule sortie. Le nombre de sous-automates pouvant être important, nous nous sommes limités aux sous-automates minimaux en longueur. Parmi ceux-ci, nous avons privilégié les sous-automates maximaux, puis les sous-automates minimaux en largeur.

Une première approche a été développée pour détecter les sous-automates maximaux en largeur que nous avons appelés "plus petits sous-automates fermés" ; elle est basée sur la propagation d'informations [Tounsi *et al.*, 2005]. Une seconde approche, plus fine, a été développée afin de détecter l'ensemble des sous-automates présents dans un automate. Cette méthode calcule, en fait, le sous-automate auquel appartient strictement un état donné et permet de distinguer deux autres types de sous-automates, les sous-automates parallèles et les sous-automates séries [Tounsi *et al.*, 2007].

3.1 Préliminaires

Cette partie regroupe les notions et définitions qui seront utilisées par la suite.

3.1.1 Définitions et notations

Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ un automate déterministe minimal acyclique à nombre fini d'états où :

- Σ est l'alphabet.
- Q est un ensemble fini d'états.
- δ est une fonction de transition: $\delta : Q \times \Sigma \to Q$.
- q_i est l'état initial ($q_i \in Q$).
- q_f est l'état final ($q_f \in Q$).

Cette définition est légèrement différente de la définition 2.3 car nous supposons ici que l'automate possède un unique état final.

Remarque 3.1
Il est aussi possible de définir l'automate par l'ensemble de ses transitions T (et non par la fonction de transition) : $T \subset Q \times \Sigma \times Q$.

- Un état p sera dit *divergent* (*resp. convergent*) si, et seulement si, il possède au moins deux transitions sortantes (resp. entrantes).
- Un état p sera dit *non divergent* (*resp. non convergent*) si, et seulement si, il possède une unique transition sortante (resp. entrante).

Soit $p \in Q$, on note :

- $Succ_A(p)$ (resp. $Pred_A(p)$) l'ensemble des successeurs (resp. prédécesseurs) immédiats de l'état p.
- $Succ^*_A(p)$ (resp. $Pred^*_A(p)$) l'ensemble de tous les successeurs (resp. prédécesseurs) de p, incluant p.

Dans ce travail, les successeurs et prédécesseurs sont toujours calculés dans l'automate initial A, ainsi, par abus de notation on décide d'écrire $Succ(p)$ pour $Succ_A(p)$ et $Pred(p)$ pour $Pred_A(p)$.

Plus précisément :

- $Succ(p) = \{q \in Q : \exists \alpha \in \Sigma : \delta(p, \alpha) = q\}$
- $Succ^*(p) = \{q \in Q : \exists w \in \Sigma^* : \underline{\delta}(p, w) = q\}$
- $Pred(p) = \{q \in Q : \exists \alpha \in \Sigma : \delta(q, \alpha) = p\}$
- $Pred^*(p) = \{q \in Q : \exists w \in \Sigma^* : \underline{\delta}(q, w) = p\}$

Remarque 3.2
Étant donné que l'automate est acyclique : $Succ^*(p) \cap Pred^*(p) = \{p\}$.

Nous employons également des notations semblables pour les ensembles d'états.
Soit $E \subset Q$:

- $Succ(E) = \bigcup_{p \in E} Succ(p) \setminus E$
- $Succ^*(E) = \bigcup_{p \in E} Succ^*(p)$
- $Pred(E) = \bigcup_{p \in E} Pred(p) \setminus E$
- $Pred^*(E) = \bigcup_{p \in E} Pred^*(p)$

3.1.2 Hauteur et cardinalité

Ces notions ont été définies par D.Revuz [Revuz, 1992].

Définition 3.1 (Hauteur)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ un automate, la hauteur d'un état $p \in Q$, notée $H(p)$, est le nombre maximum de transitions nécessaires pour atteindre l'état final à partir de p. La hauteur de q_f est nulle.
Cela correspond dans le graphe sous-jacent à la longueur en nombre d'arrêtes du plus long chemin reliant le sommet p au sommet final q_f.

$$\forall p \in Q \setminus \{q_f\},\ H(p) = Max_{\{q \in Succ(p)\}} H(q) + 1 \tag{3.1}$$

Définition 3.2 (Cardinalité)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ un automate, la cardinalité d'un état $p \in Q$, notée $C(p)$, est le nombre total de séquences reconnaissables démarrant de l'état p et arrivant à l'état final q_f. La cardinalité de q_f est égale à 1.
Cela correspond dans le graphe sous-jacent au nombre de chemins reliant le sommet p au sommet final q_f.

$$\forall p \in Q \setminus \{q_f\},\ C(p) = \sum_{q \in Succ(p)} |\{\alpha \in \Sigma / \delta(p, \alpha) = q\}| \times C(q) \tag{3.2}$$

3.2 Sous-automates recherchés

Les sous-automates recherchés sont des sous-structures de l'automate qui possèdent un seul point d'entrée et un seul point de sortie, ils peuvent ainsi être extraits et remplacés par une transition.
Pour aider à la recherche de ces sous-automates, nous avons été amenés à définir deux notions. La notion de "sous-automate fermé", qui peut être vu comme un sous-automate maximal en largeur et la notion de "plus petit sous-automate" et en particulier de "plus petit sous-automate fermé" qui est un sous-automate minimal pour l'inclusion.

3.2.1 Sous-automate

Étant donné les structures que nous souhaitons rechercher, nous avons retenu la définition suivante d'un sous-automate.

Définition 3.3 (Sous-automate)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ un automate acyclique à nombre fini d'états, $A' = < \Sigma, Q', \delta', s_i, s_f >$ est un sous-automate de A si et seulement si :

- $Q' \subset Q$
- $\{s_i, s_f\} \subset Q'$
- $\delta' : \begin{cases} Q' \times \Sigma \rightarrow Q' \\ \forall (q, \alpha) \in Q' \setminus \{s_i, s_f\} \times \Sigma \ :\ \textit{Si } \delta(q, \alpha) \textit{ est défini alors } \delta'(q, \alpha) = \delta(q, \alpha) \\ \textit{sinon } \delta'(q, \alpha) \textit{ est indéfini} \\ \forall \alpha \in \Sigma,\ \textit{si } \delta(s_i, \alpha) \in Q' \textit{ alors } \delta'(s_i, \alpha) = \delta(s_i, \alpha) \textit{ sinon } \delta'(s_i, \alpha) \textit{ est indéfini} \\ \forall \alpha \in \Sigma,\ \delta'(s_f, \alpha) \textit{ est indéfini} \end{cases}$
- $\forall q \in Q', q \in Succ^*(s_i)$ *et* $q \in Pred^*(s_f)$
- $\forall q \in Q' \setminus \{s_i, s_f\} : Succ(q) \subset Q'$ *et* $Pred(q) \subset Q'$

Rappelons que les ensembles *Succ* et *Pred* sont toujours calculés dans l'automate initial A.

Cette définition signifie que :

- Toute transition de A' est une transition de A ;
- Tout état de A' est accessible et co-accessible dans A' ;
- Toute transition de A ayant comme état de départ ou comme état but un état de Q' distinct de s_i et de s_f est une transition de A'.

Le sous-automate contient donc au moins deux états distincts, un état initial s_i et un état final s_f et il n'a de lien avec le reste de l'automate qu'au travers de ces deux états.

Avec cette définition, toute transition est un sous-automate.

Définition 3.4 (Sous-automate série pure)
Un sous-automate (s_i, s_f) est un sous automate série pure si, et seulement si :

$\exists w = s_i...s_f$ *chemin de l'automate* $:\forall q \in w \setminus \{s_i, s_f\}, |Succ(q)| = |Pred(q)| = 1$ *tel que le nombre de transitions sortantes et le nombre de transitions entrantes de q est 1.*

Définition 3.5 (Sous-automate parallèle pur)
Un sous-automate (s_i, s_f) est un sous-automate parallèle pur si, et seulement si, il est composé exactement des deux états s_i, s_f et :

$$\exists \alpha,\ \beta \in \Sigma,\ \alpha \neq \beta : \delta(s_i, \alpha) = \delta(s_i, \beta) = s_f$$

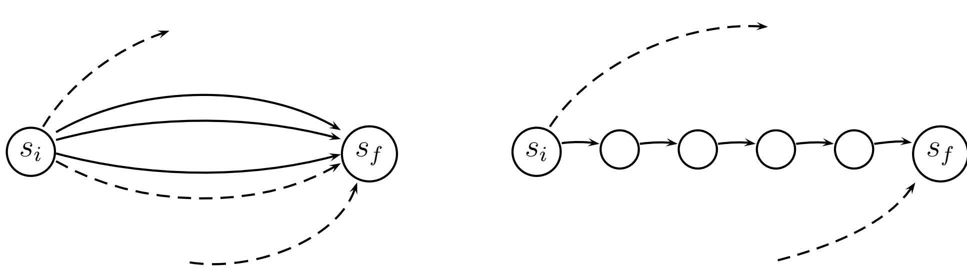


FIG. 3.1 – Sous-automate parallèle et sous-automate série

Définition 3.6 (Inclusion stricte)
Un état q est dit appartenir strictement au sous-automate A', si il n'est ni état initial, ni état final de A'. On dira aussi d'un tel état qu'il est un état interne du sous-automate.

Exemple 3.1
L'automate de la figure 3.2 est composé de 9 états et 13 transitions. Outre l'automate lui même et les treize transitions, il contient dix sous-automates. Certains états sont états initiaux de plusieurs sous-automates distincts. Ainsi, à partir de l'état initial 1, l'ensemble des états $\{1, 2, 3, 5\}$ forme un sous-automate, l'ensemble $\{1, 2, 3, 4, 5\}$ en forme un autre, le chemin reliant l'état 5 à l'état 9 passant par l'état 7 est aussi un sous-automate. L'automate initial est lui aussi un sous-automate. Par contre, le chemin reliant l'état 1 à l'état 5 passant par l'état 2 n'est pas un sous-automate car l'état 2 n'est ni état initial ni final et possède pourtant un successeur extérieur.

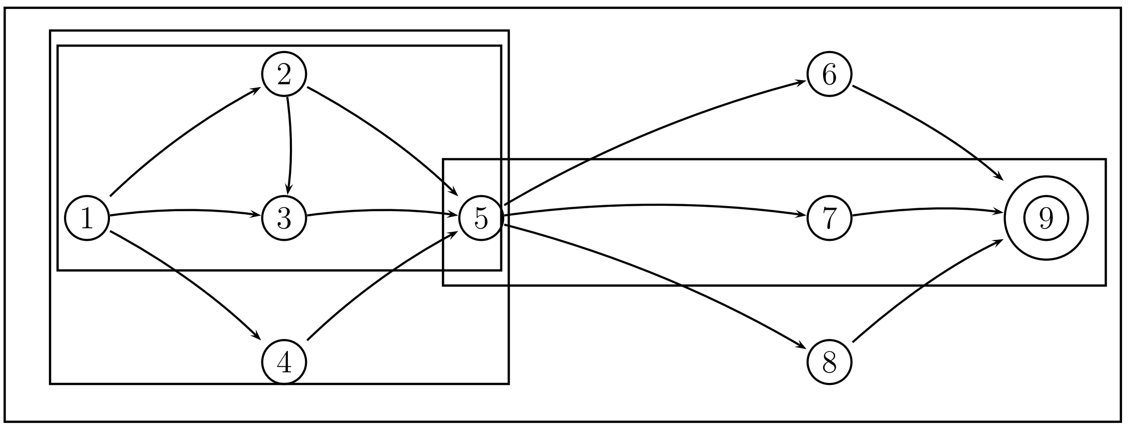


FIG. 3.2 – Exemple de sous-automates

3.2.2 Propriétés

La hauteur et la cardinalité des états d'un sous-automate respectent les propriétés suivantes :

Propriété 3.1 (Hauteur)
La hauteur d'un état q appartenant strictement à un sous-automate $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ vérifie :
$H(q) = H(s_f) + h$ *où* $h = Sup_{\underline{\delta}(q,w)=s_f}(|w|)$
(h est la longueur du plus long chemin de q à s_f)
$H(s_i) = H(q) + h'$ *où* $h' = Sup_{\underline{\delta}(s_i,w')=q}(|w'|)$
(h' est la longueur du plus long chemin de s_i à q)

En conséquence, $\forall q \in Q' \setminus \{s_i, s_f\}$, $H(s_i) > H(q) > H(s_f)$

Propriété 3.2 (Cardinalité)
La cardinalité d'un état q appartenant strictement à un sous-automate $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ est un multiple de la cardinalité de s_f :
$\forall q \in Q' \setminus \{s_i, s_f\}, \exists y \in \mathbb{N}$ *tel que* $C(q) = y \times C(s_f)$

3.2.3 Sous-automates fermés "SAF"

En toute généralité, la donnée des états initiaux et finaux ne suffit pas à caractériser un sous-automate, sauf si on impose que toutes les transitions sortant de l'état initial et toutes celles arrivant à l'état final doivent appartenir au sous-automate. On définit ainsi un sous-automate qui va être maximal en largeur et que nous appelons "sous-automate fermé". De tels sous-automates, si il en existe, seraient facile à stocker, mais bien sûr n'archiver que les deux états extrêmes oblige à recalculer le sous-automate à chaque fois que l'on en a besoin.

Définition 3.7 (Sous-automate fermé)
Soit $Q' \subseteq Q$ et s_i, s_f deux états distincts de Q' :
Un sous-automate $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ est fermé (SAF) si, et seulement si :

- $\forall q \in Q' \setminus \{s_f\} : Succ(q) \subset Q'$
- $\forall q \in Q' \setminus \{s_i\} : Pred(q) \subset Q'$

Cela signifie que tout état ou transition se situant sur un chemin issu de s_i ou aboutissant à s_f doit appartenir au sous-automate. Par abus de notation, le sous-automate A' est désigné par le couple (s_i, s_f).
Ce sous-automate est maximal en largeur car il contient tous les chemins reliant l'état initial à l'état final.

Exemple 3.2
Nous avons vu que l'automate de la figure 3.2 contient dix sous-automates ; seulement trois parmi ceux-ci sont fermés ($(1,5)$, $(5,9)$ et $(1,9)$). Le sous-automate composé des états $\{1, 2, 3, 5\}$ n'est pas fermé parce que l'état 1 et l'état 5 possèdent respectivement une transition sortante et une transition entrante en dehors du sous-automate.

L'exemple 3.3 illustre la présence de sous-automates fermés dans un automate représentant un dictionnaire.

Exemple 3.3
La figure 3.3 représente un automate de flexion qui reconnaît l'infinitif, le participe passé et le participe présent des six verbes suivants : *informer*, *insister*, *performer*, *persister*, *réformer*, *résister*.
Cet automate est composé de 20 états et 26 transitions où les états 1 et 20 sont respectivement l'état initial et l'état final.
Dans cet exemple, l'ensemble des états $Q' = \{6, 7, 8, 9, 10, 11, 12, 13\}$ définit un sous-automate fermé. Ce sous-automate pourra être noté $(6, 13)$. Il contient toutes les transitions sortantes de tous les états sauf celles de l'état final 13 et toutes les transitions entrantes de tous les états sauf celles de l'état initial 6. Au total, l'automate de flexion contient 6 SAF : $(1,6)$, $(6,13)$, $(13,20)$, $(1,13)$, $(6,20)$ et $(1,20)$.

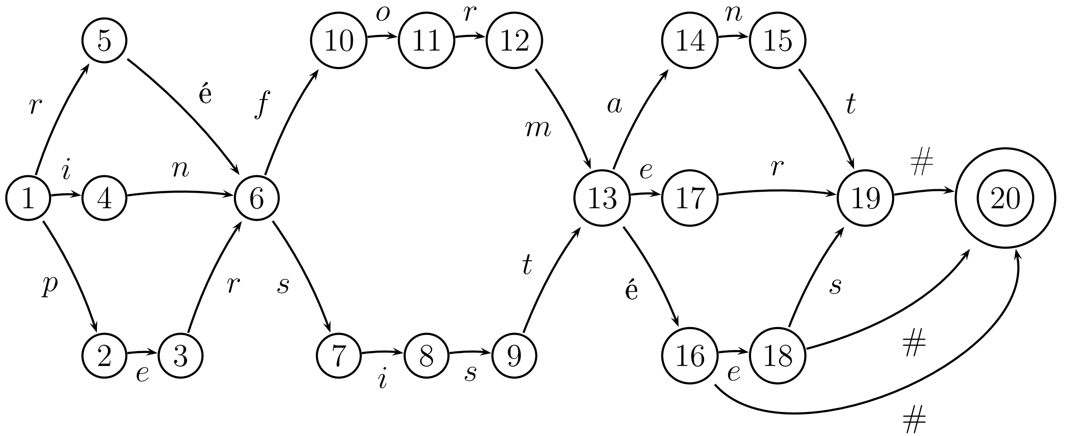


FIG. 3.3 – Automate de flexion

Remarque 3.3
Un sous automate série ou parallèle peut être aussi fermé.

FIG. 3.4 – Sous-automate parallèle fermé et sous-automate série fermé

Propriétés

Dans le cas des sous-automates fermés, les propriétés 3.1 et 3.2 s'étendent à l'état initial s_i et peuvent être complétées de la manière suivante :

Propriété 3.3 (Hauteur)
$H(s_i) = H(s_f) + h''$ *où* $h'' = Sup_{\underline{\delta}(s_i,w'')=s_f}(|w''|)$ *(*h'' *est la longueur du plus long chemin de* s_i *à* s_f*)*

Propriété 3.4 (Cardinalité)
$C(s_i)$ *est un multiple de* $C(s_f)$*.*

Corollaire 3.1 (Relation de cardinalité)
Étant donné un ensemble d'états inclus dans l'automate $A' =< \Sigma, Q', \delta_{/Q'}, s_i, s_f >$*, la cardinalité de l'état final* s_f *divise le plus grand diviseur commun (PGCD) des cardinalités de ces états.*

$$\forall q_1, .., q_n \in Q' \ : \ C(s_f)|PGCD(C(q_1), ..., C(q_n))$$

En particulier :
Si $q_1, ..., q_n$ *sont les successeurs immédiats de* s_i *:* $C(s_f)|PGCD(C(s_i), C(q_1), ..., C(q_n))$

Dans l'exemple 3.3 (figure 3.3), le couple $(6, 13)$ contenant l'ensemble des états $\{6, 7, 8, 9, 10, 11, 12, 13\}$ respecte les trois propriétés ci-dessus.

- La hauteur de tous les états H(7)=8, H(8)=7, H(9)=6, H(10)=8, H(11)=7, H(12)=6 est inférieure à celle de l'état initial H(6)=9 et supérieure à celle de l'état final H(13)=5.
- La cardinalité de tous les états C(7)=C(8)=C(9)=C(10)=C(11)=C(12)=5, C(6)=10 est un multiple de C(13)=5
- La cardinalité de l'état final 13 divise le PGCD entre la cardinalité de l'état initial 6 et celles de ses successeurs 7 et 10.

3.2.4 Plus petits sous-automates fermés ("$PPSAF$")

Dans l'exemple 3.3, l'automate contient 6 SAF, mais on remarque qu'il suffit de trouver les SAF (1,6), (6,13) et (13,20) pour reconstruire les autres. Ces trois sous automates ont la particularité d'être minimaux, plus précisément, ils ne contiennent pas d'autres SAF ayant le même état initial ou le même état final.

Définition 3.8 (Plus petit sous-automate fermé)
Soit $Q' \subset Q$ et s_i, s_f deux états distincts de Q' :
Un sous-automate fermé $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *est un plus petit sous-automate fermé* ($PPSAF$) *si, et seulement si :*

- $\forall q \in Q' : \begin{cases} (s_i, q) \textit{ est un sous-automate fermé} \Rightarrow q = s_f \\ (q, s_f) \textit{ est un sous-automate fermé} \Rightarrow q = s_i \end{cases}$

Exemple 3.4
L'automate de la figure 3.2 contient deux $PPSAF$: $(1, 5)$ et $(5, 9)$, celui de la figure 3.3 en contient trois : $(1, 6)$, $(6, 13)$ et $(13, 20)$.

Dans ces deux cas, les $PPSAF$ sont minimaux au sens de l'inclusion, c'est-à-dire qu'ils ne contiennent strictement aucun autre SAF. Ce n'est néanmoins pas toujours le cas. L'automate de l'exemple 3.5 contient trois $PPSAF$: $(7, 10)$, $(4, 11)$ et $(1, 12)$ qui sont inclus les uns dans les autres. La notion de $PPSAF$ est donc liée à l'état initial (ou à l'état final).

Exemple 3.5

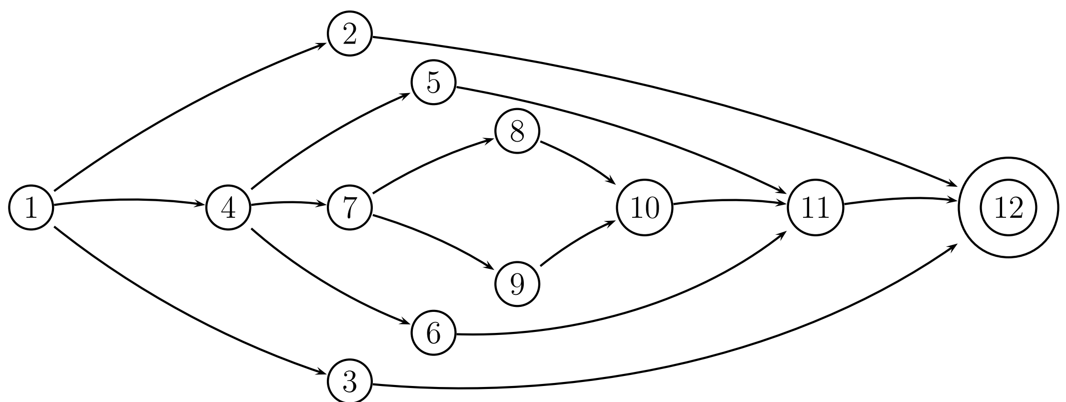


FIG. 3.5 – $PPSAF$ emboîtés

Propriété 3.5 (Relation d'inclusion entre plus petits sous-automates fermés)
Soit $A'_1 = (s_{i1}, s_{f1})$, $A'_2 = (s_{i2}, s_{f2})$ deux plus petits sous-automates fermés, A'_1 est inclus strictement dans A'_2, si et seulement si, il existe un chemin non vide reliant l'état initial de A'_2 à celui de A'_1 et un autre reliant l'état final de A'_1 à celui de A'_2.

$$A'_1 \subset A'_2 \text{ ssi } \exists w, w' \in \Sigma^* : \underline{\delta}(s_{i2}, w) = s_{i1},\ \underline{\delta}(s_{f1}, w') = s_{f2}$$

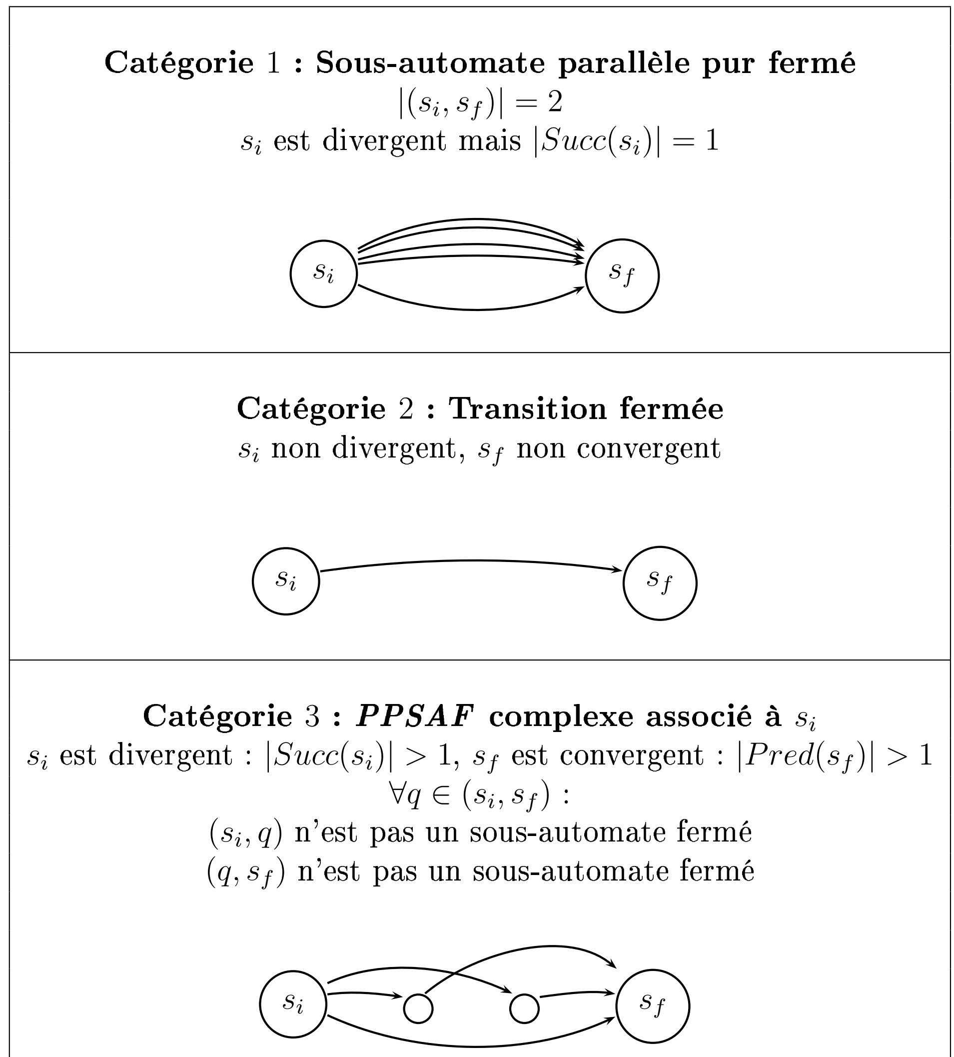
Propriété 3.6 (Coexistence de plus petits sous-automates fermés)
Étant donnés deux plus petits sous-automates fermés A'_1 et A'_2,
- *Soit ils sont transitions-disjoints*[1]*,*
- *Soit l'un est inclus strictement dans l'autre.*

Dans le cas où ils sont transitions-disjoints, ils partagent au maximum un état : l'état final de l'un peut être état initial de l'autre.

À chaque état q de l'automate initial A correspond au plus un $PPSAF$, qui est le SAF de longueur minimale issu de q. Ces $PPSAF$ peuvent être les uns à côté des autres ou former une hiérarchie d'inclusions. La donnée de son état initial s_i définit sans ambiguïté un $PPSAF$, on pourra le noter $PPSAF(s_i)$.

On est amené à distinguer trois types de $PPSAF$. Les parallèles purs fermés, les transitions fermées qui serviront à former des séries et les autres. Ces trois types sont résumés dans la table 3.1.

[1]Soit T_1 l'ensemble des transitions de A'_1 et T_2 l'ensemble des transitions de A'_2, $T_1 \cap T_2 = \varnothing$.



TAB. 3.1 – Catégories de plus petits sous-automates fermés

L'automate de la figure 3.3 ne possède aucun sous-automate de catégorie 1, par contre, il possède six sous-automates de catégorie 2 : $(2,3)$, $(7,8)$, $(8,9)$, $(10,11)$, $(11,12)$ et $(14,15)$ et trois sous-automates de catégorie 3 : $(1,6)$, $(6,13)$, $(13,20)$. Il est à noter que la série $(10,12)$ est un SAF mais pas un $PPSAF$.

3.2.5 Plus petit sous-automate ("PPSA")

La définition et la recherche des sous-automates fermés se justifient par le fait qu'ils sont archivables par le simple couple (état initial, état final), les $PPSAF$ sont même archivables par leur état initial uniquement. Néanmoins, un $PPSAF$ peut être composé de plusieurs sous-automates. Par exemple, le $PPSAF$ $(1,5)$ de l'automate représenté figure 3.2 contient les deux sous-automates $(1,2,3,5)$ et $(1,4,5)$ en parallèle. Il est donc intéressant d'élargir la recherche des sous-automates, pour cela, nous définissons la notion de plus petit sous-automate associé à un état. Connaître les plus petits sous-automates permettrait de reconstruire tous les sous-automates.

Définition 3.9 (Plus petit sous-automate associé à un état)
Étant donné un état $p \in Q \setminus \{s_i, s_f\}$*, le sous-automate* $A' =< \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *est le plus petit sous-automate associé à p (*$PPSA(p)$*) si, et seulement si :*

- *A' contient strictement p*
- *$\forall\ A'' =< \Sigma, Q'', \delta_{/Q''}, s''_i, s''_f >$ sous-automate contenant strictement p : $Q' \subset Q''$*

A' est donc le sous-automate minimal au sens de l'inclusion contenant strictement p.
Il est nécessaire d'imposer l'existence d'un état autre que initial ou final, sinon les plus petits sous-automates seraient réduits à des transitions.

Exemple 3.6
L'automate de la figure 3.6 contient sept $PPSA$.
$A'_1 = \{1, 2, 4, 6\}= PPSA(2) = PPSA(4)$
$A'_2 = \{1, 3, 5, 6\}= PPSA(3) = PPSA(5)$
$A'_3 = \{6, 7, 8, 9\}= PPSA(7) = PPSA(8)$
$A'_4 = \{9, 10, 11, 14\}= PPSA(10) = PPSA(11)$
$A'_5 = \{6, 7, 8, 9, 10, 11, 14\}= PPSA(9)$
$A'_6 = \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}= PPSA(12) = PPSA(13)$
$A'_7 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}= A = PPSA(6)$

On peut remarquer que $A'_3 \subset A'_5 \subset A'_6$ et $A'_4 \subset A'_5 \subset A'_6$. A'_5 n'est pas un plus petit sous-automate au sens intuitif du terme, il est composé de deux plus petits sous-automates mis en série. Il est obtenu en étudiant l'état 9 qui est l'état de jonction entre ces deux sous-automates.

Exemple 3.7
L'automate de l'exemple 3.5, contient lui trois PPSA :
PPSA(8)=PPSA(9)=PPASF(7)=(7,10),
PPSA(5)=PPSA(6)=PPSA(7)=PPSA(10)=PPASF(4)=(4,11)
et PPSA(2)=PPSA(3)=PPSA(4)=PPSA(11)=PPASF(1)=(1,12)

Dans cet exemple, il y a coïncidence entre les plus petits sous-automates fermés et les plus petits sous-automates. Mais ce n'est pas toujours le cas : Dans l'exemple précédent, le plus petit sous-automate associé à l'état 1 ($\{1, 2, 3, 4, 5, 6\}$) n'est pas un plus petit sous-automate associé à un état.

On peut aisément étendre la notion de plus petit sous-automate à un ensemble d'états.

Définition 3.10 ($PPSA$ **associé à un ensemble d'états)**
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ *un automate et* $E \subset Q$, $E \neq \emptyset$.
Le plus petit sous-automate associé à E ($PPSA(E)$*) est le sous-automate* $PPSA(E) =< \Sigma, Q_{PPSA}(E), \delta_{PPSA}(E), s_i, s_f >$ *tel que :*

- $E \subset Q_{PPSA}(E)$ *et*

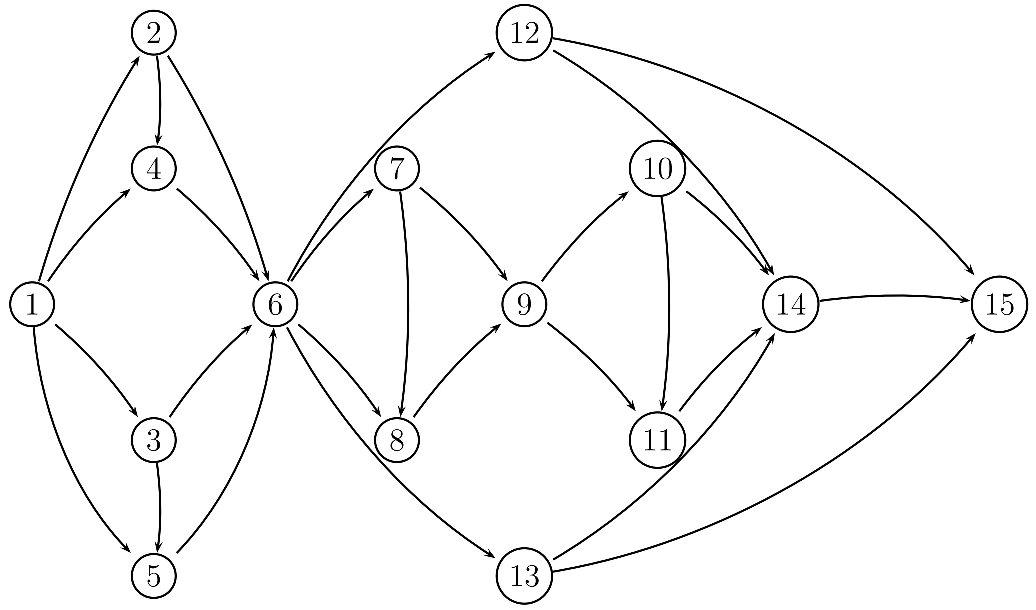


FIG. 3.6 – Recherche $PPSA$

– *$\forall A'$ sous-automate de A dont l'ensemble des états Q' contient E, $Q_{PPSA}(E) \subset Q'$.*

Exemple 3.8
En reprennant l'automate de la figure 3.6, on constate que cela permet de d'ajouter un sous-automate.
$PPSA(2,3) = \{1,2,3,4,5,6\} = PPASF(1) = PPSA(2,5) = PPSA(4,3) = ...$

Remarque 3.4
Si s_i est l'état initial d'un plus petit sous-automate fermé, alors $PPSAF(s_i) = PPSA(Succ(s_i))$.

Dans toute la suite de ce chapitre, les états sont supposés numérotés dans un ordre compatible avec l'ordre partiel induit par l'automate et nous confondrons l'état et son numéro. q_i, l'état initial de l'automate sera l'état numéro 1 et q_f, l'état final, sera l'état numéro n.

3.3 Recherche des plus petits sous-automates fermés ("PPSAF")

Nous présentons un algorithme permettant le calcul des plus petits sous-automates fermés d'un automate A donné en entrée (cf. table 3.1). Cet algorithme n'utilise que la liste des successeurs d'un état, ce qui évite d'augmenter inutilement la taille des données. En effet, dans certains modes de stockage des automates seule la liste des transitions sortantes des états est stockée, nous n'avons donc pas directement accès à l'ensemble des prédécesseurs d'un état.
Cet algorithme a été testé sur des automates représentant des dictionnaires de la langue

française, les résultats obtenus sont présentés en section 3.3.3. Comme ceux-ci montrent peu de sous-automates fermés, nous nous sommes intéressés ensuite à la recherche globale de tous les sous-automates.

3.3.1 Analyse du problème

L'algorithme de recherche des plus petits sous-automates fermés est basé sur les définitions et propriétés suivantes. Les premières sont destinées à mettre en évidence l'état initial d'un $PPSAF$.

Définition 3.11 (Chemin antérieur à un ensemble d'états)
Étant donné un ensemble non vide d'états E ($E \subset Q$), on appelle chemin antérieur à E tout chemin reliant q_i à un état quelconque de E. On note $CA(E)$ l'ensemble des chemins antérieurs à E.

$$CA(E) = \{w \text{ chemin de } q_i \text{ à } p,\ p \in E\}$$

Définition 3.12 (nœud antérieur d'un ensemble d'états)
Étant donné un ensemble non vide d'états E ($E \subset Q$), on appelle nœud antérieur de E tout état appartenant à tous les chemins antérieurs à E. On note $NA(E)$ l'ensemble des nœuds antérieurs de E.

$$NA(E) = \{p \in Q / \forall w \in CA(E), p \in w\}$$

L'état initial (q_i) est nœud antérieur de tout ensemble d'états.

Définition 3.13 (État source d'un ensemble d'états)
Étant donné un ensemble non vide d'états E ($E \subset Q$), on appelle état source de E et on note $source(E)$ le nœud antérieur de E de plus petite hauteur.

$$source(E) \in NA(E) \text{ et } H(source(E)) = Min_{q \in NA(E)}(H(q))$$

L'état $source(E)$ existe forcément car $NA(E)$ n'est pas vide et il est unique parce que deux états appartenant à $NA(E)$ sont forcément reliés par au moins un chemin, ainsi ils ne peuvent pas avoir la même hauteur.
$source(E)$ est aussi le plus grand état par la numérotation appartenant à $NA(E)$.

Soit $E = \{p_1, p_2, \cdots p_n\}$, afin de ne pas alourdir les notations, nous écrivons $source(E) = source(p_1, p_2, \cdots p_n)$ au lieu de $source(E) = source(\{p_1, p_2, \cdots, p_n\})$.

Exemple 3.9
Soit l'automate donné par la figure 3.7.
$source(q_3, q_4, q_5)= source(q_3, q_4, q_6) = source(q_3, q_4, q_5, q_6) =q_1$.
$source(q_2, q_3) = source(q_3, q_4) = q_2$.

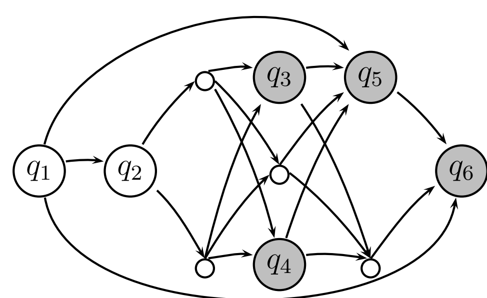


FIG. 3.7 – États source et puits associés à différents ensembles d'états

Remarque 3.5

- *L'état $source(E)$ peut représenter q_i, état initial de l'automate A.*
- *L'état $source(E)$ peut appartenir à E. Par exemple, dans le cas où E est réduit à un singleton ou bien dans le cas où E est exactement l'ensemble des états d'un sous-automate.*

Propriété 3.7 (associativité)
Si $E = E_1 \cup E_2$, avec E_1 et E_2 non vides, alors $source(source(E_1), source(E_2)) = source(E)$

Démonstration
Notons $s_1 = source(E_1)$, $s_2 = source(E_2)$, $s = source(source(E_1), source(E_2)) = source(s_1, s_2)$ et $s' = source(E)$.
- Il est clair que $s' \in NA(E_1)$ et $s' \in NA(E_2)$, donc s' précède s_1 et s_2. Soit w_0 un chemin quelconque de q_i vers $\{s_1, s_2\}$. Si w arrive en s_1, on peut le compléter par un chemin w_1 de s_1 vers E_1. Le chemin w_0w_1 va de q_i à E donc il passe par s' et donc $s' \in w_0$. On montre de même que si w arrive en s_2, il passe forcément par s'. Donc $s' \in NA(\{s_1, s_2\})$ et donc $s' \leq s$.
- Soit w un chemin quelconque de q_i à E. S'il arrive en E_1, il passe forcément par s_1, s'il arrive en E_2, il passe forcément par s_2, donc w passe forcément par l'ensemble $\{s_1, s_2\}$ ce qui signifie qu'il contient $s = source(s_1, s_2)$. Donc $s \in NA(E)$ et donc $s \leq s'$. On en déduit que $s = s'$ □

Définition 3.14 (État initial associé à un état)
Étant donné un état p, $p \neq q_i$, on appelle état initial associé à p et on note $EI(p)$ l'état $EI(p) = source(Pred(p))$. Il s'agit de l'état de hauteur minimale, différent de p qui appartient à tout chemin de q_i à p.

De façon similaire on définit des concepts liés à l'état final d'un $PPSAF$.

Définition 3.15 (Chemin postérieur à un ensemble d'états)
Étant donné un ensemble non vide d'états E $(E \subset Q)$, on appelle chemin postérieur à E tout chemin reliant un état quelconque de E à q_f. On note $CP(E)$ l'ensemble des chemins postérieurs à E.

$$CP(E) = \{w \text{ chemin de } p \text{ à } q_f,\ p \in E\}$$

Définition 3.16 (nœud postérieur d'un ensemble d'états)
Étant donné un ensemble non vide d'états E $(E \subset Q)$, on appelle nœud postérieur de E tout état appartenant à tous les chemins postérieurs à E. On note $NP(E)$ l'ensemble des nœuds postérieurs de E.

$$NP(E) = \{p \in Q / \forall w \in CP(E), p \in w\}$$

L'état n (q_f) est nœud postérieur de tout ensemble d'états.

Définition 3.17 (État puits d'un ensemble d'états)
Étant donné un ensemble non vide d'états E $(E \subset Q)$, on appelle état puits de E et on note $puits(E)$ le nœud postérieur de E de plus grande hauteur.

$$puits(E) \in NP(E) \text{ et } H(puits(E)) = Max_{q \in NP(E)}(H(q))$$

L'état $puits(E)$ existe forcément, pour les mêmes raisons que source(E) existe.
$puits(E)$ est aussi le plus petit état par la numérotation appartenant à $NP(E)$.

Exemple 3.10
Revenons vers l'automate donné par la figure 3.7.
$puits(q_3, q_4) = puits(q_3, q_4, q_6) = puits(q_3, q_4, q_5) = q_6$.

Remarque 3.6

- *L'état $puits(E)$ peut représenter q_f, état final de l'automate A.*
- *L'état $puits(E)$ peut appartenir à E. Par exemple, dans le cas où E est réduit à un singleton ou bien dans le cas où E est exactement l'ensemble des états d'un sous-automate.*

Propriété 3.8 (associativité)
Si $E = E_1 \cup E_2$, avec E_1 et E_2 non vides, alors $puits(puits(E_1), puits(E_2)) = puits(E)$

Définition 3.18 (État final associé à un état)
Étant donné un état p, $p \neq q_f$, on appelle état final associé à p et on note $EF(p)$ l'état $EF(p) = puits(Succ(p))$. Il s'agit de l'état de hauteur maximale, différent de p qui appartient à tout chemin de p à q_f.

Remarque 3.7
L'état final associé à q_f n'existe pas et l'état initial associé à q_i n'existe pas non plus.

Propriété 3.9
$\forall p < n,\ EI(EF(p)) \leq p$ et $\forall p > 1,\ EF(EI(p)) \geq p$

Démonstration
Soit $p < n$ et $q = EF(p)$. Il existe au moins un chemin de q_i à q passant par p, donc $EI(q)$ est soit un prédécesseur de p, soit un successeur de p, soit égal à p. Supposons que $EI(q)$ soit un successeur de p. Soit w un chemin quelconque de p à q_f, w passe forcément par q, par définition de $EF(p)$, on peut alors décomposer w en deux sous chemins : w_2 qui va de p à q et w_3 qui va de q à q_f ($w = w_1 w_2$). Soit en plus w_1 un chemin de q_i à p, $w_1 w_2$ est un chemin de q_i à q, il contient donc $EI(q)$, plus précisément $EI(q)$ appartient à w_2. Donc tout chemin de p à q_f passe par $EI(q)$, donc $EI(q) \geq EF(p) = q$ ce qui est impossible.
Donc $EI(q) \leq p$
La seconde inégalité de démontre de la même façon. □

Propriété 3.10
Si (s_i, s_f) est un $PPSAF$ alors $EI(s_f) = s_i$ et $EF(s_i) = s_f$.

Démonstration
• Si (s_i, s_f) est une transition alors la propriété est vraie car par définition d'un $PPSAF$, s_i est l'unique prédécesseur de s_f et s_f l'unique successeur de s_i.
• Si (s_i, s_f) n'est pas une transition, notons S son ensemble d'états
Soit $s_i' = EI(s_f)$, par définition d'un sous-automate fermé, s_i est sur tout chemin de 1 à s_f, donc $s_i \in NA(s_f)$ et donc s_i est un prédécesseur de s_i'.
Soit $S' = Succ^*(s_i') \cap Pred^*(s_f)$, montrons que $< \Sigma, S', \delta_{/S'}, s_i', s_f >$ est un sous-automate fermé.
Soient $p \in S' \setminus \{s_f\}$ et $q \in Succ(p)$.
  Il est clair que $p \in S \setminus \{s_f\}$ donc $q \in S$ (car (s_i, s_f) est un SAF),
  donc $q \in Pred^*(s_f)$,
  or $q \in Succ^*(s_i')$, donc $q \in S'$,
  donc $\forall p \in S' \setminus \{s_f\}$, $Succ(p) \subset S'$.
Soient $p \in S' \setminus \{s_i\}$ et $q \in Pred(p)$.
  $q \in Pred(p)$ et $p \in Pred^*(s_f)$ donc $q \in Pred^*(s_f)$,
  donc q est sur un chemin de 1 à s_f, chemin auquel appartient forcément s_i',
  or q ne peut pas précéder strictement s_i' sinon il y aurait un circuit dans l'automate,
  donc $q \in Succ^*(s_i')$,
  donc $q \in S'$.
  Donc $\forall p \in S' \setminus \{s_i\}$, $Pred(p) \subset S'$.
(s_i', s_f) est donc un SAF, or (s_i', s_f) est un $PPSAF$ et $s_i' \neq s_f$ donc $s_i' = s_i$.
On en conclut que $s_i = EI(s_f)$.
On montre de la même façon que $s_f = EF(s_i)$. □

Propriété 3.11
Si s_i et s_f sont deux états tels que $EI(s_f) = s_i$ et $EF(s_i) = s_f$ alors (s_i, s_f) est un $PPSAF$.

Démonstration
Soit $Q' = \{p \in Q | p$ appartient à un chemin de s_i à $s_f\}$.
- Soit $p \in Q' \setminus \{s_f\}$ et $q \in Succ(p)$.
$s_f = EF(s_i)$ donc s_f est sur tout chemin de s_i à s_f, donc q est soit un prédécesseur soit un successeur de s_f. Étant donné que p est un prédécesseur de s_f, que $p \neq s_f$ et qu'il existe une transition reliant p et q, q est forcément un prédécesseur de s_f (au pire $q = s_f$). Donc $q \in Q'$.

- On montre de la même façon que si $p \in Q' \setminus \{s_i\}$ et $q \in Pred(p)$ alors $q \in Q'$.
Donc (s_i, s_f) est un $PPSAF$. □

Un corollaire immédiat de ces deux propriétés est :

Propriété 3.12
(s_i, s_f) *est un* $PPSAF$ *si, et seulement si* $EI(s_f) = s_i$ *et* $EF(s_i) = s_f$.

3.3.2 Algorithme PPSAF

Cet algorithme se déroule en deux phases. Dans la première phase, on calcule pour chaque état son état initial associé, ainsi que des informations sur son état final associé, il s'agit de simples propagations de valeurs. Lors de la seconde phase, l'automate est parcouru une nouvelle fois afin d'extraire les plus petits sous-automates fermés à partir des informations calculées dans la passe précédente.

Rappellons que les états sont numérotés suivant un ordre induit par l'automate, c'est-à-dire qu'un état porte un numéro supérieur à celui de chacun de ses prédécesseurs, un état est donc exploré seulement après avoir exploré tous ses prédécesseurs. Á tout état p on associe trois champs, en plus de la liste de ses successeurs : ei qui est l'état initial associé à p et ef_{min} et ef_{max} qui sont les plus petits et plus grands états (par la numérotation) pouvant être état final d'un sous-automate fermé admettant p comme état initial (sans préjuger de l'existence d'un tel sous-automate). Au début de l'algorithme, $p.ei$ et $p.ef_{min}$ sont initialisés à 0 et $p.ef_{max}$ à $+\infty$.

- **Description de la phase 1 :**
  La fonction Marquage réalise la propagation des états initiaux potentiels. Chaque état transmet son numéro à chacun de ses successeurs, ce numéro correspond à l'état initial potentiel du successeur. Lorsqu'un successeur a déjà été traité, cela signifie que l'on peut arriver à cet état par au moins deux chemins différents, il faut alors remonter ces deux chemins pour trouver un état initial commun. C'est ce que fait la fonction "Trouver Source". Ces deux fonctions se chargent aussi de mettre à jour les champs ef_{min} et ef_{max}.

  À la fin de la phase 1, on a les propriétés suivantes.

  Propriété 3.13
  $\forall q > 1,\ q.ei = EI(q)$

  En conséquence, d'après la propriété 3.12, seuls les couples du type $(q.ei, q)$ ont une chance d'être des $PPSAF$.

  Propriété 3.14
  $\forall p \leq n-1,\ p.ef_{min} = Max\left(Max\left\{q / EI(q) = p\right\}; Max(Succ(p))\right)$,
  avec comme convention $Max(\emptyset) = 0$.

  Propriété 3.15
  $\forall q > 1,, (q.ei, q)$ *est un* $PPSAF$ *si, et seulement si* $q.ei.efmin \leq q \leq q.ei.efmax$.

L'application de cette phase sur l'automate de la figure 3.8 est présentée sur le tableau 3.2. Ce marquage est aussi reporté sur cette même figure. Les états qui possèdent un unique prédécesseur portent le numéro de ce prédécesseur comme état initial potentiel, ainsi 2 et 3 portent le numéro 1, 5 porte le numéro 3 et 19 porte le numéro 18... Les états qui possèdent au moins deux prédécesseurs distincts portent comme état initial potentiel le plus grand prédécesseur commun de leurs prédécesseurs (prédécesseur commun de plus petite hauteur). Par exemple, l'état 20 admet 17 et 19 comme prédécesseurs qui admettent tout deux l'état 16 comme prédécesseur commun.

Algorithm 1 PPSAF - Entrée : A - Sortie : SA

1: $n \leftarrow$ nombre d'états de l'automate A
2:
3: /* PHASE 1 */
4:
5: **for** $p = 1$ à $n-1$ **do**
6: $p.ef_{min} \leftarrow Max(Succ(p))$; /*Initialisation*/
7: **for** Chaque successeur q de p **do**
8: **if** $(q.ei = 0)$ **then**
9: $q.ei \leftarrow p$;
10: **else**
11: **if** $(q.ei \neq p)$ **then**
12: $q.ei \leftarrow$ Trouver Source $(q.ei, p, q)$; /* Algorithme 2*/
13: $q.ei.ef_{min} \leftarrow Max(q.ei.ef_{min}, q)$;
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18:
19: /* PHASE 2 */
20:
21: **for** $p = 2$ à n **do**
22: **if** $p.ei.ef_{min} \leq p \leq p.ei.ef_{max}$ **then**
23: Ajouter $(p.ei, p)$ à SA ;
24: **end if**
25: **end for**

Algorithm 2 Trouver Source - Entrée : A, ei_1, ei_2, p - Sortie : A, $source(ei_1, ei_2)$

1: $ef_1 \leftarrow p$;
2: $ef_2 \leftarrow p$;
3: **repeat**
4:
5: /*parcourir à l'envers les chemins démarrant de ei_1 et ei_2 jusqu'à la rencontre du point de départ de tous ces chemins*/
6:
7: **if** $ei_1 > ei_2$ **then**
8: $ei_1.ef_{max} \leftarrow Min(ei_1.ef_{max}, ef_1)$;
9: $ef_1 \leftarrow ei_1$;
10: $ei_1 \leftarrow ei_1.ei$;
11: **else**
12: $ei_2.ef_{max} \leftarrow Min(ei_2.ef_{max}, ef_2)$;
13: $ef_2 \leftarrow ei_2$;
14: $ei_2 \leftarrow ei_2.ei$;
15: **end if**
16: **until** $ei_1 = ei_2$
17: **return** ei_1 ;

– **Description de la phase 2 :**
La phase 2 se charge d'identifier les sous-automates fermés. Pour chaque état p le couple $(p.ei, p)$ représente potentiellement un plus petit sous-automate fermé, car $p.ei = EI(p)$, il l'est réellement si p est un état final, autrement dit si $p = EF(p.ei)$. Pour le savoir, on teste la condition $p.ei.efmin \leq p \leq p.ei.efmax$, si elle est vérifiée alors le couple $(p.ei, p)$ est ajouté à la liste des plus petits sous-automates fermés, sinon il est ignoré.

Sur l'exemple de la figure 3.8, on reconnaît sept plus petits sous-automates fermés en plus de l'automate initial : $(3, 5)$, $(5, 11)$, $(4, 6)$, $(6, 12)$, $(10, 13)$, $(13, 15)$ et $(16, 20)$.

État	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
e_i	0	1	1	2	3	4	5	6	6	2	5	6	10	1	13	1	16	16	18	16	1
ef_{min}	21	10	5	6	11	12	11	12	12	13	21	14	15	16	16	20	20	19	20	21	0
ef_{max}	$+\infty$	4	5	6	11	12	11	12	12	13	14	14	15	16	16	20	18	19	20	21	$+\infty$

TAB. 3.2 – Caractérisation de sous-automates particuliers

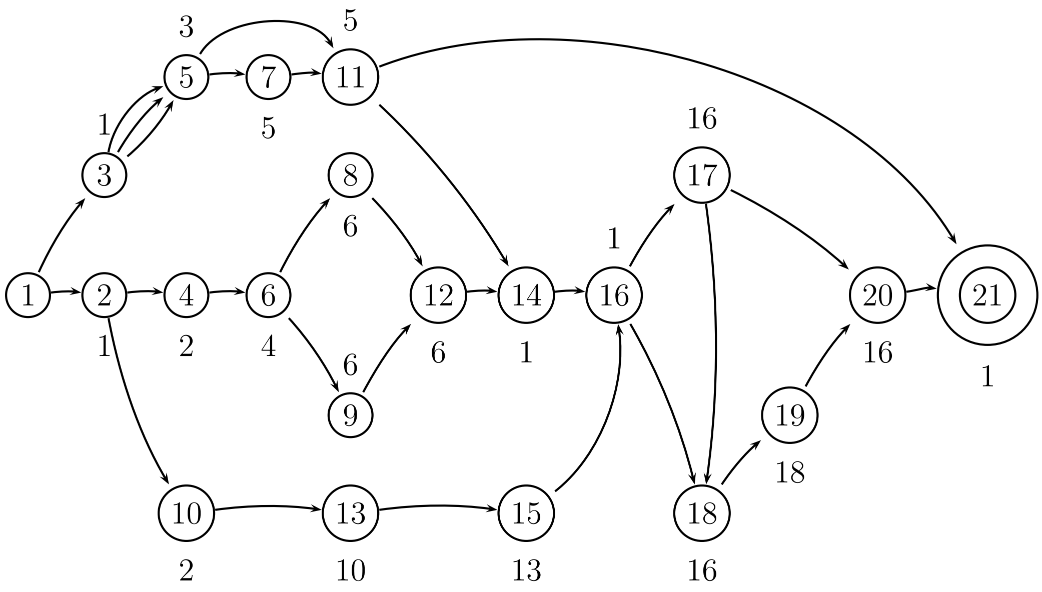


Fig. 3.8 – Recherche de sous-automates

3.3.3 Résultats expérimentaux

L'algorithme a été implanté et expérimenté sur les automates représentant les deux dictionnaires français de la table 3.3. Le dictionnaire des Noms des villes et localités françaises et le dictionnaire DELAF [Courtois et Silberztein, 1990] qui contient presque tous les mots simples de la langue française. Un post traitement a été appliqué aux résultats afin de concaténer toutes les transitions fermées pour construire des séquences les plus longues possible. Les résultats observés sont portés table 3.4.

Dictionnaire	Mots	États	Transitions
Villes françaises	38 472	61 240	95 589
DELAF français	637 283	67 995	177 465

Tab. 3.3 – Dictionnaires

Automate	Parallèles fermés (Cat1)	Séquences fermées	PPSAF complexes(Cat3)
Villes françaises	0	8 175	2
DELAF français	7	8 444	3

Tab. 3.4 – Résultats expérimentaux

L'algorithme $PPSAF$ détecte quasiment exclusivement des séries sur ces deux automates. Si ils contiennent des sous-automates parallèles purs ou des sous automates plus complexes, ils ne sont pas fermés.

Après comparaisons entre elles de ces séries, il s'avère que le nombre de séries identiques est plus important dans l'automate représentant le dictionnaire des noms des villes et localités françaises. Cela s'explique probablement par le fait que les noms des villes se composent souvent d'unités polylexicales semblables, par exemple, le mot 'Saint', la préposition 'sur', etc.
Toutes ces répétitions peuvent être intéressantes à indexer et à compresser pour optimiser la taille de l'automate.

3.3.4 Preuve de l'algorithme

Nous allons démontrer les propriétés 3.13 à 3.15 liées à la phase 1.

Propriété 3.13
$\forall q > 1,\ q.ei = EI(q)$
Démonstration
Posons l'hypothèse de récurrence

$$\mathcal{H}(k) : \text{"À l'étape } k,\ \forall q > 1 : q.e_i = source(Pred(q) \cap [1..k])\text{"}$$

avec la convention $source(\emptyset) = 0$.
• $\mathcal{H}(1)$ est vraie.
En effet, à la première étape, seuls les successeurs du sommet 1 sont mis à jour et leurs champs ei sont forcément mis à 1 puisqu'ils sont initialement nuls. Les autres états ne sont pas modifiés, leurs champs ei restent à zéro.
• supposons $\mathcal{H}(k)$ vraie.
À l'étape $k+1$ on traite l'état portant le numéro $k+1$.Étant donné un état q
- Si $q \notin Succ(k+1)$ alors $q.ei$ n'est pas modifié. Il vaut par hypothèse de récurrence $source(Pred(q) \cap [1...k])$ or $Pred(q) \cap [1..k] = Pred(q) \cap [1...k+1]$ donc $q.ei = source(Pred(q) \cap [1...k+1])$
- Si $q \in Succ(k+1)$
- Si $k+1$ est le plus petit prédécesseur de q (par la numérotation) alors $q.ei$ passe de 0 à $k+1$, or $Pred(q) \cap [1...k] = k+1$ et $source(k+1) = k+1$
- Si $k+1$ n'est pas le plus petit prédécesseur de q, alors $q.ei$ prend la valeur $source(source(Pred(q) \cap [1...k] \,; k+1) = source(source(Pred(q) \cap [1...k] \,; source(k+1))$
$source(source(Pred(q) \cap [1...k] \,; k+1) = source(Pred(q) \cap [1...k] \cup \{k+1)\}$ (par associativité de $source$)
$source(source(Pred(q) \cap [1...k] \,; k+1) = source(Pred(q) \cap [1...k+1])$.
Donc $\mathcal{H}(k+1)$ est vraie.

Donc à l'étape $n-1$: $q.ei = source(Pred(q) \cap [1,..,n-1]) = source(Pred(q)) = EI(q)$.
□

Notations
Afin d'éviter toute confusion, nous noterons $EFMIN(p)$ et $EFMAX(p)$ les valeurs de $p.efmin$ et $p.efmax$ à l'issu de l'algorithme. On vient de montrer que $EI(p)$ est la valeur finale de $p.ei$.

Propriété 3.14
$\forall p \leq n-1, EFMIN(p) = Max\left(Max\left\{q/EI(q) = p\right\}; Max(Succ(p))\right)$
avec comme convention $Max(\emptyset) = 0$.

Démonstration
• $EFMIN(p) \leq Max\left(Max\left\{q/EI(q) = p\right\}; Max(Succ(p))\right)$
Le champ $p.efmin$ ne peut être mis à jour que ligne 6 ou ligne 13. Ce qui est important, c'est la dernière fois que $p.efmin$ est modifié.
- Si c'est ligne 6, alors $EFMIN(p) = Max(Succ(p))$,
- Si c'est ligne 13, alors $EFMIN(p)$ prend une valeur q telle que $q.ei = p$. À ce moment on est en train de remonter un chemin à partir d'un certain état r, chemin qui passe par q. Cela signifie que tous les prédécesseurs de q ont été traités auparavant et que $q.ei$ a déjà pris sa valeur finale. Donc $EFMIN(p)$ prend une valeur q telle que $EI(q) = p$ et donc $EFMIN(p) \leq Max\left\{q/EI(q) = p\right\}$
• $EFMIN(p) \geq Max\left(Max\left\{q/EI(q) = p\right\}; Max(Succ(p))\right)$
À l'étape p on définit $p.ef_{min} \leftarrow Max(Succ(p))$. Donc, $p.ef_{min} \geq Max(Succ(p))$.
Soit $q \in Q$ tel que $EI(q) = p$.
- Si p est l'unique prédécesseur de q alors $q = Max(Succ(p))$, donc la propriété est vrai.
- Si q possède plusieurs prédécesseurs, soit k son plus grand prédécesseur. À l'étape k on sera ammené à mettre à jour pour la dernière fois $q.e_i$ par la fonction "Trouver Source", on obtiendra p (car la dernière mise à jour de $q.ei$ donne $EI(q)$) et la ligne suivante (ligne 13) correspondra alors à $p.ef_{min} \leftarrow Max(p.ef_{min}; q)$. Donc $p.efmin$ prendra une valeur plus grande que q. □

Lemme 3.1
Si $EFMIN(p) \leq EFMAX(p)$ alors $\forall r \in Succ^*(p)$ tel que $EI(r) < p$, tout chemin de p à r passe forcément par $EFMIN(p)$.

Démonstration
Soit $\mathcal{R} = \{r \in Succ^*(p) \mid EI(r) < p\}$ et $\tilde{r}$ le premier état de $\mathcal{R}$ pour lequel on va remonter un chemin passant par p ($\tilde{r}$ est le premier état pour lequel on aura traité un ou plusieurs prédécesseurs successeurs de p et un ou plusieurs prédécesseurs non successeurs de p). En remontant le chemin de $\tilde{r}$ vers p vers o (voir figure 3.9), p.efmax sera mis à une valeur q telle que $q.ei = p$. Si $EFMIN(p)$ n'est pas un prédécesseur de $\tilde{r}$, on aura forcément $q < EFMIN(p)$, et comme $p.efmax$ ne peut que diminuer au cours de l'algorithme, il sera impossible d'obtenir $p.efmax \geq EFMIN(p)$. □

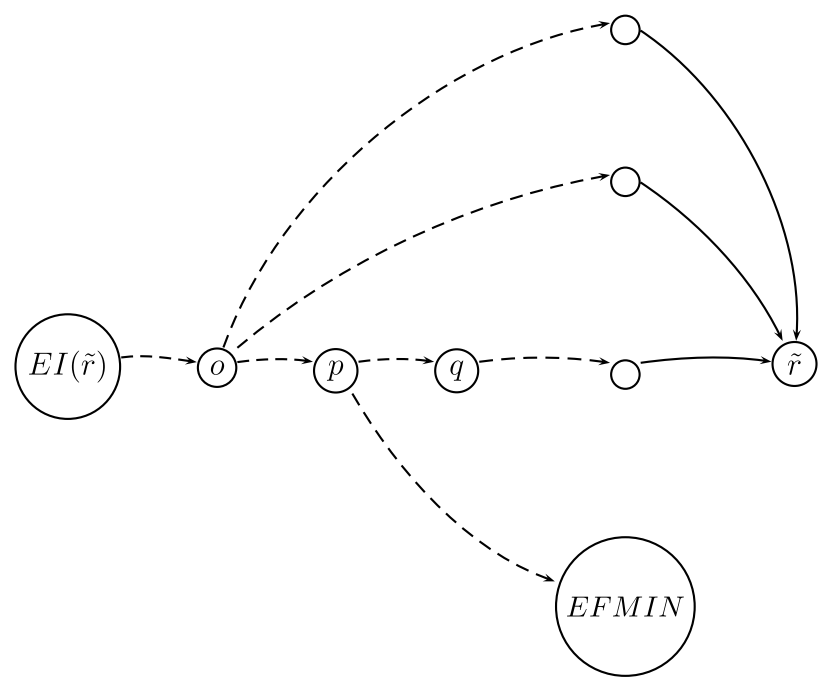


FIG. 3.9 – Cas d'une mise à jour de $p.efmax$

Lemme 3.2
Si $EI(q) = p$ et $EFMIN(p) \leq q \leq EFMAX(p)$ alors $q = EFMIN(p)$ et (p, q) est un $PPSAF$

Démonstration
• $EI(q) = p$, donc d'après la propriété 3.14, $q \leq EFMIN(p)$, or par hypothèse $q \geq EFMIN(p)$ donc $q = EFMIN(p)$.
• (p, q) est un $PPSAF$?
On sait que $EF(p) = EF(EI(q)) \geq q$ (cf. propriété 3.9).
- Si $EF(p) = q$, alors on a à la fois $EF(p) = q$ et $EF(q) = p$, donc (p,q) est un $PPSAF$.
- Si $EF(p) > q$, d'après la propriété 3.9, $EI(EF(p)) \leq p$. On ne peut pas avoir $EI(EF(p)) = p$ sinon on aurait $EFMIN(p) \geq EF(p)$ et donc $EFMIN(p) > q$ ce qui est contraire à l'hypothèse $EFMIN(p) \leq q \leq EFMAX(p)$. Donc $EI(EF(p)) < p$, donc $EFMIN(p)$ est sur tout chemin de p à $EF(p)$, d'après le lemme 3.1, or $EF(p)$ est sur tout chemin de p à q_f donc $EFMIN(p)$ est sur tout chemin de p à q_f. Donc $EFMIN(p)$ est un nœud postérieur de p, ce qui n'est pas possible car $EF(p)$ est le plus petit nœud postérieur.
Donc $EF(p) = q = EFMIN(p)$ et (p, q) est un $PPSAF$. □

Lemme 3.3
Si (p, q) est un $PPSAF$ alors $EFMIN(p) \leq q \leq EFMAX(p)$

Démonstration
• $EFMIN(p) \leq q$?
Soit r tel que $EI(r) = p$, on sait que $EF(EI(r)) \geq r$ (prop. 3.9), or $EF(EI(r)) = EF(p) = q$, donc $q = Max\{r/EI(r) = p\}$.

Soit $r = Max(Succ(p))$, r est sur un chemin de p à q car (p, q) est un SAF, donc $r \leq q$.
Donc $q \geq EFMIN(p)$.
• $q \leq EFMAX(p)$?
$p.efmax$ part de $+\infty$ et est mis à jour à chaque fois que l'on croise un état r, successeur de p tel que $EI(r) < p$. Soit r un tel état.
- Si $r < q$ alors r appartient au $PPSAF$ (p, q) et donc $EI(r) \geq q$.
- Si $r > q$ alors tout chemin de p à r passe par q, car $q = EF(p)$, donc quand la fonction "$TrouverSource$" remontera un chemin de r en direction de p, elle mettra $p.efmax$ à une valeur au moins égale à q.
Donc $q \leq EFMAX(p)$ □

Ces deux derniers lemmes démontrent la propriété 3.15.

Propriété 3.15
$\forall q > 1$,, $(q.ei, q)$ est un $PPSAF$ si, et seulement si $q.ei.efmin \leq q \leq q.ei.efmax$.

3.3.5 Complexité

La fonction $Trouver\ Source(ei_1, ei_2, p)$ parcourt au maximum $max(ei_1, ei_2)$ états[2]. Donc à l'étape p, si $Trouver\ Source(q.ei, p, q)$ est appelé, elle parcourra au maximum p états. Les autres instructions de l'algorithme s'exécutent en $O(1)$, donc la complexité maximale de l'algorithme est donnée par la formule

$$\sum_{p=1}^{n-1} p = \frac{n(n-1)}{2}$$

L'algorithme $PPSAF$ est donc au maximum en $O(n^2)$. Cette complexité est atteinte sur l'automate de la figure 3.10

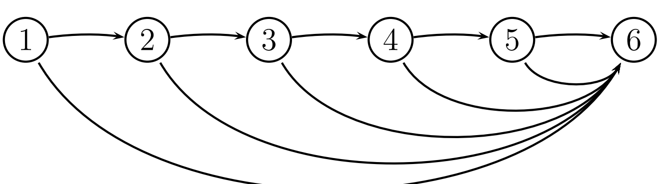


FIG. 3.10 – Exemple d'automate traité en temps quadratique par l'algorithme $PPSAF$

[2]Ce maximum est celui des numéros associés aux états ei_1 et ei_2 ; rappelons que ces numéros correspondent à l'ordre topologique.

3.4 Recherche des plus petits sous-automates "$PPSA$"

Les automates ne semblant contenir que peu de sous-automates fermés, mis à part des séries, nous avons cherché à lever la contrainte de fermeture. C'est ce qui a amené à la définition de plus petits sous-automate associé à un état ou un ensemble d'états (cf. section 3.2.5).

D'une certaine façon, cette approche prend le contre-pied de la précédente. Les $PPSAF$ sont des sous-automates maximaux en largeur et minimaux en longueur, les $PPSA$ sont minimaux en largeur et en longueur, à la différence près qu'un parallèle pur ou une transition ne peuvent pas être des $PPSA$. Le plus petit $PPSA$ est une série de longueur 2, c'est le $PPSA$ associé à l'état interne de la série. On peut alors envisager, en disposant de la liste de tous ces $PPSA$, de reconstruire l'ensemble des sous-automates présents dans l'automate.

Dans cette partie, nous étudions les propriétés d'un $PPSA$ associé à un ensemble d'états et nous en déduisons un algorithme pour calculer ce sous-automate. Cet algorithme est ensuite adapté pour calculer chaque $PPSA$ associé à chaque état interne de l'automate initial.

3.4.1 Définitions et notations

Pour nos calculs futurs de recherche de prédécesseurs et successeurs communs d'un ensemble d'états E, nous définissons pour chaque ensemble d'états E :

- L'ensemble des prédécesseurs dans E de tout élément de E, noté $E^<$.
$$E^< = E \cap \bigcap_{p \in E} Pred^*(p)$$
- L'ensemble des successeurs dans E de tout élément de E dans E, noté $E^>$.
$$E^> = E \cap \bigcap_{p \in E} Succ^*(p)$$
- L'ensemble des états interne de E, noté $\overset{o}{E}$, ces états, appartenant à E ne précèdent ni ne succèdent tous les éléments de E
$$\overset{o}{E} = E \setminus (E^< \cup E^>)$$
- L'ensemble des états de E qui ne sont pas successeurs de tout élément de E, noté $E^{\leq}$.
$$E^{\leq} = E^< \cup \overset{o}{E}$$
- L'ensemble des états de E qui ne sont pas prédécesseurs de tout élément de E, noté $E^{\geq}$.
$$E^{\geq} = E^> \cup \overset{o}{E}$$

Remarque 3.8
Quelque soit l'ensemble d'états E, $|E^<| \leq 1$ et $|E^>| \leq 1$, sinon l'automate contient un cycle.

Exemple 3.11
Soit E un ensemble d'état de l'automate donné par la figure 3.11 tel que $E = \{7, 9, 10, 14\}$.
$E^<$=$\{7\}$

$E^{>}$={14}
$\overset{o}{E}$={9, 10}
$E^{\leq}$={7, 9, 10}
$E^{\geq}$={9, 10, 14}

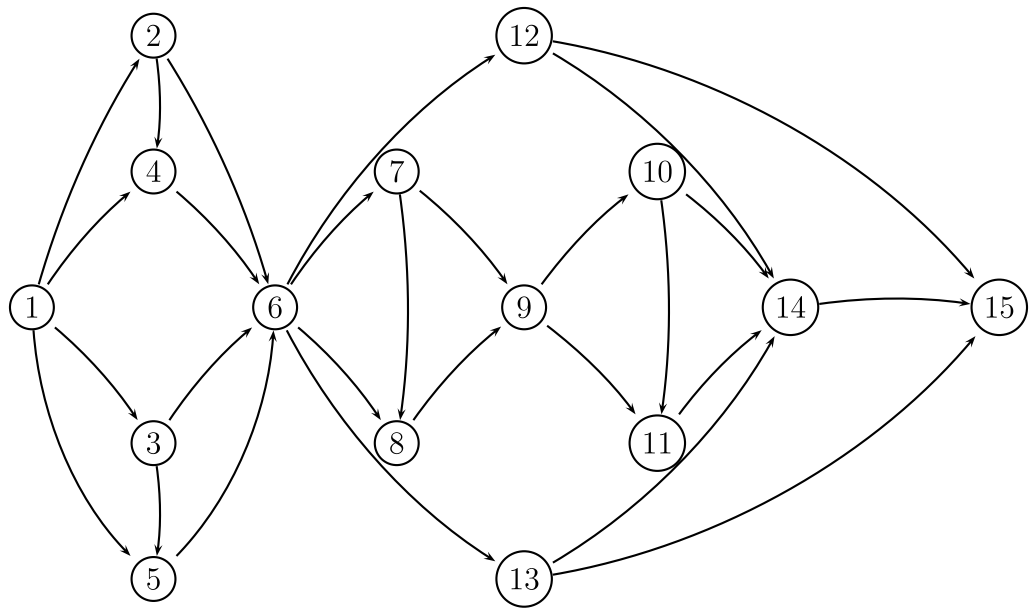


FIG. 3.11 – Recherche de sous-automates minimaux en largeur

Les algorithmes que nous allons présenter s'inspirent des propriétés suivantes qui, elles, découlent de la définition **??**.

Propriété 3.16
Soit $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *un sous-automate.*

1. $Q' \subset Pred^*(s_f) \cap Succ^*(s_i)$[3]
2. $\forall q \in Q' \setminus \{s_f\} : Succ^*(s_i) \cap Pred^*(q) \subset Q'$
3. $\forall q \in Q' \setminus \{s_i\} : Pred^*(s_f) \cap Succ^*(q) \subset Q'$
4. $Q'^{<} = \{s_i\}$ *et* $Q'^{>} = \{s_f\}$

Propriété 3.17
Soit $A' = < \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *un sous-automate.*

1. $\forall E \subset Q' \setminus \{s_f\} : Succ^*(s_i) \cap Pred^*(E) \subset Q'$
2. $\forall E \subset Q' \setminus \{s_i\} : Pred^*(s_f) \cap Succ^*(E) \subset Q'$

Dans ce qui va suivre, nous cherchons à détecter les plus petits sous-automates correspondant à la définition 3.10 rappelée ci-dessous.

[3]Déjà présent dans la définition **??**.

Définition 3.10 ($PPSA$ associé à un ensemble d'états)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ *un automate et* $E \subset Q$, $E \neq \emptyset$. *Le plus petit sous-automate associé à* E *(*$PPSA(E)$*) est le sous-automate* $A' =< \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *tel que :*
- *(i)* $E \subset Q'$ *et*
- *(ii)*$\forall A''$ *sous-automate de* A *dont l'ensemble des états* Q'' *contient* E, $Q' \subset Q''$.

Définition 3.19 (Automate Partiel Initial d'un ensemble d'états)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ *un automate et* $E \subset Q$, $E \neq \emptyset$.
On note $IPA(E) =< \Sigma, Q_{IPA}(E), \delta_{IPA}(E), source(E^{\leq}), E >$ *l'automate partiel initial associé à* E*, tel que :*
- $Q_{IPA}(E)$ *est l'ensemble de tous les états appartenant à* E *ou appartenant aux chemins reliant* $source(E^{\leq})$ *à* $E^{\leq}$
- $\delta' : \begin{cases} Q_{IPA}(E) \times \Sigma \rightarrow Q_{IPA}(E) \\ \forall (q, \alpha) \in Q_{IPA}(E) \times \Sigma : \delta_{IPA}(E)(q, \alpha) = \delta(q, \alpha) \end{cases}$
- $source(E^{\leq})$ *est l'état initial*
- E *est l'ensemble des états finaux*

Définition 3.20 (Automate Partiel Final d'un ensemble d'états)
Soit $A =< \Sigma, Q, \delta, q_i, q_f >$ *un automate et* $E \subset Q$, $E \neq \emptyset$.
On note $FPA(E) =< \Sigma, Q_{FPA}(E), \delta_{FPA}(E), E, puits(E^{\geq}) >$ *l'automate partiel final associé à* E*, tel que :*
- $Q_{FPA}(E)$ *est l'ensemble de tous les états appartenant à* E *ou appartenant aux chemins reliant* $E^{\geq}$ *à* $puits(E^{\geq})$
- $\delta' : \begin{cases} Q_{FPA}(E) \times \Sigma \rightarrow Q_{FPA}(E) \\ \forall (q, \alpha) \in Q_{FPA}(E) \times \Sigma : \delta_{FPA}(E)(q, \alpha) = \delta(q, \alpha) \end{cases}$
- E *est l'ensemble des états initiaux*
- $puits(E^{\geq})$ *est l'état final*

Les propriétés suivantes illustrent les relations entre les sous-automates et les quatre définitions précédentes : les états source et puits, automate partiel initial et final associés à l'ensemble E.

Propriété 3.18
Soit $A' =< \Sigma, Q', \delta_{/Q'}, s_i, s_f >$ *un sous-automate de* A *et* $E \subset Q'$, $E \neq \emptyset$ *:*
- $source(E^{\leq}) \in Q'$ *et* $puits(E^{\geq}) \in Q'$
- $Q_{IPA}(E) \subset Q'$ *et* $Q_{FPA}(E) \subset Q'$

Démonstration

- Dans le cas où $E \subset Q'$, la définition d'un sous-automate permet d'affirmer que $s_i \in NA(E^{\leq})$, ainsi, $source(E^{\leq})$ peut être, soit un successeur de s_i, ou bien s_i lui-même. De manière plus générale, tous les états de E sont successeurs de s_i.
  Soit $p \in E^{\leq}$: $source(E^{\leq}) \in Succ^*(s_i) \cap Pred^*(p)$ et $p \neq s_f$, ainsi, $source(E^{\leq}) \in Q'$ (cf. propriété 3.17). Le même raisonnement conduit au résultat $puits(E^{\geq}) \in Q'$.
- Soit $q \in Q_{IPA}(E)$, q appartient à un chemin reliant $source(E^{\leq})$ à $E^{\leq}$, ainsi, il existe $p \in E^{\leq}$ $(p \neq s_f)$ tel que $q \in Succ^*(s_i) \cap Pred^*(p)$. Par conséquent, $Q_{IPA}(E) \subset Q'$, idem pour $Q_{FPA}(E)$.

□

Théorème 3.1
Soit E un ensemble d'états, possédant au moins trois états, on définit la suite suivante :
$\begin{cases} E_0 = E \\ \forall k \geq 1,\ E_{2k-1} = Q_{IPA}(E_{2k-2}) \text{ et } E_{2k} = Q_{FPA}(E_{2k-1}) \end{cases}$
Il existe k_0 tel que : $\forall k \geq 2k_0, E_k = Q_{PPSA}(E)$

<u>Démonstration</u>
Étant donné la propriété 3.18, il est clair que $\forall k, E_k \subset Q_{PPSA}(E)$. De plus, la suite est non décroissante. Ainsi, elle est stationnaire et il existe k_0 tel que $\forall k \geq 2k_0, E_k = E_{2k_0}$. Ici, nous devons montrer que E_{2k_0} représente l'ensemble des états d'un sous-automate.

Soit $s_i = source(E^{\leq}_{2k_0-1})$, $s_f = puits(E^{\geq}_{2k_0})$ et $Q' = E_{2k_0} = E_{2k_0+1}$.
$\{s_i, s_f\} \in Q'$, autrement $E_{2k_0+1} \neq E_{2k_0}$.
On a :

- $\forall q \in Q', q \in E_{2k_0+1}$ ainsi, $q \in Succ^*(s_i)$
- $\forall q \in Q', q \in E_{2k_0}$ ainsi, $q \in Pred^*(s_f)$
- $\forall q \in Q' \setminus \{s_i, s_f\}$ (q existe parce que $|E| \geq 3$), $q \in E_{2k_0+1} \setminus source(E^{\leq}_{2k_0+1})$ ainsi, $Pred(q) \in Q'$.
- $\forall q \in Q' \setminus \{s_i, s_f\}$, $q \in E_{2k_0} \setminus source(E^{\leq}_{2k_0})$ ainsi, $Succ(q) \in Q'$

Q' vérifie toutes les conditions de la définition 3.3 et, par construction, il représente le plus petit ensemble d'états d'un sous-automate contenant E, ainsi $Q' = Q_{PPSA}(E)$. □

Rappelons que nous considérons un automate $A =< \Sigma, Q, \delta, q_i, q_f >$ comme un graphe acyclique orienté $G = (V, T)$ où $V = Q$ et $T = \{(q_1, q_2)/q_1, q_2 \in Q$ et $\exists(q_1, \alpha, q_2) \in Q \times \Sigma \times Q : \delta(q_1, \alpha) = q_2\}$ et que l'on suppose que les états (ou les sommets) sont numérotés en respect de l'ordre topologique du graphe.

3.4.2 Algorithmes de recherche du plus petit sous-automate associé à un ensemble d'états

Étant donné un automate $A =< \Sigma, Q, \delta, q_i, q_f >$, l'algorithme 3 permet de calculer l'automate partiel initial d'un ensemble d'états $E \subset Q$, $E \neq \emptyset$. Il utilise un ensemble L pour stocker les états restant à explorer et un ensemble Q_{IPA} pour stocker les états de $Q_{IPA}(E)$.

Il démarre de $E^{\leq}$ et d'après la définition de l'automate partiel initial, il parcourt à l'envers tous les chemins jusqu'à ce qu'il atteigne l'intersection de tous ces chemins. À la fin, L contient $source(E^{\leq})$.

Algorithm 3 Calcul de $IPA(E)$ - Entrée : A, E - Sortie : $Q_{IPA}(E)$, $source(E^{\leq})$

1: $L \leftarrow E^{\leq}$;
2: $Q_{IPA} \leftarrow E$;
3: **while** $|L| > 1$ **do**
4: Soit t un élément de L différent de $Min(L)$;
/**min selon l'ordre topologique du graphe**/
5: $L \leftarrow L \setminus \{t\}$;
6: $Q_{IPA} \leftarrow Q_{IPA} \cup \{t\}$;
7: Marquer t ; /* Pour indiquer que l'état t a déjà été visité */
8: $L \leftarrow L \cup (Pred(t) \setminus \{q \in Pred(t) | q$ marqué$\})$;
/**On ne visite que les prédécesseurs non encore inclus dans l'automate partiel**/
9: **end while** /**ici, L contient $source(E^{\leq})$**/
10: $Q_{IPA} \leftarrow Q_{IPA} \cup L$;
11: $source(E^{\leq}) \leftarrow min(L)$;

L'algorithme de calcul de $FPA(E)$ (algorithme 4) est similaire à l'algorithme 3 : il retourne en sortie $Q_{FPA}(E)$ et $puits(E^{\geq})$ par un calcul itératif des ensembles $CP(E^{\geq})$ et $NP(E^{\geq})$ des définitions 3.15 et 3.16. À la fin L contient $puits(E^{\geq})$.
Étant donné un automate $A =< \Sigma, Q, \delta, q_i, q_f >$ et un ensemble $E \subset Q$, $E \neq \emptyset$, l'algorithme 5 effectue la recherche du plus petit sous-automate associé à E, d'après le théorème 3.1. Il prend en entrée A et E et calcule itérativement IPA et FPA jusqu'à ce qu'il atteigne un point de fixe. À la fin Q' contient tous les états du plus petit sous-automate, s_i son état initial et s_f son état final.

Algorithm 4 Calcul de $FPA(E)$ - Entrée : A, E - Sortie : $Q_{FPA}(E)$, $puits(E^{\geq})$

1: $L \leftarrow\ E^{\geq}$;
2: $Q_{FPA} \leftarrow\ E$;
3: **while** $|L| > 1$ **do**
4: Soit t un élément de L différent de $max(L)$;
*/*max respectivement à l'ordre topologique du graphe*/*
5: $L \leftarrow L \setminus \{t\}$;
6: $Q_{FPA} \leftarrow Q_{FPA} \cup \{t\}$;
7: Marquer t ; /* Pour indiquer que l'état t a déjà été visité */
8: $L \leftarrow L \cup (Succ(t) \setminus \{q \in Succ(t) | q$ marqué$\})$;
*/*On ne visite que les successeurs non encore inclus dans l'automate partiel*/*
9: **end while** */*ici, L contient $puits(E^{\geq})$*/*
10: $Q_{FPA} \leftarrow Q_{FPA} \cup L$;
11: $puits(E^{\geq}) \leftarrow max(L)$;

Algorithm 5 Calcul de $PPSA(E)$ - Entrée : A, E - Sortie : $Q_{PPSA}(E)$, s_i, s_f

1: $Q' \leftarrow E$;
2: **repeat**
3: $Qtemp \leftarrow Q'$;
4: $CalculIPA(A, Q', Q_{IPA}, s_i)$;
5: $Q' \leftarrow Q_{IPA}$;
6: $CalculFPA(A, Q', Q_{FPA}, s_f)$;
7: $Q' \leftarrow Q_{FPA}$;
8: **until** $Q' = Qtemp$

Exemple 3.12
La figure 3.12 illustre les premières étapes de l'algorithme 5 sur un exemple : l'ensemble initial d'états E se compose des états gris du graphe *(a)*.
Le graphe *(b)* représente les états de $Q_{IPA}(E)$, récupérés après le premier appel de *Calcul IPA*. A cette étape, s_i a une première valeur. Les états gris du graphe *(c)* sont ceux dans $Q_{FPA}(E)$, récupérés après le premier appel de *Calcul FPA*. A cet instant, s_f désigne une première valeur (c'est la fin de la première itération).
A la deuxième itération, $Q_{IPA}(E)$ (resp. $Q_{FPA}(E)$) est représenté par le graphe *(d)* (resp. le graphe *(e)*). Un point fixe est atteint à la fin de la deuxième itération et le plus petit sous-automate est totalement recouvert.

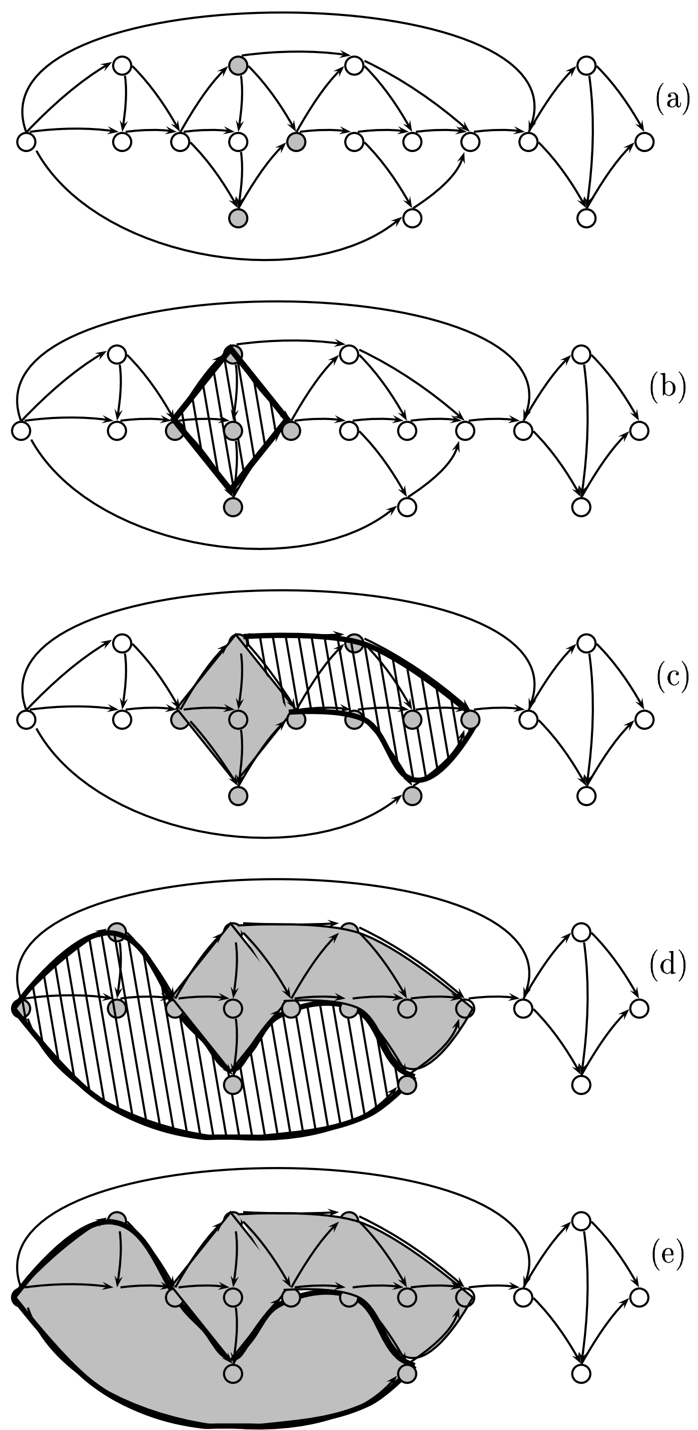


FIG. 3.12 – Recherche du plus petit sous-automate associé aux états gris de l'automate (a)

3.4.3 Recherche de tous les plus petits sous-automates associé à un état

L'obectif est de trouver tous les $PPSA$ associés aux états internes d'un automate $\mathcal{A}$. Un $PPSA$ contient au moins trois états, son état initial, son état final et un état interne. L'algorithme 5 calcule un sous-automate qui contient strictement $\overset{o}{E}$, nous allons l'appliquer à un ensemble E pour lequel $\overset{o}{E}$ est un singleton $\{p\}$ (dans le but d'obtenir le plus petit sous-automate). Une possibilité est de définir E comme étant composé de p, d'un prédécesseur de p et d'un successeur de p.
Si on considère tous les états $p \in Q \setminus \{q_i, q_f\}$, l'exécution de l'algorithme 5 pour $E = \{pred, p, succ\}$ ou $pred \in Pred(p)$ et $succ \in Succ(p)$ permet d'énumérer l'ensemble des plus petits sous-automates de $\mathcal{A}$.

L'application de cette idée mène à l'algorithme suivant (Algorithme 6).

Algorithm 6 CalculBasique $PPSA(p)$ - Entrée : A, p - Sortie : $Q_{PPSA}(p)$, s_i, s_f

1: $E \leftarrow \{p\} \cup \{pred\} \cup \{succ\}$;
2: CalculPPSA(E)

Si on n'y prend pas garde, cet algorithme peut engendrer des calculs inutiles. L'algorithme Calcul$PPSA$ (algorithme 5) consiste en une succession d'appels à CalculIPA et CalculFPA. Après une exécution de CalculIPA, on obtient un ensemble Q_{IPA} que l'on passe en paramètre, au travers de Q', à CalculFPA. CalculFPA va passer en revue tous les états de $Q_{IPA}^{\leq}$ pour étudier leurs successeurs, y compris ceux dont tous les successeurs appartiennent déjà à Q_{IPA}. L'algorithme CalculIPA présente le même travers. On aura donc tendance à réexplorer inutilement plusieurs fois les mêmes chemins.

L'algorithme 7 est une variation de l'algorithme 5 qui traite le cas particulier $|\overset{o}{E}| = 1$, dans laquelle un certain nombre de re-calculs sont évités. Il calcule itérativement les sous-automates partiels initial et final, en utilisant deux listes locales d'états $LPred$ et $LSucc$. Lors d'un calcul d'automate partiel initial, par exemple, il anticipe le calcul de l'automate partiel final suivant en incorporant dans LSucc les successeurs des états visités qui n'appartiennent pas à Q_{IPA}. Le calcul de l'automate partiel final suivant se fera à partir de LSucc, évitant ainsi de repasser inutilement par des états qui n'ont aucun successeur hors de Q_{IPA}. À la fin de l'algorithme, les états marqués sont les états du plus petit sous-automate associé à $E = \{pred, p, succ\}$.
Chaque état exploré est marqué à sa première visite pour n'être testé qu'une seule fois, ses prédecessseurs non marqués sont ajoutés à $Lpred$ et ses successeurs non marqués sont ajoutés à $LSucc$, quel que soit l'action en cours (détermination d'un IPA ou d'un FPA). Ainsi un état n'appartiendra au maximum qu'à une seule de ces deux listes. Tout état testé est retiré de sa liste d'appartenance. La recherche s'arrête lorsque les deux listes sont réduites à un élément. Les états visités lors de cette exploration correspondent alors exactement à l'ensemble des états du sous-automate à détecter, on calcule ainsi le plus petit sous-automate associé à un état. En appliquant ceci à tous les états on obtient l'ensemble des sous-automates minimaux présents dans un automate A.

Algorithm 7 Calcul $PPSA(p)$ - Entrée : A, p - Sortie : $Q_{PPSA}(p)$, s_i, s_f

1: $LPred \leftarrow Pred(p)$;
2: $LSucc \leftarrow Succ(p)$;
3: Marquer p par p
4: **while** $|LPred| > 1 \mid\mid |LSucc| > 1$ **do**
5:    **while** $|LPred| > 1$ **do**
6:       $t \leftarrow max(LPred)$;
7:       $LPred \leftarrow LPred \setminus \{t\}$
8:       Marquer t par p
9:       $LPred \leftarrow LPred \cup Pred(t)$
10:       $L \leftarrow$ tous les élément de $Succ(t)$ non marqués par p ;
11:       $LSucc \leftarrow LSucc \cup L$
12:    **end while**
13:    **while** $|LSucc| > 1$ **do**
14:       $t \leftarrow min(LSucc)$
15:       $LSucc \leftarrow LSucc \setminus \{t\}$
16:       Marquer t par p
17:       $LSucc \leftarrow LSucc \cup Succ(t)$
18:       $L \leftarrow$ tous les élément de $Pred(t)$ non marqués par p ;
19:       $LPred \leftarrow LPred \cup L$
20:    **end while**
21: **end while**
22: $s_i(p) \leftarrow$ l'élément restant dans $LPred$;
23: $s_f(p) \leftarrow$ l'élément restant dans $LSucc$;

Les états explorés sont marqués par p, l'état pour lequel on cherche le $PPSA$ associé. Cela évite de réinitialiser les marquages lorsque l'on itère la méthode à tous les états internes de l'automate.

Exemple 3.13
Soit l'automate donné par la figure 3.13. La recherche du plus petit sous-automate associé à l'état 5 passe par les étapes suivantes :

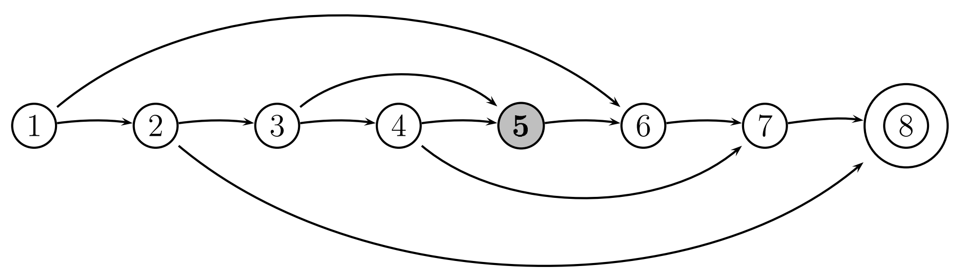
- Initialement $LPred(5) = \{3, 4\}$ et $LSucc(5) = \{6\}$.
  Le traitement des prédécesseurs examine d'abord l'état maximal 4, le retire de $LPred(5)$, le marque comme étant déjà visité et inclut ses prédécesseurs dans $LPred(5)$. L'état 4 ne possède qu'un seul prédécesseur, l'état 3, qui est déjà dans la liste $LPred(5)$. Ensuite, la liste des successeurs de l'état 5 est complétée par les successeurs de l'état 4 non encore visités. Cet ajout permet de parcourir l'ensemble des chemins du sous-automate recherché. Ainsi, $LSucc(5) = \{6, 7\}$.

- $LPred(5) = \{3\}$ et $LSucc(5) = \{6, 7\}$.
  La liste des prédécesseurs de l'état 5 ne contient plus qu'un seul élément, on passe donc au traitement de ses successeurs.
  On examine ainsi le plus petit état 6, on le marque comme étant déjà visité et on inclut ses successeurs dans $LSucc(5)$ et ses prédécesseurs non marqués dans la liste $LPred(5)$.

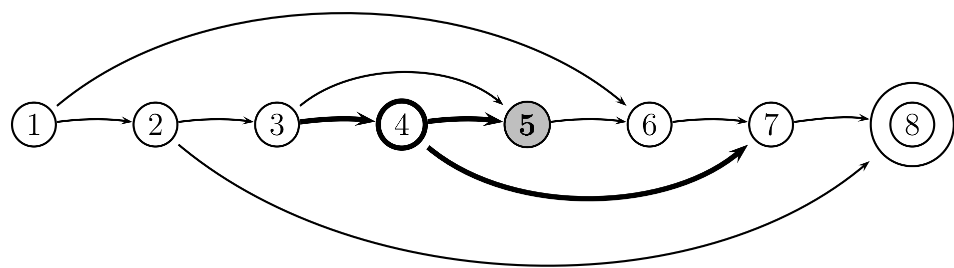
- $LPred(5) = \{1, 3\}$ et $LSucc(5) = \{7\}$.
  La liste des successeurs de l'état 5 ne contient plus qu'un seul élément, on passe donc au traitement de ses prédécesseurs.
  On examine l'état maximal 3, on le marque et on insère ses prédécesseurs dans $LPred(5)$ et ses successeurs non marqués dans la liste $LSucc(5)$.

- $LPred(5) = \{1, 2\}$ et $LSucc(5) = \{7\}$.
  On continue de traiter la liste des prédécesseurs de 5 dans l'ordre décroissant, ainsi, on examine l'état 2, on le marque, puis on ajoute ses prédécesseurs dans $LPred(5)$ et ses successeurs non marqués dans la liste $LSucc(5)$.

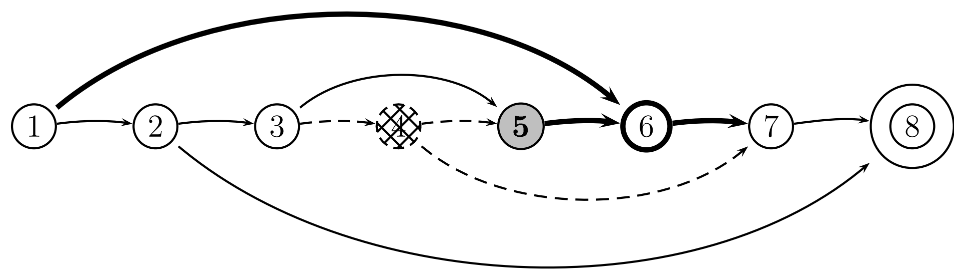
- $LPred(5) = \{1\}$ et $LSucc(5) = \{7, 8\}$.
  La liste des prédécesseurs de l'état 5 ne contient plus qu'un seul élément, on passe donc au traitement de ses successeurs.
  On examine le plus petit état de cette liste, l'état 7, tous ses successeurs sont déjà dans $LSucc(5)$ et ses prédécesseurs sont tous marqués. Les deux listes ne contiennent plus qu'un seul élément, ce qui indique que le plus petit sous-automate associé à 5 est trouvé, il s'agit de l'automate initial.



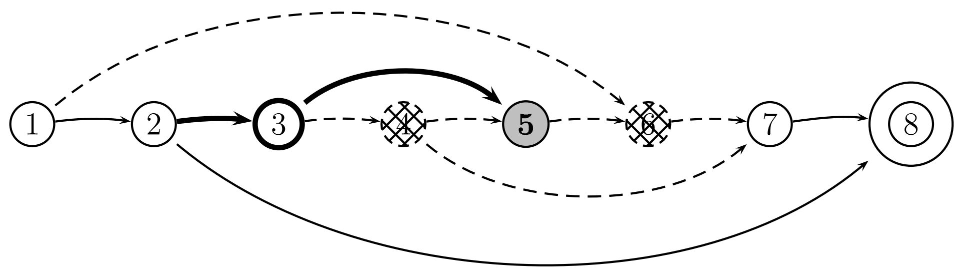
$LPred(5) = \{3,4\}$, $LSucc(5) = \{6\}$, Traitement de $LPred$



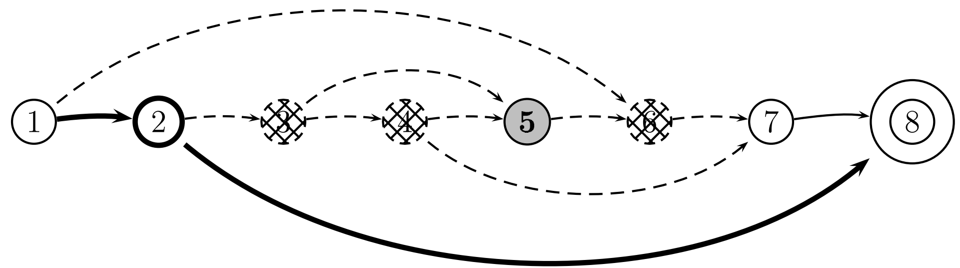
$LPred(5) = \{3\}$, $LSucc(5) = \{6,7\}$, Traitement de $LSucc$



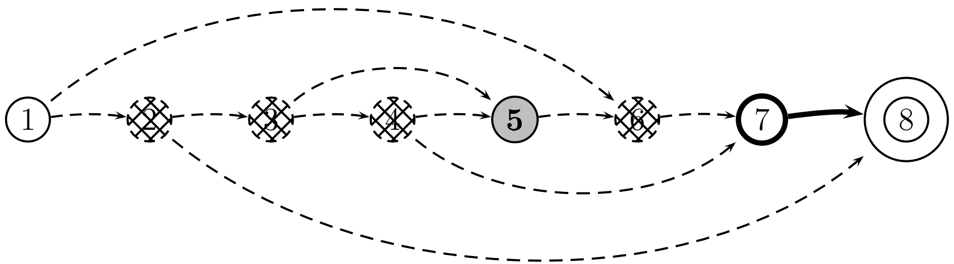
$LPred(5) = \{1,3\}$, $LSucc(5) = \{7\}$, Traitement de $LPred$



$LPred(5) = \{1,2\}$, $LSucc(5) = \{7\}$, Traitement de $LPred$



$LPred(5) = \{1\}$, $LSucc(5) = \{7, 8\}$, Traitement de $LSucc$



$LPred(5) = \{1\}$, $LSucc(5) = \{8\}$, le sous-automate contenant strictement l'état 5 est défini, son état initial est l'état 1 et son état final est l'état 8

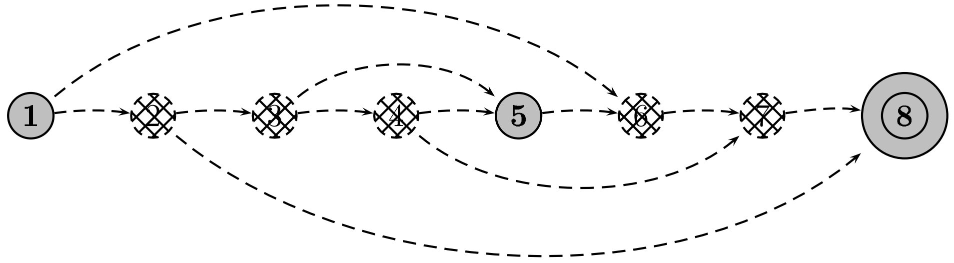


FIG. 3.13 – Recherche du $PPSA$ associé à l'état 5

3.4.4 Complexité de l'algorithme CalculPPSA

Soit n le nombre d'états et m le nombre de transitions de l'automate de départ A. La complexité de l'algorithme 7 est calculée en fonction de la complexité de ses boucles principales.
En supposant que la boucle principale soit exécutée r fois, et qu'à l'exécution k ($k \leq r$), la première boucle interne sur $LPred$ soit exécutée A_k fois et la deuxième boucle interne sur $LSucc$ soit exécutée B_k fois : il est clair que $\sum_{k=1}^{r}(A_k + B_k) \leq n$ parce que chaque état est traité seulement une fois.
Dans chacune des deux boucles internes, la complexité vient des fusions de listes. Or, la fusion de deux listes triées est proportionnelle au nombre d'éléments à fusionner. Donc dans chacune des deux boucles internes la complexité est en $O(|LSucc| + |LPred| + |Succ(t)| + |Pred(t)|)$. Or, $|LSucc| + |LPred| \leq n$ et $|Succ(t)| + |Pred(t)| \leq n$. On obtient donc une complexité maximale en $O(n)$ pour chacune des deux boucles.
Par conséquent, la complexité maximale de l'algorithme 7 est de $\sum_{k=1}^{r}(A_k + B_k) \times n$, qui est en $O(n^2)$.
La recherche de tous les sous-automates minimaux en largeur présents dans un automate donné s'effectue en appliquant l'algorithme 7 à tous les états $p \in Q \setminus \{q_i, q_f\}$, ce qui conduit à une complexité de l'ordre de $O(n^3)$.

Remarque 3.9
Le nombre de transitions sortantes de chaque état est inférieur ou égal à $|\Sigma|$ (la taille de l'alphabet) puisque l'automate est déterministe ; chaque transition de l'automate est franchie deux fois au maximum (1 fois dans chaque boucle). La complexité de l'algorithme 7 est donc au minimum de $O(m)$. Elle est en $O(n * m)$ pour tous les états de l'automate.

Remarque 3.10
La complexité maximale est atteinte pour un automate complet.

Il est possible de réduire la complexité dans la pratique en employant la propriété 3.19.

Propriété 3.19
Soit $A = < \Sigma, Q, \delta, q_i, q_f >$ et $p, p', q, q' \in Q$.
Si les deux conditions suivantes sont satisfaites alors p et q appartiennent au même plus petit sous-automate (voir figure 3.14).
- $\{p, p'\} \subset Pred(q)$ *et* $\{q, q'\} \subset Succ(p)$
- $H(p') \geq H(p)$ *et* $H(q') \leq H(q)$

Démonstration
$H(p') \geq H(p)$, ainsi $p \neq q_i$, mais aussi $q \neq q_f$. Soit r un prédécesseur de p et s un successeur de q et soit $E_1 = \{r, p, q\}$ et $E_2 = \{p, q, s\}$: nous souhaitons montrer que $Q_{PPSA}(E_1) = Q_{PPSA}(E_2)$.
L'état $puits(E_1^{\geq})$ appartient à tous les chemins reliant $\{p, q\}$ à q_f, ainsi, c'est un successeur commun de q et q'. De plus, $puits(E_1^{\geq})) \neq q$ parce que $H(q') \leq H(q)$, ainsi $puits(E_1^{\geq})$ est également un successeur de s. Cela signifie que $s \in Q_{FPA}(E_1)$, mais $Q_{FPA}(E_1) \subset Q_{PPSA}(E_1)$, ainsi $s \in Q_{PPSA}(E_1)$ et $Q_{PPSA}(E_1) = Q_{PPSA}(\{r, p, q, s\})$. Pour les mêmes raisons $Q_{PPSA}(E_2) = Q_{PPSA}(\{r, p, q, s\})$.
Nous pouvons conclure que $Q_{PPSA}(E_1) = Q_{PPSA}(E_2)$. □

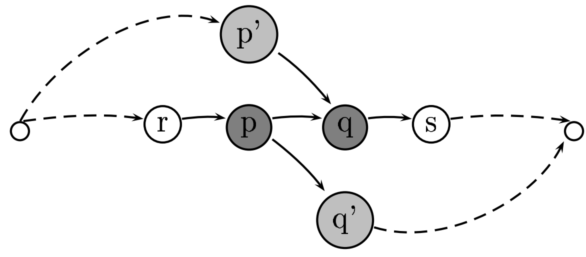


FIG. 3.14 – Illustration de la propriété 3.19

Il est possible d'examiner les deux conditions de la propriété 3.19 avant d'exécuter l'algorithme 7 pour chaque état de A visité dans l'ordre topologique. Plus précisément, l'algorithme 7 sera appelé uniquement pour les états qui ne vérifient pas les conditions de la propriété 3.19. Ceci est illustré par l'algorithme 8.

Algorithm 8 Recherche des $PPSA$- Entrée : A - Sortie : Liste des $PPSA$

```
1: for q = 2 to n − 1 do
2:    if |Pred(q)| ≥ 2 then
3:       p ← max(Pred(q)) ;
4:       p′ ← un prédécesseur de q différent de p ;
5:       H ← min{H(q′) / q′ ∈ Succ(p) ∖ {q}} ;
6:       if H > H(q) then
7:          CalculPPSA(q) ; /*algorithme 7 */
8:       end if
9:    else
10:      CalculPPSA(q) ; /*algorithme 7 */
11:   end if
12: end for
```

Il est plus efficace de lancer la recherche des plus petits sous-automates sur un automate dénué de parallèles et de séries. En effet, d'une part l'algorithme *Recherche des PPSA* proposé ne détecte pas des transitions parallèles, et, d'autre part, si l'automate donné contient des séries, l'algorithme calcule uniquement des séries de longueur deux alors qu'il serait plus intéressant d'extraire les plus longues séries.

3.5 Algorithmes de recherche des sous-Automates séries et sous-automates parallèles

La recherche des sous-automates séries et sous-automates parallèles a été développée à partir des grandes lignes des algorithmes de reconnaissance des sous-graphes séries-parallèles [Valdez *et al.*, 1982], [Bachelet, 2003]. En effet, ces travaux permettent de réduire les parties séries et parallèles d'un graphe G jusqu'à l'obtention d'une unique transition reliant le sommet initial et le final de G ; à cette transition s'ajoute un arbre binaire

de décomposition. Ces méthodes impliquent la suppression de tout sommet ou arc qui gênerait la réduction, ce qui dans notre cas n'est pas envisageable.
Les graphes séries-parallèles sont une classe particulière des graphes [Möhring, 1989], ils sont utilisés entre autre pour la décomposition arborescente d'un graphe, la construction d'algorithmes polynomiaux, la modélisation des synchronisations, etc.

Caractéristiques des sous-automates à détecter

- Les sous-automates parallèles détectés doivent être de largeur maximale.
- Les sous-automates séries doivent de longueur maximale.

- L'algorithme *RechercheParallele* est chargé d'identifier les sous-automates parallèles maximaux en largeur. Chaque état de l'automate est analysé, s'il possède au moins deux transitions sortantes qui mènent vers un même état alors un sous-automate est ajouté à SA.
  L'automate A est mis à jour en remplaçant toutes ses transitions parallèles par une seule transition portant un identifiant unique pour faire référence au sous-automate courant.

Algorithm 9 Recherche des parallèles

for $p \in Q$ **do**
    for tout $q \in Succ(p)$ tel que $|In(q) \cap Out(p)| \geq 2$ **do**
        Créer le sous-automate parallèle (p, q) et lui associer un identifiant unique. Si (p, q) existe déjà on récupère son ancien indice ;
        Mise à jour de A et insertion de (p, q) dans SA s'il n'existe pas déjà ;
    end for
end for

- L'algorithme *RechercheSerie* est chargé d'identifier les sous-automates séries les plus longs en parcourant l'automate A en profondeur. Chaque chemin démarrant d'un état divergent ou convergent, passant par une succession d'états non divergents et non convergents, définit un sous-automate série. Il est construit et inséré à SA par la concaténation des transitions franchies. On le remplace dans A par une unique transition. Cette concaténation engendre la suppression des états intermédiaires contenus dans le chemin formant le sous-automate série.

Algorithm 10 Recherche des séries

for $p \in Q$ tel que $(|Succ(p)| \neq 1$ et $|Succ(p)| \neq 1)$ **do**
    Soit q un successeur immédiat de p non divergent et non convergent ;
    while $|In(q)| = |Out(q)| = 1$ **do**
        Concaténer les deux transitions appartenant aux ensembles $In(q)$ et $Out(q)$;
        Supprimer l'état q ;
    end while
    Créer le sous-automate série (p, q) et lui associer un identifiant unique. Si (p, q) existe déjà on récupère son ancien indice ;
    Mise à jour de A et insertion de (p, q) dans SA s'il n'existe pas déjà ;
end for

La méthode s'exécute en plusieurs passes et permet de fusionner les transitions en parallèles et de concaténer les transitions en séries. L'idée conductrice est de remplacer itérativement les transitions parallèles et les transitions séries par une unique transition qui référence un sous automate série ou parallèle. La recherche considère d'abord les transitions parallèles, puis les transitions séries, à nouveau les transitions parallèles, suivies des séries, etc. Commencer par les parallèles permet de mettre en évidence les séries constituées d'une succession de parallèles. [Tounsi *et al.*, 2006a], [Tounsi *et al.*, 2006b].

Algorithm 11 Recherche SA-SP

```
while Il existe au moins un sous-automate série ou parallèle do
    RechercheParallele() ;
    RechercheSerie() ;
end while
```

3.5.1 Exemple

L'automate A_1 de la figure 3.15 possède trois sous-automates parallèles qui sont aplatis à la première passe et remplacés par α_1, α_2 et α_3. Cela permettra ainsi de définir de plus longues séries à la deuxième passe. A cette étape, on concatène toutes les séries en une unique transition qui donne l'automate A_3, en aplatissant de nouveau les parallèles on obtient l'automate A_4 et en concaténant ensuite les séries on obtient l'automate A_5. L'algorithme se termine donc sur l'automate A_5 qui ne présente plus aucune série ni aucun parallèle. Cet automate reconnaît le même dictionnaire que l'automate A_1 (le même langage), mais ses transitions sont étiquetées soit par des lettres, soit par des sous-automates.

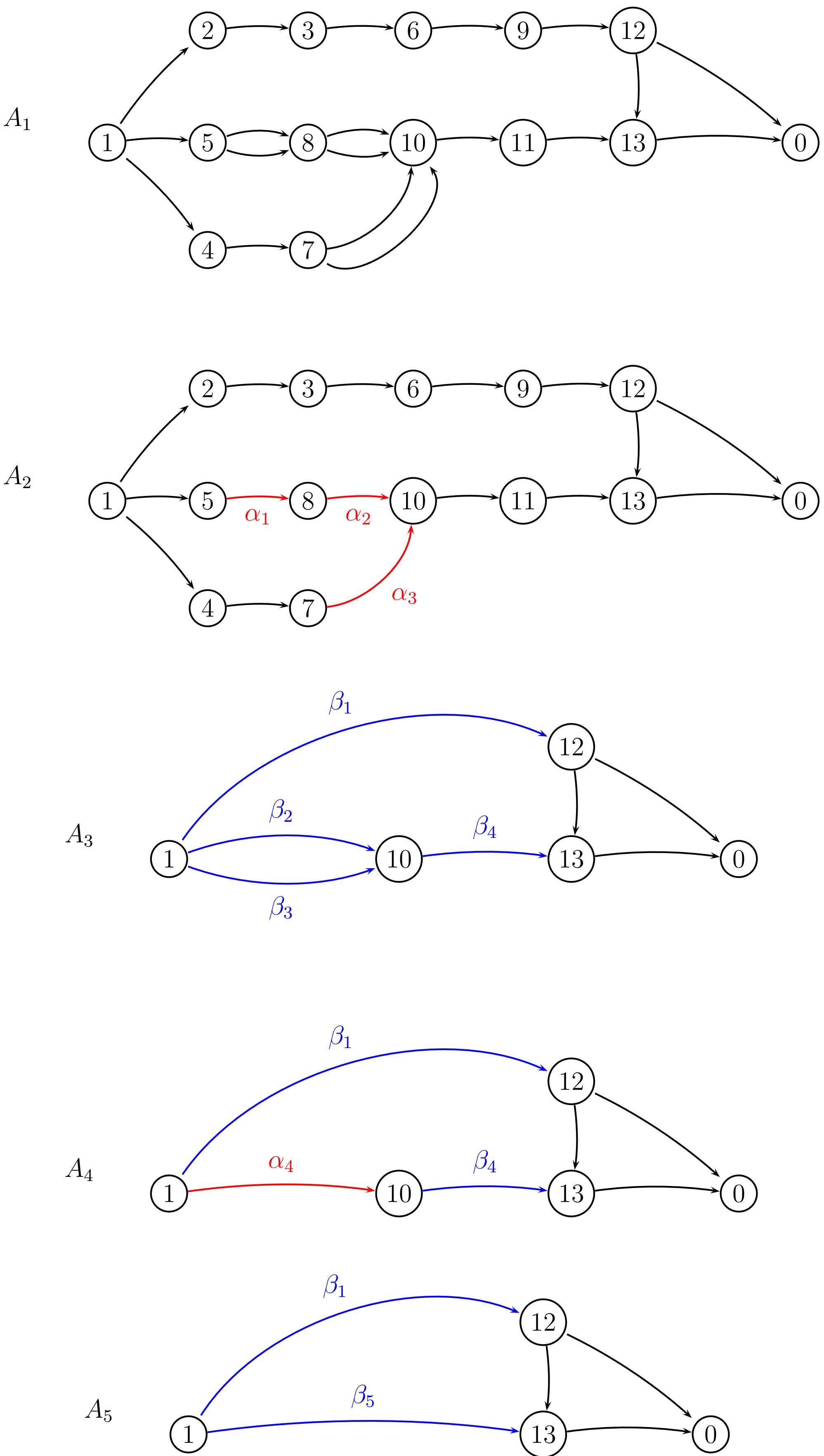


FIG. 3.15 – Recherche de sous-automates séries et sous-automates parallèles

3.5.2 Algorithme général de recherche de sous-automates

Une fois que l'automate courant est dénué de ses parallèles et de ses séries, nous pouvons calculer ses $PPSA$ en appliquant l'algorithme *Recherche* $PPSA$, et ensuite, les réduire (c'est-à-dire les stocker et les remplacer par une transition). Cette étape peut générer de nouveaux sous-automates parallèles ou sous-automates séries, ainsi nous réitérons l'opération comme l'indique l'algorithme 12.

Algorithm 12 Énumération et réduction de tous les sous-automates - Entrée : A - Sortie : sous-automates

---

1: **repeat**
2:     **repeat**
3:         Détecter, stocker et remplacer chaque sous-automate parallèle par une transition ;
4:         Détecter, stocker et remplacer chaque sous-automate série par une transition ;
5:     **until** l'automate soit libéré de tous ses sous-automates parallèles et sous-automates séries
6:     Détecter, stocker et remplacer chaque $PPSA$ par une transition ;
7: **until** l'automate A soit réduit à une unique transition

---

Dans la suite nous ferons référence à cet algorithme sous forme plus condensée :

Algorithm 13 Recherche SA

1: **repeat**
2:     Recherche $SA - SP$; /*Lignes 2-5 de l'algorithme 12*/
3:     Recherche $PPSA$; /*Ligne 6 de l'algorithme 12*/
4: **until** l'automate A soit réduit à une unique transition

---

Exemple 3.14
La figure 3.16 illustre la recherche de sous-automates, l'automate initial composé de 14 états et 21 transitions est réduit à une seule transition reliant deux états en 5 étapes.
D'abord, les sous-automates parallèles et les sous-automates séries sont détectés et remplacés par une transition, ensuite les deux sous-automates minimaux en largeur sont localisés et réduits, finalement une réduction d'un sous-automate série finit l'exécution.
On utilise les lettres α pour indiquer les sous-automates parallèles, les lettres β pour sous-automates les séries et γ pour les autres.

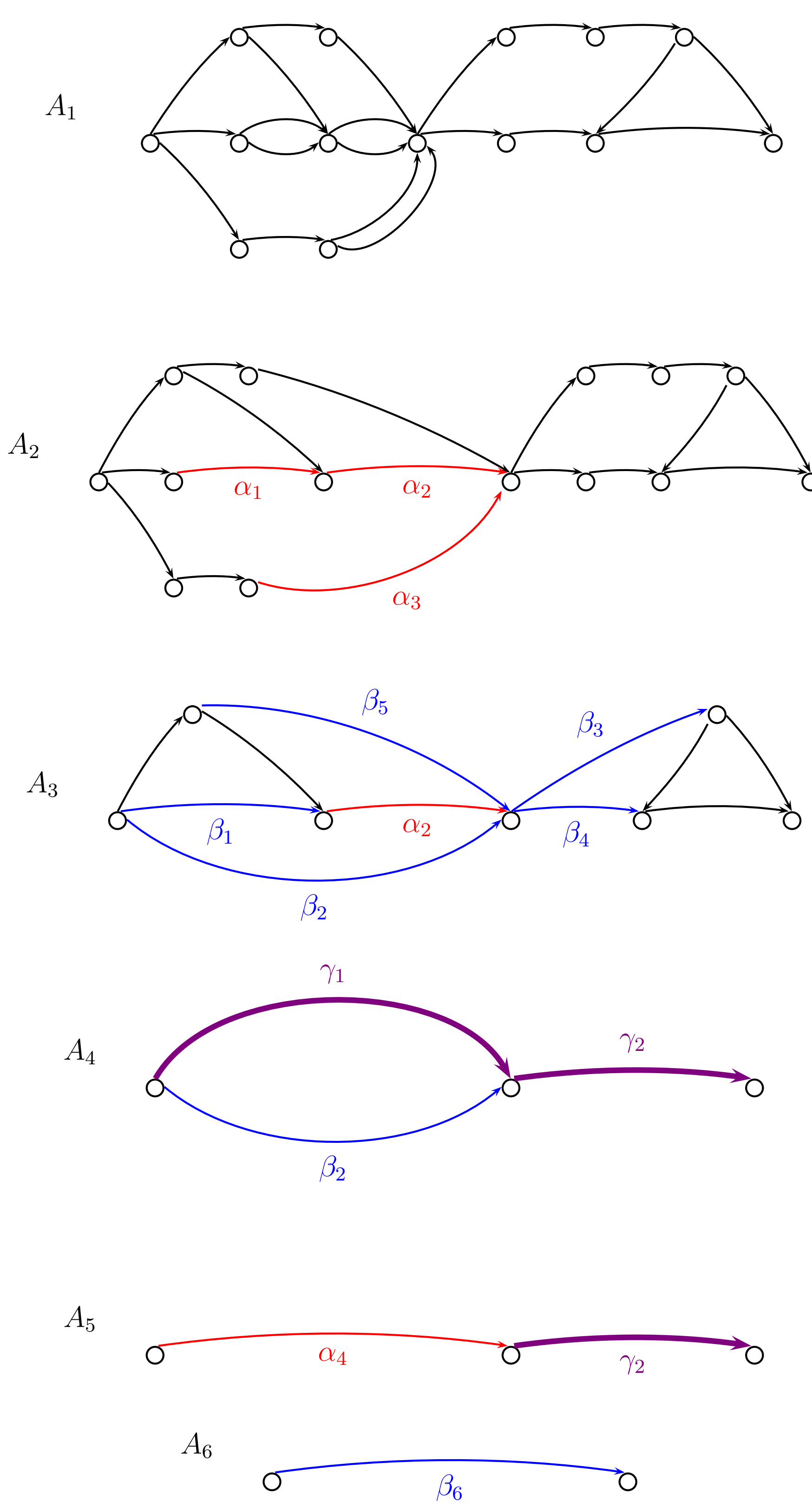


FIG. 3.16 – Réduction de sous-automates

3.6 Conclusion

Dans ce chapitre nous avons détaillé le problème de déterminer la structure interne d'un automate. Nous avons défini la notion de sous-automate et nous avons présenté des algorithmes pour trouver tous les sous-automates d'un automate donné. Une première approche permet d'analyser la structure de l'automate en s'appuyant sur la recherche de sous-automates maximaux en largeur et minimaux en longueur, appelés "sous-automates fermés". La méthode proposée est correcte et calcule l'ensemble des sous-automates fermés présent dans un automate, toutefois les résultats expérimentaux indiquent un nombre réduit de ses sous-structures. Ainsi, dans le but de détailler encore plus la structure interne d'un automate, une seconde approche a été développée pour détecter l'ensemble des sous-automates présents dans l'automate originel.
La méthode de la seconde approche s'applique de préférence à un automate qui ne contient aucun parallèle, ni aucune série ; ainsi, deux états sont reliés au maximum par une seule transition et aucun état n'est à la fois non convergent et non divergent. Au besoin, un pré-traitement de l'automate est nécessaire. Notre algorithme caractérise trois types de sous-automate : *(i) sous-automate parallèle*, *(ii) sous-automate série* et *(iii) plus petit sous-automate* qui n'est ni parallèle, ni série. Tous ces sous-automates peuvent être inclus les uns dans les autres. Pour détecter tout sous-automate d'un FSA A, notre méthode calcule et réduit les parallèles et les séries, avant de calculer et réduire les sous-automates minimaux (en largeur), et elle réitère le processus jusqu'à ce que l'automate donné A soit réduit à une transition. À ce stade, tous ses sous-automates sont connus.

Notre méthode est correcte, c'est-à-dire qu'elle ne détecte que des sous-structures correspondant à la définition des sous-automates, elle calcule tous les sous-automates d'un automate donné A.
La complexité théorique de recherche des sous-automates parallèles et séries est de $O(m^2)$, où m représente le nombre de transitions dans A et dans le pire des cas, la complexité de nos algorithmes de recherche des sous-automates minimaux en largeur est de $O(n^3)$ où n représente le nombre d'états de A. Néanmoins, les résultats expérimentaux démontrent que ces algorithmes ont un bon comportement pour les automates représentant des dictionnaires de langue.
Des représentations particulières du graphe de l'automate pourraient conduire à des simplifications et donc à réduire la complexité comme par exemple dans [Jaumard et Minoux, 1986], mais l'algorithme est appelé à s'appliquer une seule fois et sur des automates de taille réduite, du moins dans les expérimentations que nous avons menées liées au traitement des dictionnaires.

Ces algorithmes ont été testés sur des automates représentant de grands dictionnaires de langues naturelles (travail présenté au chapitre suivant), ces expériences prouvent que ces algorithmes ont un bon comportement dans la pratique. En effet, l'analyse des résultats montre que la réduction de ces automates se produit en peu d'itérations et que les sous-automates sont inclus dans d'autres.

Ce travail conduit naturellement à la compression d'automates, idée qui peut se comparer aux travaux effectués dans [Ristov et Laporte, 1999] ou [Daciuk, 2000]. En effet, la recherche de sous-automates a plusieurs applications potentielles : localiser et

indexer des sous-parties d'un un automate, recherche de données redondantes pour réduire la consommation de mémoire, décomposer un automate de très grande taille en plusieurs plus petit automates, etc.
Nos travaux futurs seront d'adapter nos algorithmes pour des formes plus générales d'automate, par exemple les automate cycliques, et comparer notre méthode aux algorithmes d'extraction, dont le but est de découvrir les motifs fréquents dans les graphes [Yan et Han, 2003], [Barbu *et al.*, 2004] and [Liquière, 2006].

Chapitre 4

Structure interne d'automates représentant des dictionnaires

On appellera ici *dictionnaire* tout ensemble fini W de chaînes non vides sur un vocabulaire V. Les éléments de W sont appelés *mots* et ces mots sont rangés suivant un certain ordre afin d'être retrouvés plus facilement. Les dictionnaires sont souvent représentés par des automates acycliques, car ils fournissent une structure compacte [Revuz, 1991] et des fonctions d'accès et de recherche de motifs des plus rapides. Le temps d'accès est linéaire en fonction de la taille du texte et ne dépend pas de la taille du dictionnaire.
Dans ce chapitre, nous proposons une étude statistique sur la structure interne de onze automates représentant des dictionnaires électroniques. La première partie rappelle quelques résultats classiques sur les automates. Les onze dictionnaires sont présentés dans la deuxième partie, ainsi que des statistiques très générales sur leurs automates. Dans la troisième partie, l'algorithme *Recherche SA* a été appliqué à ces automates, l'étude des sous-automates détectés met en évidence des similarités entre eux.

4.1 Généralités sur les automates

4.1.1 Représentation d'un dictionnaire par un automate

Les automates étudiés sont acycliques car il sont utilisés pour reconnaître des séquences finies (mots, séquences ADN...). Ainsi, tous les chemins d'un automate sont de longueur finie.
Les algorithmes présentés au chapitre précédent s'appliquent à des automates avec un seul état final. Pour construire l'automate déterministe minimal à partir d'un dictionnaire avec un seul état terminal, plusieurs méthodes existent, nous utilisons l'algorithme de Daciuk-Mihov tel qu'il a été présenté dans [Maurel et Guenthner, 2006]. Ce programme considère que chaque mot du dictionnaire est suivi par un caractère de fin de mot "#", la fonction de transition de l'automate δ devient : $\delta : Q \times \Sigma \cup \{\#\} \rightarrow Q$. Ainsi, l'automate ne possédera plus qu'un seul état terminal. Cet ajout ne modifie pas le langage reconnu par l'automate. Ce programme calcule aussi la hauteur et la cardinalité de chaque état.

Propriété 4.1
Tout automate A_1 à nombre fini d'états peut être considéré comme équivalent à un automate A_2 ayant un seul état final et un caractère de fin de mot. Les mots reconnus par le langage de A_2 sont ceux reconnus par le langage A_1 auxquels est concaténé le caractère de fin de mot.

Exemple 4.1

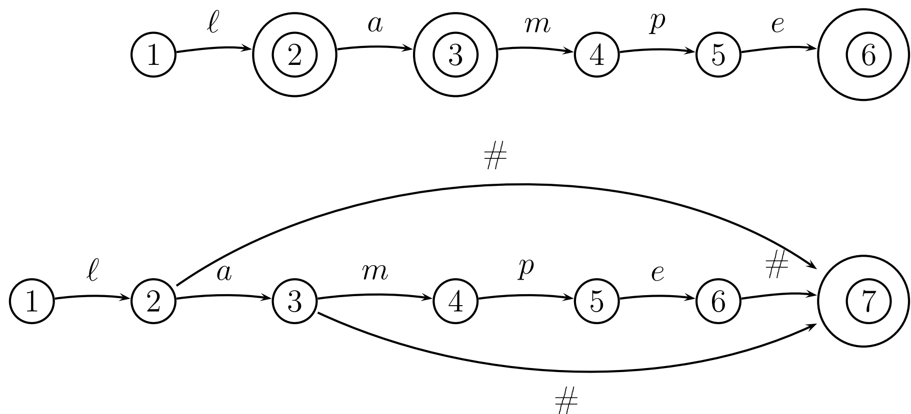


FIG. 4.1 – Automates représentant la liste de mots : ℓ, ℓa, $\ell ampe$

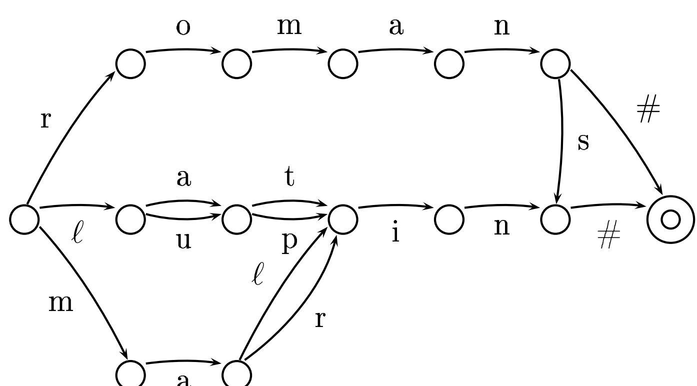
4.1.2 Représentation d'un automate

Contrairement au chapitre précédent, les automates ne sont plus traités uniquement comme des graphes orientés, nous prenons en considération les étiquettes associées aux transitions pour étudier les dictionnaires.
Initialement, à chaque état de l'automate est associé la liste ordonnée de ses transitions sortantes. Cette organisation permet de réduire les temps de calcul pour vérifier l'appartenance d'un mot au dictionnaire.
En mémoire, nous représentons l'automate par la liste des transitions sortantes de chaque état. Chacune de ces transitions porte trois informations : un booléen mis à zéro ou à 1 si elle appartient ou n'appartient pas au même état source que celle qui la précède, son étiquette et l'adresse de la première transition de son état but (cf. table 4.1).

Propriété 4.2
Un mot appartient au dictionnaire si et seulement si il existe un chemin dans l'automate débutant à l'état initial et arrivant à l'état terminal qui reconnaît ce mot.

Exemple 4.2
Le tableau 4.1 correspond à la liste des transitions de l'automate représentant les mots *lapin, lutin, latin, lupin, malin, marin, roman, romans.* Le mot *latin* est reconnu par le parcours des lignes 1, 7, 12, 13, 15, 16 et 19. (Les lignes 12 et 13 sont successivement parcourues pour reconnaître la lettre "t").

1	1	ℓ	7
2	0	m	6
3	0	r	4
4	1	o	5
5	1	m	9
6	1	a	10
7	1	a	12
8	0	u	12
9	1	a	14
10	1	ℓ	15
11	0	r	15
12	1	p	15
13	0	t	15
14	1	n	17
15	1	i	16
16	1	n	19
17	1	#	0
18	0	s	19
19	1	#	0



TAB. 4.1 – Représentation d'un automate

Cette représentation peut s'utiliser avec des étiquettes généralisées correspondant à des expressions rationnelles. En effet, un résultat de Kleene dit que :

Théorème 4.1 (Kleene)
Tout langage reconnu par un automate à nombre fini d'états peut être représenté par une expression rationnelle, et réciproquement.

Une expression rationnelle est basée sur un alphabet et trois opérateurs. Elles sont par exemple utilisés dans des filtres comme l'utilitaire Grep sous Unix.
Les expressions rationnelles permettent d'engendrer des langages dits rationnels. Ces langages sont reconnus formellement par des automates finis.

Définition 4.1 (Expressions rationnelles ou expressions régulières)
Étant donné un alphabet Σ, on appelle expression rationnelle une expression construite par les règles suivantes :
- *tout élément de Σ est une expression rationnelle ;*
- *le mot vide noté ϵ est une expression rationnelle ;*
- *si E_1 et E_2 sont des expressions rationnelles, alors la concaténation, notée E_1E_2, est une expression rationnelle ;*
- *si E_1 et E_2 sont des expressions rationnelles, alors l'union, notée $E_1 + E_2$, est une expression rationnelle ;*
- *si E est une expression rationnelle, l'itération, notée E^*, est une expression rationnelle (L'opérateur '*' est appelé itérateur de Kleene).*

Remarque 4.1
Lorsque l'automate est acyclique, l'expression rationnelle correspondante ne comporte pas d'itération, et réciproquement, une expression rationnelle qui ne contient pas d'itération est représentée par un automate acyclique.

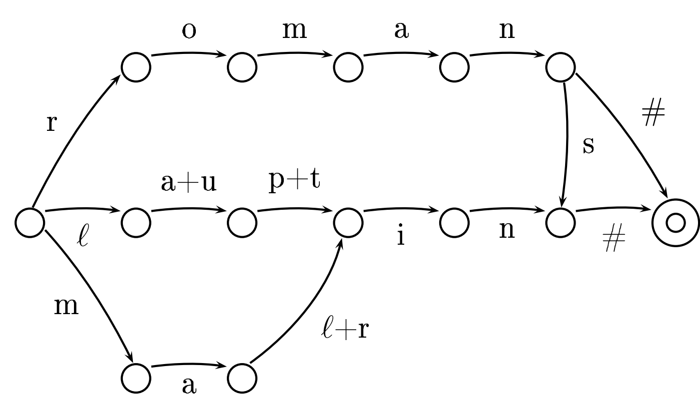
Remarque 4.2
Un sous-automate est un automate (voir définition **??***), donc il peut aussi être représenté par une expression rationnelle.*

Remarque 4.3
En appliquant l'algorithme recherche SA-SP à un automate, les sous-automates détectés sont réduits à une seule transition. Puisque nous tenons compte des étiquettes, cette transition doit porter une expression rationnelle en tant qu'étiquette.

Exemple 4.3
Les tableaux suivants représentent toutes les étapes de déroulement de l'algorithme *Recherche SA-SP* à l'automate précédent. Cette construction s'effectue de manière itérative.

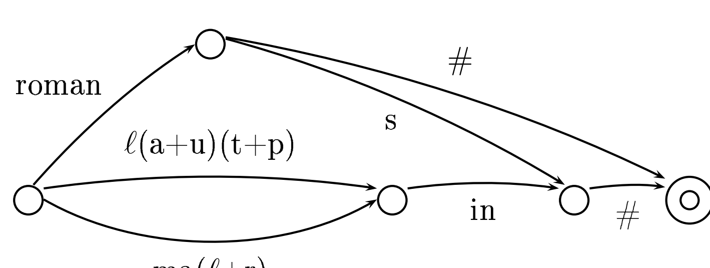
Étape 1

1	1	ℓ	7
2	0	m	6
3	0	r	4
4	1	o	5
5	1	m	8
6	1	a	9
7	1	a+u	10
8	1	a	11
9	1	ℓ+r	12
10	1	p+t	12
11	1	n	15
12	1	i	13
13	1	n	16
14	1	#	0
15	0	s	16
16	1	#	0



Étape 2

1	1	ℓ(a+u)(p+t)	4
2	0	ma(ℓ+r)	4
3	0	roman	5
4	1	in	7
5	1	#	0
6	0	s	7
7	1	#	0



Étape 3

1	1	(ℓ (a+u)(t+p)) +(ma(ℓ+r))	3
2	0	roman	4
3	1	in	6
4	1	#	0
5	0	s	6
6	1	#	0

roman

#

s

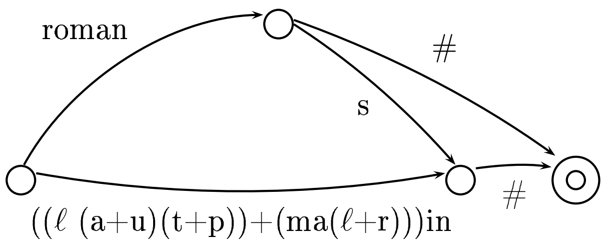
in

#

(ℓ (a+u)(t+p))+(ma(ℓ+r))

Étape 4

1	1	((ℓ (a+u)(t+p))+ (ma(ℓ+r)))in	5
2	0	roman	3
3	1	#	0
4	0	s	5
5	1	#	0



TAB. 4.2: Algorithme *Recherche SA-SP* et expressions rationnelles

L'algorithme *Recherche PPSA* détecte un seul *PPSA*, une étape supplémentaire consisterait à le réduire à une transition portant une nouvelle expression rationnelle, par exemple par une lecture en profondeur (voir la figure ci-dessous).

((ℓ ((ℓ (a+u)(t+p))+(ma(ℓ+r)))in # + roman#+ romans

FIG. 4.2 – Expression rationnelle finale

Il existe plusieurs méthodes de constructions d'expressions régulières [McNaughton et Yamada, 1960], [Glushkov, 1961] et [Thompson, 1968]. En s'appuyant sur l'élimination des états de l'automate donné, les travaux de Han et Wood [Hana et Wood, 2006] permettent de construire des expressions régulières plus courtes, à partir d'un automate à nombre fini d'états.
La longueur de chaque expression régulière dépend du choix de la séquence d'états à éliminer. Ainsi, toutes les séquences possibles sont testées pour aboutir à la solution optimale. La première étape de la construction consiste à examiner la minimisation de l'automate, basé sur l'équivalence des états et la suivante, repose sur des heuristiques permettant les hachages verticales et horizontaux de l'automate.
Ce partitionnement répétitifs permet de déconnecter, puis de fusionner les parties séquentielles et parallèles de l'automate. Ainsi, ce processus présente une ressemblance avec la réduction des sous-automates séries et des sous-automates parallèles.
La réduction des sous-automates séries et des sous-automates parallèles possède aussi des analogies avec la caractérisation des automates Glushkov en termes de graphes. Cette caractérisation permet le passage d'un automate de Glushkov à une expression rationnelle courte. Les automates de Glushkov ont des propriétés particulières : i) L'état initial de l'automate ne possède pas de transition entrante et ii) l'ensemble des transitions entrantes à un état portent la même étiquette.
Pour déterminer si le graphe est de Glushkov, il est nécessaire de le transformer en un graphe sans composantes fortement connexes, avant de le réduire en utilisant trois règles de réduction. Si le graphe d'un automate est réduit à une état unique alors il s'agit d'un automate de Glushkov [Caron, 1997], [Caron et Ziadi, 2000].

4.2 Présentation des automates étudiés

Dans cette partie, nous présentons nos dictionnaires et leurs automates associés ainsi que quelques statistiques très générales telles que : le nombre de mots, le nombre de transitions, la taille de l'alphabet, etc.

4.2.1 Les dictionnaires

Cette partie présente les dictionnaires électroniques sur lesquels nous effectuons notre étude, ils y sont décrits et regroupés en trois catégories.

– **Catégorie 1 :**
Cette catégorie contient des dictionnaires électroniques utilisant le formalisme "DELA" des dictionnaires électroniques du LADL. Ce modèle réunit un ensemble de mots et associe à chaque mot une série d'informations morphologiques, syntaxiques et sémantiques (B. Courtois, 1990, 1999). Les dictionnaires construits peuvent être des dictionnaires de formes fléchies[1] ou des dictionnaires de formes non fléchies[2].
Les dictionnaires examinés relevant de cette catégorie sont le DELAF de la langue française, le DELAF de la langue anglaise, le DELAF de la langue serbe, le DELAF de la langue allemande, un dictionnaire de noms propres représentants les noms des villes et localités françaises et un dictionnaire de mots polylexicaux anglais.
Les tailles de ces dictionnaires sont présentées sur le tableau 4.3. À titre d'exemple, le DELAF français référence 680 000 mots et utilise 6 834 916 caractères pour les représenter. Une entrée d'un dictionnaire DELA se présente sous la forme de l'exemple 4.4.

Exemple 4.4 (Une entrée d'un dictionnaire DELA)
avions, avion.N+CONC:mp.

"avions" est associé au lemme "avion", la lettre "N" indique qu'il s'agit d'un nom, "CONC" signale que la classe distributionnelle est *Concret* et "mp" représente le code flexionnel : masculin pluriel.

Toutefois, dans la suite du travail, il sera tenu compte uniquement de l'ensemble des mots présents dans un dictionnaire et non des informations associées. En effet, celles-ci seront considérées comme les sorties d'un transducteur et l'adaptation de notre travail aux transducteurs est une des perspectives de cette thèse.

Catégorie 1						
	DELAF français	DELAF anglais	DELAF serbe	DELAF allemand	Villes françaises	Polylexicaux anglais
Nombre de mots	637 283	282 338	1 214 417	3 713 121	35 391	320 424
Nombre de caractères	6 834 916	3 098 051	12 739 222	49 382 754	422 476	6 684 899
Taille alphabet	70	73	51	88	74	30

TAB. 4.3 – Dictionnaires électroniques de la catégorie 1

– **Catégorie 2 :**
Cette catégorie contient des dictionnaires électroniques élaborés par une équipe de

[1] "DELAF" DELA de formes fléchies et le "DELACF" DELA de formes composées fléchies.
[2] "DELAS" DELA de formes simples et le "DELAC" DELA de formes composées.

recherche de l'Université de Neuchâtel en Suisse (linguistes et informaticiens). Ces dictionnaires représentent une collection de mots simples et de mots composés de plusieurs langues étrangères collectés sur des sites web et essentiellement à partir d'éditions électroniques de journaux, comme celle du journal "Le Monde". Les langues dépouillées sont le français, le portugais, le hongrois et le bulgare. La particularité de ces dictionnaires est de contenir des chiffres et plusieurs caractères spéciaux tels que le l'arobase, l'esperluette, le point, etc.

Catégorie 2				
	Français	Hongrois	Bulgare	Portugais
Nombre de mots	236 057	191 738	165 073	398 839
Nombre de caractères	1 734 484	1 734 227	1 351 567	3 471 142
Taille alphabet	42	42	41	48

TAB. 4.4 – Dictionnaires électroniques de la catégorie 2

- **Catégorie 3 :**
  Cette catégorie contient un dictionnaire représentant des données génomiques, c'est-à-dire un fichier d'ADN contenant des séquences composées de quatre lettres, "a", "t", "g" et "c". Contrairement aux mots d'une langue, les séquences ADN ont la particularité d'être de grande taille. Le nombre moyen de caractères par séquence de ce dictionnaire est de 784 caractères, alors que celui du DELAF français est de 10.

Catégorie 3	
	Séquences ADN
Nombre de mots	3 500
Nombre de caractères	2 743 488
Taille alphabet	4

TAB. 4.5 – Dictionnaire électronique de la catégorie 3

Je remercie le professeur Franz Guenthner, le professeur Éric Laporte, le professeur Jacques Savoy et le professeur Duško Vitas pour la libre utilisation de ces différents dictionnaires. Je remercie également Pierre-Emmanuel gros qui m'a fournit une liste de gènes.

4.2.2 Les automates associés

La table 4.6 présente les caractéristiques globales des automates examinés, c'est-à-dire les caractéristiques qui peuvent être extraites aisément de la représentation de l'automate (cf. table 4.1, page 91) : la taille de l'alphabet, le nombre de transitions, le nombre d'états, le nombre d'états d'un type donné (divergent et/ou convergent). À titre d'exemple, l'automate qui modélise le DELAF français contient 75 lettres (une de plus que le dictionnaire, car on a ajouté le caractère de fin de mot "#"), 177 465 transitions et 67 995 états dont 2 559 simultanément divergents et convergents, 32 924 divergents et non convergents, etc. On remarque que les automates représentant ces dictionnaires sont très grands et ce, malgré leur minimisation. L'efficacité de la minimisation varie selon le dictionnaire, ainsi elle

est moins efficace sur les dictionnaires qui possèdent un petit nombre de suffixes communs. On remarque aussi que ces automates sont fortement croisés[3]. À l'exception du dictionnaire représentant des séquences ADN, les états divergents possèdent en moyenne 3,75 transitions sortantes et les états convergents 11,29 transitions entrantes.

Automate	Taille alphabet	Nombre transitions	Nombre états	Div Conv	Div Non Conv	Non Div Conv	Non Div Non Conv
Catégorie 1							
DELAF Fr	71	177 465	67 995	2 559	32 924	7 275	25 237
DELAF En	74	252 664	116 848	3 388	50 790	12 775	49 895
DELAF Sr	52	193 668	61 383	3 784	34 274	5 240	18 085
DELAF De	89	335 284	142 795	6 168	67 752	14 421	54 454
Villes Fr	75	95 589	61 240	357	16 023	5 941	38 919
Poly En	31	717 112	435 940	7 295	11 0445	29 807	288 393
Catégorie 2							
Web Fr	43	298 117	101 837	3 799	60 697	8 263	29 078
Web Hu	43	270 495	113 678	3 099	57 638	11 528	41 413
Weg Bg	42	209 209	85 661	2 666	41 690	8 055	33 250
Web Pt	49	538 697	214 992	5 598	10 5944	19 734	83 716
Catégorie 3							
ADN	5	2 680 516	2 677 033	0	2 194	2 249	2 672 590

TAB. 4.6 – Statistiques générales

Ces information nous ont permis une première comparaison des automates. Dans les graphiques suivants, à chaque fois un automate est pris en référence (indiqué en rouge). Les autres automates sont "normalisés" proportionnellement au nombre de transitions de l'automate de référence. Ceci s'effectue en deux étapes : la première divise chaque statistique descriptive de l'automate par la statistique descriptive correspondante de l'automate de référence, et la seconde recentre la valeur obtenue vers l'unité par application d'un coefficient multiplicateur qui ramène le "nombre de transitions" à 1.

Exemple 4.5
Le tableau 4.7 présente la normalisation de l'automate DELAF anglais relativement à l'automate DELAF français. Dans un premier temps, les valeurs des six dernières colonnes de l'automate DELAF anglais sont divisés par les valeurs des colonnes de l'automate DELAF français leurs correspondants. Ensuite, une deuxième étape consiste à ramener à 1 le nombre de transitions de l'automate. Par exemple : La valeur 1,2 représente le produit de la division du nombre d'états de l'automate DELAF anglais par le nombre d'états de l'automate DELAF français et de la division du nombre de transition de l'automate DELAF français par le nombre de transitions de l'automate DELAF anglais, (116 848/67 995)*(177 465/252 664).

Un automate sera ainsi considéré comme similaire à l'automate de référence si la courbe qui le représente est globalement de hauteur 1. L'objectif est d'identifier des automates

[3]Chaque automate compte un nombre considérable d'états divergents et/ou convergents, et par conséquent, possède des transitions qui croisent ses chemins.

Automate	Transitions	État	Div Conv	Div Non Conv	Non Div Conv	Non Div Non Conv
Valeurs avant normalisation						
DELAF Fr	177 465	67 995	2 559	32 924	7 275	25 237
DELAF En	252 664	116 848	3 388	50 790	2 775	49 895
Valeurs après normalisation						
DELAF En	1	1,20	0,92	1,08	1,23	1,38

TAB. 4.7 – Normalisation de l'automate DELAF En relativement à l'automate DELAF Fr

se ressemblant en se basant uniquement sur leurs statistiques descriptives. La pertinence de ces similarités sera discutée ultérieurement.
La démarche est essentiellement visuelle, néanmoins voici les seuils que nous avons globalement utilisés pour comparer la courbe d'un automate A à celle de l'automate de référence :

- $I_1 = [0,75 - 1,25]$: L'automate A est "très similaire" à l'automate de référence lorsque toutes les statistiques de A sont comprises entre les deux valeurs de l'intervalle I_1.
- $I_2 = [0,5 - 1,5]$: L'automate A est "similaire" à l'automate de référence lorsque toutes les statistiques de A sont dans I_2 tel que il existe au moins une statistique de A qui n'appartient pas à I_1.
- $I_3 = [0 - 2]$: L'automate A est "peu similaire" à l'automate de référence lorsque toutes les statistiques de A sont dans I_3 tel que il existe au moins une statistique de A qui n'appartient pas à I_2.
- L'automate A est "dissimilaire" à l'automate de référence lorsqu'il existe au moins une statistique de A qui n'appartient pas à I_3.

L'automate de référence est toujours indiqué en rouge sur les graphiques :

1. **Comparaison à l'automate DELAF français :**
   Le graphique 1 de la figure 4.3 met en évidence une forte dissimilarité avec les trois automates des polylexicaux anglais, villes françaises et séquences ADN. En effet, chacune des trois courbes possède au moins une statistique extérieure à I_3.
   Pour apporter plus de clarté dans la lecture du graphique, les courbes correspondant aux automates signalés dissimilaires à l'automate de référence, sont supprimées. Le résultat est présenté par le deuxième graphique. Le graphique 2 met en évidence d'autres dissimilarités avec les automates DELAF serbe, DELAF anglais et web français. En effet, ces derniers possèdent des statistiques appartenant à l'intervalle I_3.
   Ces automates signalés peu similaires ont été éliminés pour construire le graphique 3. On observe sur ce graphique que l'automate le plus proche du DELAF français est le web bulgare (statistiques appartenant à I_1) suivie du web hongrois et dans une moindre mesure du web portugais et du DELAF allemand (statistiques appartenant à I_2) .

La même méthode a été successivement appliquée aux autres automates ; seuls les graphiques restreints aux automates les plus similaires apparaissent dans le texte, les autres ont été portés en annexe A, page 176.

2. **Comparaison à l'automate DELAF anglais :**
   L'automate DELAF anglais est très similaire des automates web hongrois et web bulgare. Il se rapproche aussi des automates DELAF allemand et web portugais, mais il est globalement dissimilaire de tous les autres.

3. **Comparaison à l'automate DELAF serbe :**
   L'automate DELAF serbe présente une similarité uniquement avec l'automate web français.

4. **Comparaison à l'automate DELAF allemand :**
   Le DELAF allemand est fortement dissimilaire des automates polylexicaux anglais, villes françaises, séquences ADN. L'automate le plus proche du DELAF allemand est le DELAF français. Les autres automates forment un groupe intermédiaire plutôt similaire.

5. **Comparaison à l'automate ville françaises :**
   Aucune similarité n'est mise en évidence.

6. **Comparaison à l'automate polylexicaux anglais :**
   Aucune similarité n'est mise en évidence là non plus.

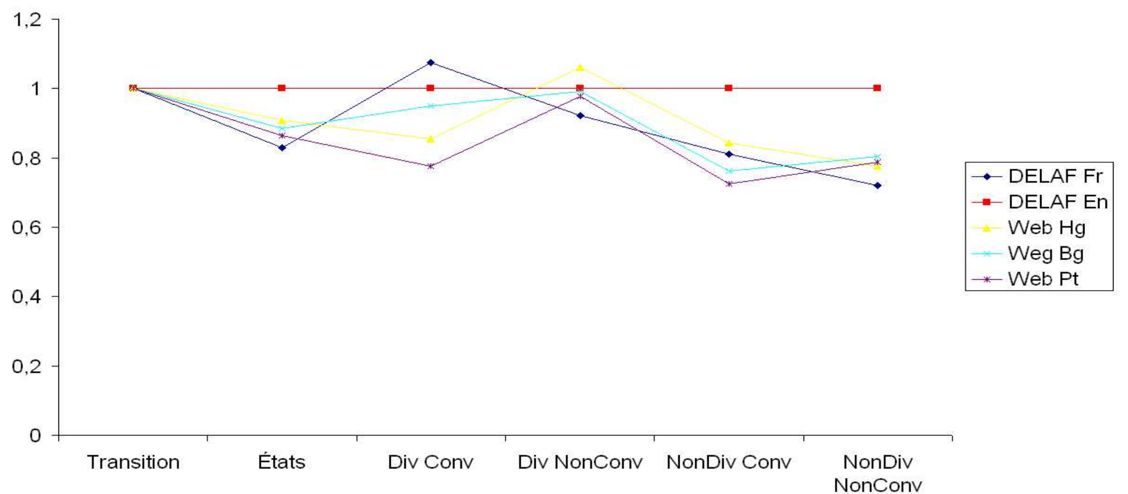
7. **Comparaison à l'automate web français :**
   Dans ce graphique, le seul automate proche de l'automate web français est le DELAF français.

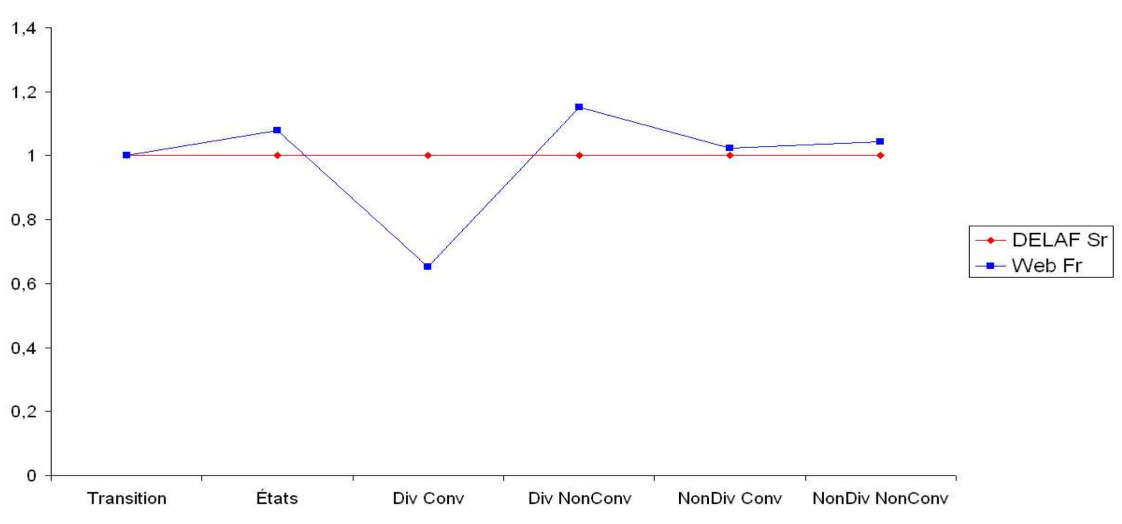
8. **Comparaison à l'automate web hongrois :**
   L'automate web hongrois est proche des automates web bulgare et web portugais et dans une moindre mesure du DELAF français, DELAF anglais et web français.

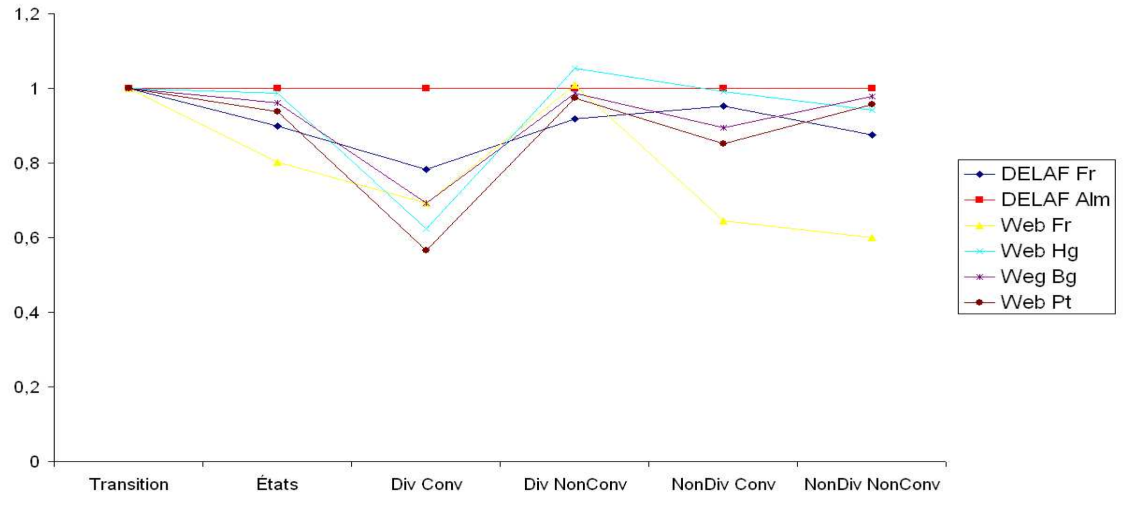
9. **Comparaison à l'automate web bulgare :**
   L'automate web bulgare est très proche des automates web hongrois, web portugais et DELAF français.

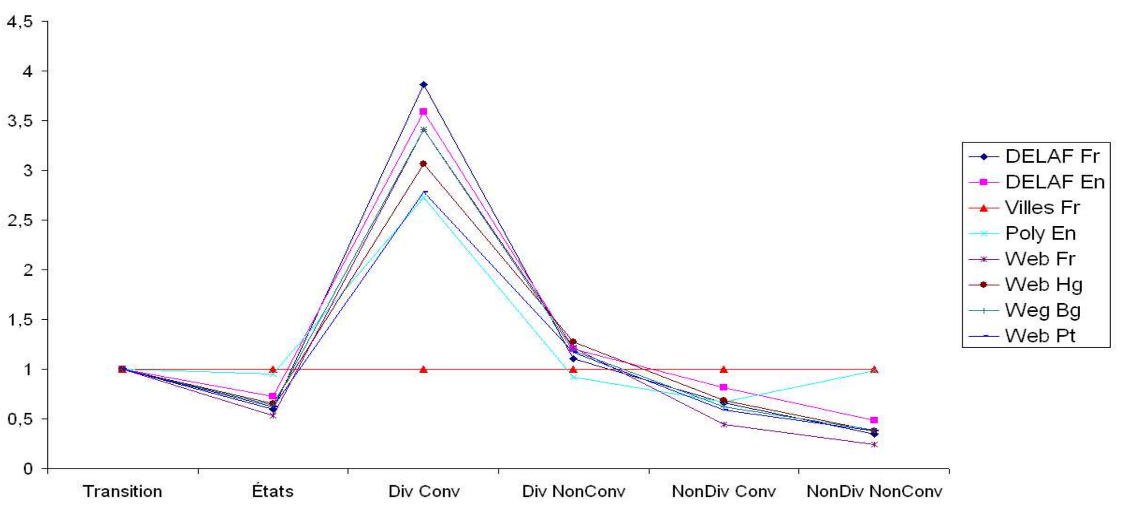
10. **Comparaison à l'automate web portugais :**
    L'automate web portugais est très proche des automates web hongrois et web bulgare.

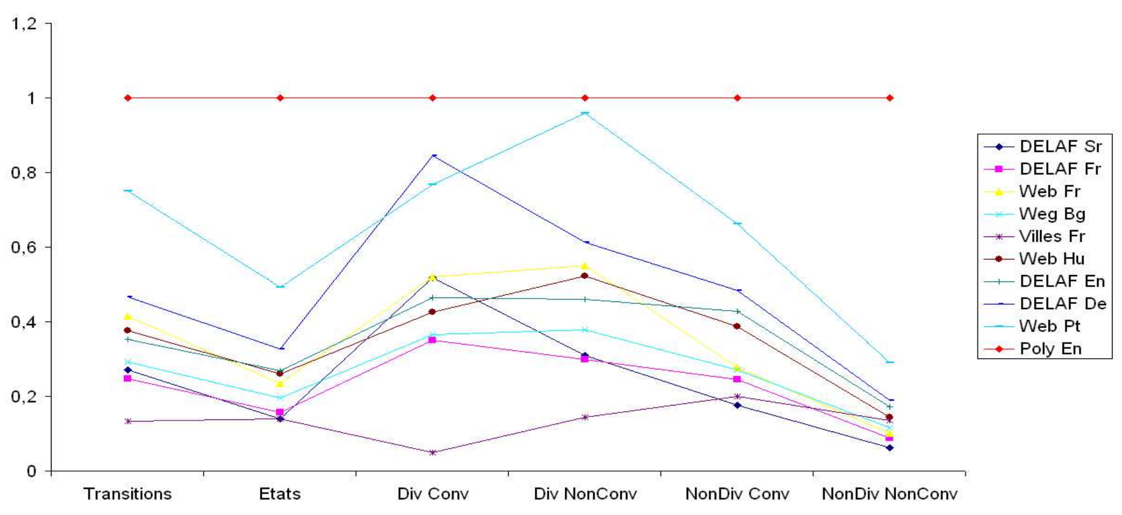
11. **Comparaison à l'automate des séquences ADN :**
    L'automate représentant les séquences ADN ne possède pas d'états divergents et convergents à la fois, cette statistique a du être mise à l'écart dans la comparaison. Aucune similarité n'est mise en évidence. Cela s'explique par sa forte proportion d'états non divergents et non convergents. C'est un automate qui est essentiellement composé de séries pures.

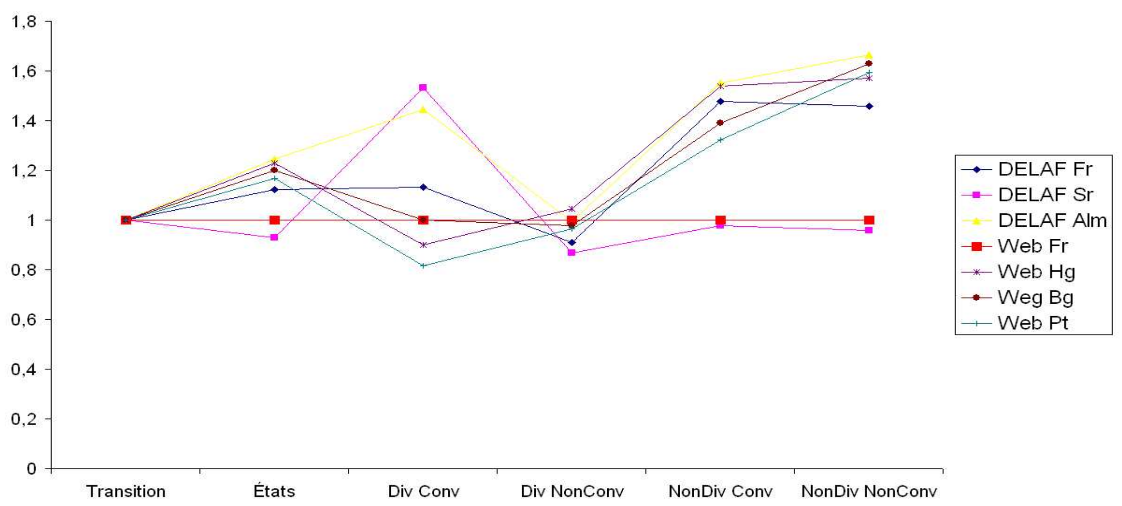

Comparaison de la structure des automates
par rapport à l'automate DELAF En
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
Web Hg
Weg Bg
Web Pt

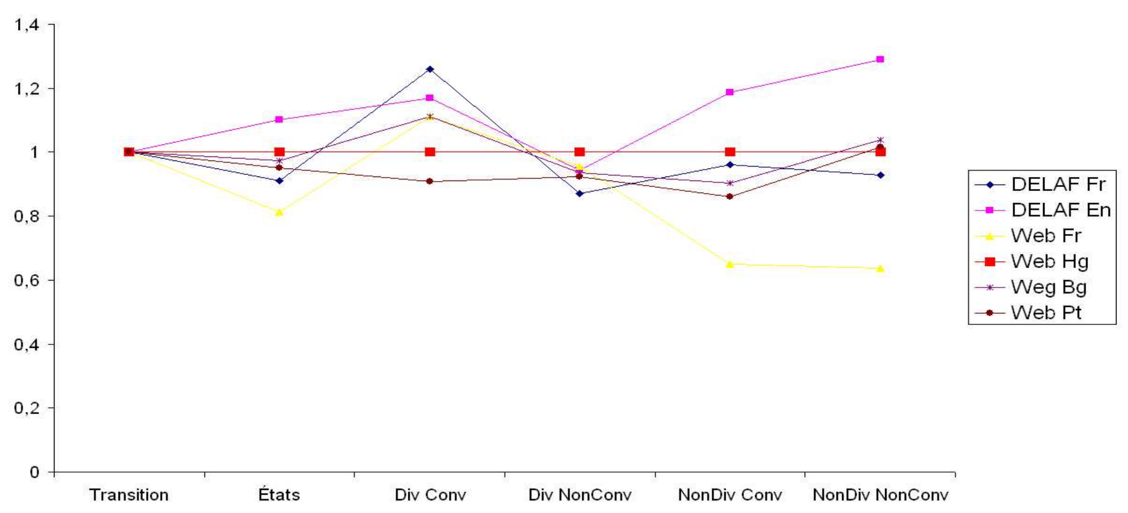

Comparaison de la structure des automates
par rapport à l'automate DELAF Sr
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Sr
Web Fr

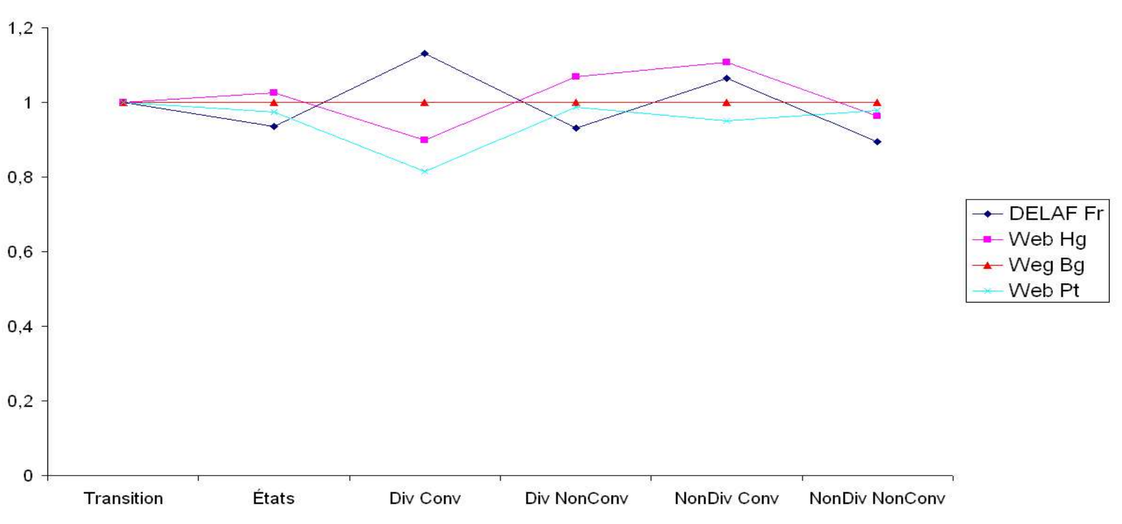

Comparaison de la structure des automates
par rapport à l'automate DELAF De
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF Alm
Web Fr
Web Hg
Weg Bg
Web Pt

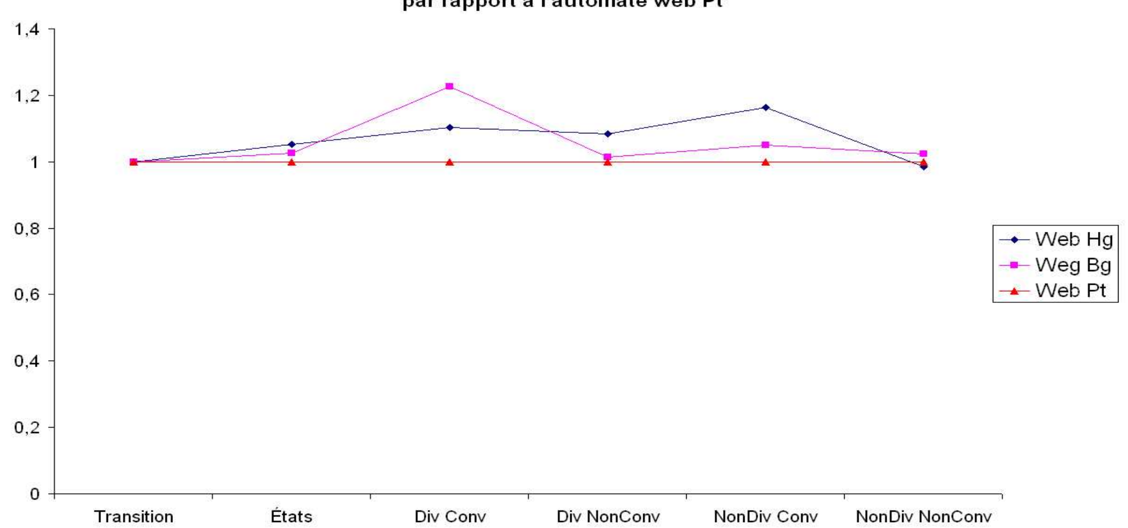

Comparaison de la structure des automates
par rapport à l'automate des villes françaises
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

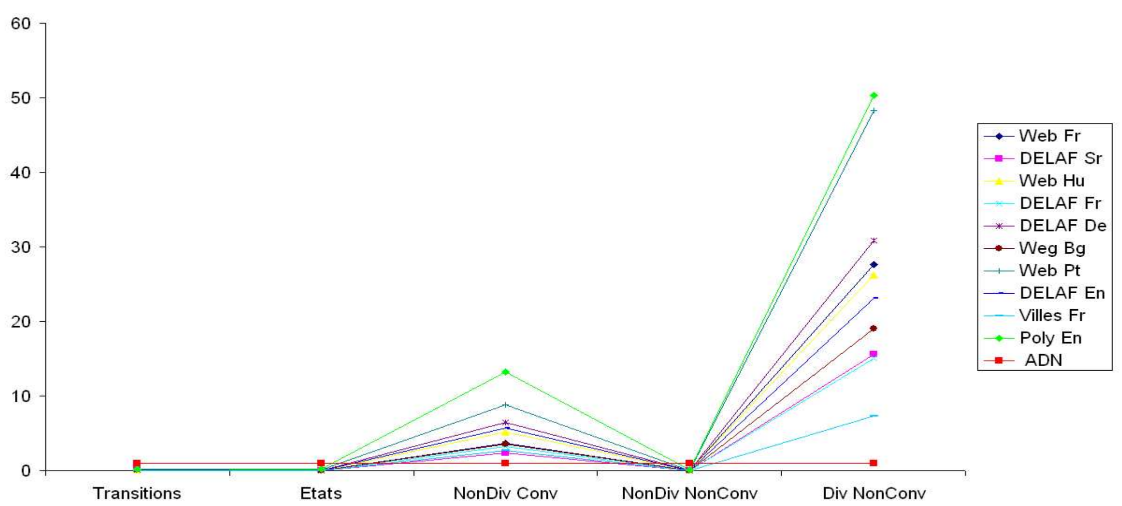

Comparaison de la structure des automates par rapport
à l'automate des polylexicaux anglais
1,2
1
0,8
0,6
0,4
0,2
0
Transitions
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Sr
DELAF Fr
Web Fr
Weg Bg
Villes Fr
Web Hu
DELAF En
DELAF De
Web Pt
Poly En


Comparaison de la structure des automates
par rapport à l'automate web Fr
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF Sr
DELAF Alm
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Hg
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Bg
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Pt
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates par rapport
à l'automate ADN
60
50
40
30
20
10
0
Transitions
Etats
NonDiv Conv
NonDiv NonConv
Div NonConv
Web Fr
DELAF Sr
Web Hu
DELAF Fr
DELAF De
Weg Bg
Web Pt
DELAF En
Villes Fr
Poly En
ADN

Une synthèse de ces observations est proposée au tableau 4.8, les couples de dictionnaires ont été qualifiés de "très similaires" (cases noires) à "dissimilaires" (cases blanches).

	DELAF				Villes	Poly	Web				
	Fr	En	Sr	De	Fr	En	Fr	Hu	Bg	Pt	ADN
DELAF Fr											
DELAF En											
DELAF Sr											
DELAF De											
Villes Fr											
Poly En											
Web Fr											
Web Hu											
Web Bg											
Web Pt											
ADN											

Très similaire — Similaire

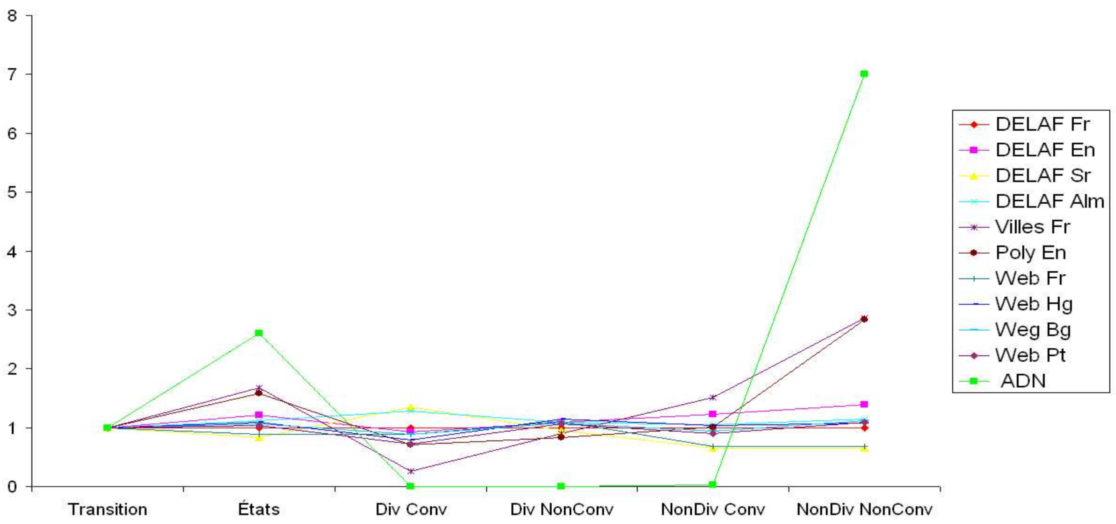
Peu similaire — Dissimilaire

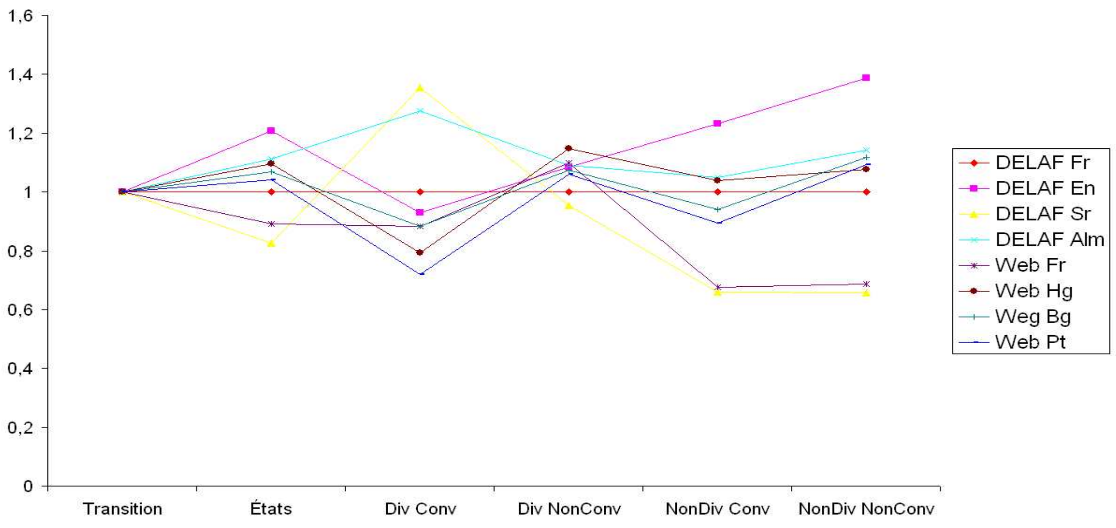
TAB. 4.8 – Étude de la similarité des automates

Cette étude s'appuie sur des observations graphiques. De plus, elle a tendance à privilégier l'automate choisi comme automate de référence, ce qui induit une dissymétrie des résultats. En fait, elle revient à chercher des relations linéaires entre les statistiques de l'automate de référence et les autres automates. Pour poursuivre dans cette voie en palliant ces problèmes, nous avons décidé d'utiliser des régressions linéaires.
Pour chaque paire d'automates une régression linéaire a été effectuée sur les cinq statistiques suivantes :

1. Nombre de transitions.
2. Nombre d'états divergents et convergents.
3. Nombre d'états divergents et non convergents.
4. Nombre d'états non divergents et convergents.
5. Nombre d'états non divergents non convergents.

Le tableau 4.9 présente le calcul des coefficients de corrélation de la régression linéaire.





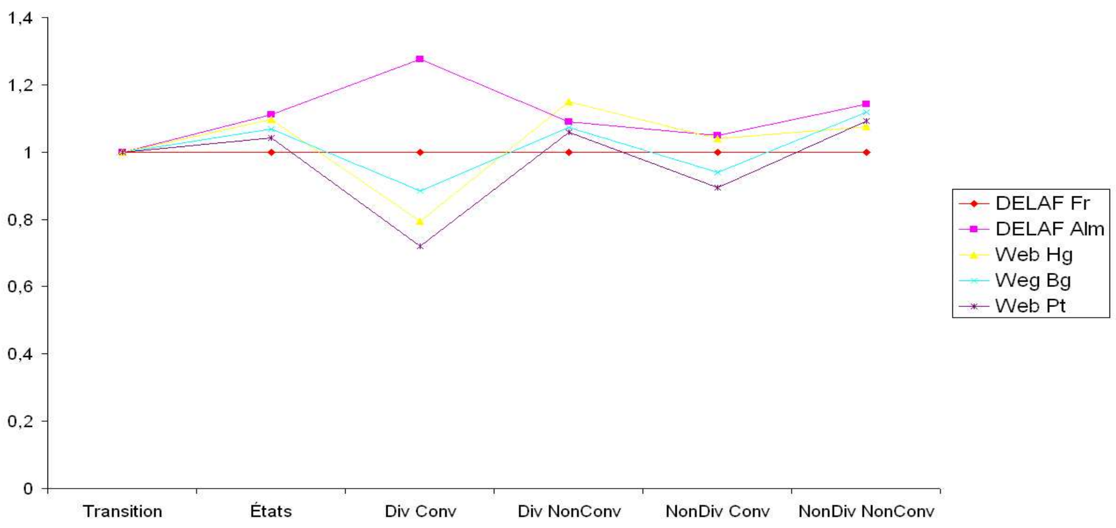


FIG. 4.3 – Comparaison à l'automate DELAF français

Dico	DELAF Fr	DELAF En	DELAF Sr	DELAF De	Villes Fr	Poly En	Web Fr	Web Hg	Weg Bg	Web Pt	ADN
DELAF Fr	1	0,99851	0,99879	0,99978	0,95670	0,95649	0,99851	0,99955	0,99975	0,99979	0,65639
DELAF En	0,99851	1	0,99467	0,99923	0,97036	0,96990	0,99475	0,99867	0,99925	0,99920	0,69259
DELAF Sr	0,99879	0,99467	1	0,99785	0,94191	0,94208	0,99953	0,99796	0,99778	0,99788	0,62227
DELAF De	0,99978	0,99923	0,99785	1	0,96034	0,96008	0,99798	0,99982	0,99999	0,99999	0,66533
Villes Fr	0,95670	0,97036	0,94191	0,96034	1	0,99969	0,94068	0,95680	0,96047	0,96008	0,84519
Poly En	0,95649	0,96990	0,94208	0,96008	0,99969	1	0,94046	0,95628	0,96017	0,95978	0,84738
Web Fr	0,99851	0,99475	0,99953	0,99798	0,94068	0,94046	1	0,99857	0,99795	0,99803	0,61708
Web Hg	0,99955	0,99867	0,99796	0,99982	0,95680	0,95628	0,99857	1	0,99984	0,99984	0,65517
Weg Bg	0,99975	0,99925	0,99778	0,99999	0,96047	0,96017	0,99795	0,99984	1	0,99999	0,66554
Web Pt	0,99979	0,99920	0,99788	0,99999	0,96008	0,95978	0,99803	0,99984	0,99999	1	0,66451
ADN	0,65639	0,69259	0,62227	0,66533	0,84519	0,84738	0,61708	0,65517	0,66554	0,66451	1

Tab. 4.9 – Calcul des coefficients de corrélation

Les coefficients de corrélation peuvent être considérés comme des mesures de la similarité, pour définir des automates similaires entre eux nous nous sommes appuyés sur la courbe 4.4 qui représente les coefficients de corrélation par ordre décroissant, en définissant des seuils aux endroits de brusque décroissance (calcul de la pente).

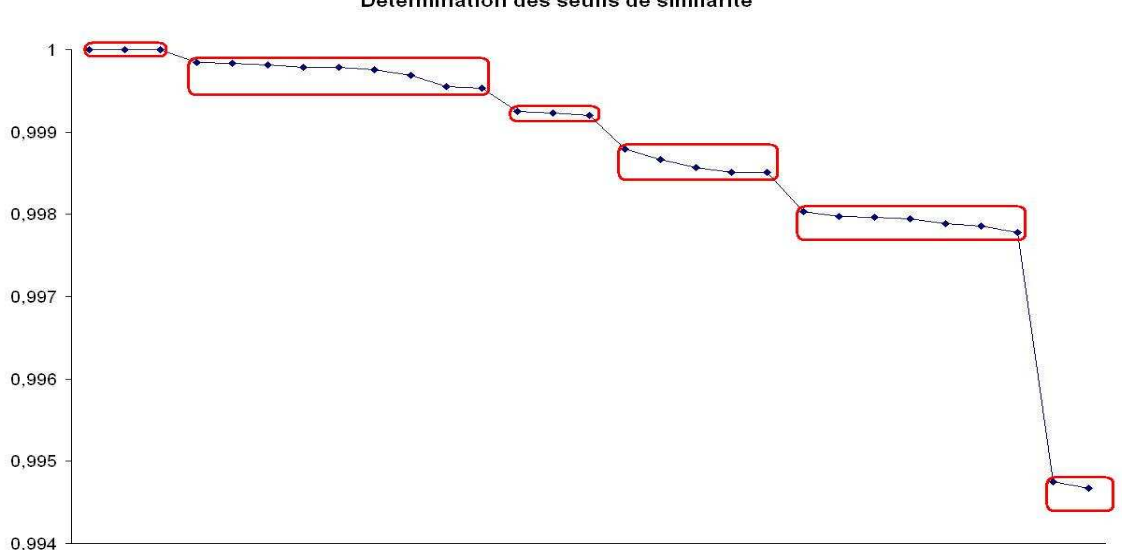


FIG. 4.4 – Corrélation

Le tableau 4.10 résume les quatre premiers groupes obtenus ainsi. Les cases les plus foncées déterminent les automates les plus similaires. Ce premier groupe est constitué des trois automates DELAF allemand, web hongrois et web portuguais. Notons à nouveau qu'on a représenté les quatre premiers groupes et qu'il en existe plusieurs autres qui sont de moins en moins similaires.

	DELAF				Villes	Poly	Web				
	Fr	En	Sr	De	Fr	En	Fr	Hu	Bg	Pt	ADN
DELAF Fr											
DELAF En											
DELAF Sr											
DELAF De											
Villes Fr											
Poly En											
Web Fr											
Web Hu											
Web Bg											
Web Pt											
ADN											

TAB. 4.10 – Similarité décroissante des automates du plus foncé au plus clair

Les deux approches mènent à des regroupements différents[4], mais on retrouve par exemple le groupe des automates de catégorie 2 ("les automates du web") avec le DELAF français, plus le DELAF allemand qui vient s'y joindre. Par contre, les deux automates des villes françaises et des polylexicaux anglais qui étaient considérés comme peu similaires dans la première approche sont plutôt similaires dans la deuxième approche.

4.3 Application de l'algorithme de recherche des sous-automates ***Recherche*** SA

Nous avons implémenté l'algorithme *Recherche* SA en langage C, sur un Pentium 4, 4,80 GHZ et 512 Mo de RAM. Ensuite nous l'avons appliqué aux différents automates représentant les 11 dictionnaires des catégories 1, 2 et 3 (voir table 4.6, page 97).
Des résultats généraux sont d'abord présentés qui mettent en évidence l'absence de $PPSA$ (qui ne contiennent ni série, ni parallèle). Les parties suivantes développent une analyse des sous-automates séries et des sous-automates parallèles présents.

4.3.1 Expérimentations numériques

La recherche des sous-automates s'effectue en plusieurs itérations de *Recherche* $SA - SP$ et une seule itération de *Recherche* $PPSA$ (cf. Algorithme 13). L'algorithme *Recherche* $SA - SP$ se déroule en itérations successives, une itération consistant à trouver tous les parallèles, les archiver et les aplatir en une transition, ensuite toutes les séries, les archiver et les remplacer par une transition. Sur chacun des automates testés, l'algorithme *Recherche* SA stoppe à la première itération, ce qui indique que l'automate ne contient aucun $PPSA$ qui ne soit déjà obtenu par *Recherche* $SA - SP$.

La table 4.11 présente les caractéristiques, en terme de transitions et d'états, de l'automate après application de l'algorithme *Recherche* $SA - SP$. Elle indique également le nombre d'itérations effectuées par *Recherche* $SA - SP$, c'est-à-dire le niveau maximal d'imbrication des sous-automates détectés par *Recherche* $SA - SP$. Un 0,5 signifie que dans sa dernière itération, *Recherche* $SA - SP$ a trouvé des parallèles mais aucune série.
À titre d'exemple, l'automate qui modélise le DELAF français est réduit en 2 itérations à 150 299 transitions (gain transitions = 15,3%) et 42 625 états (gain états = 37,3%). Le nombre moyen de transitions sortantes par état divergent est 4 et le nombre moyen de transitions entrantes par état convergent est 11,9.
La valeur élevée du nombre moyen de transitions entrantes par état convergent explique peut être l'absence des $PPSA$ qui ne contiennent ni série, ni parallèle.

Pour chaque automate, le gain en nombre de transitions et le gain en nombre d'états sont exprimés en pourcentage en annexe par la table A.1, page 174.
Les automates représentant les dictionnaires contiennent un nombre considérable de séries, de parallèles et de sous-automates, la recherche de ces structures s'effectue par plusieurs itérations de l'algorithme *Recherche* $SA - SP$. Selon le dictionnaire deux à trois itérations sont nécessaires. Ce résultat révèle que chaque dictionnaire contient des

[4]Dans le premier cas, cf. tableau 4.8, on recherche une relation linéaire et dans le second, cf. tableau 4.10, on recherche une relation affine.

sous-automate détectés dès la première itération et des sous-automates détectés à la n^{eme} itération possédant plusieurs niveaux d'imbrications. Chacune de leurs transitions est étiquetée par une lettre de l'alphabet ou une expression rationnelle. Ces sous-automates sont intéressants à analyser car ils constituent une information impossible à extraire directement du dictionnaire. On distinguera trois types de sous-automates : les séries pures (resp. parallèles purs) qui sont des séries (resp. parallèles) composées par des transitions de l'automate initial et les sous-automates qui sont des série ou des parallèles contenant au moins une réduction série ou une réduction parallèle.

La table 4.12 présente les sous-structures localisées dans chacun des dictionnaires ; ainsi l'automate DELAF français contient 14 624 séries pures dont 4 222 distinctes, 1 706 parallèles purs dont 178 distincts et 183 sous-automates dont 174 distincts.

Automate	Recherche SA-SP	Automate réduit			
	Itérations	Transitions	États	Nb moyen de transitions sortantes par état divergent	Nb moyen de transitions entrantes par état convergent
Catégorie 1					
DELAF Fr	2	150 299	42 625	4	11,9
DELAF En	2	199 332	66 635	3,5	9,2
DELAF Sr	2	163 466	43 018	4,2	14,5
DELAF De	3,5	276 223	87 772	3,56	10,18
Villes Fr	2,5	56 064	22 224	3,1	6,4
Poly En	2,5	427 269	147 423	3,4	8,5
Catégorie 2					
Web Fr	3	253 595	72 102	3,8	16,1
Web Hu	2,5	221 908	72 006	3,5	11,3
Web Bg	2,5	167 045	52 052	3,6	11,8
Web Pt	3	435 147	130 430	3,7	13,1
Catégorie 3					
ADN	4	7 913	4 435	2,6	2,5

TAB. 4.11 – Caractéristiques des automates après application de *Recherche SA-SP*

Automate	Séries pures		Parallèles purs		Sous-automates (SA)	
	Total séries	Total séries distinctes	Total parallèles	Total parallèles distincts	Total SA	Total SA distincts
Catégorie 1						
DELAF Fr	14 624	4 222	1 706	178	183	174
DELAF En	26 132	7 125	2 871	250	421	365
DELAF Sr	11 106	2 963	8 372	209	377	276
DELAF De	292 599	5 549	3 384	333	746	440
Villes Fr	12 754	6 268	448	156	119	117
Poly En	74 888	17 401	1 264	170	152	141
Catégorie 2						
Web Fr	17 110	4 726	10 235	2 048	891	819
Web Hu	23 811	5 733	5 112	1 054	356	347
Web Bg	18 069	5 146	5 792	1 056	498	484
Web Pt	38 392	11 152	13 270	2 365	1 164	1 050
Catégorie 3						
ADN	4 133	3 612	0	0	10	10

TAB. 4.12 – Structure interne des automates représentant des dictionnaires

Exemple 4.6
La figure 4.5 représente plusieurs sous-automates détectés dans les différents dictionnaires. On remarque que les informations représentées par ces sous-automates sont impossibles à extraire directement du dictionnaire.

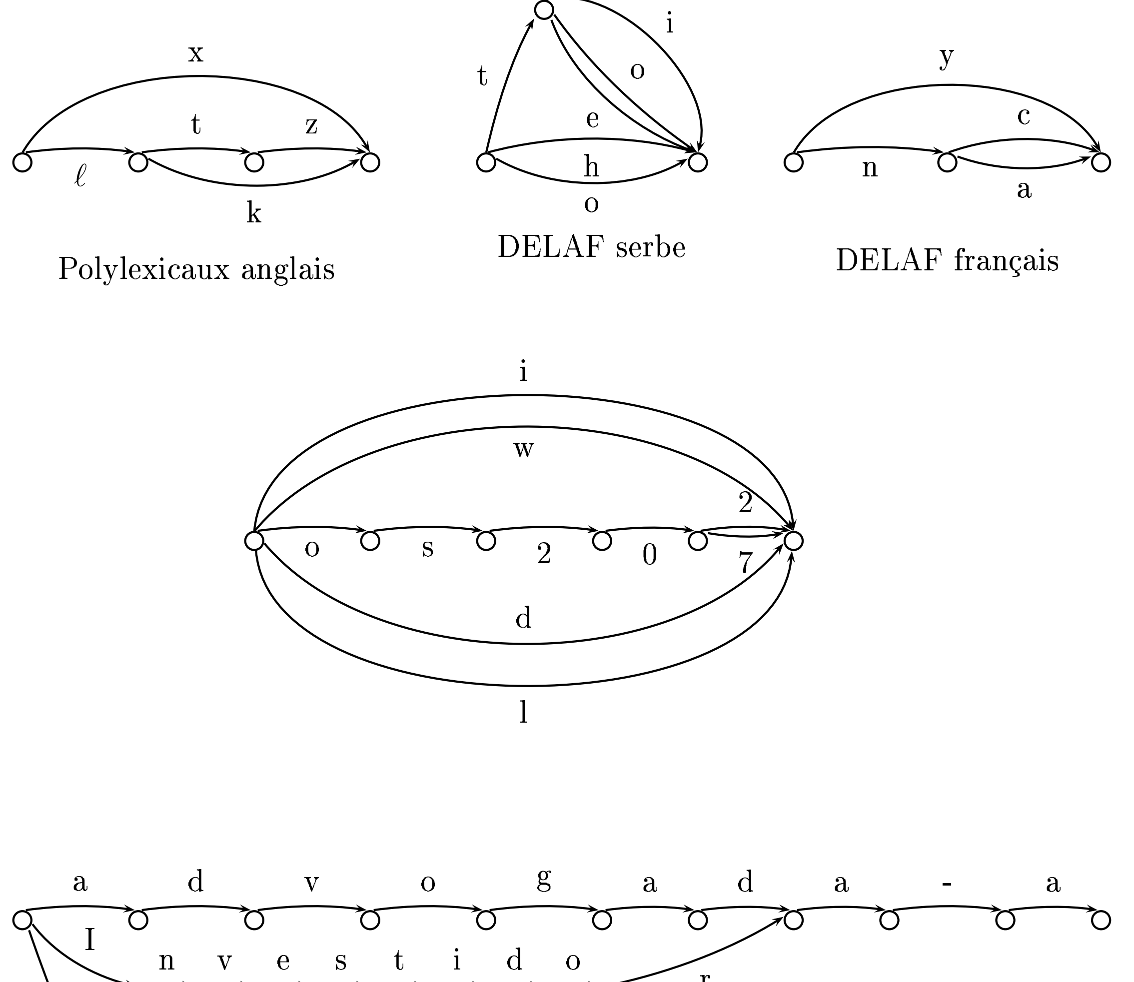


FIG. 4.5 – Exemples de sous-automate

4.3.2 Parallèles purs

La table 4.13 développe les caractéristiques des parallèles purs. Ces parallèles sont ceux détectés à la première itération de l'algorithme *Recherche* $SA - SP$. Les parallèles purs présents au sein de l'automate du DELAF français sont de largeur moyenne 2,14 et de fréquence moyenne 9,58. La largeur maximale est atteinte avec 4 transitions et la fréquence maximale est de 760. L'automate représentant les séquences ADN ne contient aucun parallèle pur.

Automate	Largeur max	Largeur moy	Fréquence Max	Fréquence moy
Catégorie 1				
DELAF Fr	4	2,14	760	9,58
DELAF En	4	2,21	632	11,48
DELAF Sr	5	2,4	1 885	40,05
DELAF De	7	2,45	284	10,61
Villes Fr	5	2,2	34	2,8
Poly En	5	2,17	172	7,43
Catégorie 2				
Web Fr	14	3,63	239	4,99
Web Hu	12	3,26	514	4,85
Web Bg	17	3,38	300	5,48
Web Pt	18	3,64	563	5,61
Catégorie 3				
ADN	Ne contient pas de parallèles purs			

TAB. 4.13 – Parallèles purs

En détaillant le contenu de ces parallèles, on découvre qu'ils sont le plus souvent composés uniquement de consonnes ou uniquement de voyelles, 60% des parallèles purs des automates suivent ce principe, les mélanges consonnes voyelles sont plus rares.
Naturellement, après une consonne dans un mot, une voyelle est généralement attendue[5] et la modification de cette seule voyelle peut permettre de former différents mots, comme dans : sec, sac, suc, sic, soc ou gros, gris, gras.

La langue serbe présente aussi le même phénomène avec la particularité que les parallèles localisés sont plus fréquents et plus larges. Cette observation est à lier au fait que tous les mots se déclinent et construisent des dérivations régulières (régularité morphologique).
En effet, le serbe est une langue hautement flexionnelle qui possède sept cas : nominatif, génitif, datif, accusatif, vocatif, instrumental et locatif. Les noms sont groupés en quatre classes de déclinaison, d'après leur désinence au nominatif singulier[6].
Le mode de construction des dictionnaires semble influencer sur les parallèles présents dans les automates correspondants. La largeur moyenne des parallèles dans les

[5]d'après Philippe Boula de Mareüil : "1,6 consonnes par syllabe en français, contre 2,1 pour l'anglais et 2,5 pour l'allemand" [de Mareüil, 2007].

[6]Consultation de http ://fr.wikipedia.org/wiki/Serbe

automates de catégorie 1 ("DELAF") est comprise entre 2 et 2,5. Dans les automates de catégorie 2 ("Web"), elle est située entre 3 et 4 et l'automate représentant les séquences ADN ne contient aucun parallèle. À part l'automate des villes françaises, la fréquence moyenne des parallèles purs est plus importante dans les automates de catégorie 1, mais moins homogène.

La table 4.14 représente la répartition des parallèles purs en fonction de leurs fréquences. À titre d'exemple, l'automate DELAF Français possède 80 parallèles purs distincts non redondants et 63 parallèles qui se répètent deux à cinq fois. On remarque que, plus la fréquence augmente, plus le nombre de parallèles purs diminue à l'exception des automates DELAF serbe et web portugais. En effet, l'automate DELAF serbe contient seulement 4 parallèles purs qui se répètent entre 101 à 1 000 fois, alors qu'il en possède 5 qui se répètent 31 à 50 fois ; mais il possède aussi 6 parallèles purs qui se répètent entre 101 à 1 000 fois. L'automate web portugais contient uniquement 99 parallèles purs de fréquence entre 2 et 5 alors qu'il en possède 101 qui se répètent entre 6 à 10 fois.
Les automates testés ont rarement des parallèles purs de fréquence supérieure à 30. En moyenne, seulement 4% des parallèles purs possède une fréquence supérieure à 30. Le seul qui se distingue est l'automate DELAF serbe avec une moyenne de 9%. De plus, cet automate possède la plus haute fréquence d'un parallèle pur, 1 885. Ce phénomène peut être expliqué par la régularité morphologique de la langue serbe.

La figure 4.6 présente respectivement la répartition des parallèles purs en fonction des fréquences et la répartition des parallèles purs en fonction des largeurs. Les données ayant servi au tracé des graphiques sont présentées en annexe A, page 174. Lorsque l'on observe les parallèles en fonction de leur largeur on peut distinguer trois groupes. Le premier groupe comprend les automates du DELAF anglais, villes françaises et polylexicaux anglais. Le deuxième groupe est composé des automates DELAF allemand et DELAF serbe où les largeurs sont un peu plus étalées et le troisième groupe est constitué par les automates du web où la largeur la plus représentée et 3. Ceci est représenté par une courbe croissante dans sa première partie, puis décroissante pour des valeurs supérieures à 3.
L'observation des parallèles purs en fonction des fréquences dénote un étalement des proportions de parallèles pour l'ensemble des automates de catégorie 1 et la non redondance des données des automates de la deuxième catégorie. Une exception est l'automate des polylexicaux anglais qui reprend le chemin de la hausse sur le graphique pour une fréquence de 8.

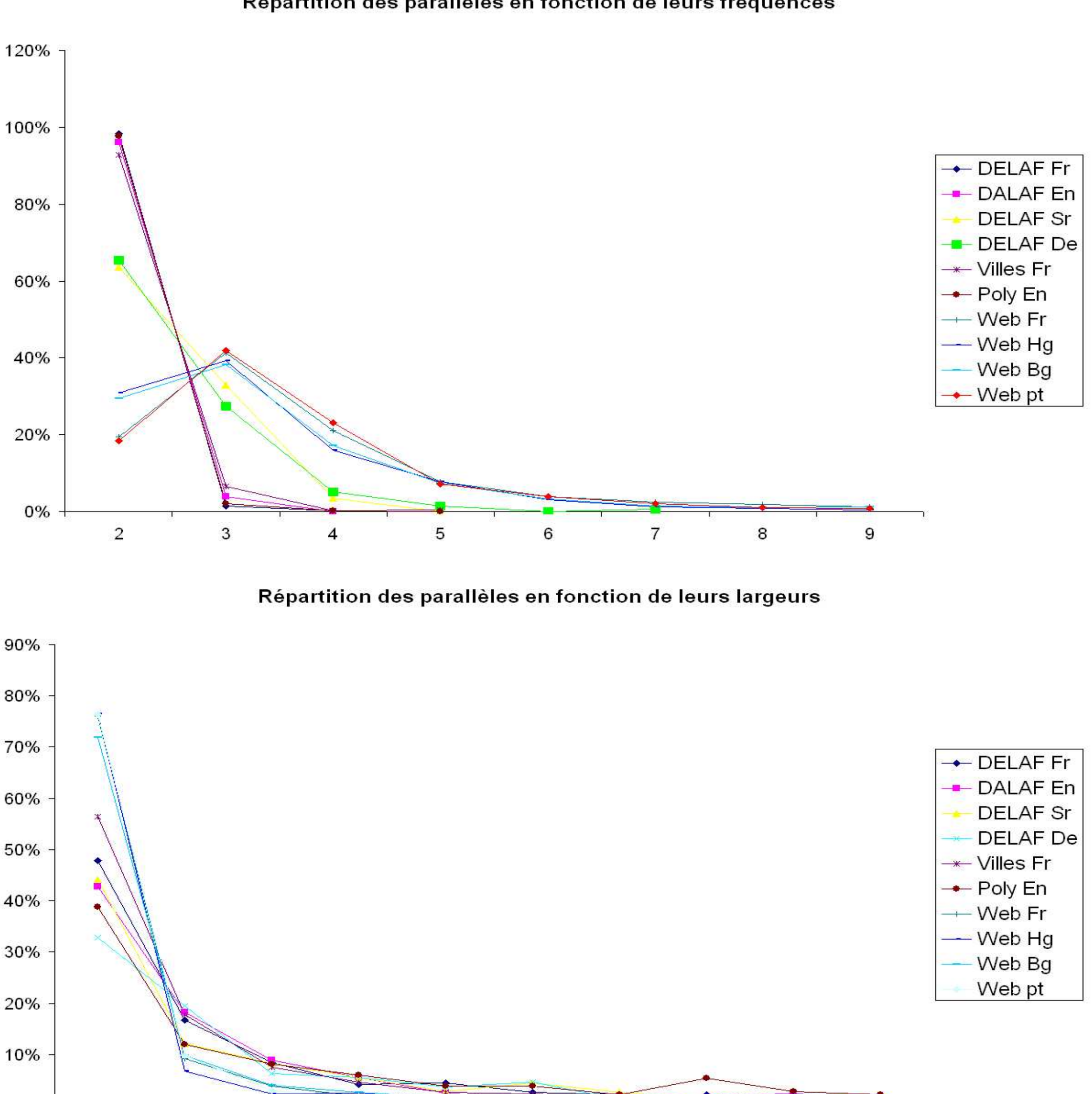


FIG. 4.6 – Répartition des parallèles purs en fonction des fréquences et des largeurs

Automate	Nombre de répétions Nombre de parallèles purs								
Catégorie 1									
DELAF Fr	760 1	138 1	100-51 2	50-31 1	30-21 5	20-11 7	10-6 18	5-2 63	1 80
DELAF Anglais	632-613 2	263 1	100-51 4	50-31 2	30-21 5	20-11 26	10-6 19	5-2 90	1 101
DELAF Sr	1885-1001 3	1000-101 6	100-51 4	50-31 5	30-21 6	20-11 13	10-6 24	5-2 66	1 82
DELAF De	284 1	143-101 4	100-51 3	50-31 15	30-21 16	20-11 45	10-6 42	5-2 118	1 89
Villes Fr	0	0	100-51 0	50-31 0	30-21 1	20-11 6	10-6 12	5-2 55	1 82
Poly En	172 1	118 1	100-51 0	50-31 3	30-21 7	20-11 16	10-6 30	5-2 53	1 49
Catégorie 2									
Web Fr	239 1	173-101 14	100-51 24	50-31 30	30-21 34	20-11 83	10-6 92	5-2 414	1 1356
Web Hu	514 1	271-101 3	100-51 5	50-31 15	30-21 16	20-11 65	10-6 76	5-2 171	1 702
Web Bg	300 1	265-101 6	100-51 14	50-31 17	30-21 16	20-11 46	10-6 60	5-2 241	1 655
Web PT	563 1	261-101 23	100-51 26	50-31 33	30-21 44	20-11 101	10-6 99	5-2 456	1 1582
Catégorie 3									
ADN	Ne contient pas de parallèles purs								

TAB. 4.14 – Répartition des parallèles purs

4.3.3 Séries pures

Dans cette partie, il est nécessaire de préciser que nous nous intéressons aux *séries pures extraites directement d'un automate représentant un dictionnaire* et non aux séries extraites des mots du dictionnaire. De ce fait, ces séries ne correspondent pas forcément aux informations contenues dans les mots du dictionnaire, de plus, elles sont biaisées par la minimisation qui agit sur la nature des informations recueillies.

Exemple 4.7
Dans le dictionnaire des noms des villes françaises, en respectant la casse, la séquence "sur" apparaît 2 163 fois : 2 149 fois encadrée par deux tiret dans chaque nom où elle apparaît "-sur-" et 14 fois totalement incluse dans un nom (par exemple : Lassur, Troussures, etc).
Dans l'automate, les informations recueillies sont différentes à cause de la déterminisation et de la minimisation. Aussi, il n'existe aucune série pure de longueur supérieure à trois qui contient la séquence "sur" sans lui associer au moins un tiret.
La séquence "sur" est localisée au sein de 260 séries pures, sous les formes suivantes :

"sur"(série pure entière), "sur-*" (seule ou uniquement en début de série, exemples : sur-, sur-Mos, sur-la-L), "*-sur-*"(en début, au milieu et en fin de série, exemples : -sur-le-Blais, e-sur-Ni, court-sur-) ,et "*-sur" (seule ou en fin de série uniquement, exemples : -sur, la-Roche-sur).
Cette diversité de formes s'explique par la déterminisation de l'automate ; par exemple pour les noms "Bagnols-les-Bains" et "Bagnols-sur-Cèze", la divergence commence à "sur-" . . . .

4.3.3.1 Caractéristiques des séries pures

Automate	Longueur max	Longueur moy	Fréquence max	Fréquence moy
Catégorie 1				
DELAF Fr	18	3,82	256	3,46
DELAF En	27	4,31	325	3,66
DELAF Sr	14	3,69	288	3,74
DELAF De	69	4,45	1155	5,27
Villes Fr	33	5,51	165	2,03
Poly En	41	8,45	1 798	4,3
Catégorie 2				
Web Fr	30	3,97	384	3,62
Web Hu	22	4,19	696	4,15
Web Bg	24	4,2	663	3,5
Web Pt	68	5,28	533	3,44
Catégorie 3				
ADN	1 898	740,74	57	1,14

TAB. 4.15 – Séries pures

La table 4.15 illustre les caractéristiques obtenues sur les séries pures présentes au sein des automates de dictionnaire. Les séries pures de l'automate DELAF français sont de longueur moyenne 3,82 et de fréquence moyenne 3,46. La longueur maximum est atteinte avec 18 transitions successives et la fréquence maximum d'une série pure est de 256.
Une étude plus approfondie de la répartition de ces séries en fonction de leurs fréquences et de leurs longueurs a été menée sur chacun des automates testés. Le graphique de la figure 4.7 illustre le pourcentage des séries pures en fonction des quinze premières longueurs dans chacun des automates.
On observe que, pour chaque automate, la longueur des séries les plus fréquentes est 2. Toutefois, la pente des courbes est moins élevée pour les automates des séquences ADN, des polylexicaux anglais et des villes françaises. Effectivement, lorsqu'on observe la répartition des fréquences des séries en fonction de leur longueur, on remarque que les séries les plus fréquentes sont de longueur 2 pour la majorité des automates (environ 60%) sauf pour l'automate des villes françaises et l'automate des polylexicaux anglais ; ces deux derniers en ont moitié moins sur cette longueur et une distribution plus étalée sur les autres longueurs. Les autres automates ont des répartitions qui deviennent négligeables après la longueur 7, le stade des 1% est atteint pour une longueur 12.

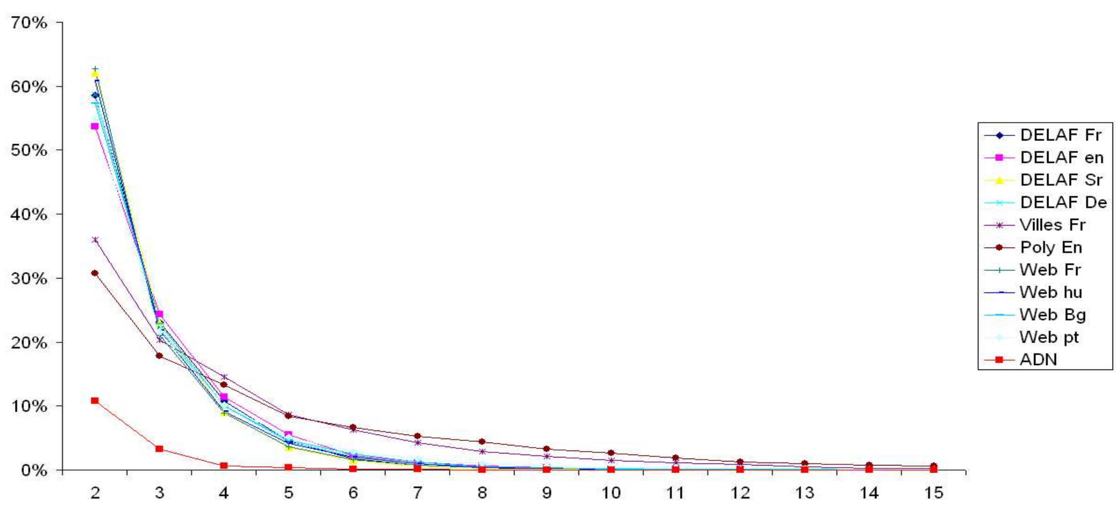
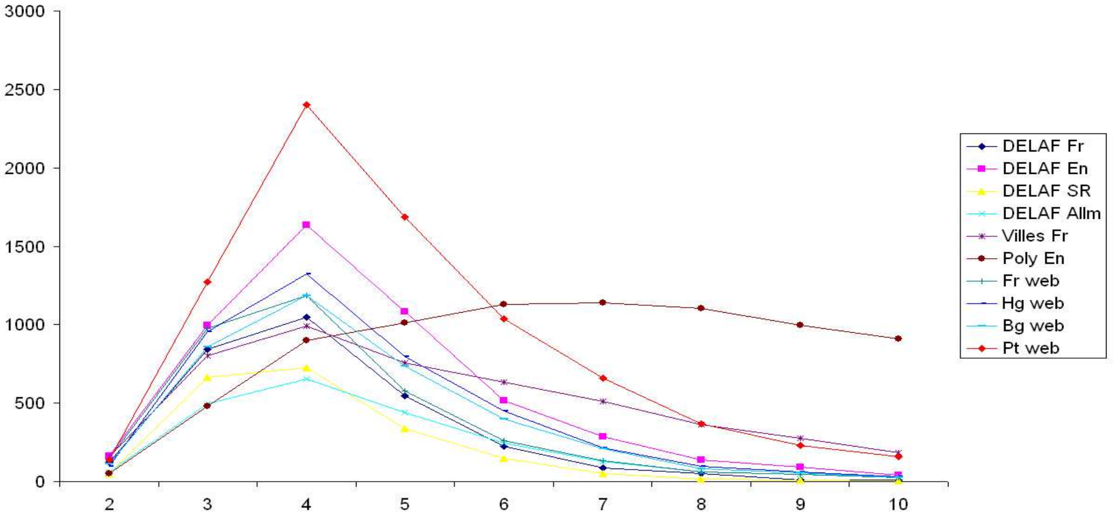
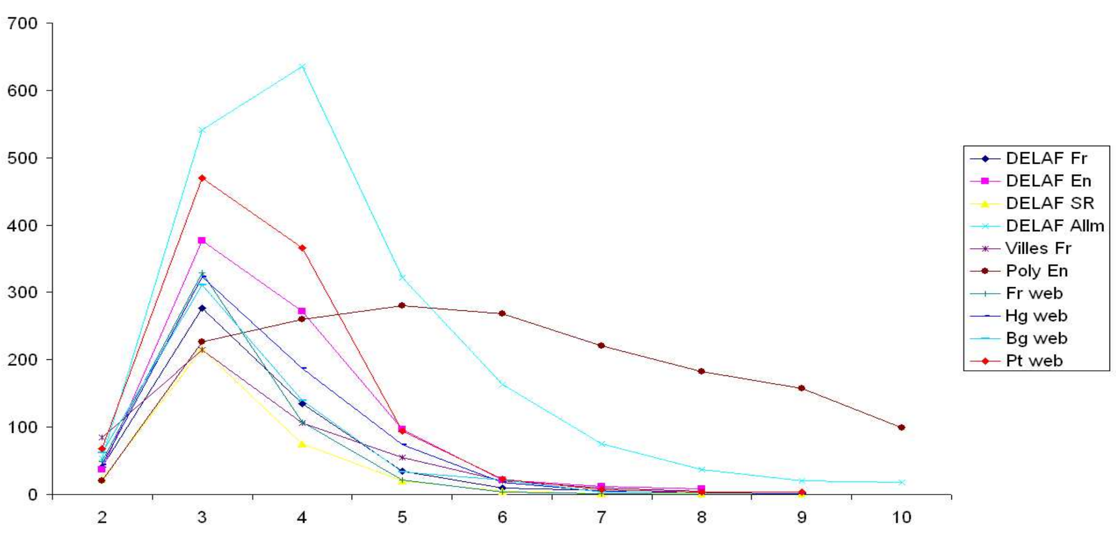


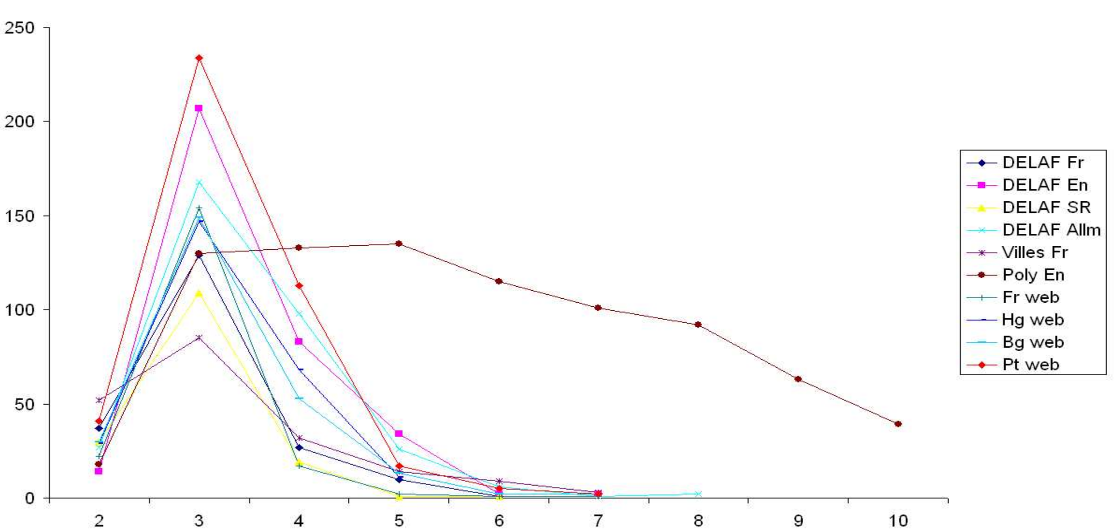
FIG. 4.7 – Répartition des séries pures en fonction de leurs longueurs

Ce résultat confirme l'étude menée en section 4.2.2 qui annonce une structure similaire de leurs automates.

Les graphiques de la figure 4.8 illustrent la répartition des séries pures en fonction de leurs fréquences d'apparition dans chaque automate testé. On remarque que, plus la fréquence augmente, plus la distribution des longueurs se recentre autour d'une valeur dont la proportion gagne en importance. On remarque également qu'un automate se distingue des autres par l'allongement de sa distribution. En effet, nous pouvons observer que, quelle que soit la fréquence, la courbe représentant l'automate des polylexicaux anglais est beaucoup plus aplatie que les autres. Enfin, nous pouvons noter que l'ordre des courbes reste à peu près toujours le même, à l'exception de la fréquence 2 où la courbe du DELAF allemand a le sommet le plus élevé. De plus, sur ce graphique de fréquence 2, le sommet de l'automate DELAF allemand se trouve à la longueur 4 contre la longueur 3 pour les autres automates. Ceci s'explique par le fait que la proportion de séries pures de fréquence 2 est plus importante dans cet automate que dans les autres. Une étude comparative des différents automates sur chaque fréquence indique des comportements identiques des courbes des "automates du web", ce qui confirme aussi les résultats présentés en section 4.2.2 sur la comparaison des structures des automates.


Répartition des séries pures de fréquence 1
3000
2500
2000
1500
1000
500
0
2
3
4
5
6
7
8
9
10
DELAF Fr
DELAF En
DELAF SR
DELAF Allm
Villes Fr
Poly En
Fr web
Hg web
Bg web
Pt web


Répartition des séries pures de fréquence 2
700
600
500
400
300
200
100
0
2
3
4
5
6
7
8
9
10
DELAF Fr
DELAF En
DELAF SR
DELAF Allm
Villes Fr
Poly En
Fr web
Hg web
Bg web
Pt web


Répartition des séries pures de fréquence 3
250
200
150
100
50
0
2
3
4
5
6
7
8
9
10
DELAF Fr
DELAF En
DELAF SR
DELAF Allm
Villes Fr
Poly En
Fr web
Hg web
Bg web
Pt web

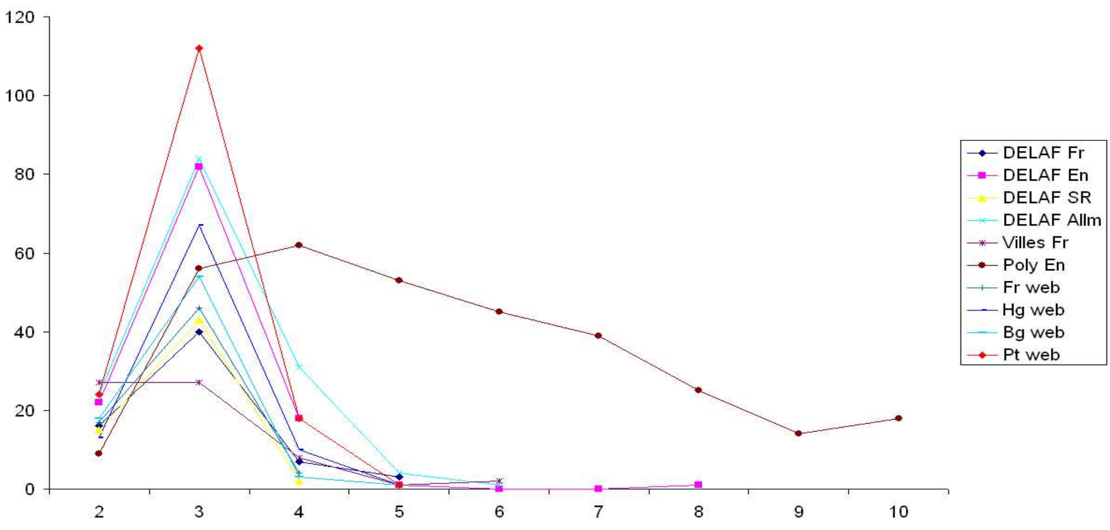
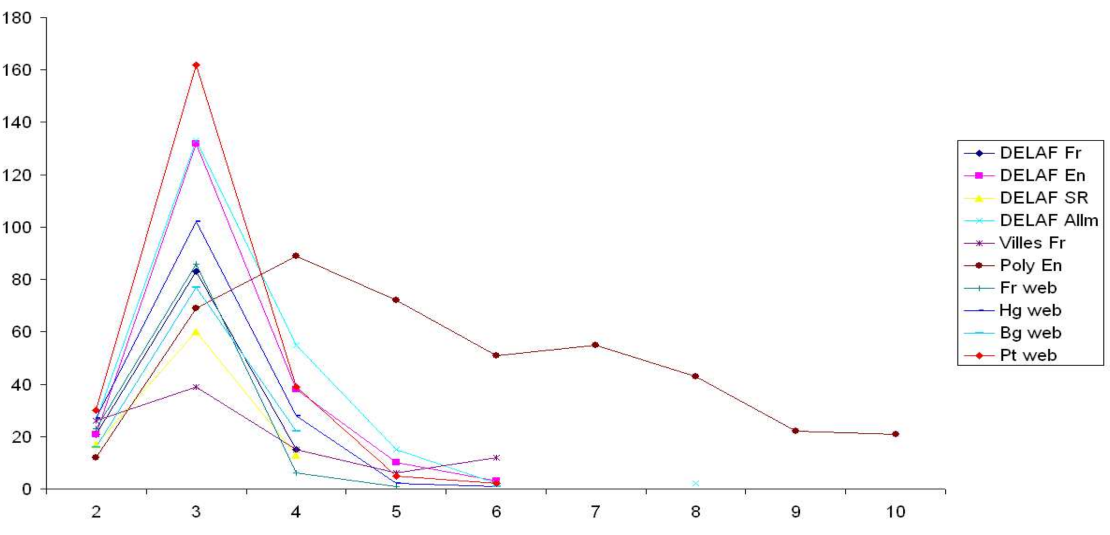


FIG. 4.8 – Répartition des séries pures en fonction de leurs fréquences



Les tables 4.16 et 4.17 représentent la répartition des séries en fonction de leurs fréquences et de leurs longueurs. Ainsi, on peut lire dans la table 4.16 que l'automate DELAF français contient 2 953 séries pures non redondantes et 892 séries pures qui se répètent 2 à 5 fois. Même si la fréquence augmente, le nombre de séries ne diminue pas forcément. Les automates DELAF français, DELAF serbe et polylexicaux anglais sont les seuls à avoir un nombre de séries continuellement décroissant (3 automates /11). On observe que quelle que soit la longueur des séries, la fréquence la plus représentée est de valeur 1 sauf pour le DELAF allemand pour lequel on observe que la fréquence 2 est plus représentée pour la longueur 3 et la différence entre les fréquences 1 et 2 est moins importante que pour les autres dictionnaires quelle que soit la longueur des séries.
De la table 4.17 on peut lire que l'automate DELAF français possède 490 séries pures distinctes de longueur 2 qui se répètent au total 8 568 fois. On remarque qu'un

dictionnaire se distingue des autres par la répartition homogène de ses séries pures. En effet, nous pouvons observer que sur plusieurs longueurs, le nombre de séries pures de l'automate des polylexicaux anglais varie très faiblement.

Automate	Nombre de répétions								
	Nombre de séries pures								
Catégorie 1									
DELAF	256	170-101	100-51	50-31	30-21	20-11	10-6	5-2	1
Fr	1	13	26	41	47	105	144	892	2 953
DELAF	125	318-101	100-51	50-31	30-21	20-11	10-6	5-2	1
En	1	32	50	64	52	160	234	1 494	5 038
DELAF	288	271-101	100-51	50-31	30-21	20-11	10-6	5-2	1
Sr	1	8	20	30	45	64	136	652	2 007
DELAF	1 155	567-101	100-51	50-31	30-21	20-11	10-6	5-2	1
De	1	44	45	62	59	174	278	2 610	2 276
Villes	165	111-101	100-51	50-31	30-21	20-11	10-6	5-2	1
Fr	1	2	17	28	26	63	157	853	5 121
Poly	1 798	1 684-101	100-51	50-31	30-21	20-11	10-6	5-2	1
En	1	77	110	143	153	506	779	3 636	11 996
Catégorie 2									
Web Fr	384	336-101	100-51	50-31	30-21	20-11	10-6	5-2	1
	1	21	26	63	42	89	128	894	3 462
Web Hg	696	479-101	100-51	50-31	30-21	20-11	10-6	5-2	1
	1	29	46	42	53	123	188	1 160	4 091
Web Bg	663	328-101	100-51	50-31	30-21	20-11	10-6	5-2	1
	1	22	29	51	48	86	169	1 014	3 726
Web Pt	533	509-101	100-51	50-31	30-21	20-11	10-6	5-2	1
	1	54	73	70	60	163	285	1 839	8 607
Catégorie 3									
ADN			57	50-31	30-21	20-11	10-6	5-2	1
			1	5	5	6	3	40	3 552

TAB. 4.16 – Répartition des séries pures en fonction de leurs fréquences

Automate	Longueur de la série							
	Nombre de séries pures							
Catégorie 1								
DELAF Fr	18	14	11-8	7	6	5	4	3	2
Nb séries	1	1	74	101	250	667	1 583	3 379	8568
Nb Distinctes	1	1	74	95	238	595	1 242	1 487	490
DELAF En	27	25-12	11-8	7	6	5	4	3	2
Nb séries	1	67	317	315	584	1 457	2 987	6 370	14 034
Nb Distinctes	1	67	305	299	543	1 226	2 070	2 055	559
DELAF Sr		14	11-8	7	6	5	4	3	2
Nb séries		1	32	54	161	395	1 006	2 558	6 899
Nb Distinctes		1	30	53	154	362	836	1 177	350
DELAF De	69	68-12	11-8	7	6	5	4	3	2
Nb séries	1	153	340	288	620	1286	2 944	6 460	17 167
NbDistinctes	1	124	245	204	421	809	1 509	1 712	524
Villes Fr	33	29-12	11-8	7	6	5	4	3	2
Nb séries	1	305	973	537	806	843	1 848	2 591	4 586
Nb Distinctes	1	305	968	522	680	1107	1 180	1 226	543
Poly En	41	40-12	11-8	7	6	5	4	3	2
Nb séries	1	4 275	9 050	3 988	4 934	6 263	9 950	13 369	23 058
Nb Distinctes	1	3 685	4 987	1 682	1 784	1 784	1 747	1 355	376
Catégorie 2								
Web Fr	30	17-12	11-8	7	6	5	4	3	2
Nb séries	1	31	155	136	274	631	1 529	3 616	10 737
Nb Distinctes	1	31	153	133	268	602	1 323	1 682	533
Web Hu	22	21-12	11-8	7	6	5	4	3	2
Nb séries	2	34	215	227	490	993	2 179	5 184	14 487
Nb Distinctes	2	34	212	222	469	887	1 632	1 798	477
Web Bg	24	18-12	11-8	7	6	5	4	3	2
Nb séries	1	45	187	220	450	845	1 822	4 134	10 365
Nb Distinctes	1	44	185	213	422	782	1 418	1 593	488
Web Pt	68	67-12	11-8	7	6	5	4	3	2
Nb séries	1	524	900	681	1 106	1 954	3 909	8 292	21 025
Nb Distinctes	1	524	894	670	1 067	1 802	2 961	2 589	644
Catégorie 3								
ADN	1898	1896-12	11-8	7	6	5	4	3	2
Nb séries	1	3 492	4	3	5	17	28	137	446
Nb Distinctes	1	3 492	4	3	5	14	26	51	16

TAB. 4.17 – Répartition des séries pures en fonction de leurs longueurs

4.3.3.2 Étude des fréquences

Selon la langue, un texte comporte une répartition particulière des fréquences de ses lettres, ses bigrammes (séquence de deux lettres consécutives) et ses trigrammes (séquence de trois lettres consécutives). L'analyse et l'interprétation de ces fréquences permet d'identifier le plus souvent la langue dans un texte. Cette analyse dépend aussi d'autres paramètres tels que le niveau de langue du texte, ainsi que le style d'écriture.
La fréquence des lettres dans un texte diffère de celle de la liste des mots d'un dictionnaire. À titre d'exemple, en français, la fréquence de la lettre "s" est moins importante dans un dictionnaire où les mots apparaissent au singulier, que dans un corpus où de nombreux mots sont au pluriel ; c'est le cas aussi de certaines lettres accentuées, par exemple, "ù" apparaît dans un seul mot "où" relativement fréquent dans les corpus. Cependant, nous allons voir que l'étude des fréquences d'un dictionnaire permet aussi d'identifier la langue.

Pour tenter de répondre à la question : "à partir d'un automate peut on identifier la langue du dictionnaire ?", une étude des fréquences a été menée, d'une part, sur nos dictionnaires, et, d'autre part, sur les séries pures extraites des automates représentant les dictionnaires. Ainsi, pour chaque dictionnaire, ont été calculées la fréquence des lettres, bigrammes et trigrammes sur les mots, puis, sur les séries pures présentes dans l'automate représentant ce dictionnaire. Ainsi, si on considère les 50 bigrammes (resp. trigrammes) les plus fréquents dans un dictionnaire et les 20 bigrammes (resp. trigrammes) les plus fréquents dans les séries pures de l'automate de ce dictionnaire, 80% des bigrammes (resp. trigrammes) des séries pures apparaissent au sein des 50 bigrammes (resp. trigrammes) des dictionnaires. De plus, on retrouve des différences significatives d'une langue à une autre. Par exemple, la lettre "h" est très utilisée en anglais, beaucoup moins en français. La fréquence des digrammes qui dupliquent uniquement des voyelles ou uniquement des consonnes est aussi plus présente en français, etc (cf. annexe A, page 185).
Cette étude a permis de constater que l'étude des fréquences à partir des séries pures de l'automate peut être une première étape pour déterminer la langue d'un dictionnaire.

4.4 Conclusion

Dans ce chapitre, nous avons proposé une étude statistique sur la structure interne d'automates représentant des dictionnaires électroniques. Onze automates répartis en trois catégories ont été examinés et comparés, avant et après l'application de l'algorithme *Recherche SA*.
La recherche des sous-automates a ainsi permis de déterminer la structure interne des automates ; il s'avère qu'elle se compose de séries pures, de parallèles purs et de sous-automates imbriqués les uns dans les autres.
La fréquence de ces sous-structures au sein des automates est souvent élevée et représente des informations susceptibles d'être factorisées. Une compression nous permettrait d'une part, de réduire l'espace mémoire nécessaire au stockage des automates et d'autre part de conserver un accès efficace aux données.
L'étude de ces sous-automates a permis de mettre en évidence des similarités entre les divers automates. De plus, une application naturelle utilisant les sous-automates série a été testée, il s'agit de "l'étude des fréquences". Cette étude indique qu'une langue

donnée se signe par la fréquence d'apparition de ses lettres et séquences. L'étude des fréquences des séries pures peut constituer une première étape pour reconnaître les langues des automates représentant des dictionnaires. En effet, une analyse de la fréquence des occurrences des caractères, des bigrammes et des trigrammes dans les séries pures permet, en les comparant aux fréquences connues pour chaque dictionnaire, de déterminer la langue du dictionnaire.

Chapitre 5

Indexation des automates acycliques à nombre fini d'états et application à la compression

Les automates à nombre fini d'états sont utilisés entre autres pour le stockage des dictionnaires de langues. L'atout majeur de cette représentation est l'accès rapide à l'information. Cependant, le volume de tels automates reste imposant. Ainsi, comme dans un texte, quand la quantité d'information est suffisamment importante, il peut devenir judicieux d'envisager de l'indexer ou de la classer.
L'objectif principal de ce chapitre est de constituer une palette d'outils pour compresser et indexer des automates représentant des dictionnaires. Cette compression doit nous permettre, d'une part, de réduire l'espace mémoire nécessaire au stockage des automates et, d'autre part, de conserver un accès efficace aux données pour la reconnaissance des mots par exemple.

Ce travail se découpe en deux parties : La première correspond à la conception et la réalisation d'une méthode de compression à taille fixe, appliquée directement aux automates. La deuxième, combine l'algorithme de factorisation *RechercheSA* et celui de compression. En outre, nous étudions, d'une part, la possibilité d'indexer des automates et, d'autre part, la possibilité d'utiliser des algorithmes dédiés initialement à l'indexation des textes pour le faire. Le choix des factorisations s'appuie sur le calcul de l'automate des suffixes associé aux sous-structures décelées dans l'automate. Le processus est itératif, il utilise une heuristique qui permet de sélectionner à chaque itération, à partir de l'automate des suffixes, la sous-structure la plus intéressante à factoriser ; celle-ci maximise le gain de mémoire et réduit la taille de l'automate initial. Cette heuristique sera définie en fonction des caractéristiques de l'automate après sa compression. Cette partie du chapitre sera suivie des résultats de différents tests réalisés sur différentes variantes du programme. Ces variantes sont construites en fonction du type de données sur lesquelles elles s'appliquent. En effet, trois méthodes de factorisation ont été testées. La première s'applique directement sur des automates minimisés, la deuxième considère les automates non minimisés et la troisième traite directement du texte et s'applique aux dictionnaires avant même de générer leurs automates associés. Pour pousser plus loin l'expérimentation, nous avons aussi considéré le problème à l'envers en inversant les mots des dictionnaires. Ainsi, les trois méthodes de factorisation ont été, à nouveau, testées.

L'automate des séquences ADN (catégorie 3) sera absent de ce chapitre en raison de la taille de ses mots. En effet la méthode de compression utilise un $DAWG$ pour comparer les sous-structures et dans le cas de cet automate, la génération du $DAWG$ prend trop de temps.

Notations

- $Arr[]$: arrondi à l'entier supérieur.
- $log_2()$: logarithme en base 2.
- $ValMax$: valeur maximale de l'ensemble des adresses des états buts.

Abréviations

- FCM : Factorisation et compression d'un automate minimal.
- $FCNM$: Factorisation et compression d'un automate non minimal.
- $FCDic$: Factorisation et compression d'un dictionnaire.
- $FCM_{inverse}$: Factorisation et compression d'un automate minimal de mots inversés.
- $FCNM_{inverse}$: Factorisation et compression d'un automate non minimal de mots inversés.
- $FCDic_{inverse}$: Factorisation et compression d'un dictionnaire de mots inversés.

5.1 Représentation et stockage d'un automate

Rappelons qu'un automate est représenté en mémoire par la liste de ses transitions sortantes et qu'à chaque transition sont associées trois informations : i) un indicateur d'appartenance à l'état courant, ii) une étiquette et iii) l'adresse de son état d'arrivée (voir Chapitre 4).

5.1.1 Optimisation de l'occupation mémoire d'un automate

Les automates sont souvent stockés dans des fichiers textes ; dans le but de réduire l'espace mémoire nécessaire, nous avons codé chaque donnée du même type sur un nombre de bits minimal, mais constant. Cette méthode permet, d'une part, de stocker les automates à moindre coût et, d'autre part, de conserver un accès efficace aux données.

Le codage réservé à chaque donnée est :

- **Codage du booléen :** 1 bit est utilisé pour le booléen. Ce booléen est un indicateur d'appartenance à l'état courant. La valeur 1 signale que la transition correspond à la première transition d'un nouvel état et la valeur 0 caractérise une nouvelle transition démarrant de l'état courant.

- **Codage des étiquettes :** Le nombre de bits nécessaires pour coder l'alphabet initial est noté $Taille_{alphabet}$ et il correspond à :

$$Taille_{alphabet} = Arr[log_2(|\Sigma|)] \tag{5.1}$$

– **Codage de l'adresse de l'état d'arrivée :** Le même procédé est utilisé pour représenter l'adresse par sa valeur numérique (en base 2). Un pré-traitement permet de calculer le nombre $ValMax$, correspondant à l'adresse de l'état but de valeur maximale et donc d'ajuster le nombre de bits nécessaires et suffisant pour réaliser ce codage. Ce nombre est augmenté de un car la valeur zéro est utilisée pour désigner l'état final[1]. Chaque valeur binaire correspond ainsi à l'entier du fichier d'origine.
Le nombre de bits nécessaires pour coder les adresses est noté $Taille_{adresse}$ et il correspond à :

$$Taille_{adresse} = Arr[log_2(ValMax + 1)] \tag{5.2}$$

L'adresse est une adresse absolue puisque le dictionnaire contient des mots très courts. Ce qui nécessite des grands sauts d'adresse et qui rend inutile un adressage relatif. En effet, la valeur maximale des adresses relatives serait très proche de la valeur maximale des adresses absolues.

Dans l'exemple 5.1, page 130, 10 bits sont nécessaires pour coder l'automate initial tel que : $Taille_{alphabet} = 4$ et $Taille_{adresse} = 5$.

Les informations nécessaires au décodage seront toutes placées dans un en-tête inséré dans le fichier binaire. Cet en-tête représente d'une manière générale les choix effectués pour compresser les données. On y retrouve des informations telles que le nombre de transitions de l'automate et le nombre de bits utilisés pour coder chaque type d'élément $Taille_{alphabet}$ est $Taille_{adresse}$. Pour plus de détails, un schéma de l'en-tête utilisé est présent en annexe B, page 187.

5.1.2 Résultats expérimentaux

L'implémentation des méthodes a été réalisée en langage C avec le logiciel Windows Visual C++ 6.0. Le code C obtenu respecte la norme ANSI, il est donc très facilement portable sur un autre système d'exploitation.
Les automates que nous utilisons sont ceux présentés dans le chapitre précédent, 11 automates représentant des dictionnaires répartis en 3 catégories.
Les fichiers correspondant aux automates de ces dictionnaires sont très gourmands en mémoire ; en effet, la colonne 1 du tableau 5.1 indique la taille initiale de ces automates. La compression des données a permis d'optimiser la place requise pour le stockage, la colonne 2 du tableau 5.1 présente la taille des automates après compression (compression obtenue par le codage présenté en section 5.1.1), les en-têtes y sont inclus. Les gains de place sont appréciables.

[1]Étant donné que les adresses commencent à la valeur 0, l'utilisation de $ValMax + 1$ permet de coder la valeur exacte de chaque adresse et non la valeur -1.

Automate	Taille Initiale (ko)	Taille après compression (ko)	Taux compression
Catégorie 1			
DELAF Fr	2 178	563	74,15%
DELAF En	3 081	801	73,97%
DELAF Sr	2 340	591	74,75%
DELAF De	4 133	1 105	73,27%
Villes Françaises	1 108	291	73,74%
Polylexicaux Anglais	8 963	2 451	72,65%
Catégorie 2			
Web Fr	3 561	946	73,43%
Web Bg	2 504	638	74,52%
Web Hg	3 236	858	73,48%
Web Pt	6 510	1 776	72,72%

TAB. 5.1 – Occupation mémoire des automates

5.1.3 Décompression

La compression doit toujours s'accompagner de la décompression qui permet de retrouver les données d'origine.
L'accès à l'information au sein du fichier compressé est facile grâce aux informations présentes dans l'en-tête. En effet, chaque transition de l'automate est codée sur une taille fixe. Pour décompresser ce fichier, nous avons seulement besoin de trois informations : le nombre de transitions de l'automate, $Taille_{adresse}$ et $Taille_{alphabet}$. Ces informations sont placées dans l'en-tête du fichier binaire lors d'un prétraitement.
Chaque transition peut ainsi être reconstituée en récupérant son caractère à partir de la table des correspondances et son adresse par la conversion de son code binaire en valeur décimale.
Dans l'exemple 5.1, la compression de l'automate initial utilise 10 bits pour le codage des données : $Taille_{alphabet} = 4$ et $Taille_{adresse} = 5$. La décompression de l'automate récupère cette information à partir de l'entête et décode le fichier binaire par la conversion des données en valeur décimal.

Dans ce qui va suivre nous allons nous intéresser uniquement aux sous-automates séries et aux sous-automates parallèles, étant donné que le troisième type de sous-automate n'existe pas au sein des différents automates testés.
Pour alléger notre propos, nous utiliserons directement les termes "séries" et "parallèles" pour désigner les sous-automates séries et les sous-automates parallèles.

5.1.4 Représentation étendue et stockage d'un automate

En s'appuyant sur la méthode décrite au chapitre 2, on peut remplacer les sous-automates séries et les sous-automates parallèles par de simples transitions, cela revient en fait à étendre l'alphabet. Pour cela on numérote à partir de 1 l'alphabet initial et on associe les numéros au delà de la dernière lettre aux sous-structures traitées. Le stockage se décompose en deux parties, le fichier habituel décrivant un automate auquel s'ajoute un fichier répertoriant l'alphabet étendu.

Une série est représentée par la concaténation des étiquettes de ses transitions dans leur ordre d'apparition. Un parallèle est lui aussi représenté par la concaténation des étiquettes de ses transitions selon un ordre pré-établi sur l'alphabet, par exemple, par numéro croissant. Cet ordre permet de comparer les parallèles pour localiser les redondances. De plus, il est associé à chaque mot un booléen précisant la nature de la sous-structure qu'il représente, il prend la valeur *vrai* pour désigner une série et $faux$ pour désigner un parallèle, noté respectivement S et P sur la figure 5.2.

Le codage réservé au booléen et le codage réservé à l'adresse de l'état d'arrivée sont maintenus. Cependant le codage des étiquettes évolue pour tenir compte de l'extention de l'alphabet.

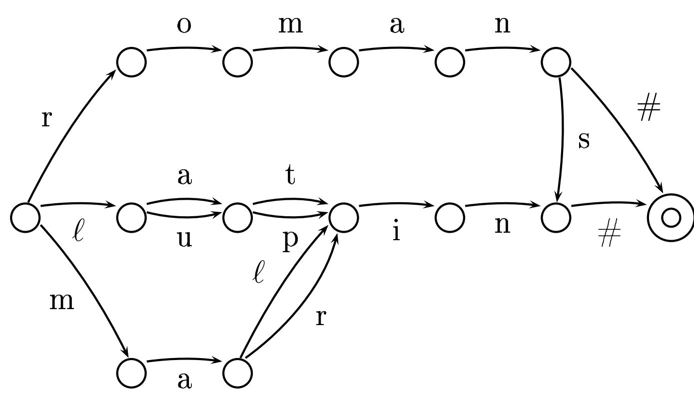
$$Taille_{alphabet} = Arr[log_2(|\Sigma'|)] \tag{5.3}$$

Où Σ' est l'union de l'alphabet initial Σ et des nouveaux caractères alloués aux séries et parallèles.
Dans l'exemple 5.1, 9 bits sont nécessaires pour coder l'automate réduit tel que : $Taille_{alphabet} = 5$ $Taille_{adresse} = 3$.

Exemple 5.1
La table 5.2 présente la recherche des sous-structures à partir de l'automate initial. Elle conclut à un automate réduit composé de 4 états et 5 transitions. À cet automate est associé un alphabet étendu contenant l'alphabet initial, 12 caractères, et 9 sous-structures distribuées en 4 parallèles et 5 séries. À titre d'exemple, le "caractère" 13 représente le parallèle composé des transitions a et u.

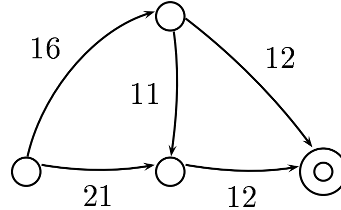
Automate initial

1	1	ℓ	7
2	0	m	6
3	0	r	4
4	1	o	5
5	1	m	9
6	1	a	10
7	1	a	12
8	0	u	12
9	1	a	14
10	1	ℓ	15
11	0	r	15
12	1	p	15
13	0	t	15
14	1	n	17
15	1	i	16
16	1	n	19
17	1	#	0
18	0	s	19
19	1	#	0



Automate réduit

1	1	16	3
2	0	21	5
3	1	11	5
4	0	12	0
5	1	12	0



Alphabet		
Alphabet initial		
1	ℓ	
2	m	
3	r	
4	o	
5	a	
6	u	
7	p	
8	t	
9	n	
10	i	
11	s	
12	#	
Alphabet étendu		
13	P	5 6
14	P	7 8
15	P	1 3
16	S	3 4 2 5 9
17	S	1 13 14
18	S	2 5 15
19	S	10 9
20	P	17 18
21	S	20 19

TAB. 5.2: Représentation étendue d'un automate

5.2 Lecture d'un mot à partir de l'automate réduit

L'automate réduit A est déterministe, il conserve ainsi la possibilité de vérifier si un mot w appartient ou non au langage. L'algorithme *Lecture* permet cette vérification.

Notons que l'alphabet d'un l'automate réduit A est composé de l'alphabet initial qui contient tous les caractères présents dans le dictionnaire auquel s'ajoute l'alphabet étendu qui liste toutes les sous-structures factorisées dans l'automate.

$$\text{Alphabet } (A) = \text{Alphabet initial } (A) + \text{Alphabet étendu } (A).$$

Pour les besoins de l'algorithme *Lecture*, notons aussi que l'alphabet d'un état p de A est constitué des étiquettes de toutes ses transitions sortantes. Ces étiquettes appartiennent soit à l'alphabet initial de A soit à l'alphabet étendu de A.

$$\text{Alphabet } (p) = \text{Alphabet initial } (p) + \text{Alphabet étendu } (p).$$

Un prétraitement est nécessaire avant d'exécuter cet algorithme, du fait que l'alphabet initial de A a été recodé par des entiers naturels, à partir de 1. On commence par compléter un mot w avec le caractère de fin de mot ($w\#$), puis on le code en utilisant la table des correspondances de l'alphabet initial de A placé dans l'en-tête (cf. annexe B, page 187). Si w contient un caractère qui n'appartient pas à l'alphabet initial de A, il est rejeté.

L'algorithme *Lecture* démarre de l'état initial q_i à la recherche de l'unique chemin de w dans A reliant q_i à q_f. À chaque état p, il compare les étiquettes de ses transitions sortantes au caractère courant w_i. Si w_i appartient à l'alphabet initial de p alors la transition portant l'étiquette w_i est franchie. Dans le cas contraire, l'alphabet étendu de p est analysé. Chaque élément de cet alphabet est examiné pour retrouver l'alphabet initial dont il est composé grâce aux deux procédures *Analyser Série* et *Analyser Parallèle.*
La procédure *Analyser Série* permet de vérifier si w_i appartient à une série et la procédure *Analyser Parallèle* vérifie si w_i est inclus dans un parallèle.

Exemple
La recherche du mot "w=malin" dans l'automate réduit A de la figure 5.2, page 131, commence d'abord par coder w : "w= 2 5 1 10 9 12" tel que $car_1 = 2$, $car_2 = 5$, $car_3 = 1$, $car_4 = 10$, $car_5 = 9$ et $car_6 = 12$.
Ensuite, la procédure *Lecture* démarre de l'état initial (p est instancié à q_i) à la recherche d'un chemin qui valide w. L'état p possède deux transitions sortantes t_1 et t_2 dont les étiquettes, 16 et 21, appartiennent à l'alphabet étendu.
L'analyse de $t_1 = 16$ indique qu'elle représente une série dont la lettre initiale est 3 alors que le caractère de départ de w est 2. t_1 ne peut pas valider w, elle ne sera donc pas franchie /*Lignes 7-14* de l'algorithme 14/
L'analyse de $t_2 = 21$ conduit à des appels récursifs des procédures *Analyser Série* et *Analyser Parallèle* parce que les caractères de t_2 appartiennent eux aussi à l'alphabet étendu de A. Selon l'ordre de lecture, l'analyse de 21 par la procédure *Analyser Série*

examine d'abord le parallèle 20, la procédure *Analyser Parallèle* montre qu'il contient lui aussi des éléments de l'alphabet étendu 17 et 18. Pour examiner 17, on fait appel à la procédure *Analyser Série*, qui indique que la série 17 débute par un caractère différent de car_1. Ce chemin est alors écarté. L'examen de 18 révèle que les deux premiers caractères de la série 18 sont car_1 et car_2, elle correspond ainsi à un chemin qui pourrait valider w. L'analyse du caractère suivant 15 appelle la procédure *Analyser Parallèle* et valide car_3. Les caractères car_4 et car_5 sont reconnus par l'analyse de la série 19. Ainsi, partant de l'état initial, la transitions 21 est franchie pour aboutir à son état but, cet état possède une seule transition sortante et son étiquette correspond à car_5. Elle est franchie et permet d'atteindre l'état final.
Le mot "w" appartient donc au langage.

Algorithm 14 Lecture - Entrée : A, w - Sortie : $Existe$

```
1: p ← q_i ; i ← 1 ;
2: while (p ≠ q_f) et (w_i ≠"\0") do
3:    Existe ← Faux ;
4:    if w_i ∉ alphabet initial de p then
5:       for j = 1 à Nb/* Nb= L'ensemble des transitions t sortantes de p dont l'étiquette
         appartient à l'alphabet étendu/* do
6:          if t_j est une série then
7:             Analyser Série (t_j,i,Existe)
8:          else
9:             Analyser Parallèle (t_j,i,Existe)
10:         end if
11:         if Existe = Vrai then
12:            p ← État but de t_j ; j ← Nb+1 ; /*Franchir t_j et arrêt de la boucle FOR */
13:         end if
14:      end for
15:      if Existe = Faux then
16:         p ← q_f ; /* w_i n'existe pas dans A, arrêt de la boucle WHILE*/
17:      end if
18:   else
19:      Existe ← Vrai ;
20:      p ← État but de la transition étiquetée par w_i ; /*Franchir la transition w_i*/
21:      i + +;
22:   end if
23: end while
24: if (p = q_f) et (i = |w| + 1) et (Existe = Vrai) then
25:    w ∈ L(A) /* Validation de w dans A*/
26: end if
```

Algorithm 15 Analyser série - Entrée : u, $Existe$, i, w - Sortie : $Existe$, i

1: **for** $s = 1$ à $|u|$ **do**
2: **if** $u_s \in$ alphabet initial de A **then**
3: **if** $u_s = w_i$ **then**
4: $Existe \leftarrow Vrai$; $i++$; /*$w_i \in s$*/
5: **else**
6: $Existe \leftarrow Faux$; $s \leftarrow |u| + 1$; /*$w_i \notin s$*/
7: **end if**
8: **else**
9: **if** u_s est une série **then**
10: Analyser Série $(u_s, Existe, i, w)$
11: **else**
12: Analyser Parallèle $(u_s, Existe, i, w)$
13: **end if**
14: **end if**
15: **end for**

Algorithm 16 Analyser Parallèle - Entrée : u, $Existe$, i, w - Sortie : $Existe$, i

1: **if** (u possède une transition portant l'étiquette w_i) **then**
2: $Existe \leftarrow Vrai$; $i++$; /*$w_i \in s$*/
3: **else**
4: $Existe \leftarrow Faux$
5: **for** $j = 1$ à Nb /*Nb= Nombre de transition t du parallèle u dont l'étiquette appartient à l'alphabet étendu A*/ **do**
6: **if** t_j est une série **then**
7: Analyser Série $(t_j, Existe, i, w)$;
8: **else**
9: Analyser Parallèle $(t_j, Existe, i, w)$;
10: **end if**
11: **if** $Existe \leftarrow Vrai$ **then**
12: $j \leftarrow Nb + 1$; /*$w_i \in s$, arrêt de la boucle FOR */
13: **end if**
14: **end for**
15: **end if**

Nous avons comparé le temps nécessaire pour réécrire directement le dictionnaire DELAF français à partir de l'automate initial et à partir de l'automate réduit.
Le temps de reconstruction augmente légèrement, il est de 0,8 secondes à partir de l'automate réduit et il est de 0,7 secondes à partir de l'automate initial.

Dans cette partie, nous avons mis en œuvre une méthode pour réduire l'occupation mémoire des automates. Cette méthode de compression est à taille fixe et à adressage absolu (compression obtenue grâce au codage présenté en section 5.1.1. En effet, la compression à taille variable est beaucoup moins efficace sur ce type de données car les informations, en nombre considérable, sont trop dissemblables ; la table de correspondance prendrait trop de place dans le fichier compressé. Ce résultat a été vérifié en appliquant le codage de Huffman.
L'adressage relatif a lui aussi été écarté parce que les mots très courts atteignent rapidement l'état final dans l'automate et induisent ainsi des grands sauts d'adresse qui pénalisent l'adressage relatif.
Il existe d'autres méthodes de compression ; cependant, dans la plupart des cas, la lecture du fichier compressé est rendue très difficile. C'est le cas notamment des codages de type dictionnaire tel que LZW. Ces codages considèrent le fichier bloc par bloc et non élément par élément. Une table des correspondances est créée dynamiquement en même temps que la compression. Le même principe est utilisé lors de la décompression, la table des correspondances est recalculée au fur et à mesure. Il est alors impossible d'interpréter les données du fichier compressé sans le parcourir séquentiellement pour recréer la table des correspondances.

5.3 Indexation automatique des automates acycliques à nombre fini d'états

L'indexation automatique d'un document texte recense toutes les occurrences des mots de ce document. Toutes ces données représentent l'index du texte et ne sont soumises à aucune contrainte (hiérarchisation ou autres). Par extrapolation, dans un automate, la liste de ses sous-automates accompagnés de leurs adresses (positions dans l'automate) représente son index. Ainsi, les sous-automates correspondants à la définition **??** constituent une information accessible, représentative du contenu de l'automate mais, aussi, facile à extraire grâce à l'algorithme *Recherche SA*.
Chacun des sous-automates visités lors de la recherche est examiné afin de retenir ses particularités : type, taille, nombre d'états, nombre de transitions, ensemble de ses positions dans l'automate initial, etc. Toutes ces particularités représentent le formulaire idéal pour naviguer dans l'automate et faciliter la recherche d'information, par exemple, la recherche de grammaires locales.

La figure 5.1 résume la transposition de l'indexation de texte à l'indexation d'automate.

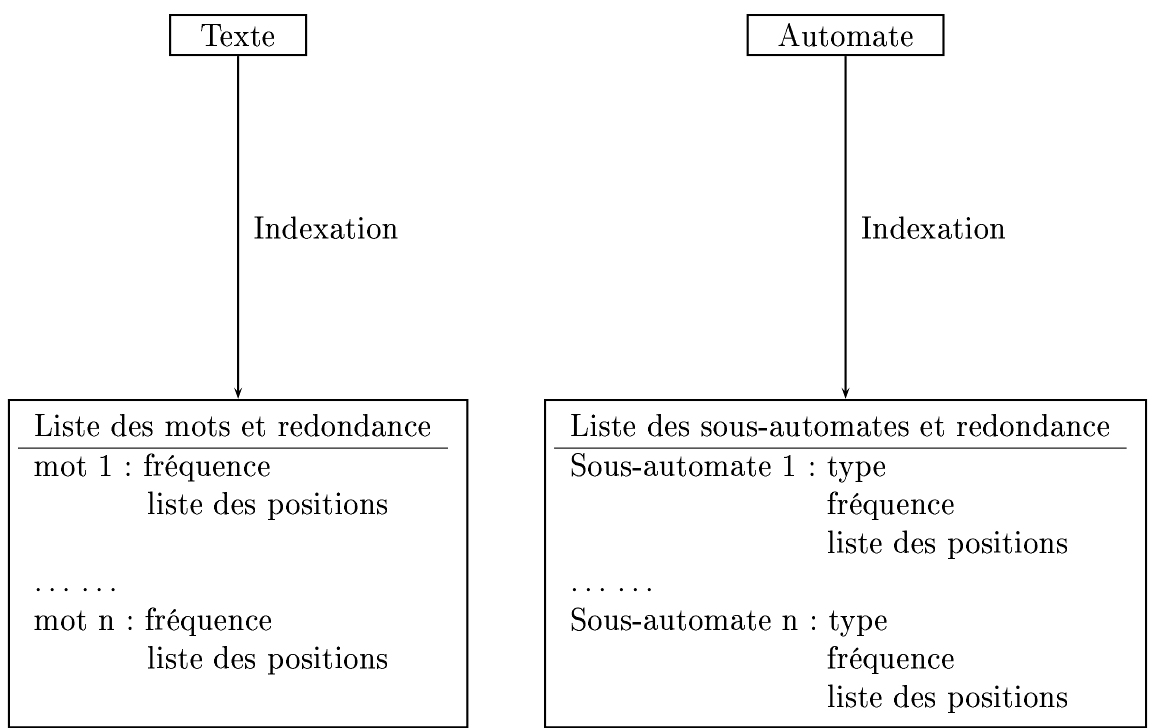


FIG. 5.1 – Indexation texte vs automate

5.3.1 Indexation des séries et parallèles

Rappelons qu'au chapitre 3 nous avons distingué plusieurs types de sous-automates et qu'au chapitre 4 nous avons présenté la recherche de ces sous-automates au sein des automates représentant des dictionnaires. Cette recherche a révélé la présence massive de sous-automates de type série ou parallèle, raison pour laquelle nous nous intéressons spécifiquement à ces deux sous-structures et à leur indexation.
Pour indexer les séries, une liste est réalisée associant à chaque série l'ensemble de toutes ses positions dans l'automate. La position d'une série dans l'automate est donnée par le numéro de ligne de sa transition de départ dans le fichier représentant l'automate. Une liste similaire est dressée pour indexer les parallèles.

5.3.2 Indexation des sous-séries et sous-parallèles

Pour aller plus loin dans l'indexation des parties internes d'un automate, il serait intéressant d'avoir accès, non seulement aux positions des séries et des parallèles, mais aussi à l'ensemble des positions des sous-séries et des sous-parallèles présents dans l'automate. Nous avons étudié la possibilité d'utiliser des systèmes dédiés initialement à l'indexation des textes pour le faire. Une technique très intéressante, qui répond à nos besoins, est l'automate des suffixes (Directed Acyclic Word Graph "DAWG"). En effet, plusieurs études autour de l'indexation des répétitions de structures, des mots et des textes, font usage d'un DAWG, voir par exemple [Crochemore et Hancart, 1997] et [Crochemore *et al.*, 2001]. Dans notre cas, l'utilisation d'un *DAWG* assurerait un avantage certain, parce que ce procédé nous permettrait de calculer les positions de toutes les occurrences d'une sous-structure, en y détectant des inclusions, des chevauchements, etc.

5.3.3 Automate des suffixes "*DAWG*"

Un *DAWG* est un automate des suffixes qui représente une structure efficace pour le traitement et l'analyse des répétitions dans un texte. Son algorithme de construction repose sur une lecture de gauche à droite du texte pour créer au fur et à mesure l'automate minimal des suffixes en temps linéaire. Il représente ainsi le texte sous forme d'un graphe.
Cette structure possède une grande variété d'applications parmi lesquelles l'indexation, la comparaison de séquences, le calcul des positions des mots et sous-mots dans un texte et le calcul des fréquences. En effet, dans la version présentée par Mehryar Mohri [Mohri, 1996], à chaque état du *DAWG* est associée une liste d'entiers représentant l'ensemble des positions des mots reconnus, à cet état, dans le texte. Ainsi, une fois qu'un mot ou sous-mot est lu à partir du *DAWG*, l'ensemble de ses positions peut être directement obtenu à partir de la liste des nombres entiers associés à l'état atteint.

Exemple 5.2
L'automate de la figure 5.2, extraite de [Mohri, 1996], représente le *DAWG* du texte T où $T = aabba$. La lecture de la séquence "a" aboutit à l'état 2 du *DAWG*, cet état possède un index contenant trois positions qui correspondent aux positions de la séquence "a"

dans le texte T et indiquent aussi sa fréquence : "a" apparaît trois fois dans le texte. L'état 2 est un état final, cette information indique que la séquence reconnue à cet état est un suffixe de T.
L'état 5 peut être atteint par trois chemins distincts, la longueur de la séquence parcourue associée aux positions de l'index permet de la situer dans le texte. Par exemple, la lecture de la séquence "*abb*", séquence de longueur 3, aboutit à l'état 5 qui indique la position 4 dans le texte ; la différence entre la position et la longueur permet d'obtenir le point de départ de la séquence dans T. L'état 5 n'étant pas final, aucune des séquences qu'il valide n'est un suffixe de T.

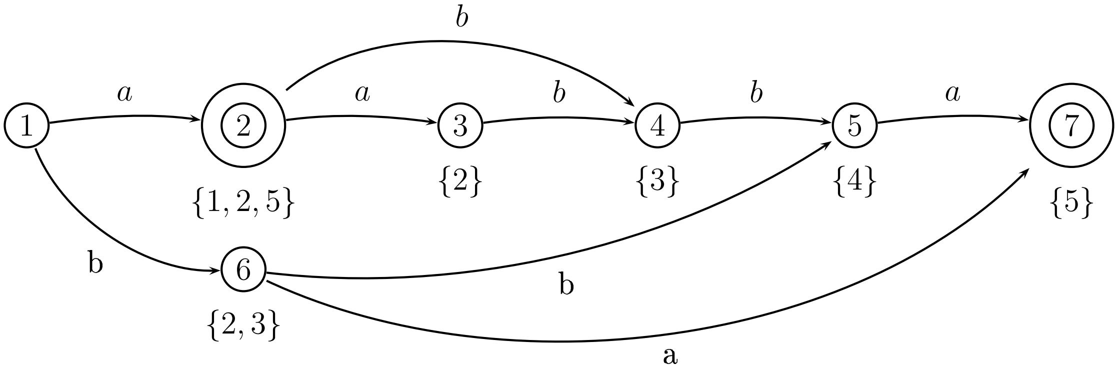


FIG. 5.2 – DAWG de "*aabba*"

5.3.4 Comment adapter le *DAWG* ?

Nous avons adapté ce procédé à notre problème dans le but d'indexer et de comparer les sous-structures détectées dans les automates (séries ou parallèles). Pour alléger la recherche et faciliter la lecture des informations de l'index d'un *DAWG*, les séries et les parallèles sont traités séparément. Un *DAWG* est généré pour analyser les séries et un autre pour examiner les parallèles.
Les étapes ci-dessous décrivent l'adaptation aux séries, pour les parallèles les mêmes étapes s'appliquent, mais un traitement supplémentaire est nécessaire, qui est décrit plus loin.

1. **Transformation des données d'entrée**
   Les séries sont converties en mots pour s'adapter aux formats des données manipulées par le *DAWG*. Un mot représentant une série est formé par la concaténation des étiquettes des transitions de cette série dans l'ordre de leur apparition.
2. **De l'intérêt des états finaux**
   À chaque état d'un *DAWG* correspond au moins une sous-série qui peut être préfixe, infixe ou suffixe d'une série de la liste initiale. Ainsi, chaque état de ce *DAWG* est à prendre en compte, qu'il soit final ou non.
3. **Utilisation d'un index double dans le *DAWG***
   À chaque état d'un *DAWG* correspond un index qui indique les positions (dans l'automate initial) des sous-séries associées à l'état. Cet index comporte deux parties :

(a) La liste des séries contenant ces sous-séries.
(b) Les positions des sous-séries dans les séries (pour chaque série, les positions de ces sous-séries).

Dans l'exemple 5.3, l'état 9 correspond aux sous-séries "d", "cd" et "ccd". Son index associé est $Index(9){=}S_1\{3\}, S_3\{3\}$, ce qui indique que chacune des trois sous-séries est incluse dans S_1 et dans S_3 et se termine en position 3 dans les deux séries.

4. **Traitement des parallèles**

Comme pour les séries, les parallèles sont transformés en mots. Un mot représentant un parallèle est formé par la concaténation ordonnée des étiquettes, par exemple l'ordre alphabétique, permettant ainsi de détecter les parallèles identiques.

Néanmoins, un traitement supplémentaire est nécessaire car le $DAWG$ associé au mot représentant un parallèle ne permet pas de produire tous ses sous-parallèles. Pour les créer, il faut associer plusieurs mots à un parallèle. Plus exactement 2^{n-2} mots, où n représente le nombre de transitions du parallèle.

Par exemple, pour générer tous les sous-parallèles du parallèle "*abcde*", on va créer le $DAWG$ associé aux huit mots : "*ae*", "*abe*", "*ace*", "*abce*", "*ade*", "*abde*", "*acde*" et "*abcde*".

Si le parallèle est composé des étiquettes $a_1, a_2, \cdots, a_n$, on indexe tous ses sous-parallèles en mettant en entrée du $DAWG$ les mots de la forme $a_1\ a_2^{\alpha 2}\ a_3^{\alpha 3} \cdots\ a_{n-1}^{\alpha n-1}\ a_n$, où $\alpha_i = 1$ si la lettre a_i est présente dans le mot, 0 sinon.

<u>Démonstration</u>

Par récurrence :

- $n = 2$: L'ensemble des mots (sous-parallèles) d'un parallèle composé de deux transitions "a_1" et "a_2", est le parallèle lui même "$a_1\ a_2$". Il suffit donc de créer le DAWG associé à $a_1\ a_2$.
- Supposons la proposition vrai pour un parallèle composé de $n-1$ transitions et considérons un parallèle P composé de n transitions portant les étiquettes $a_1, a_2, \cdots, a_n$.

  Un sous-parallèle de P contient ou ne contient pas l'étiquette a_2 et il y a autant de sous-parallèles contenant a_2 que de sous-parallèles ne le contenant pas.

  Indexer tous les sous-parallèles ne contenant pas a_2 revient à indexer tous les sous-parallèles du parallèles composé des $n-1$ étiquettes $a_1, a_3, \cdots, a_n$. Il faut donc mettre en entrée du $DAWG$ les 2^{n-3} mots de la forme $a_1 a_3^{\alpha 3} \cdots a_{n-1}^{\alpha n-1} a_n$.

  Pour indexer les sous-parallèles contenant a_2, il suffit de mettre en entrée du DAWG les 2^{n-3} mots de la forme $a_1\ a_2\ a_3^{\alpha 3} \cdots a_{n-1}^{\alpha n-1} a_n$. C'est également nécessaire car si on ne met pas un mot $a_1 a_2 w a_n$, alors le sous-parallèle composé des étiquettes a_1, a_2 et a_n et de toutes les lettres de w ne pourra jamais être indexé. Ce qui fait bien au total 2^{n-2} mots de la forme $a_1 a_2^2 a_3^3 \cdots a_{n-1}^{\alpha n-1} a_n$.

□

Les parallèles présents dans les automates étudiés sont peu larges, cet aspect exponentiel n'est donc pas trop pénalisant.

En l'adaptant ainsi, le *DAWG* constitue un outil bien adapté à l'indexation des séries et des parallèles.

Exemple 5.3
Soient S_1, S_2 et S_3 trois séries détectées dans un automate donné en entrée A.

- S_1= *ccdbaab*
- S_2= *baab*
- S_3= *ccdb*

Supposons que S_1 apparaisse quatre fois dans A et que, en dehors de S_1, S_2 apparaisse une fois dans A et S_3 y apparaisse deux fois. Donc au total, S_2 apparaît cinq fois dans A et S_3 six fois.
La figure 5.3 présente le *DAWG* construit pour S_1, S_2 et S_3. A chaque état du *DAWG* est associé un index représentant les positions des sous-séries qu'il reconnaît dans S_1, S_2 et S_3. Cet index indique aussi la fréquence des sous-séries reconnues à cet état.

Liste des index :

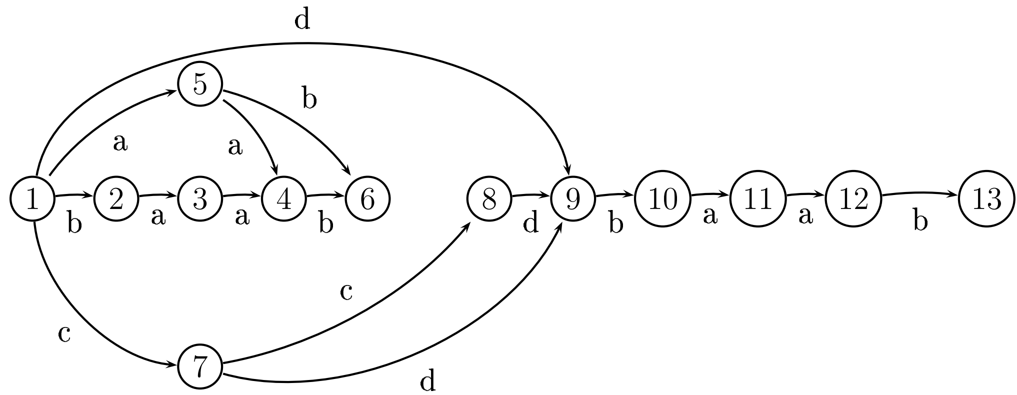


FIG. 5.3 – $DAWG(S_1, S_2, S_3)$

$Index(1)$=∅
$Index(2)$=$S_1\{4,7\}, S_2\{1,4\}, S_3\{3\}$
$Index(3)$=$S_1\{5\}, S_2\{2\}$
$Index(4)$=$S_1\{6\}, S_2\{3\}$
$Index(5)$=$S_1\{5,6\}, S_2\{2,3\}$
$Index(6)$=$S_1\{7\}, S_2\{4\}$
$Index(7)$=$S_1\{1,2\}, S_3\{1,2\}$
$Index(8)$=$S_1\{2\}, S_3\{2\}$
$Index(9)$=$S_1\{3\}, S_3\{3\}$
$Index(10)$=$S_1\{4\}, S_3\{4\}$
$Index(11)$=$S_1\{5\}$
$Index(12)$=$S_1\{6\}$
$Index(13)$=$S_1\{7\}$

5.4 Factorisation et compression d'automates acycliques à nombre fini d'états

Le travail effectué dans cette partie est combiné à l'indexation et il se situe en amont de la compression. En effet, la factorisation permet de réorganiser l'automate dans le but de réduire la quantité de données nécessaires pour le représenter. Ainsi, l'objectif est d'alléger l'automate en diminuant son nombre d'états et son nombre de transitions sous la condition stricte de ne pas modifier le langage initialement reconnu.

Le tableau 5.3 illustre les résultats de factorisation et compression des automates étudiés au chapitre 3[2]. À titre d'exemple, la taille initiale de l'automate représentant le DELAF français est 2 178 Ko, après une recodage des données, l'automate est réduit à 563 Ko (voir le tableau 5.1, page 129). Si on factorise toutes les séries et les parallèles avant de le comprimer, sa taille indique 631 Ko. Le résultat est donc meilleur lorsque l'on ne factorise pas l'automate !
Ce qu'il faut retenir de ce tableau est qu'en pratique, toutes les sous-structures ne sont pas forcément intéressantes à factoriser, on remarque ainsi que, en général, les taux de compression sont plus faibles comparés à ceux présentés dans la table 5.1.
Effectivement, lorsqu'on factorise une série ou un parallèle non redondant on augmente forcément la taille de l'alphabet en la remplaçant dans l'automate par une transition lui faisant référence. Pour éviter d'augmenter la taille de stockage, il est indispensable de choisir les bons éléments à factoriser ; pour cela on utilise un indicateur qui permet d'évaluer si une sous-structure est intéressante à factoriser ou non.

Automate	Taille après factorisation et compression (ko)	Taux compression
Catégorie 1		
DELAF Français	631	70,02%
DELAF Anglais	893	71%
DELAF Serbe	647	72,35%
DELAF Allemand	1 320	68%
Villes Françaises	31	97,02%
Polylexicaux Anglais	2 509	72%
Catégorie 2		
Français	1 064	70,12%
Hongrois	939	70,98%
Bulgare	719	71,28%
Portugais	1 989	69,45%

TAB. 5.3 – Factorisation et compression de toutes les séries et tous les parallèles

[2] Rappelons que ces automates sont totalement dénués de séries et de parallèles.

5.4.1 Comment choisir les sous-structures à factoriser ?

Pour choisir les sous-structures à factoriser, nous utilisons une fonction appelée "fonction gain" et notée "Δ". Elle calcule le gain qui sera obtenu lors de la factorisation de chaque sous-structure, elle permet ainsi de comparer les plus intéressantes et de sélectionner la sous-structure qui sera factorisée.
En effet, pour chaque sous-structure, la fonction gain calcule, d'une part, l'espace mémoire économisé en supprimant toutes ses occurrences de l'automate et, d'autre part, l'espace mémoire consommé en l'insérant à l'alphabet. Puis, la différence indique s'il est avantageux de factoriser cette sous-structure ou non.

Pour calculer le gain, il sera associé à chaque sous-structure présente dans un automate A les informations suivantes :
- L : la longueur de la sous-structure (le nombre de ses transitions).
- F : la fréquence de la sous-structure (le nombre de ses occurrences dans A).
- $Taille_{alphabet}$: le nombre de bits nécessaires pour stocker l'alphabet final.
- $Taille_{adresse}$: le nombre de bits nécessaires pour stocker les adresses (correspond aussi au nombre de transitions de A).
- $Taille_{sous-structure}$: le nombre de bits nécessaires pour stocker le nombre de transitions de la sous-structure.

5.4.1.1 Fonction gain Δ

Initialement, chacune des transitions de l'automate est représentée par trois informations (voir section 5.1, page 127). Ainsi, la taille allouée à ces informations, notée $Taille_{transition}$, peut être calculée par la formule suivante :

$$Taille_{transition} = 1 + Taille_{alphabet} + Taille_{adresse} \tag{5.4}$$

Calcul du bénéfice

Le bénéfice est obtenu en supprimant la série et l'ensemble de ses occurrences de l'automate.

$$bénéfice = F * (L * Taille_{transition}) \tag{5.5}$$

Calcul du préjudice

Le préjudice est obtenu en insérant la sous-structure dans l'alphabet.

$$préjudice = L * Taille_{transition} + 1 + Taille_{sous-structure} + (L * Taille_{alphabet}) \tag{5.6}$$

Ainsi, la fonction de gain est représentée par l'équation suivante :

$$\Delta = (F - 1) * L * Taille_{transition} - 1 - Taille_{sous-structure} - (L * Taille_{alphabet}) \tag{5.7}$$

Les paramètres de la fonction gain sont déterminés à partir du codage du fichier compressé. Étant donné que les tailles calculées varient au fur et à mesure de la factorisation, il est nécessaire d'émettre plusieurs hypothèses, dès le départ, pour fixer ces tailles.
Les hypothèses vont porter principalement sur les paramètres $Taille_{alphabet}$ et $Taille_{adresse}$. L'impact de $Taille_{sous-structure}$ est négligeable pour l'instant car les sous-structures des automates sont en moyenne de petite taille pour les dictionnaires de catégorie 1 et 2 (voir tableau 4.6, page 97).

5.4.2 Calcul des fréquences à l'aide d'un $DAWG$

Dans le but de comparer les différentes sous-structures et calculer leurs fréquences, nous avons étudié la possibilité d'utiliser l'automate des suffixes pour le faire. Déjà utilisé à l'étape de l'indexation, le $DAWG$ permet encore une fois de traiter notre problème. En effet, les index associés à chaque état représentent les occurrences des sous-structures.
Dans l'exemple 5.3, l'état 8 possède une sous-série "cc", l'index de l'état 8, $Index(8){=}S_1\{2\}, S_3\{2\}$, indique que "cc" est présente 1 fois dans S_1, à la position 2 et une fois aussi dans S_3 à la position 2. De plus S_1 apparaît 4 fois dans A et S_3 deux fois. Ainsi, la fréquence de "cc" dans l'automate A correspond à $4 * 1 + 2 * 1 = 6$.

5.4.3 Comment factoriser ?

Une sous-structure peut être incluse dans une autre sous-structure, aussi, on ne peut pas prévoir l'impact d'une factorisation, ou d'une combinaison de factorisations, sur les autres. La marche à suivre pour traiter ce problème consiste à mettre en place des heuristiques qui, à chaque étape, calculent le gain et proposent les factorisations les plus intéressantes.

Dans l'exemple 5.3, page 140, plusieurs scénarios de factorisation sont possibles et chacun engendre un gain différent. Rappelons que S_1 apparaît quatre fois dans A et que en dehors de S_1, S_2 apparaît une fois dans A et S_3 y apparaît deux fois. Donc au total, S_2 apparaît cinq fois dans A et S_3 six fois. Soient x, y, z trois variables représentant les fréquences respectives de S_1, S_2 et S_3 dans A :
$x = 4$, $y = 1$ et $z = 2$.

1. Si on factorise (x) S_1 puis $(z)S_3$ le gain sera de $(4*6-7)+(2*3-4) = 17+2 = 19$.
2. Si on factorise $(x + y)S_2$ puis $(x + z)ccd$ le gain sera de $11 + 9 = 20$.
3. Si on factorise $(x+y)S_2$ puis $(x)ccd$ et ensuite $(z)S_3$ le gain sera de $11+5+2 = 18$.
4. Si on factorise $(x + y)aab$ puis $(x + z)S_3$ le gain sera de $5 + 14 = 19$.

À chaque état p du $DAWG$ correspond une ou plusieurs sous-structures et à chaque sous-structure est associé son gain (calculé en fonction de sa longueur et de sa fréquence). Donc, l'état p possède plusieurs gains possibles.
Puisque le but est bien évidemment d'alléger le traitement, nous proposons une première heuristique pour limiter l'intervalle des valeurs à explorer et favoriser la factorisation des

plus longues séries. Ainsi, il sera associé à chaque état p le gain de la factorisation de la plus longue de ses sous-structures.

5.4.4 Comment adapter le $DAWG$ à la factorisation ?

La factorisation d'une sous-structure w_i implique des mises à jour de l'automate initial mais aussi du $DAWG(w_1, .., w_n)$. En effet, lorsqu'elle est choisie, w_i est retirée du $DAWG(w_1, .., w_n)$ par la suppression de tous les index lui faisant référence. Lorsqu'un état du $DAWG(w_1, .., w_n)$ possède un index vide, il est lui aussi éliminé.
Si $\exists w_j \in \{w_1, .., w_n\} : w_i \subset w_j$ et $|w_i| \prec |w_j|$ alors la factorisation de w_i crée une nouvelle sous-structure w_{n+1} en remplaçant w_i par la nouvelle lettre de l'alphabet lui faisant référence dans w_j.
Le $DAWG$ évolue au fur et à mesure de cette extension et intègre aisément tout nouvel élément.

Remarque 5.1
Lorsqu'un état d'un DAWG ne référence plus aucune sous-structure (il possède un index vide), il est débarrassé de ses transitions entrantes et sortantes pour se transformer en état non accessible avant d'être éliminé.

5.4.5 Algorithme glouton de factorisation

Nous proposons un algorithme glouton pour factoriser l'automate. Le principe général de cet algorithme est de déterminer à chaque étape, le meilleur choix local (optimum local), sans aucun retour arrière, en espérant obtenir la meilleure factorisation globale. Il localise ainsi, au fur et à mesure, la sous-structure la plus intéressante à factoriser à l'instant t. Les étapes ci-dessous décrivent le principe de cet algorithme pour les séries. Pour les parallèles les mêmes principes s'appliquent.

Notre méthode de factorisation est basée sur la construction et la réduction d'un $DAWG$ qui indexe toutes les séries localisées dans un automate donné en entrée A.
Elle passe par les étapes suivantes :

- Examiner les états du $DAWG$ pour déterminer la sous-série la plus intéressante à factoriser : celle qui produit le meilleur taux de compression à l'instant t (optimum local).
  À cette étape, on propose de factoriser la sous-série qui correspond au gain maximal du $DAWG$.
- Factoriser la sous-série dans A. L'index du $DAWG$ permet de localiser toutes les positions de la sous-série dans l'automate.
  A cette étape, on propose de réduire toutes les occurrences de la sous-série dans A.
- Supprimer la sous-série du $DAWG$ en éliminant tous les index lui faisant référence, ensuite supprimer les états qui possèdent un index vide.
- mettre à jour le $DAWG$ en y insérant les nouvelles séries : celles produites par la substitution de la sous-série dans toutes les séries où elle apparaît. La nouvelle série est représentée par un nouveau caractère de l'alphabet.

Ce processus est itératif et il s'achève lorsqu'il n'existe plus de sous-série intéressante à factoriser ou lorsque la taille maximale de l'alphabet est atteinte.

À partir de l'automate donné en entrée A, cet algorithme calcule l'ensemble CSA qui contient toutes les séries ou sous-séries choisies pour la factorisation.
En premier, il parcourt A en profondeur d'abord pour calculer l'ensemble SA contenant toutes les séries de A. Ensuite, il construit le $DAWG$ à partir des éléments de SA en prenant soin d'associer à chaque état p du $DAWG(SA)$ un gain. Ce gain est calculé grâce à la fonction Δ et il correspond au gain de la factorisation du plus long chemin arrivant à p (la plus longue série ou sous-série reconnue à p).

L'étape suivante de l'algorithme sélectionne un seul état q du $DAWG(SA)$ qui possède le meilleur gain du $DAWG(SA)$ (meilleure économie de l'espace mémoire). La sous-structure lui correspondant est alors supprimée du $DAWG(SA)$ et remplacée dans A et SA par une transition étiquetée par une nouvelle lettre de l'alphabet. Cette étape permet de générer de nouvelles séries qui seront à leurs tours insérées à l'ensemble SA et dans le $DAWG(SA)$. Ainsi, toute factorisation est suivie de mises à jour de A, SA et $DAWG(SA)$.

L'algorithme réitère le processus de factorisation jusqu'à ce que le gain de n'importe quel état du $DAWG(SA)$ ne soit plus intéressant pour la factorisation.

Algorithm 17 Choix et factorisation des séries — entrée/sortie : A - sortie : CSA

Soit SA l'ensemble des séries : $SA \leftarrow \varnothing$;
Soit CSA l'ensemble des séries factorisées : $CSA \leftarrow \varnothing$;
Calculer l'ensemble SA à partir de l'automate A ;
Créer $DAWG(SA)$ et calculer le gain pour chacun de ses états ; /**Utiliser* Δ*/
Sélectionner l'état q qui possède le meilleur gain à partir du $DAWG(SA)$; /**valeur max positive**/
while $DAWG(SA)$ contient au moins un état q possédant un gain attractif **do**
  Soit w le plus long chemin arrivant à q et α un nouveau caractère ; /**alphabet étendu**/
  Supprimer w du $DAWG(SA)$;
  Remplacer w par une transition étiquetée par α dans A ;
  Remplacer w par une transition étiquetée par α dans SA ;
  Ajouter la série w à CSA ;
  Mise à jour du $DAWG(SA)$ avec les nouveaux éléments de SA ; /* *éléments contenant* α*/
  Sélectionner l'état q qui possède le meilleur gain à partir du $DAWG(SA)$;
end while

5.5 Mise en oeuvre de la factorisation et de la compression

Dans ce qui va suivre plusieurs méthodes de factorisation et compression seront présentées dans le but de réduire la taille des automates. Pour ces applications, nous nous sommes limités à la factorisation des sous-automates séries étant donné que les sous-automates parallèles ne représentent en moyenne que 1% des transitions de l'automate. Le gain obtenu en les factorisant serait donc minime. Cependant le processus est exactement le même pour traiter les parallèles.
Pour être en mesure d'analyser et de comparer l'efficacité de l'algorithme "Choix et factorisation des séries" (algorithme 17) sur des bases différentes, indépendamment de la structure d'entrée d'un automate (qu'il soit minimal ou non), on a constitué plusieurs ensembles d'entrée possibles, auxquelles peut s'appliquer directement notre algorithme de factorisation. Trois méthodes de factorisation employant l'algorithme 17 ont été testées. La première s'applique directement sur des automates minimisés, la deuxième considère les automates non minimisés et la troisième traite directement du texte et s'applique aux dictionnaires avant même de générer leur automate associé. Pour pousser plus loin l'expérimentation sans remettre en question les différentes approches, nous avons considéré le problème à l'envers en inversant les mots des dictionnaires. Ainsi, les trois méthodes de factorisation ont été, à nouveau, testées sur des ensembles d'entrée métamorphosés.

5.5.1 Factorisation et compression d'un automate minimal "FCM"

Cette première méthode prend en entrée un automate minimisé, calcule ses séries en utilisant l'algorithme $RechercheSA$, puis le factorise en utilisant l'algorithme "Choix et factorisation des séries" (algorithme 17) avant de le compresser. Le résultat de la compression correspond toujours à un automate, il est réduit et muni d'un alphabet étendu qui répertorie toutes les sous-structures factorisées dans l'automate.

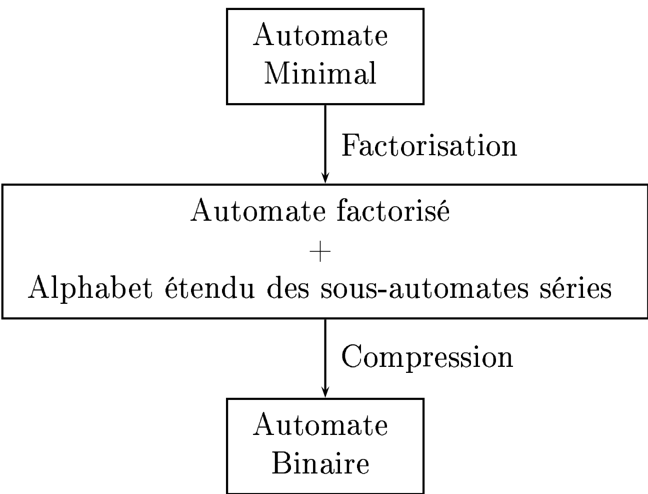


FIG. 5.4 – Diagramme de factorisation d'un automate minimal

Résultats expérimentaux

Plusieurs séquences de tests ont été réalisées sur les automates minimaux représentant les dictionnaires des catégories 1 et 2 (tableau 4.6, page 97). Les hypothèses envisagées portent sur la taille de l'alphabet étant donné son impact sur la fonction gain. Ainsi, on a appliqué notre méthode de factorisation en fixant l'extension de l'alphabet à 7 bits, puis 8 bits, ensuite 9 jusqu'à atteindre 14 bits pour pouvoir traiter tous les cas possibles de factorisation de ces automates.

Le tableau 5.4 présente un résumé des résultats de la factorisation des automates. À titre d'exemple, la taille initiale de l'automate minimal représentant le DELAF français est égale à 2 178 Ko, après sa compression, sa taille diminue à 563 Ko, cependant cette compression est dénuée de factorisation. La factorisation de 128 sous-structures réduit la taille de l'automate à 529 Ko, cette taille est la meilleure solution obtenue par l'algorithme. La factorisation de 256 sous-structures augmente la taille mémoire nécessaire pour le stockage de l'automate à 533 Ko, mais reste quand même inférieure à sa taille après un codage simple (voir tableau 5.1, page 129). La suite des hypothèses faisant varier la taille de l'alphabet n'améliore pas la solution.

L'unité de mesure de la table 5.4 est le "kilo octect" et les solutions optimales y sont indiquées en rouge.

Tous les résultats obtenus sont meilleurs comparés aux valeurs de la table 5.1. En effet, dès la première étape où la taille de l'alphabet est fixée à $Taille_{alphabet} = 128$, on obtient une amélioration du taux de compression de $0,2\%$ à $2,2\%$. Les tableaux détaillant les résultats de chaque implantation sont présentés en annexe B, page 187.
Le tableau 5.4 indique que la majorité des séries choisies par la fonction gain sont de longueur 2, en raison du nombre très élevé de leurs occurrences. Les différents tests prouvent que le nombre de factorisation optimum est atteint très rapidement pour la plupart des automates. En effet, dès le départ, en fixant la taille de l'alphabet à 7 bits, on obtient les meilleurs résultats pour le DELAF français, le DELAF serbe, le web français, le web bulgare, le web hongrois et le web portugais. Ensuite, en ajoutant 1 bit à l'alphabet, on obtient les meilleurs résultats pour le DELAF anglais et le DELAF allemand. L'automate des polylexicaux anglais atteint son meilleur résultat de compression lorsque l'alphabet est fixé à 10 bits. Enfin, celui de l'automate des noms des villes françaises est atteint à un alphabet borné à 11 bits.

L'évolution de certains automates lors de la factorisation est assez similaire :
- L'automate des villes françaises et de l'automate des polylexicaux anglais.
- L'automate du web bulgare et du web hongrois.
- Les automates DELAF serbe, DELAF allemand et web portugais.

Leurs taux de compression, la proportion du nombre de transitions éliminé (voir en annexe B) indique que l'évolution est assez semblable.
Le résultat de cette étude correspond partiellement aux résultats du chapitre 4. En effet, on retrouve certains regroupement des automates présents déjà au chapitre 4.

Automate minimal	Taille (ko) initiale	Taille (ko) après codage simple	Taille après factorisation et compression (ko) index borné : $Taille_{alphabet}$ fixée aux valeurs							
			128	256	512	1024	2048	4096	8192	16384
Catégorie 1										
DELAF Fr	2 178,22	563	**529,44**	533,32	543,36	556,97	572,74	591,38	610,97	
DELAF En	3 081,12	801	738,96	**735**	743,03	757,01	774,32	795,65	821,54	
DELAF Sr	2 340,47	591	**585,33**	598,16	614,71	632,96	653,37	675,08		
DELAF De	4 133,67	1 105	1040,56	**1036,72**	1 049,77	1 069,7	1 093,05	1 120,27	1 155,7	
Villes Fr	1 108,25	291	240,97	231,22	228,62	228,58	**222,5**	227,48		
Poly En	8 963,46	2 451	1 919,01	1 869,04	1 773,72	**1 761,14**	1 762,15	1 779,12	1 809,83	1 860,69
Catégorie 2										
Web Fr	3 560,97	946	**935,38**	954,48	979,52	1 007,3	1 036,56	1 069,41	1 103,25	
Web Bg	2 503,96	638	**610,06**	618,69	632,19	647,62	664,88	685,95	708,54	
Web Hu	3 235,67	858	**788,35**	800,85	818,4	839,2	862,48	889,36	918,43	
Web Pt	6 510,09	1 776	**1 653,47**	1 671,96	1 703,4	1 742,01	1 783,4	1 828,42	1 881,55	

TAB. 5.4 – Factorisation et compression d'un automate minimal

5.5.2 Factorisation et compression d'un automate minimal des mots inversés "$FCM_inverse$"

Le même processus a été appliqué sur des automates construits à partir des mots inversés des dictionnaires. Cette méthode bouleverse l'automate initial car les préfixes récupèrent la place des suffixes et les suffixes adoptent la position des préfixes. L'automate obtenu est toujours déterministe et minimal.
On remarque que la taille de ces automates est plus grande, en effet, l'automate du DELAF français contient 20% de transitions supplémentaires, le DELAF anglais 25%, le DELAF serbe 79%, le DELAF allemand 48% et 1% pour l'automate des polylexicaux anglais. En moyenne les automates des dictionnaires de la catégorie 2 contiennent 5% de transitions supplémentaires. Le seul qui échape à cet accroissement du nombre de transitions est l'automate des villes française, cet automate possède 117 transitions en moins par rapport à l'automate représentant les mots non inversés. On constate alors que lorsqu'on inverse les mots de ce dictionnaire la minimisation réduit encore plus l'automate, ce qui s'explique par l'utilisation importante de préfixe (Saint-, Pont-, Port-, etc) dans la construction des noms de ville.

Résultats expérimentaux

Le tableau 5.5 présente un résumé des tests effectués sur les automates minimaux des mots inversés. À titre d'exemple, la taille initiale de l'automate représentant le DELAF français est égale à 2 620 Ko et dès la première étape de factorisation, où la taille de l'alphabet est fixée à $Taille_{alphabet} = 128$, on réduit la taille de l'automate à 651 Ko. C'est d'ailleurs le meilleur résultat d'économie en mémoire de stockage, car on remarque qu'une extension plus importante de l'alphabet ne permet pas d'éliminer assez de transitions dans l'automate pour réduire sa taille. La factorisation s'interrompt lorsque la taille de l'alphabet atteint 13 bits, cette information indique que le calcul du gain a atteint ses limites pour l'automate du DELAF français, car il ne décèle plus aucune série intéressante à factoriser, le calcul du gain est négatif ou nul.

Les résultats de la factorisation et compression démontrent, qu'en général, la méthode est moins efficace lorsqu'elle est appliquée aux automates représentant les mots inversés des dictionnaire (résultats présentés en annexe B). En effet, même si la taille des automates diminue elle reste supérieure comparée aux résultats du tableau 5.1. Cependant, deux automates se distinguent, l'automate représentant les villes françaises qui observe une réduction de taille passant à 217 Ko au lieu de 222 Ko et l'automate représentant les polylexicaux anglais qui diminue sa taille jusqu'à atteindre 1 709 Ko au lieu des 1 761 Ko obtenus plus haut.

Les tableaux détaillant les résultats de chaque implantation sont présentés en annexe B.

Automate minimal	Taille (ko) initiale	Taille après factorisation et compression (ko) index borné : $Taille_{alphabet}$ fixée aux valeurs							
		128	256	512	1024	2048	4096	8192	16384
Catégorie 1									
DELAF Fr	2 620,85	**651,57**	663,73	681,36	701,85	724,54	748,64		
DELAF En	3 879,99	**994,02**	1 003,28	1 024,66	1 050,94	1079,7	1112,99	1148,23	
DELAF Sr	4 292,5	**1 123,17**	1 158,04	1 196,16	1 236,21	1 277,19			
DELAF De	6 176,49	**1 579,49**	1 601,99	1 637,93	1 680,97	1 727,92	1 780,05	1 836,16	
Villes Fr	1 109,33	238,81	228,95	226,02	**217,98**	220,07	225		
Poly En	9045,26	1928,13	1790,31	1742,56	1717,13	**1709,62**	1720,5	1747,29	1795,04
Catégorie 2									
Web Fr	3726,56	**969,79**	992,36	1020,06	1050,1	1082,06	1116,3	1151,78	
Web Bg	2777,86	**677,89**	690,22	708,06	727,62	748,98	774,04	799,59	
Web Hu	3490,55	874,25	**854,75**	873,89	896,73	921,86	951,05	982,14	
Web Pt	6931	1816,07	**1774,57**	1810,92	1854,11	1900,08	1949,81	2008,25	

Tab. 5.5 – Factorisation et compression d'un automate minimal de mots inversés

5.5.3 Factorisation et compression d'un automate non minimal "FCNM"

Cette méthode prend en entrée un automate non minimal (déterministe). Elle calcule ses séries et le factorise en utilisant le $DAWG$, puis elle minimise l'automate obtenu. Cet automate minimisé sera ensuite compressé. À cet automate est associé un alphabet qui répertorie toutes les sous-structures factorisées dans l'automate.

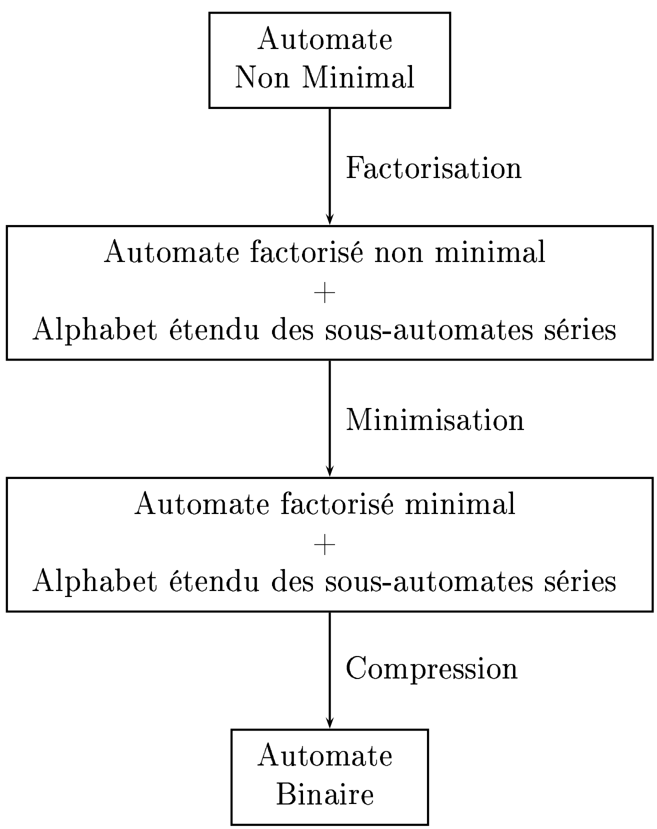


FIG. 5.5 – Diagramme de factorisation d'un automate non minimal

Adapter la fonction gain

Le programme implémenté commence par calculer les séries de l'automate non minimal, ensuite, en utilisant le $DAWG$ et la fonction gain, il sélectionne celles qui sont intéressantes à factoriser. La fonction gain utilisée dans la méthode FCM est basée sur des constantes dépendant du fichier récupéré en sortie (automate binaire). Une de ces constantes est invalide pour la factorisation courante $Taille_{adresse}$. En effet, Normalement, le nombre total de transitions de cet automate est utilisé par la fonction Δ pour décrire la variable $Taille_{adresse}$. Cependant cet automate n'est pas minimal et la fonction gain a été définie dans le cas où l'automate est minimal. Aussi, on ne peut pas associer à $Taille_{adresse}$ la valeur représentant le nombre total de transitions de l'automate non minimal car elle serait erronée. Étant donné que l'automate final qu'on

souhaite récupérer après la factorisation sera minimal, il se rapprochera forcément des résultats obtenus par la méthode FCM, la solution adoptée est donc d'associer les mêmes valeurs calculées pour FCM à $Taille_{adresse}$. Ce choix permettra de sélectionner les séries selon les mêmes critères.
Rappelons qu'une fois factorisé, l'automate non minimal est minimisé en prenant soin de sauvegarder l'alphabet étendu des sous-structures factorisées.

Exemple 5.4
La figure 5.6 illustre cette nouvelle méthode de factorisation. Étant donné un automate A non minimal représentant la liste des mots "aabc", "acabc", "bc", "bab", "caba" et "ccabc", la factorisation des séries en utilisant la fonction gain permet de réduire la série abc, elle sera placée en cascade dans l'index juste après l'alphabet initial. Une fois factorisé, l'automate qui est toujours non minimal est minimisé.

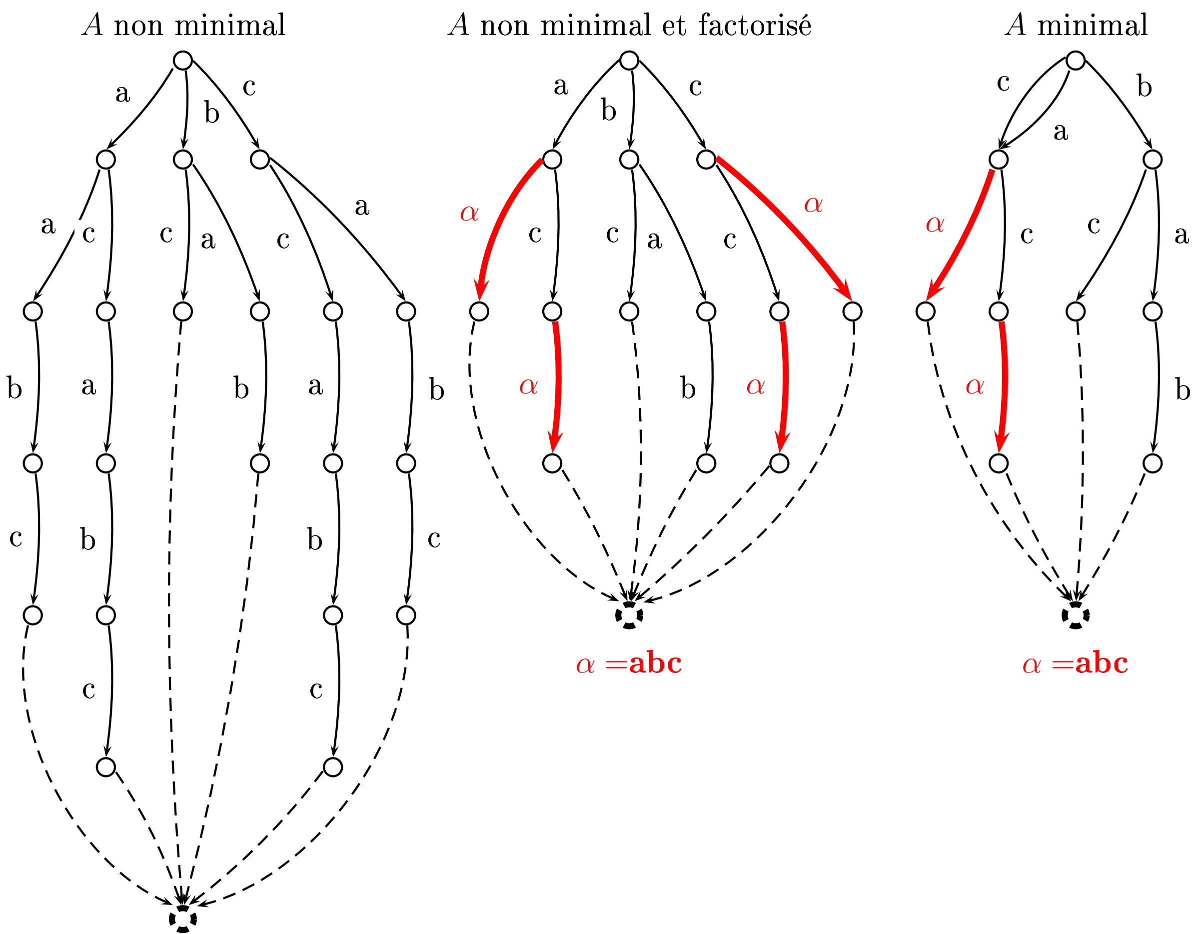


FIG. 5.6 – Factorisation d'un automate non minimal

Résultats expérimentaux

Deux séries de tests ont été réalisées, la première fixe une taille maximale de l'alphabet à 7 bits ($Taille_{alphabet} = 128$), la seconde à 8 bits ($Taille_{alphabet} = 256$).

L'unité de mesure de la table 5.6 est le "kilo octect" et les solutions optimales y sont indiquées en rouge.

Le tableau 5.6 représente les résultats obtenus suite à la factorisation, minimisation et compression des automates non minimaux, correspondant aux dictionnaires de catégorie 1 et 2. À titre d'exemple, la taille de l'automate non minimal représentant le DELAF français est 18 723 Ko. Dans le cas où l'extension maximale de l'alphabet est fixée à 7 bits ($Taille_{alphabet} = 128$), l'étape de factorisation réduit la taille de l'automate non minimal à 12 869 Ko, sa minimisation suivie de sa compression diminuent cette taille à 538 Ko. Ce résultat explique le taux très élevé de compression 97%. Au total 57 séries ont été factorisées (initialement, l'index contient 71 caractères représentant l'alphabet de départ). La moyenne des longueurs des séries factorisées est $2,18$ transitions et la plus longue série à avoir été factorisée se compose de 4 transitions.

En comparant aux résultats de la méthode FCM et dans le cas où l'extension maximale de l'alphabet est fixée à 7 bits, on remarque que les tailles des automates des villes françaises, web français, web bulgare, web hongrois et web portugais sont meilleures (plus petites). En effet, les séries sélectionnées permettent d'éliminer plus de transitions dans l'automate sans desservir la minimisation.

Un point très important à signaler est le gain de place obtenu sur les dictionnaires de la deuxième catégorie. En comparaison des tailles minimales calculées selon la méthode FCM, les tailles mesurées des quatre dictionnaires en utilisant cette nouvelle méthode $FNCM$ de factorisation ont légèrement évolué à la baisse. Le web français est réduit à 934 Ko au lieu de 935 Ko avec la méthode FCM, le web bulgare rapetisse à 608 Ko au lieu de 610 Ko, le web hongrois atteint 775 Ko au lieu de 788 Ko et enfin, le web portugais diminue à 1 652 Ko au lieu de 1 653 Ko.

Cette méthode ($FNCM$) confirme que factoriser, avant de minimiser un automate, peut être plus efficace et permet de réduire des sous-structures qui ne sont pas détectées sur un automate minimal.

À l'étape suivante, on a augmenté la taille maximale de l'alphabet à 8 bits et on a constaté que deux automates continuent de progresser et diminuent leur taille, il s'agit de l'automate des villes françaises et l'automate des polylexicaux anglais. Cependant elles sont toujours supérieures à celles calculées en utilisant la méthode FCM.

Automate non minimal	Taille initiale	Taille après factorisation	Taille après factorisation et minimisation et compression	Taux compression	Caractéristique de la factorisation		
					Séries factorisées	Longueur moyenne	Longueur maximale
$Taille_{alphabet} = 128$							
DELAF Fr	18 723,87	12 869,18	538,24	97,13%	57	2,18	4
DELAF En	9 991,52	7 226,51	740,64	92,59%	54	2,15	4
DELAF Sr	26 753,38	21 711,77	588,52	97,80%	76	2,05	3
DELAF De	44 836,96	38 195,69	1 050,93	97,66%	38	2,13	3
Villes Fr	2 559,35	1 747,3	240,72	90,59%	53	2,28	5
Poly En	30 230,98	17 474,43	1938,41	93,59%	97	2,18	6
Web Fr	7 454,46	5 572,12	934,99	87,46%	85	2,01	3
Web Bg	5 558,38	3 997,17	608,69	89,05%	86	2,03	5
Web Hu	7 428,11	5 240,37	775,78	89,56%	85	2,01	3
Web Pt	15 359,34	10855,66	1 652,05	89,24%	79	2	2
$Taille_{alphabet} = 256$							
DELAF Fr		1 2439,72	541,16	97,11%	152	2,12	5
DELAF En		6 687,13	740,95	92,58%	149	2,1	4
DELAF Sr		2 1528,14	601,98	97,75%	171	2,06	4
DELAF De		3 6990,46	1 048,95	97,66%	133	2,11	5
Villes Fr		1 497,47	233,35	90,88%	148	2,18	5
Poly En		1 5325,96	1 913,46	93,67%	192	2,29	6
Web Fr		5 237,29	957,45	87,16%	180	2,01	3
Web Bg		3 767,24	619,54	88,85%	181	2,03	5
Web Hu		4 882,01	792,66	89,33%	180	2,06	8
Web Pt		1 0145,47	1682,73	89,04%	174	2,2	33

TAB. 5.6 – Factorisation, minimisation et compression d'un automate non minimal

5.5.4 Factorisation et compression d'un automate non minimal des mots inversés "$FNCM_{inverse}$"

L'idée est la même que celle décrite en section 5.5.2 : inverser les mots du dictionnaire avant de générer l'automate. Dans notre cas, l'automate sera non minimal et déterministe. On observe que la taille des automates générés à partir des mots inversés est plus grande que celle des automates générés directement à partir des mots du dictionnaire. Le nombre d'états et le nombre de transitions sont plus élevés. Ce constat précise que ce sont les suffixes des mots des dictionnaires, qui, lorsqu'ils sont transformés en préfixes, génèrent plus de transitions, ce qui indique qu'ils sont très hétérogènes. On remarque également que la factorisation et la compression ne sont pas aussi efficaces lorsqu'elles sont appliquées sur des automates générés à partir des mots inversés du dictionnaire. En effet, un seul automate profite de cette méthode, il s'agit de l'automate des polylexicaux anglais, sa taille est réduite à 1 905 Ko (1 913 Ko auparavant).

Le tableau 5.7 présente les résultats produits après la factorisation, minimisation et compression des automates non minimaux, correspondant aux dictionnaires des mots inverses de catégorie 1 et 2. Initialement, la taille de l'automate non minimal représentant le DELAF français des mots inverses est 24 735 Ko. Lorsque l'extension maximale de l'alphabet est limitée à 7 bits, le processus de factorisation diminue la taille de l'automate non minimal à 17 951 Ko. L'étape de minimisation suivie de la compression le réduisent à 648 Ko. Le taux de compression correspond ainsi à 97%. Au total 57 séries sont factorisées, de longueur moyenne $2,05$ et la plus longue série à avoir été factorisée se compose de 4 transitions.

Lorsque la taille de l'alphabet est limitée à 7 bits, l'analyse des résultats ne révèle aucune amélioration en comparaison, des résultats de FCM et des résultats de $FNCM$. À l'étape suivante, on augmente la taille maximale de l'alphabet à 8 bits et on constate que seul l'automate représentant les polylexicaux anglais perd du poids. Cependant ce n'est pas suffisant pour concurrencer les résultats obtenus par la méthode FCM.
Ces résultats démontrent que la méthode $FNCM$ est moins performante lorsqu'elle est appliquée aux automates non minimaux, représentant les mots inversés des dictionnaires parce que les séries factorisées n'éliminent pas assez de transitions pour obtenir un gain global intéressant.

Automate non minimal	Taille initiale	Taille après factorisation	Taille après factorisation et minimisation et compression	Taux compression	Caractéristique de la factorisation		
					Séries factorisées	Longueur moyenne	Longueur maximale
$Taille_{alphabet} = 128$							
DELAF Fr	24735,06	17951,2	648,79	97,38%	57	2,05	4
DELAF En	11828,69	8954,54	989,29	91,64%	54	2,06	3
DELAF Sr	46571,44	33370,02	1099,02	97,64%	76	2,01	3
DELAF De	66055,01	54450,33	1578,46	97,61%	38	2,21	5
Villes Fr	2686,03	1773,01	251,78	90,63%	53	2,6	6
Poly En	33045,55	17125,7	1979,18	94,01%	97	2,52	8
Web Fr	7892,49	5991,66	968,34	87,73%	85	2,01	3
Web Bg	6331,37	4441,56	673,72	89,36%	86	2,06	5
Web Hu	8943,03	6253,55	867,57	90,30%	85	2,01	3
Web Pt	16986,38	11911,99	1706,07	89,96%	79	2	2
$Taille_{alphabet} = 256$							
DELAF Fr	24735,06	16047,97	660,74	97,33%	152	2,05	4
DELAF En	11828,69	7890,53	997,14	91,57%	149	2,05	4
DELAF Sr	46571,44	31073,6	1132,25	97,57%	171	2,01	3
DELAF De	66055,01	50945,93	1599,56	97,58%	133	2,11	5
Villes Fr	2686,03	1510,12	234,42	91,27%	148	2,62	8
Poly En	33045,55	14462,08	1905,49	94,23%	192	2,55	8
Web Fr	7892,49	5538,49	992,53	87,42%	180	2,02	3
Web Bg	6331,37	4110,44	688,53	89,13%	181	2,06	5
Web Hu	8943,03	5652,83	849,87	90,50%	180	2,03	4
Web Pt	16986,38	10874,76	1738,78	89,76%	174	2,02	3

TAB. 5.7 – Factorisation, minimisation et compression d'un automate non minimal de mots inversés

5.5.5 Factorisations et compression des mots d'un dictionnaire "$FCDic$"

Cette méthode s'applique directement à un dictionnaire et non pas à son automate. Elle génère le $DAWG$ des mots et allonge l'alphabet pour effectuer les factorisations directement sur les mots. Ensuite, elle crée l'automate minimal associé à cette liste des nouveaux mots avant de le compresser.

Adapter la fonction gain

Étant donné que la factorisation est réalisée sur une liste de mots, nous ne pouvons plus émettre d'hypothèse sur la taille des automates et donc des codages. La fonction gain utilisée précédemment perd un peu de sens pour cette méthode. Ainsi, nous utilisons une fonction simplifiée, prenant en compte uniquement les caractéristiques des séries (dans notre cas des mots), à savoir la longueur et la fréquence (obtenues par le $DAWG$).

$$\Delta = (F - 1) * L \tag{5.8}$$

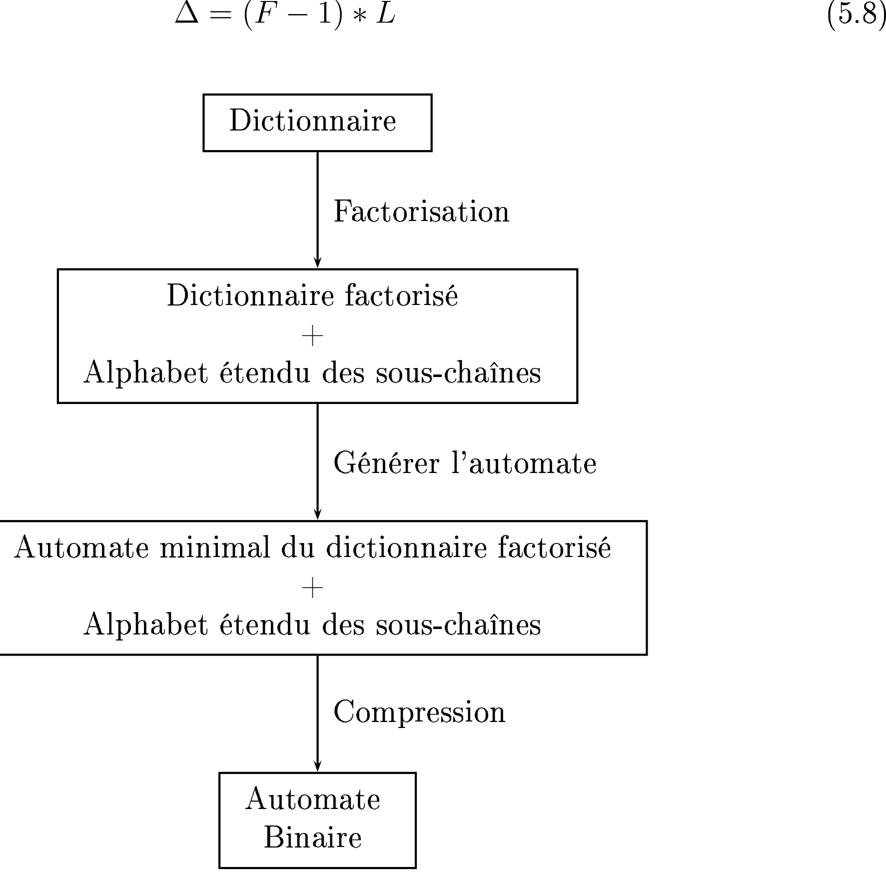


FIG. 5.7 – Diagramme de la factorisation des mots d'un dictionnaire

Des tests ont été réalisés sur l'ensemble des dictionnaires, cependant certains résultats n'ont pas pu être obtenus, en raison du temps de calcul du $DAWG$. En effet, pour sélectionner les sous-mots intéressants à factoriser, cette méthode crée le $DAWG$ de

tout le dictionnaire, cependant le nombre total de mots est très élevé pour certains. Par exemple, rappelons que le DELAF Allemand contient 3 713 121 mots.

La table 5.8 présente les résultats obtenus. On remarque que cette factorisation profite à certains dictionnaires. En effet, la taille de l'automate représentant le dictionnaire web hongrois a baissé jusqu'à atteindre 757 Ko, alors que le meilleur résultat obtenu jusqu'à cette étape était 775 Ko ($FNCM$).
Plus important encore, l'automate des villes françaises qui passe de 222 Ko obtenus avec FCM à 67 Ko. L'examen des sous-mots factorisés explique ce résultat par la présence de nombreuses sous-structures communes dans les noms de villes françaises telles que, Saint, Bourg, etc. Ainsi, appliquer la factorisation sur les mots du dictionnaire est plus efficace que sur l'automate.
La factorisation des autres dictionnaires ne permet pas d'améliorer les résultats, mais ils restent assez proches de ceux obtenus avec les méthodes précédentes.

Dictionnaire	Taille dictionnaire	Taille dictionnaire après factorisation	Taille automate minimal	Taille automate compressé	Taux de compression
DELAF En	3 335,06	2 410,27	2 897,53	776,28	76,72%
Villes Fr	481,7	88,34	260,04	67,24	86,04%
Web Fr	2 154,88	1 567,71	2 289,75	939,7	56,39%
Web Bg	1 642,3	1 162,98	3 317,91	621,26	62,17%
Web Hu	2 068,07	1 430,27	2 767,26	757,55	63,37%
Web PT	4 268	3 414	4 600	1 600	62,5%

TAB. 5.8 – Factorisation des mot d'un dictionnaire

5.6 Discussion et Conclusion

Avant de clore ce chapitre une discussion s'impose sur la factorisation. En effet, les résultats obtenus correspondant aux différentes méthodes testées démontrent (ou confirment) l'importance du choix des sous-structures à factoriser. Ce choix dépend du nombre de transitions que contiennent ces sous-structures et de leur fréquence dans l'automate, mais il dépend aussi de l'impact d'une factorisation sur les suivantes.
Revenons à l'exemple 5.3, page 140, la figure 5.3 permet de visualiser le $DAWG(S_1, S_2, S_3)$ associé aux trois séries détectées dans un automate donné. Les différents scénarios proposés prouvent que l'application d'une factorisation influence la, ou les suivantes.
Rappelons que S_1= *ccdbaab*, S_2= *baab* et S_3= *ccdb* et que les variables x, y, z représentant les fréquences respectives de S_1, S_2 et S_3 : $x = 4$, $y = 1$ et $z = 2$.
La fonction gain associée à chaque état du $DAWG$ permet de calculer le gain de la factorisation de la plus longue série reconnue à cet état, cependant un état peut reconnaître plusieurs séries ou sous-série. L'état 6 reconnaît les sous-séries "baab" et "aab".
Comme l'indique le scénario 4, il serait intéressant d'évaluer le gain assigné à "aab" même si elle n'est pas de longueur maximale à l'état 6 et ne pas se concentrer uniquement sur la série de longueur maximale. On constate ainsi que le gain obtenu au $4^{ème}$ scénario est égal à celui obtenu au 1er, même si les séries sélectionnées au premier scénario sont les plus longues associées aux états 6 et 10.

Il est aussi possible de factoriser uniquement une partie des occurrences d'une série dans l'automate pour anticiper l'impact des combinaisons de factorisation.

Exemple 5.5
Étant données trois séries "$S_1 = acdacd$", "$S_2 = cdcd$" et "$S_3 = acda$".
La sous-série "*cd*" apparaît, au total, cinq fois. Deux fois dans S_1, deux fois dans S_2 et une seule fois S_3.
Il serait peut être plus intéressant de ne pas la factoriser directement aux cinq positions et envisager de la réduire uniquement sa position finale dans S_1 et ses deux position dans S_2. Ce choix permettrait de réduire par la suite la sous-série *acda*.

Cette remarque ouvre la réflexion sur la possibilité de sélectionner différentes factorisations lors d'une même étape.

Pour parachever les résultats des factorisations obtenues sur les différentes méthodes, le tableau 5.9 présente un résumé des meilleurs résultats. On constate qu'en général, les factorisations les plus efficaces des automates se réalisent en utilisant un alphabet assez court, de 7 à 8 bits. Toutefois, on remarque qu'il existe un intrus dans ce tableau, il s'agit de l'automate des polylexicaux anglais qui atteint son meilleur taux de compression lorsque la taille maximale de l'alphabet est fixée à 1 024. Ce dictionnaire est composé d'expressions anglaises, à l'instar du dictionnaire des villes française, de nombreux mots y sont récurrents. On peut donc émettre l'hypothèse d'améliorer son résultat en utilisant $FCDic$ (factorisation des mots avant de générer l'automate).
Pour résumer, la méthode FCM est efficace lorsqu'elle s'applique aux automates du

DELAF. La méthode $FCM_{inverse}$ est performante sur les automates des villes françaises et le polylexicaux anglais, toutefois, inverser les mots pour les autres automates n'est pas profitable et la méthode $FNCM$ est efficace lorsqu'elle s'applique aux automates du Web.

On constate aussi que les méthodes FCM et $FNCM$ proposent des résultats intéressants. Il serait donc justifié d'étudier la possibilité de les combiner. En déminimisant une partie de l'automate minimal avant la factorisation on pourrait peut être y faire apparaître d'autres sous-structure plus intéressantes à réduire.

Automate	Meilleure compression	Méthode utilisée	$Taille_{alphabet}$
Catégorie 1			
DELAF Fr	529 Ko (30%)	FCM	128
DELAF En	735 Ko	FCM	256
DELAF Sr	585 Ko	FCM	128
DELAF De	1036 Ko	FCM	256
Villes Fr	222 Ko	FCM (cf. $FCDic$)	2 048
Poly En	1 709 ko	$FCM_{inverse}$	2 048
Catégorie 2			
Web Fr	934.99 Ko	$FNCM$	128
Web Bg	608.69 Ko	$FNCM$	128
Web Hu	757 Ko	FCM (cf. $FCDic$)	128
Web Pt	1652.05 Ko	$FNCM$(cf. $FCDic$)	128

TAB. 5.9 – Récapitulatif des meilleurs résultats des différentes méthodes de factorisation

Dans ce chapitre nous avons présenté une méthode itérative pour factoriser et comprimer les automates acycliques à nombre finis d'états. Étant donné un automate A, cette méthode utilise la recherche des sous-automates présentée au chapitre 2 pour localiser les sous-structures de A et ensuite calculer, à chaque étape, les gains d'un ensemble de factorisations possibles en utilisant un $DAWG$.
Nous avons ainsi proposé un algorithme glouton qui détermine le meilleur choix local pour tenter obtenir la meilleure factorisation globale de A. Il localise ainsi, au fur et à mesure, la sous-structure la plus intéressante à factoriser, à partir d'un $DAWG$ qui indexe toutes les sous-structures de A. Une fois choisie, cette sous-structure est remplacée dans A par une extension de son alphabet. Ce processus est itératif et il s'achève lorsqu'il n'existe plus de sous-structure intéressante à factoriser ou lorsque la taille maximale de l'alphabet est atteinte.

Les mises à jours de l'automate factorisé A sont possible, il demeure ainsi toujours utilisable.

- **Suppression d'un mot $w \in L(A)$:**
  Le parcours de w dans A passe par les états q_i, q_1, $\cdots$,q_n, q_f. Les états q_j tel que $j = 1$ à n sont appelés les états interne du chemin w.
  Le déterminisme de l'automate factorisé et compressé permet la suppression de w. Trois cas sont possibles :
  Cas 1 : Les états internes de w sont tous non divergents et non convergents, alors la suppression de w consiste à les élimine tous de A,
  Cas 2 : Aucun des états q_j n'est convergent mais certains sont divergents, la suppression de w consiste alors à éliminer la dernière portion du w partant du dernier état divergent q_i.
  Cas 3 : Il existe des états q_j convergents (conséquence de la minimisation de l'automate) alors il nécessaire de déminimiser complètement le chemin de w dans A avant de supprimer w selon le cas 1 ou 2.

- **Insertion d'un mot $w \notin L(A)$:**
  L'insertion d'un mot w dans A doit conserver le déterminisme de A comme suit :
  Cas 1 : Si le premier caractère de w n'existe pas à partir de q_j (ni dans son alphabet initial, ni dans son alphabet étendu) alors un chemin composé d'état non divergent et non convergent représentant w sera ajouter à A (l'alphabet étendu peuvent être utilisé).
  Cas 2 : Si une portion de w existe à partir de q_j, alors cette insertion s'effectue en trois temps : 1) Déminimiser et recomposer cette portion dans A par des transitions étiquetées uniquement par l'alphabet initial de A. 2) Franchir la portion de w qui existe dans A à partir de q_j . 3) En admettant que cette portion s'arrête à un état p dans A il suffit alors de compléter A par une succession d'état non divergent et non convergent reliant p à q_f représentant les caractères suivants de w

Remarque 5.2
Ajouter ou supprimer de l'information à l'automate influence la fréquence des caractères et donc influence le choix des factorisations.

Parmi les travaux existants traitant le problème de la compression d'automates représentant des dictionnaires de langues présentés au chapitre 1, on peut citer Ristov et Laporte [Ristov et Laporte, 1999].

Rappelons que pour représenter et stocker les données, Ristov et Laporte n'utilisent pas les automates finis acycliques mais des arbres lexicographiques. Une fois généré, un arbre est transformé en une liste chaînée avant d'être compressé suivant la méthode de Ziv et Lempel. La recherche des structures similaires dans l'arbre lexicographique s'adapte aux données et utilise des arbres de suffixes ou des tableaux de suffixes pour y détecter les répétitions. La compression du DELAF français réduit l'arbre à 385 Ko.
En comparaison au travail proposé dans ce chapitre, le résultat final obtenu par Ristov et Laporte est meilleur, cependant il ne correspond plus à un automate. Notre structure de données n'est pas modifiée, ni en entrée, ni en sortie : on utilise des automates finis acycliques pour représenter les dictionnaires.

Exemple 5.6
La figure 5.8 se compose d'une part, de l'automate représentant les quatre mots "abcde", "abcfg", "abhcde" et "abhcfi", et d'autre part, de l'arbre lexicographique représentant la même liste de mots.
Cet exemple illustre la différence entre la méthode de compression présentée dans [Ristov et Laporte, 1999] et la notre qui s'appuie sur la recherche de sous-automates.
Les partie indiquées en gras sur l'arbre lexicographique sont détectées pour être factorisées selon la méthode de Ristov et Laporte alors que la représentation par un automate ne permet pas de retrouver cette redondance car aucun sous-automate n'apparaît.

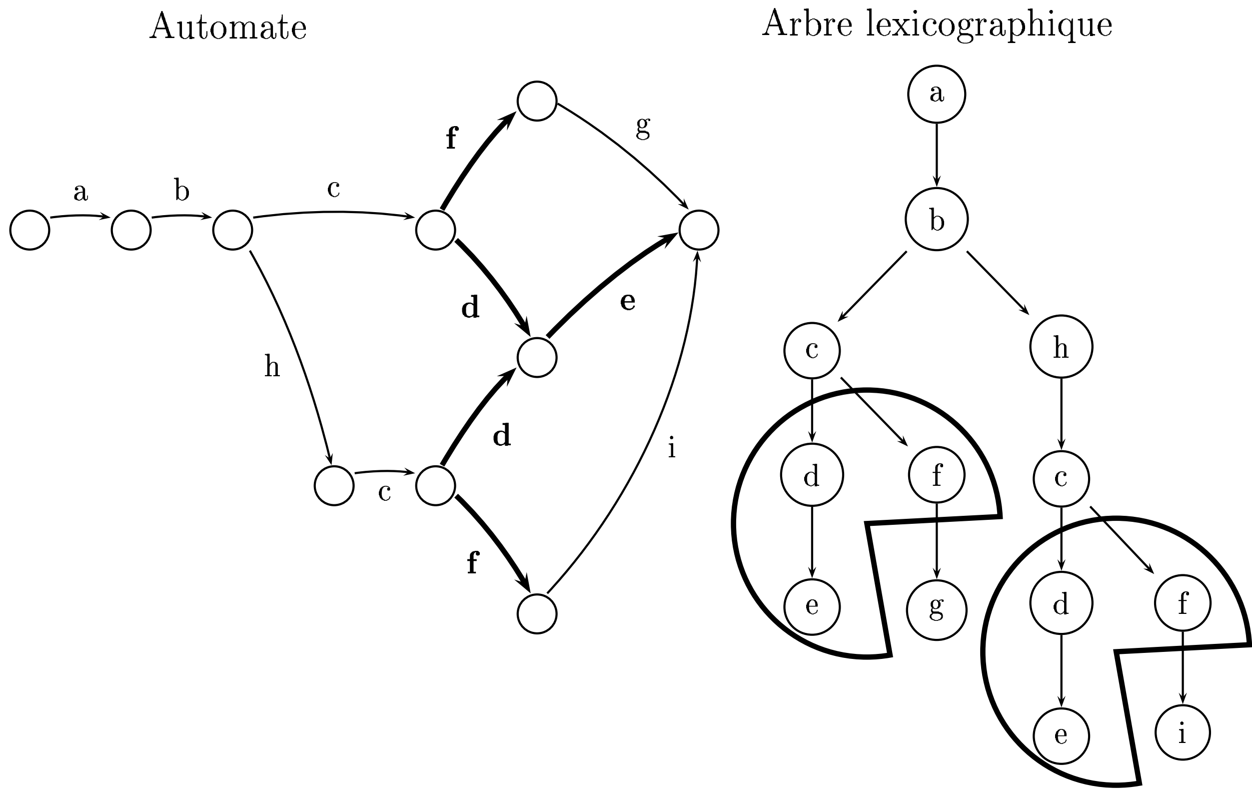


FIG. 5.8 – Représentation d'un automate vs arbre lexicographique

Conclusion

Nos travaux de recherche ont porté sur l'étude de la structure interne des automates à nombre finis d'états représentant des dictionnaires électroniques, de différentes langues. Cette problématique se situe entre deux domaines qui sont la théorie des automates et le traitement automatique des langues.

Au niveau théorie des automates, nous avons présenté une méthode qui permet l'étude de la structure interne d'un automate acyclique donné A à partir de la recherche de tous ses sous-automates. Un sous-automate est défini comme un automate isolé possible à extraire de A. Notre méthode considère trois types de sous-automates : i) un sous-automate parallèle, (ii) un sous-automate série et (iii) un sous-automate minimal qui n'est ni parallèle, ni série.
Elle est basée sur un algorithme itératif qui calcule et factorise par une seule transition les parallèles et les séries jusqu'à ce que l'automate en soit totalement dénué, puis qui calcule et factorise par une seule transition les sous-automates minimaux, et ainsi de suite, jusqu'à ce que l'automate donné soit réduit à deux états reliés par une seule transition.
La complexité théorique de recherche des sous-automates séries et parallèles est de $O(m^2)$, où m représente le nombre de transitions dans A et dans le pire des cas, la complexité de l'algorithme de recherche des sous-automates minimaux qui ne sont ni parallèles ni séries est de $O(n^3)$ où n représente le nombre d'états de A.

Au niveau du traitement automatique des langues, nous avons appliqué notre méthode à plusieurs automates déterministes, acycliques et minimaux représentant des dictionnaires électroniques de différentes langues ; les résultats obtenus montrent que ces automates contiennent un nombre considérable de séries, de parallèles et dans une moindre mesure, des sous-automates issus de la réduction des parallèles et séries. De plus, certains de ces sous-automates possèdent plusieurs niveaux d'imbrications.
A l'instar de la construction automatique d'un index à partir d'un document texte, qui recense toute occurrence des mots de ce document, grâce à l'algorithme de recherche des sous-automates, la construction automatique d'un index à partir d'un automate énumère la liste de ses sous-automates qui représente son index. Cet index est partiel car il est construit en fonction des sous-automates séries et parallèles.

Cette étude a révélé plusieurs applications possibles telles que l'indexation et la compression d'automates. Pour alléger l'automate (déjà minimal) sans modifier le langage reconnu, nous avons proposé un algorithme glouton qui calcule et compare les gains de la factorisation de chaque sous-structure en utilisant un "Directed acyclic word graphe", puis sélectionne celle qui maximise ce gain pour la factorisation. Cet algorithme a aussi été mis en oeuvre pour étudier plusieurs compressions possibles :

compression d'automates minimisés, compression d'automates non minimisés et compression de dictionnaire. De plus, les trois méthodes ont aussi été testées en inversant les mots du dictionnaire.

Les perspectives de nos travaux sont multiples : d'un point de vue théorique, il serait intéressant d'adapter l'algorithme de recherche des sous-automates aux automates cycliques et élargir ainsi nos définitions. D'un point de vue applicatif, actuellement, la recherche de sous-structures intéressantes à factoriser s'effectue grâce à la fonction gain associée à chaque état du $DAWG$; cette fonction dépend de deux paramètres : la fréquence et la longueur de la sous-structure dans l'automate initial. Une amélioration du calcul de ce gain pourrait aussi être envisagée dans de futurs travaux.
L'objectif de nos travaux actuels est d'élaborer des formats de stockage les plus compacts possibles. En effet, un format de stockage largement utilisé consiste à sauvegarder l'automate sous forme d'une liste de transitions. Les transitions sont regroupées par état de départ commun. La représentation d'un état et de ses transitions sortantes peut faire penser à un faisceau lumineux. L'automate est donc vu comme un ensemble de faisceaux interconnectés. L'approche envisagée est de repérer, en s'appuyant sur les $DAWG$, les faisceaux identiques ou partiellement identiques pour limiter par une compression de type LZW les redondances dans le format de stockage. Les résultats dépendront certainement de l'ordre adopté, une étude devra être menée sur ce point.
Dans un deuxième temps, on s'intéresse aussi à la représentation de l'automate en mémoire c'est-à-dire, à modifier la représentation actuelle pour y faire apparaître plus de redondances avant d'appliquer un algorithme de compression de type LZW.

Bibliographie

[Aho *et al.*, 1974] AHO, A., HOPCROFT, J. et ULLMAN, J. (1974). The design and analysis of computer algorithms. *Addison Wesley, Reading, MA.*

[Aho *et al.*, 1988] AHO, A. V., SETHI, R. et ULLMAN, J. D. (1988). *Compilers : Principles, Techniques and Tools.* Addison-Wesley, Reading.

[Babai *et al.*, 1980] BABAI, L., ERDOS, P. et SELKOW, S. M. (1980). Random graph isomorphism. *SIAM Journal on Computing*, pages 628–635.

[Bachelet, 2003] BACHELET, B. (2003). *Modélisation et optimisation de problèmes de synchronisation dans les documents hypermédia.* Thèse de doctorat, Université Blaise Pascal, Clermont-Ferrand.

[Balkenhol *et al.*, 1999] BALKENHOL, B., KURTZ, S. et SHTARKOV, Y. (1999). Modification of the burrows and wheeler data compression algorithm. *In Proceedings of the IEEE Data Compression Conference, Snowbird, Utah, IEEE Computer Society Press*, B. Balkenhol, S. Kurtz, Y. Shtarkov,:188–197.

[Barbu *et al.*, 2004] BARBU, E., HÉROUX, P., ADAM, S. et TRUPIN, E. (2004). Découverte de motifs fréquents : Application à l'analyse de documents graphiques. *Colloque International Francophone sur l'Ecrit et le Document*, pages 143–148.

[Barlaud et Labit, 2002] BARLAUD, M. et LABIT, C. (2002). Compression et codage des images et des vidéos. *Traitement du signal et de l'image, Hermes Science Publications.*

[Bellot, 2000] BELLOT, P. (2000). *Méthodes de classification et de segmentation locales non supervisées pour la recherche documentaire.* Thèse de doctorat, Université d'Avignon.

[Blum, 1996] BLUM, N. (1996). An o(n log n) implementation of the standard method for minimizing n-state finite automata. *Information Processing Letters*, 57(2):65–69.

[Blumer *et al.*, 1984] BLUMER, A., BLUMER, J., EHRENFEUCHT, A., HAUSSLER, D. et MCCONNELL, R. (1984). Building the minimal dfa for the set of all subwords of a word online in linear time. *J. Paredaens (Ed.), Lecture Notes in Computer Science*, 172:109–118.

[Blumer *et al.*, 1985] BLUMER, A., BLUMER, J., HAUSSLER, D., EHRENFEUCHT, A., CHEN, M. et SEIFERAS, J. (1985). The samllest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55.

[Blumer *et al.*, 1987] BLUMER, A., BLUMER, J., HAUSSLER, D. et MCCONNELL, R. (1987). Complete inverted files for efficient text retrieval and analysis. *the Association for Computing Machinery*, 34:578–595.

[Blumer *et al.*, 1989] BLUMER, A., EHRENFEUCHT, A. et HAUSSLER, D. (1989). Average size of suffix trees and dawgs. *Discrete Applied Mathematics*, 24:37–45.

[Bourigault et Charlet, 1999] BOURIGAULT, D. et CHARLET, J. (1999). Construction d'un index thématique de l'ingénierie des connaissances. *Régine Teulier, éditeur, Actes d'IC'99.*

[Brémaud, 1984] BRÉMAUD, P. (1984). Introductions aux probabilités. *Springer-Verlag.*

[Brzozowski, 1962] BRZOZOWSKI, J. A. (1962). Canonical regular expressions and minimal state graphs for definite events. *In Mathematical Theory of Automata*, 12 of MRI Symposia Series:529–561.

[Burrows et Wheeler, 1994] BURROWS, M. et WHEELER, D. (1994). A block- sorting lossless data compression algorithm. Rapport technique, Digital Systems Research Center, Research Report 124.

[Caron, 1997] CARON, P. (1997). *Langages rationnels et automates : de la théorie à la programmation.* Thèse de doctorat, Université de Rouen.

[Caron et Ziadi, 2000] CARON, P. et ZIADI, D. (2000). Characterization of glushkov automata. *Theoret. Comput. Sci*, 233(1-2):75–90.

[Chomsky, 1956] CHOMSKY, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory.*

[Chomsky, 1957] CHOMSKY, N. (1957). Syntactic structures. *Mouton & Co.*

[Ciura et Deorowicz, 2001] CIURA, M. et DEOROWICZ, S. (2001). How to squeeze a lexicon. *Software : Practice and Experience.*

[Cleary et Teahan, 1997] CLEARY, J. G. et TEAHAN, W. J. (1997). Unbounded length contexts for ppm. *Computer Jrnl*, 40(2/3):67–75.

[Courtois et Silberztein, 1990] COURTOIS, B. et SILBERZTEIN, M. (1990). Dictionnaires électroniques du français. *Langues française*, 87:11–22.

[Crochemore et Hancart, 1997] CROCHEMORE, M. et HANCART, C. (1997). Automata for matching patterns. *Handbook Formal Languages, Springer.*

[Crochemore *et al.*, 2001] CROCHEMORE, M., HANCART, C. et LECROQ, T. (2001). *Algorithmique du texte.* Vuibert.

[Crochemore et Rytter, 1994] CROCHEMORE, M. et RYTTER, W. (1994). *Text Algorithms.* Oxford University Press.

[Crochemore et Vérin, 1997a] CROCHEMORE, M. et VÉRIN, R. (1997a). Direct construction of compact directed acyclic word graphs. *Proc. Eighth Annual Symp. CPM 97*, pages 116–129.

[Crochemore et Vérin, 1997b] CROCHEMORE, M. et VÉRIN, R. (1997b). On compact suffix dawg. *J. Mycielski, G. Rozenberg, and A. Salomaa, editors, Mathematical Logic and Theoretical Computer Science, Lecture Notes in Computer Science.*

[Curcio et Chauvin, 1987] CURCIO, M. et CHAUVIN, Y. (1987). Le classement efficace. dictionnaire et méthodes. *Paris : Ed. d'organisation.*

[Daciuk, 2000] DACIUK, J. (2000). Experiments with automata compression. *CIAA 2000*, pages 105–112.

[Daciuk *et al.*, 2000] DACIUK, J., MIHOV, S., WATSON, S. et WATSON, B. (2000). Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16.

[de Mareüil, 2007] de MAREÜIL, P. B. (2007). Linguistique. Rapport technique, LIMSI-CNRS, Master recherche informatique, Parcours sciences cognitives.

[Durand *et al.*, 2005] Durand, P., Mahé, F., Valin, A. et Nicolas, J. (2005). Pyramid diagram : visualizing the organization of repetitive sequences in genome. *Rapport de recherche INRIA*, 5798.

[Faller, 1973] Faller, N. (1973). An adaptive system for data compression. *IEEE Transaction on Information Theory.*

[Ferragina et Grossi, 1999] Ferragina, P. et Grossi, R. (1999). The string b-tree : A new data structure for string search in external memory and its applications. *JACM*, 46(2):236–280.

[Gallager, 1978] Gallager, R. G. (1978). Variations on a theme by huffman. *IEEE Trans. Inform. Theory, Vol. IT-24, pp. 668-674, 1978*, IT-24:668–674.

[Glushkov, 1961] Glushkov, V. (1961). The abstract theory of automata. *Russian Mathematical Surveys*, pages 1–53.

[Gondran et Minoux, 1985] Gondran, M. et Minoux, M. (1985). *Graphes et algorithmes.* Eyrolles.

[Gries, 1973] Gries, D. (1973). Describing an algorithm by hopcroft. *Acta Informatica*, 2:97–109.

[Gros et Assadi, 1997] Gros, C. et Assadi, H. (1997). Intégration de connaissance dans un système de consultation de documents techniques. *Actes des premières recontres du chapitre Français de l'ISKO, Presses universitaires du Septentrion.*

[Gross et Perrin, 1987] Gross, M. et Perrin, D. (1987). Electronic dictionaries and automata in computational linguistics. *LITP Spring School on Theoretical Computer Science.*

[Guazzo, 1980] Guazzo, M. (1980). A general minimum redundancy source coding algorithm. *IEEE Trans. Information Theory*, 26 (1):15–25.

[Guiasu et Theodorescu, 1971] Guiasu, S. et Theodorescu, R. (1971). Incertitude et information. *Les Presses de l'Université LAVAL, Québec.*

[Hamrouni, 1996] Hamrouni, B. M. (1996). *Méthodes et algorithme de représentation et de compression de grands dictionnaires de formes.* Thèse de doctorat, Université josef Fourier, Grenoble1.

[Hana et Wood, 2006] Hana, Y. et Wood, D. (2006). Obtaining shorter regular expressions from finite-state automata. *Theorical Computer Science.*

[Holub et Melichar, 2000] Holub, J. et Melichar, B. (2000). Approximate string matching using factor automata. *Theoretical Computer Science 249 pp.305-311, 2000.*, 249:305–311.

[Hopcroft, 1971] Hopcroft, J. E. (1971). An n log n algorithm for minimizing the states in a finite automaton. *The Theory of Machines and Computations,Academic Press*, pages 189–196.

[Hopcroft et Ullman, 1979] Hopcroft, J. E. et Ullman, J. D. (1979). Introduction to automata theory, languages, and computation. *Addison-Wesley.*

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimim redundancy codes. *Proc of the IRE 40.*

[Huffman, 1954] Huffman, D. A. (1954). The synthesis of sequential switching circuits. *Journal of Franklin Institute*, 257:161–190.

[Jaumard et Minoux, 1986] JAUMARD, B. et MINOUX, M. (1986). An efficient algorithm for the transitive closure and a linear worstcase complexity result for a class of sparse graphs. *Information processing letters*, 22:163–169.

[Kleene, 1956] KLEENE, S. C. (1956). Representation of events in nerve nets and finite automata. *In C E Shannon and J McCarthy (eds.), Automata Studies, Princeton University Press.*

[Knuth, 1985] KNUTH, D. E. (1985). Dynamic huffman coding. *Journal of Algorithms*, 6:163–180.

[Kowaltowski *et al.*, 1993] KOWALTOWSKI, T., LUCCHESI, C. L. et STOLFI, J. (1993). minimisation binary automata. *Work-shop on Implementing Automata WIA99*, 19:1–31.

[Kucherov et Rusinowitch, 1997] KUCHEROV, G. et RUSINOWITCH, M. (1997). Matching a set of strings with variable length don't cares. *Theoretical Computer Science*, 178:129–154.

[Lacouture et Mori, 1991] LACOUTURE, R. et MORI, R. D. (1991). Lexical tree compression. *Proc. 2nd European Conf. on Speech Communications and Techniques*, pages 581–584.

[Liquière, 2006] LIQUIÈRE, M. (2006). Propositionalisation formelle et fouille de graphes. *Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA).*

[Lopresti et Wilfong, 2001] LOPRESTI, D. et WILFONG, G. (2001). Evaluating document analysis results via graph probing. *Sixth International Conference on Document Analysis and Recognition*, 116 - 120.

[Manber, 1993] MANBER, U. (1993). A text compression scheme that allows fast searching in the compressed file. *Software- Practice and experience*, 19(2):11–124.

[Manber et Myers, 1990] MANBER, U. et MYERS, G. W. (1990). Suffix arrays : A new method for on-line string searches. *Proceedings of the first annual ACM-SIAM Symposium on Discrete Algorithms*, pages 319–327.

[Martineau, 2001] MARTINEAU, C. (2001). *Compression de textes en langue naturelle.* Thèse de doctorat, Université de Marne-la-Vallée, Institut Gaspard-Monge.

[Maurel et Guenthner, 2006] MAUREL, D. et GUENTHNER, F. (2006). *Automata and Dictionaries.* King's College Publications.

[McKay, 1981] MCKAY, B. (1981). Practical graph isomorphism. *Congressus Numerantium*, 30:45–87.

[McNaughton et Yamada, 1960] MCNAUGHTON, R. et YAMADA, H. (1960). Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, pages 39–47.

[Mekki et Nazarenko, 2001] MEKKI, T. E. et NAZARENKO, A. (2001). Quel index pour le document électronique ? document électronique : méthodes et techniques. *Colloque International sur le Document Électronique.*

[Möhring, 1989] MÖHRING, R. H. (1989). Computationally tractable classes of ordered sets. *I. Rival editor, Algorithms and Order*, 255 of NATO ASI series C:105–194.

[Moffat, 1990] MOFFAT, A. M. (1990). Implementing the ppm data compression scheme. *IEEE Trans. Commun*, 38:401–406.

[Mohri, 1996] Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Journal of Natural Language Engineering*, 2:1–20.

[Moore, 1956] Moore, E. F. (1956). Gedanken experiments on sequential machines. *In Automata Studies, Annals of Mathematical Studies, Princeton University Press*, 34.

[Myhill, 1957] Myhill, J. (1957). Finite automata and the representation of events. *Wright Patterson AFB*, pages WADD TR 57–624.

[Nazarenko et Mekki, 2005] Nazarenko, A. et Mekki, T. E. (2005). Building back-of-the-book indexes in the terminology. *special issue on "Application-driven Terminology engineering", John Benjamins.*

[Nelson, 1992] Nelson, M. (1992). *The data compression book.* M&T Books.

[Perrin, 1995] Perrin, D. (1995). Technique et science informatique. *Technique et Science Informatique*, 14:409–433.

[Raffinot, 1997] Raffinot, M. (1997). Asymptotic estimation of the average number of terminal states in dawgs. *N Ziviani, R. Baeza-Yates and K. Guimaraes, editors Proceedings, Recife.*

[Revuz, 1991] Revuz, D. (1991). *Dictionnaires et lexiques : méthodes et algorithmes.* Thèse de doctorat, Université de Paris 7.

[Revuz, 1992] Revuz, D. (1992). Minimization of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189.

[Réfrégier, 1993] Réfrégier, P. (1993). Théorie du signal. *Signal-Information-Fluctuations, Masson, Paris.*

[Rissanen et Langdon, 1979] Rissanen, G. G. et Langdon, J. (1979). Arithmetic coding. *IBM J. Res. Devel*, 3, no. 2:149–162.

[Ristov, 2005] Ristov, S. (2005). Lz trie and dictionary compression. *Software : Practice and Experience.*

[Ristov et Laporte, 1999] Ristov, S. et Laporte, E. (1999). Ziv lempel compression of huge natural language data tries using suffix arrays. *Combinatorial Pattern Matching, 10th Annual Symposium, Warwick University, Proceedings, M. Crochemore, M. Paterson, eds*, LNCS 1645:196–211.

[Salomon, 2000] Salomon, D. (2000). Data compression : the complete reference. *Spinger.*

[Sayood, 2000] Sayood, K. (2000). Introduction to data compression. *Morgan Kaufmann, 2nd edition.*

[Sgarbas *et al.*, 2003] Sgarbas, K. N., Fakotakis, N. et Kokkinakis, G. K. (2003). Optimal insertion in deterministic dawgs. *Theoretical Computer Science*, 1(3):103 – 117.

[Shannon, 1948] Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656.

[Shannon et Weaver, 1949] Shannon, C. et Weaver, W. (1949). The mathematical theory of communications. *University of Illinois Press.*

[Sieminski, 1988] Sieminski, A. (1988). Fast decoding of the huffman codes. *Inf. Process. Lett. 26, 237-241,1988.*, 26:237–241.

[Tanaka, 1987] Tanaka, H. (1987). Data structure of huffman codes and its application to efficient encoding and decoding. *IEEE Trans. Inf. Theory*, 33:154–156.

[Thompson, 1968] THOMPSON, K. (1968). Regular expression search algorithm. *Communications of the ACM 11*, pages 419–422.

[Tounsi *et al.*, 2007] TOUNSI, L., BOUCHOU, B., LENTÉ, C. et MAUREL, D. (2007). Internal structure computation of finite state automaton. *AutoMathA 2007 : Conference of the program AutoMathA of the European Science Foundation*, page 12 pages.

[Tounsi *et al.*, 2005] TOUNSI, L., BOUCHOU, B. et MAUREL, D. (2005). Basic search of sub automata ; application to electronic dictionaries. *IEEE International Conference on Natural Language Processing and Knowledge Engineering*, pages 543–548.

[Tounsi *et al.*, 2006a] TOUNSI, L., LENTÉ, C., BOUCHOU, B. et MAUREL, D. (2006a). Décomposition d'automates représentant des dictionnaires électroniques. *7ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, page 2 pages.

[Tounsi *et al.*, 2006b] TOUNSI, L., LENTÉ, C. et MAUREL, D. (2006b). Analyse statistique de la structure des automates représentant des dictionnaires électroniques. *Huitièmes Journées internationales d'Analyse statistique des Données Textuelles*, pages 527–935.

[Ukkonen, 1995] UKKONEN, E. (1995). On-line construction of suffix-trees. *Algorithmica*, 14(3):249–260.

[Valdez *et al.*, 1982] VALDEZ, J., TARJAN, R. E. et LAWLER, E. L. (1982). The recognition of series-parallel digraphs. *SIAM J. Comput*, 11(2):298–313.

[van Lint, 1982] van LINT, J. H. (1982). Introduction to coding theory. *Springer-Verlag.*

[Vérin, 1998] VÉRIN, R. (1998). Algorithmes de recherche de motifs dans les sequences d'adn. Mémoire de maîtrise, Université Marne la vallée.

[Watson, 1995] WATSON, B. W. (1995). *Taxonomies and Toolkits of Regular Language Algorithms*. Thèse de doctorat, Eindhoven University of Technology.

[Welch, 1984] WELCH, T. (1984). A technique for high-performance data compression. *IEEE Computer*, 17 No 6:8–19.

[Williams et Sloane, 1978] WILLIAMS, F. J. M. et SLOANE, N. J. A. (1978). The theory of error-correcting codes. *North Holland Publishing Company.*

[Witten *et al.*, 1987] WITTEN, I. H., NEAL, R. M. et CLEARLY, J. G. (1987). Arithmetic coding for data compression. *Commun. ACM*, 30:520–540.

[Yan et Han, 2003] YAN, X. et HAN, J. (2003). Closegraph : Mining closed frequent graph patterns. *Proc. of the 9th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 286–295.

[Ziv et Lempel, 1977] ZIV, J. et LEMPEL, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, It-23, No3:337–343.

[Ziv et Lempel, 1978] ZIV, J. et LEMPEL, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transaction on Information Theory*, 24(5).

Annexe A

Statistiques des automates étudiés

A.1 Caractéristique des automates après application de *Recherche SA-SP*

Automate	Automate initial		Automate Réduit			
	Transitions	États	Transitions	**Gain des transition**	États	**Gain des états**
Catégorie 1						
DELAF Fr	177 465	67 995	150 299	**15,3%**	42 625	**37,3%**
DELAF En	252 664	116 848	199 332	**21,1%**	66 635	**42,9%**
DELAF Sr	193 668	61 383	163 466	**15,6%**	43 018	**29,9%**
DELAF De	335 284	142 795	276 223	**17,6%**	87 772	**38,5%**
Villes Fr	95 589	61 240	56 064	**41,3%**	22 224	**63,7%**
Poly En	717 112	435 940	427 269	**40,4%**	147 423	**66,1%**
Catégorie 2						
Web Fr	298 117	101 837	253 595	**14,9%**	72 102	**29,2%**
Web Hu	270 495	113 678	221 908	**17,9%**	72 006	**36,6%**
Weg Bg	209 209	85 661	167 045	**20,1%**	52 052	**39,2%**
Web Pt	538 697	214 992	435 147	**19,2%**	130 430	**39,3%**
Catégorie 3						
ADN	2 680 516	2 677 033	7 913	**99,7%**	4 435	**99,8%**

TAB. A.1 – Gain du nombre de transitions et du nombre d'états

A.2 Parallèles Purs

Les tables A.2 et A.3 présentent respectivement la répartition des parallèles purs en fonction des fréquences et la répartition des parallèles purs en fonction des largeurs. À titre d'exemple, les parallèles purs du DELAF Fr sont répartis en fréquence comme suit : 47,91% apparaissent une seule fois, 16,75% ont une fréquence de 2, 8,38% une fréquence de 3, etc. La répartition de ces parallèles purs en fréquence fonction de la largeur est comme suit : 98,36% de ces parallèles se composent de deux transitions, 1,41% se composent de

trois transitions et 0,23% possèdent une largeur de 4.
Ces résultats sont représentés par les graphiques de la figure 4.6.

	DELAF				Villes	Poly	Web			
	Fr	En	Sr	De	Fr	En	Fr	Hg	Bg	Pt
1	47,91%	42,78%	44,14%	32,84%	56,4%	38,75%	75,98%	76,48%	71,94%	76,31%
2	16,75%	18,23%	12,13%	19,49%	17,73%	11,92%	9,27%	6,69%	9,77%	9,85%
3	8,38%	8,84%	8,35%	6,25%	7,56%	8,13%	3,87%	2,3%	3,98%	2,89%
4	4,19%	5,6%	5,57%	5,51%	4,65%	5,96%	1,8%	2,41%	2,6%	1,81%
5	4,45%	2,53%	2,78%	3,68%	2,62%	3,79%	1,05%	1,19%	1,6%	0,73%
6	2,62%	1,08%	4,37%	4,53%	2,33%	3,79%	0,87%	0,9%	1,51%	0,64%
7	2,09%	1,81%	2,58%	1,23%	1,74%	2,17%	0,73%	0,64%	0,64%	0,8%
8	2,09%	1,44%	1,19%	1,23%	1,16%	5,42%	0,44%	0,9%	0,76%	0,62%
9	1,57%	2,53%	0,8%	2,21%	1,74%	2,71%	0,35%	1,16%	0,34%	0,3%
10	1,05%	0,36%	1,19%	1,72%	0,58%	2,17%	0,5%	1,05%	0,64%	0,42%
11	0,52%	1,08%	0,8%	1,35%	0,58%	0,54%	0,43%	0,52%	0,48%	0,38%
12	1,57%	1,44%	1,39%	1,84%	0,58%	1,63%	0,43%	0,73%	0,48%	0,31%
13	0%	0,9%	0,8%	0,74%	0%	2,17%	0,21%	0,52%	0,28%	0,29%
14	0%	0,36%	0,4%	0,74%	0%	1,08%	0,15%	0,41%	0,42%	0,28%
15	0,52%	1,99%	1,19%	1,47%	0%	1,63%	0,38%	0,47%	0,22%	0,16%
16	0,52%	0,36%	0,4%	1,23%	1,74%	1,08%	0,16%	0,12%	0,17%	0,3%
17	0,52%	0,72%	0,4%	0,98%		0%	0,09%	0,35%	0,22%	0,21%
18		0,36%		2,08%		0%	0,13%	0,17%	0,28%	0,28%
19		1,08%		0,98%		0%	0,23%	0,35%	0,2%	0,15%
20		1,44%				0,54%	0,16%	0,26%	0,11%	0,21%

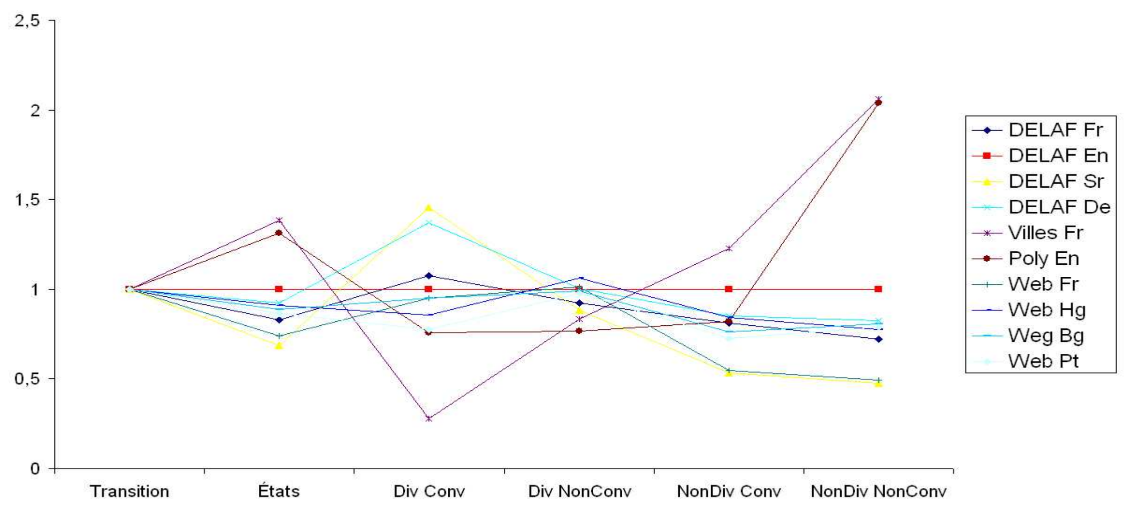
TAB. A.2 – Répartition des parallèles purs en fonction des fréquences

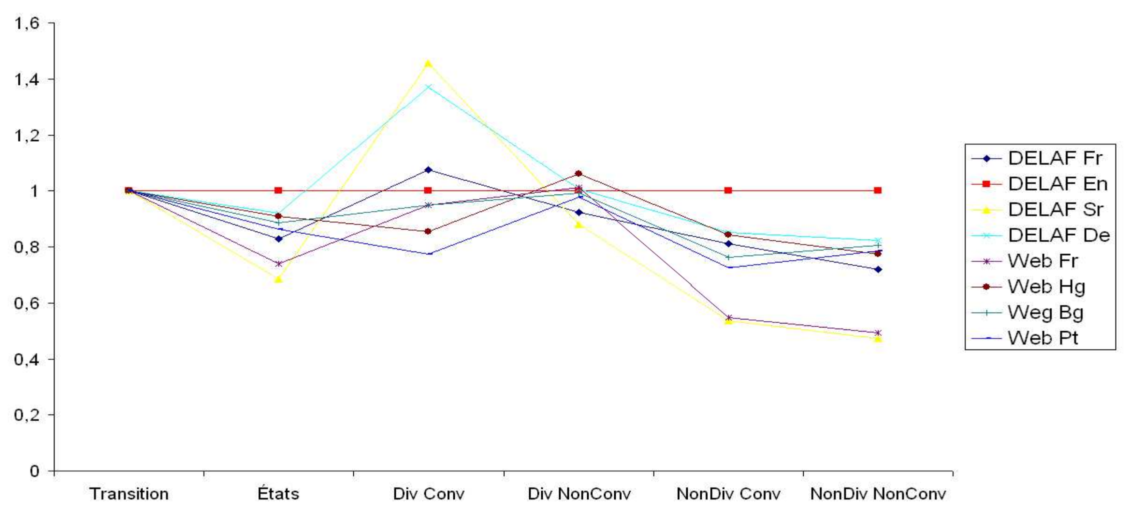
	DELAF				Villes	Poly	Web			
	Fr	En	Sr	De	Fr	En	Fr	Hg	Bg	Pt
2	98,36%	96,03%	63,59%	65,47%	92,86%	97,71%	19,34%	30,83%	29,45%	18,31%
3	1,41%	3,9%	32,97%	27,33%	6,47%	2,06%	41,26%	39,28%	38,16%	41,99%
4	0,23%	0,07%	3,4%	5,11%	0,22%	0,16%	21,14%	15,94%	17,23%	23,09%
5			0,04%	1,5%	0,45%	0,08%	7,86%	7,69%	7,29%	7,23%
6				0%			3,86%	3,13%	3,22%	3,97%
7				0,6%			2,44%	1,23%	1,52%	1,95%
8							1,81%	0,85%	1,04%	0,97%
9							1,32%	0,47%	1,23%	0,72%
10							0,34%	0,28%	0,19%	0,72%
11								0,19%	0,47%	0,21%
12								0,09%	0%	0,21%
13									0%	0,3%
14									0,09%	0,13%
15									0%	0,04%
16									0%	0,08%
17									0,09%	0,04%
18										0,04%

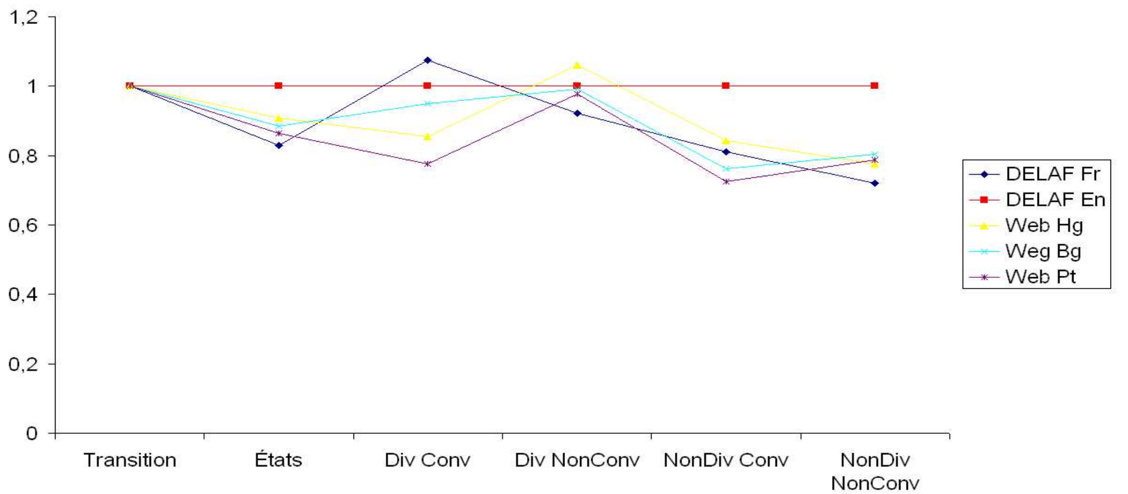
TAB. A.3 – Répartition des parallèles purs en fonction des largeurs

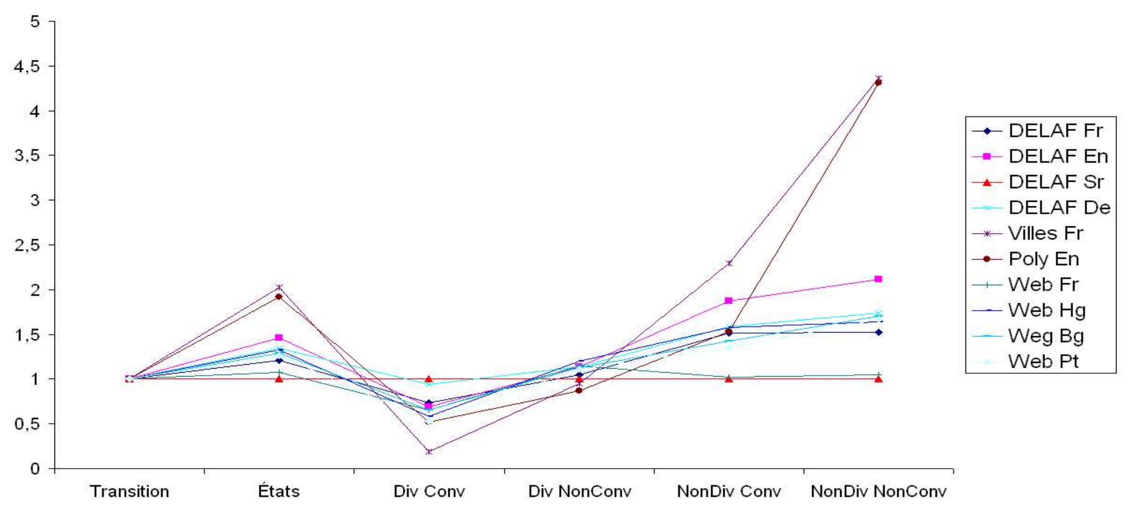
A.3 Similarité et dissimilarité des automates étudiés

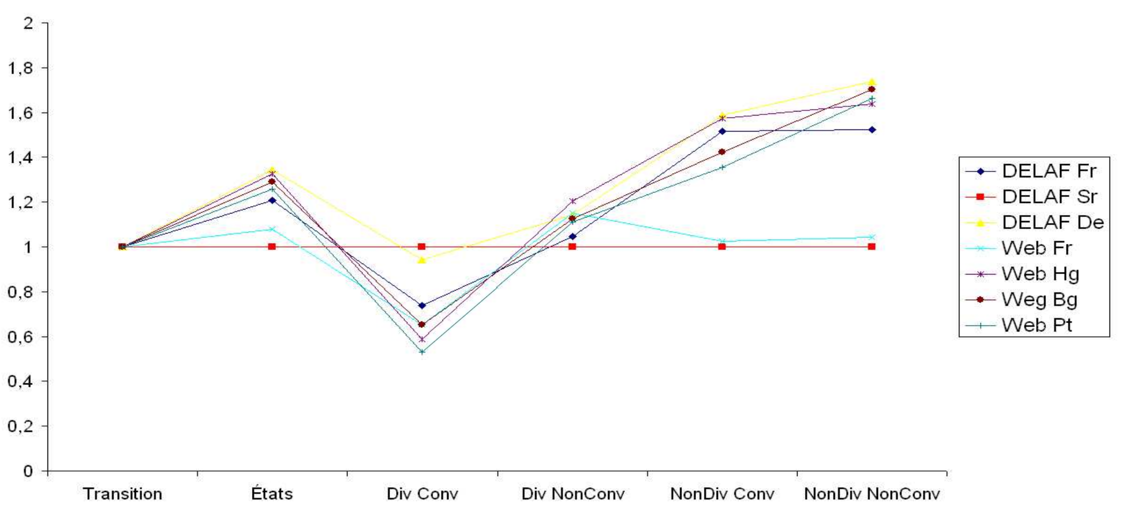
Les graphiques suivants présentent les différentes phases de comparaison de la structure d'un automate par rapport aux autres. L'objectif est d'identifier des automates se ressemblant en se basant uniquement sur leurs statistiques descriptives.

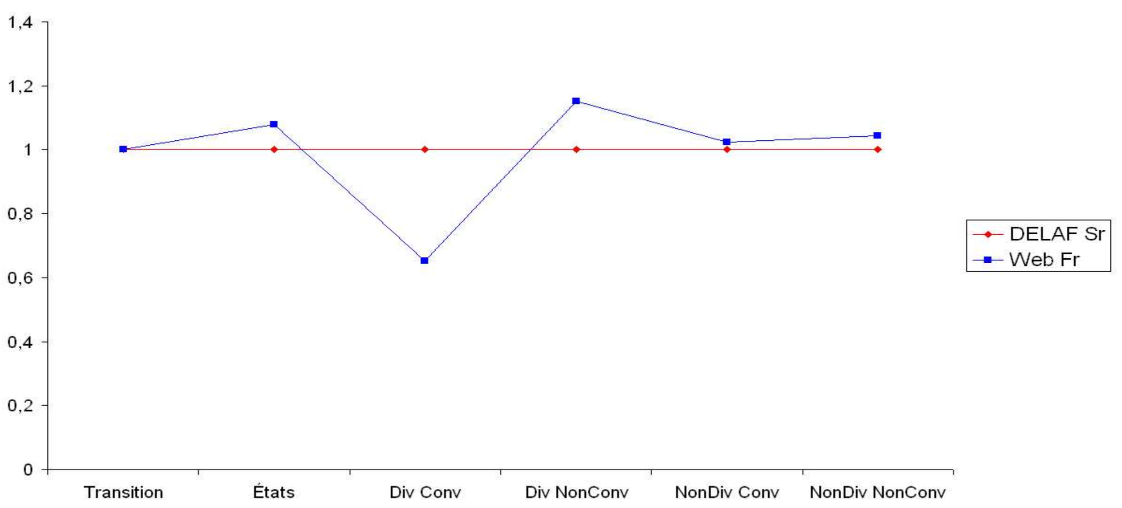

Comparaison de la structure des automates
par rapport à l'automate DELAF En
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

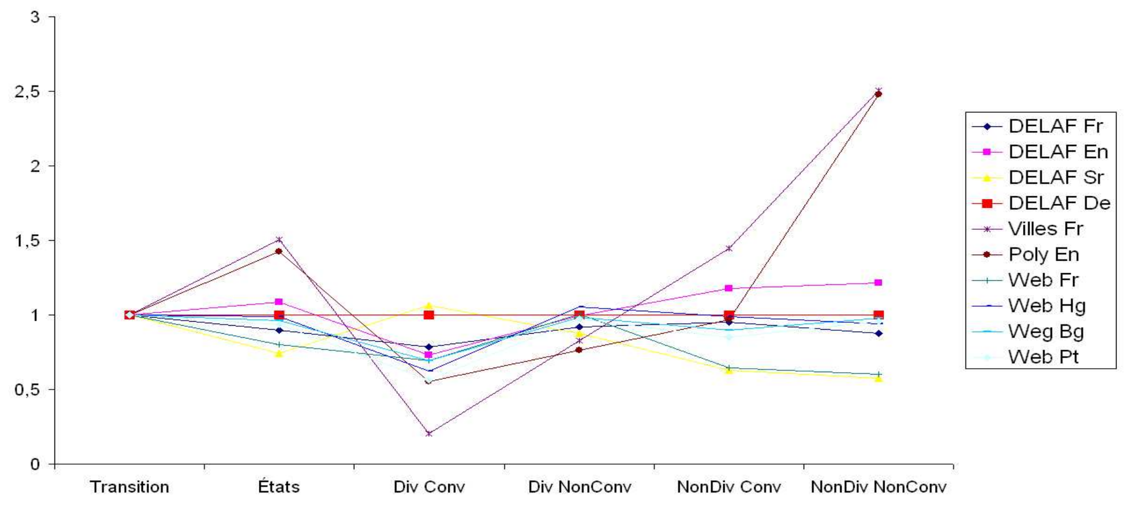

Comparaison de la structure des automates
par rapport à l'automate DELAF En
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Web Fr
Web Hg
Weg Bg
Web Pt

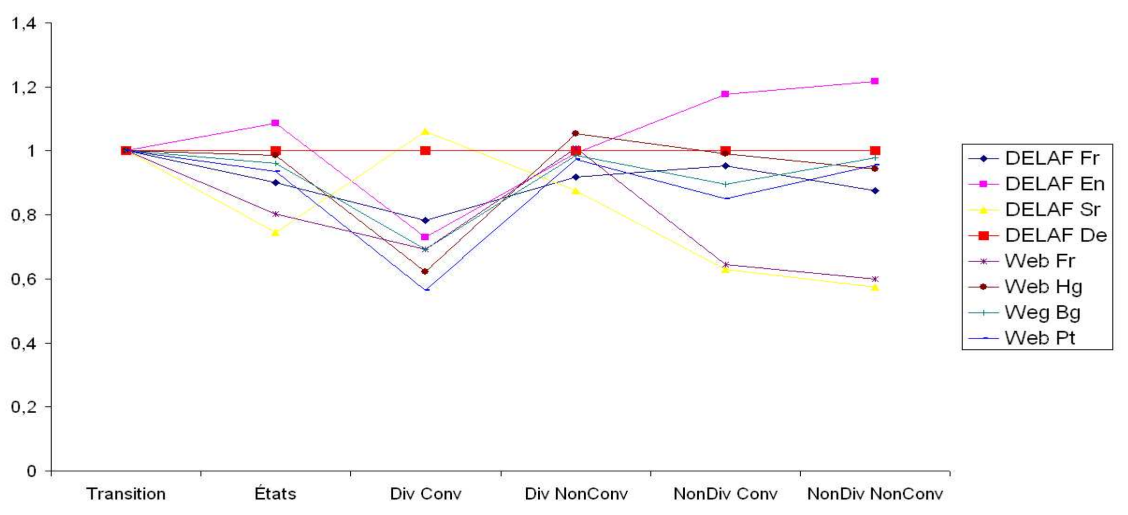

Comparaison de la structure des automates
par rapport à l'automate DELAF En
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv
NonConv
DELAF Fr
DELAF En
Web Hg
Weg Bg
Web Pt

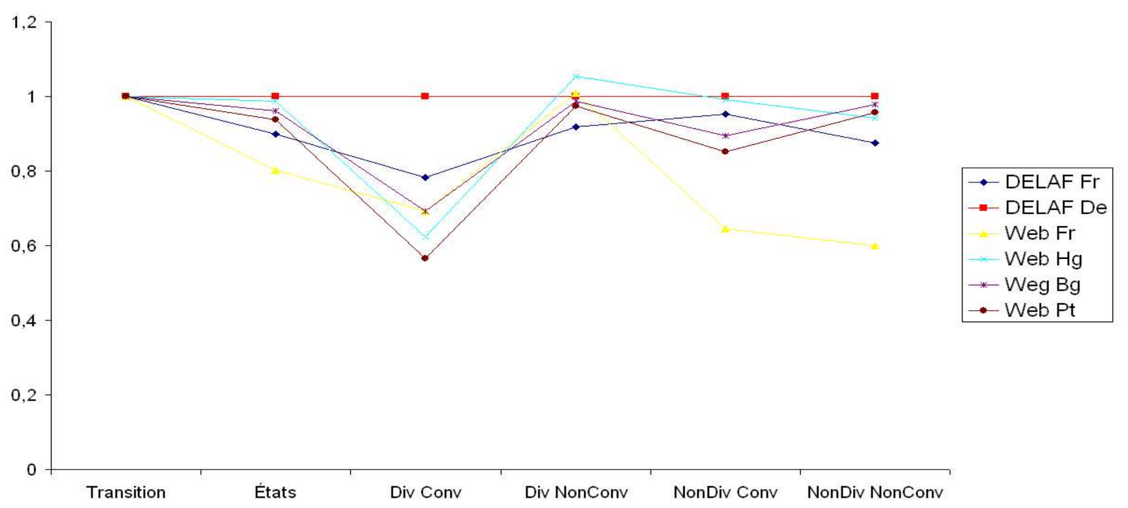

Comparaison de la structure des automates
par rapport à l'automate DELAF Sr
5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

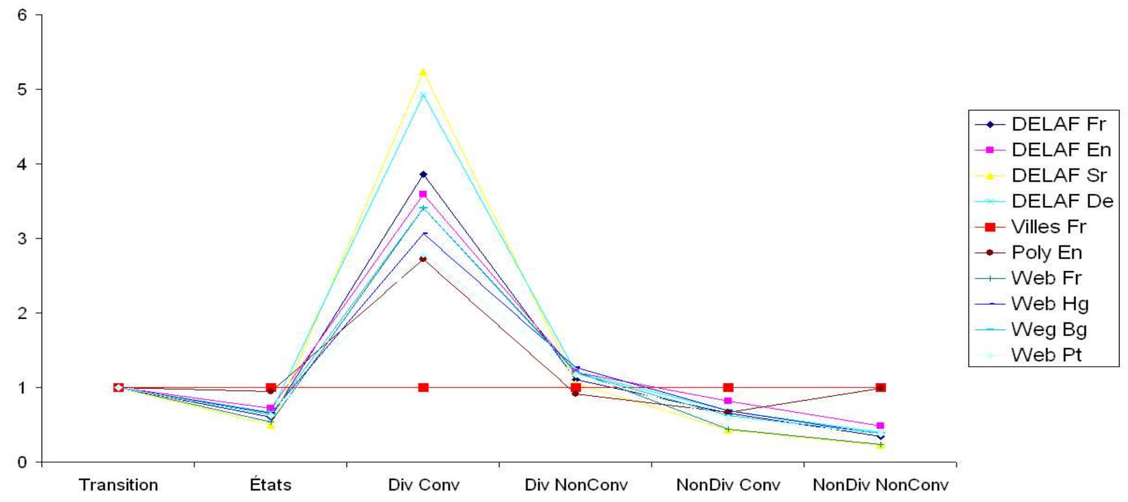

Comparaison de la structure des automates
par rapport à l'automate DELAF Sr
2
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF Sr
DELAF De
Web Fr
Web Hg
Weg Bg
Web Pt

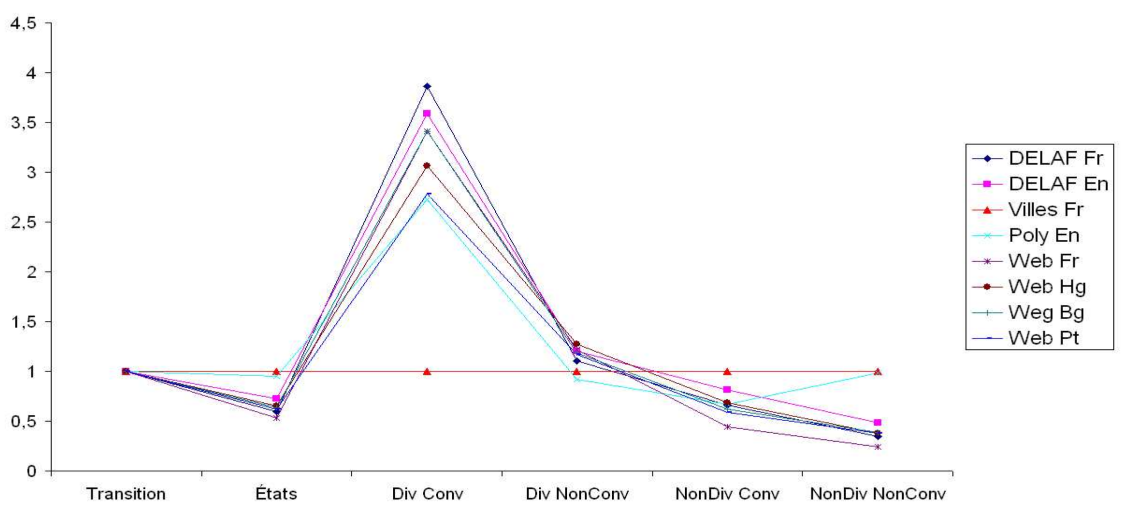

Comparaison de la structure des automates
par rapport à l'automate DELAF Sr
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Sr
Web Fr

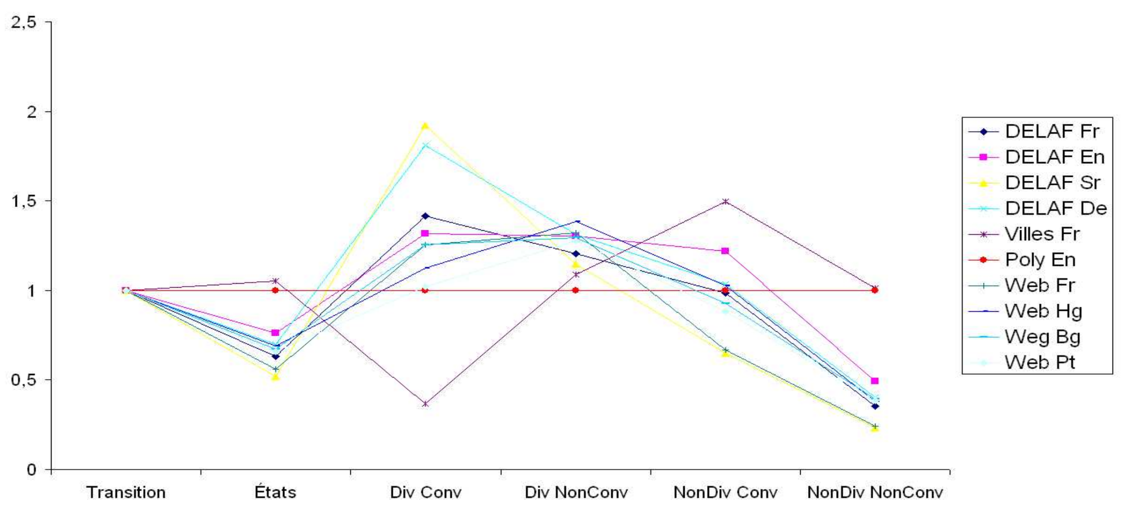

Comparaison de la structure des automates
par rapport à l'automate DELAF De
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

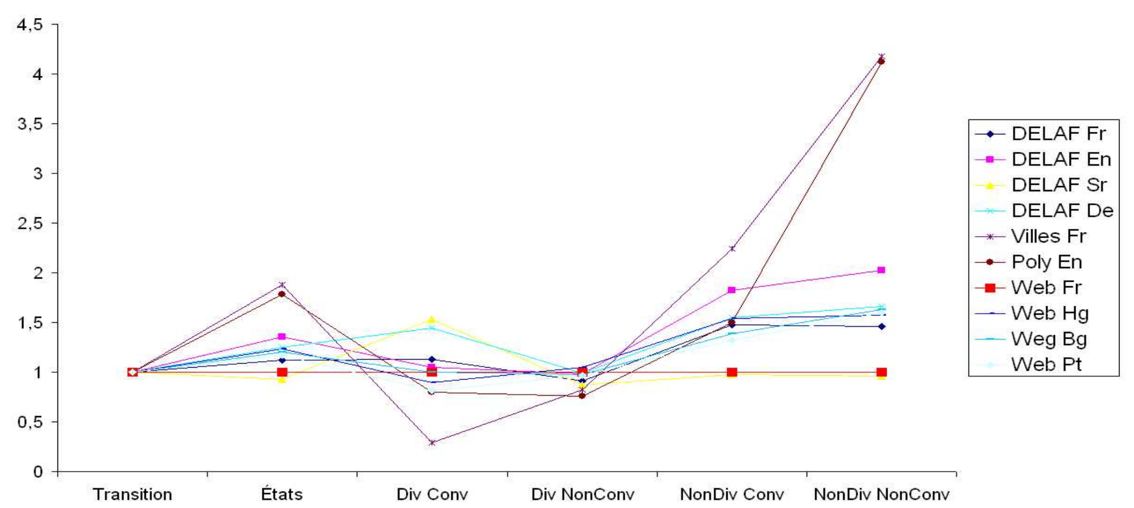

Comparaison de la structure des automates
par rapport à l'automate DELAF De
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Web Fr
Web Hg
Weg Bg
Web Pt

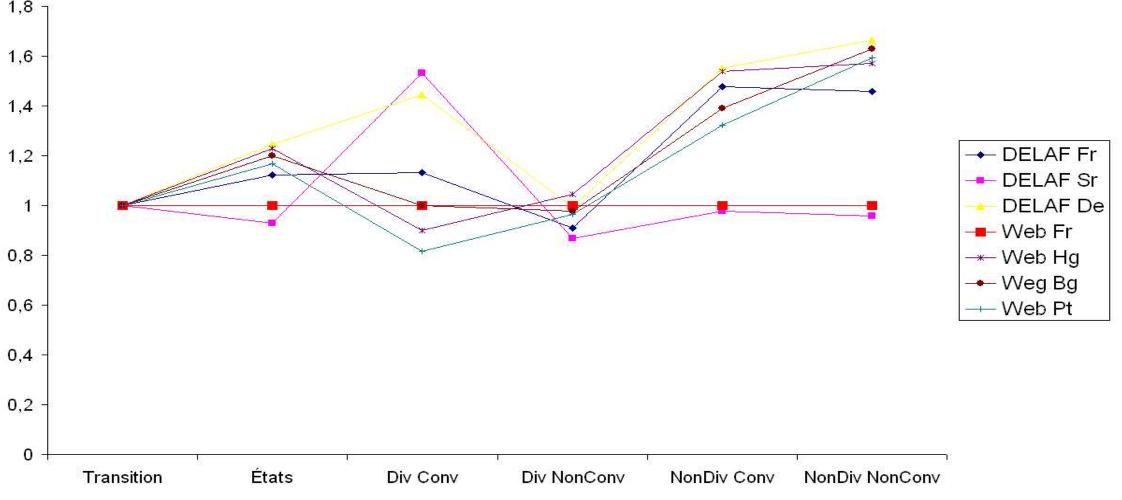

Comparaison de la structure des automates
par rapport à l'automate DELAF De
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF De
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate des villes françaises
6
5
4
3
2
1
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

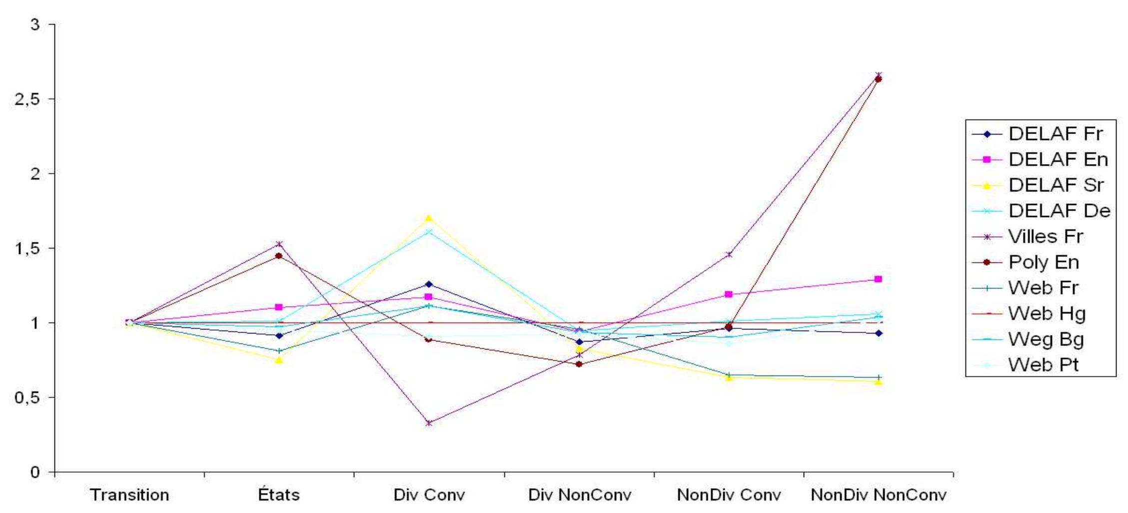

Comparaison de la structure des automates
par rapport à l'automate des villes françaises
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate des polylexicaux anglais
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

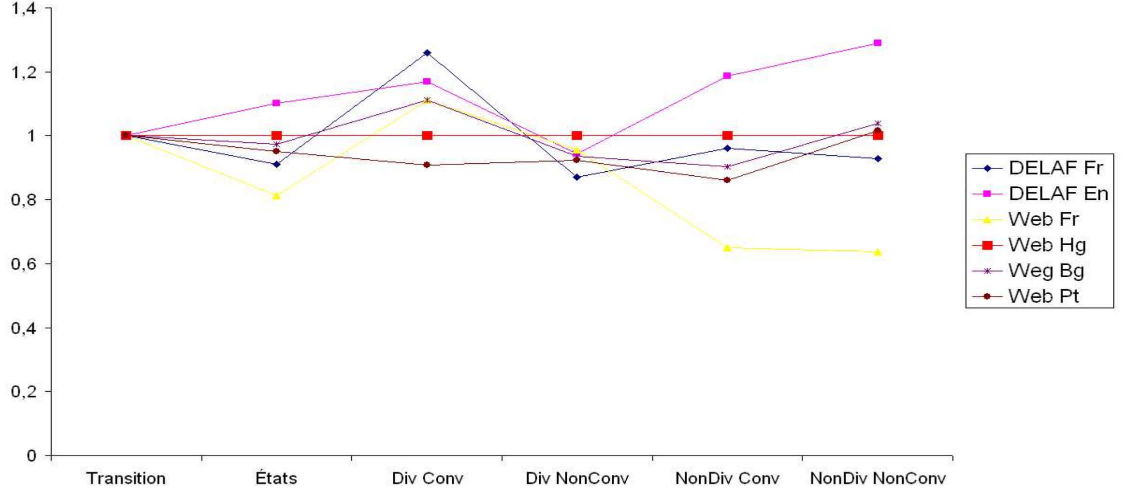

Comparaison de la structure des automates
par rapport à l'automate web fr
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt

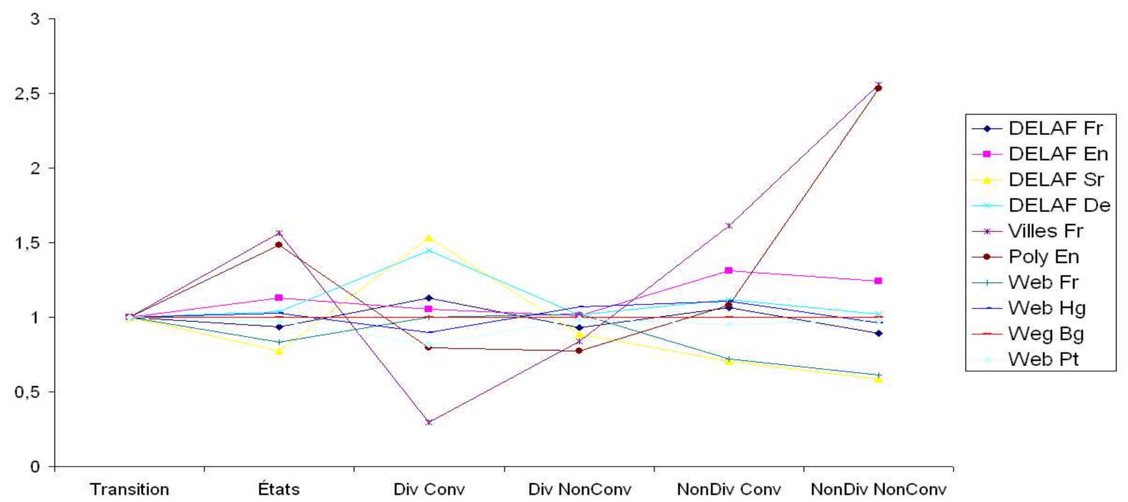

Comparaison de la structure des automates
par rapport à l'automate web Fr
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF Sr
DELAF De
Web Fr
Web Hg
Weg Bg
Web Pt

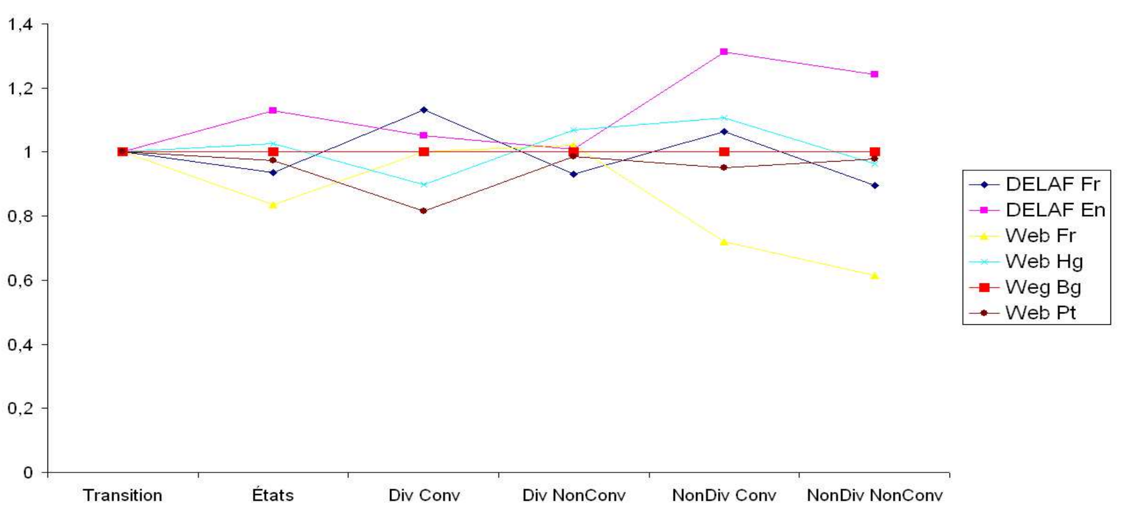
Comparaison de la structure des automates par rapport à l'automate web Hg

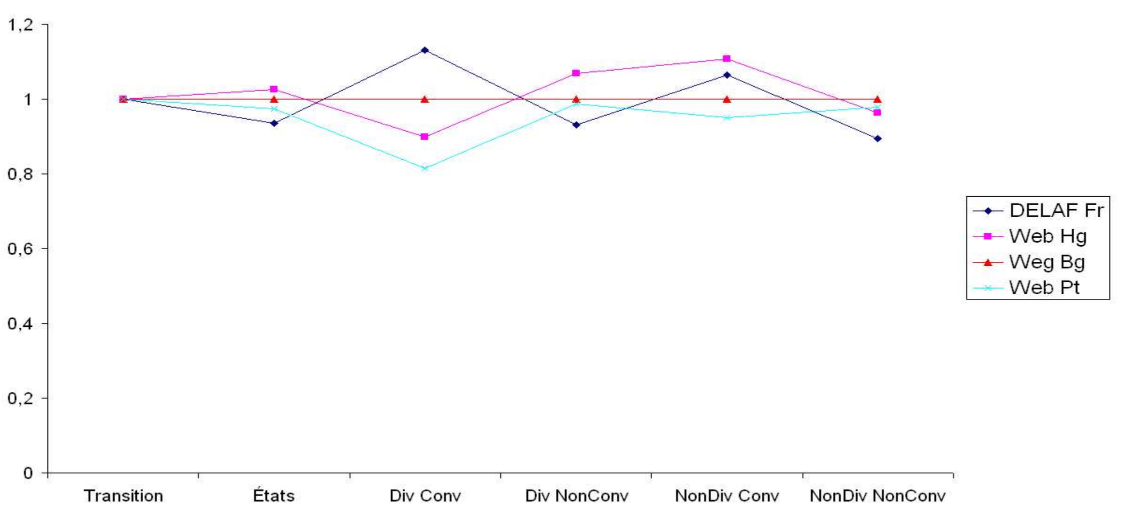


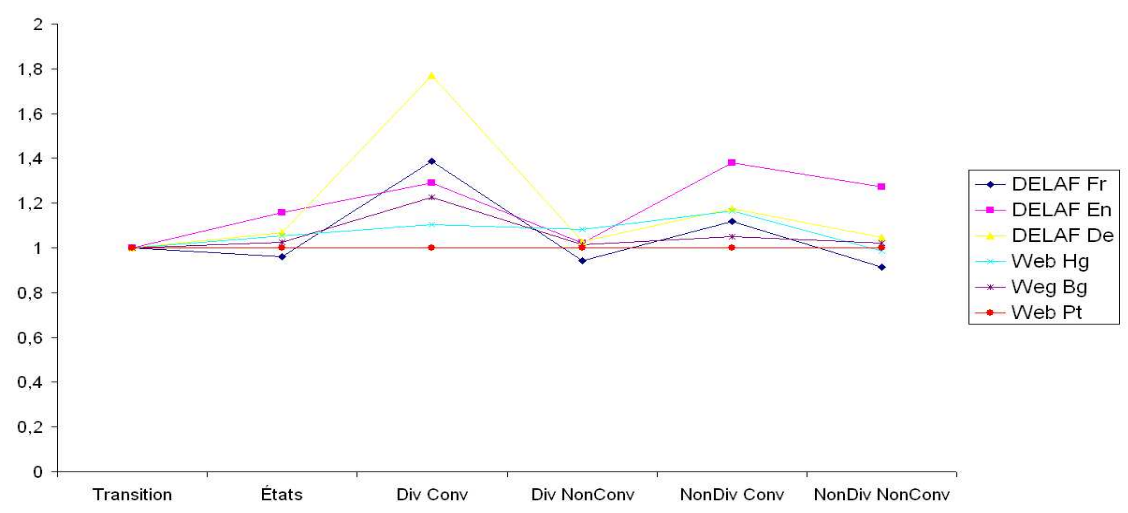
Comparaison de la structure des automates par rapport à l'automate web Hg

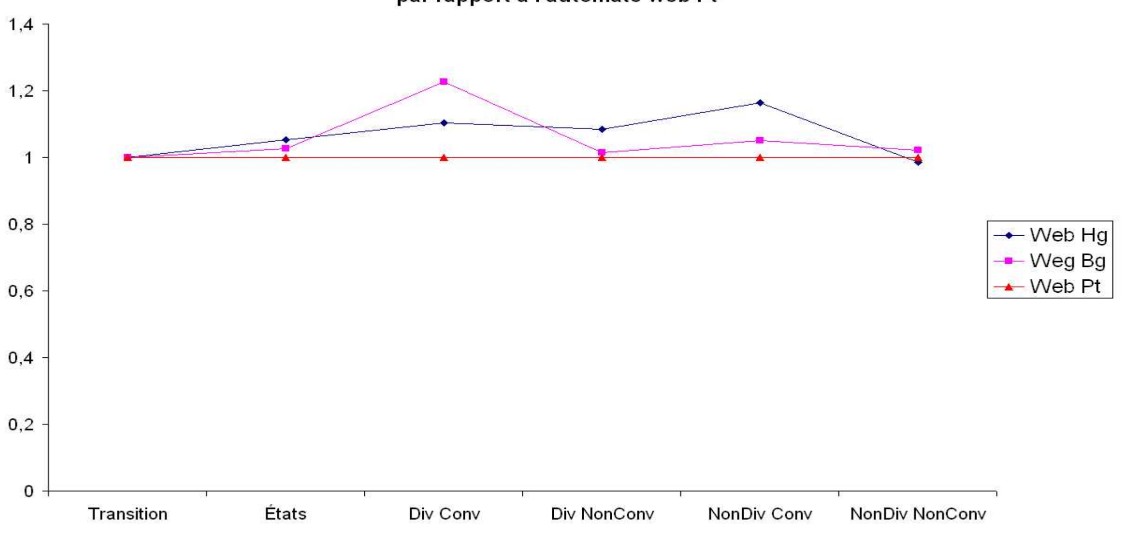


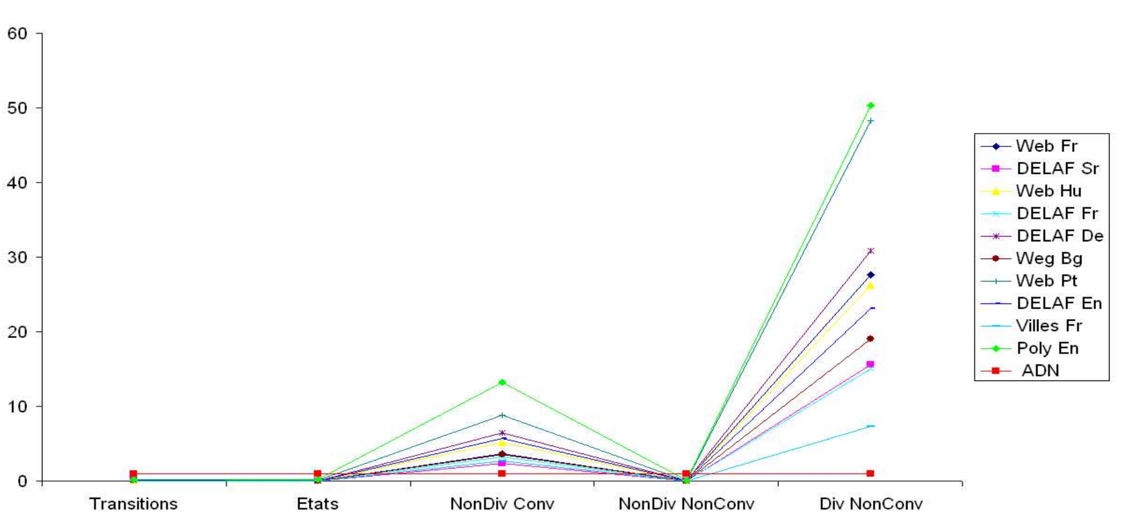

Comparaison de la structure des automates
par rapport à l'automate web Bg
3
2,5
2
1,5
1
0,5
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF Sr
DELAF De
Villes Fr
Poly En
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Bg
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
Web Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Bg
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Pt
2
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
DELAF Fr
DELAF En
DELAF De
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates
par rapport à l'automate web Pt
1,4
1,2
1
0,8
0,6
0,4
0,2
0
Transition
États
Div Conv
Div NonConv
NonDiv Conv
NonDiv NonConv
Web Hg
Weg Bg
Web Pt


Comparaison de la structure des automates par rapport
à l'automate ADN
60
50
40
30
20
10
0
Transitions
Etats
NonDiv Conv
NonDiv NonConv
Div NonConv
Web Fr
DELAF Sr
Web Hu
DELAF Fr
DELAF De
Weg Bg
Web Pt
DELAF En
Villes Fr
Poly En
ADN

A.4 Étude des fréquences

La table A.4 présente les bigrammes et trigrammes les plus fréquents au sein des séries pures des automates DELAF français, DELAF anglais, DELAF allemand et DELAF Serbe.

Bigrammes fréquents				Trigrammes fréquents			
Fr	En	De	Sr	Fr	En	De	Sr
ro	er	er	er	ant	the	sch	str
on	in	ch	en	pha	ter	ter	ter
an	an	en	ra	ion	ent	ich	icy
in	ra	in	sx	tro	and	ent	ija
ra	en	on	an	oph	ion	cht	ent
ch	ti	an	st	ill	ati	che	sxt
en	on	te	cy	ent	ran	ste	isx
ph	ar	st	li	ect	tra	cha	ali
ti	ri	ro	ri	ist	ist	ver	ije
ri	th	ri	in	and	str	der	cyi
er	at	ti	al	rop	men	pho	sti
ar	or	ar	re	tal	est	rop	sta
la	al	ra	lx	pho	tio	ion	lxa
st	st	re	ro	lan	res	ell	ist
th	ro	al	la	thr	ing	pha	ver
lo	re	le	ij	tér	era	oph	ran
al	te	li	ar	ron	tic	hro	oli
or	nt	at	or	ran	ect	osp	men
tr	li	or	ti	onn	ant	ing	eri
at	ic	la	on	hro	rat	pro	sto

Tab. A.4 – Bigrammes et trigrammes fréquents des séries pures

Annexe B

Factorisation et compression d'un automate

B.1 Description de l'en-tête utilisé pour le codage et le décodage d'un automate

Pour pouvoir décompresser le fichier binaire représentant un automate A, 4 informations sont nécessaires dont une table de correspondance pour les caractères portés par les transitions de A. Ces informations sont schématisées dans l'entête ci-dessous.

	Nombre de transitions de l'automate	
	$Taille_{adresse}$	$Taille_{alphabet}$
Table de correspondance	1er caractère (ASCII)	code du 1er caractère
	$2^{ème}$ caractère (ASCII)	code du $2^{ème}$ caractère
	...	...
	Dernier caractère (ASCII)	code du dernier caractère

Tab. B.1 – En-tête utilisé pour le codage et le décodage d'un automate

B.2 Caractéristiques des factorisations d'un automate minimal (Factorisations FCM)

La table B.2 présente les caractéristiques de la compression lorsque l'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 128$. À titre d'exemple, la taille initiale de l'automate DELAF français est 2 178 Ko ; suite à la factorisation, sa taille diminue à 2 031 Ko et ensuite vient la compression qui réduit l'automate à 529 Ko, ce qui représente un taux de compression de 75%. Ainsi, le nombre de transitions initialement à 177 465 diminue jusqu'à atteindre 166 759. La longueur moyenne des séries factorisées est 2 et la plus longue série factorisée est de longueur 2.
La factorisation est plus efficace sur les automates hongrois et portugais parce qu'elle a fait perdre suffisamment de transitions aux automates, ce qui explique les meilleures résultats en comparaison à la compression simple. Dans ce tableau, on peut aussi remarquer que la majorité des séries qui ont été choisies par la fonction gain sont de

longueur 2. Ce choix vient de la forte redondance de ces petites séries.

Dans le tableau B.3 on a fixé la taille maximale de l'alphabet à $Taille_{alphabet} = 256$, on peut voir que certains résultats ont été améliorés en comparaison de la factorisation précédente ($Taille_{alphabet} = 128$), c'est le cas des automates DELAF anglais, DELAF allemand, villes française et Polylexicaux anglais. Pour les autres, le taux de compression a diminué parce que le gain obtenu par la factorisation des séries n'a pas compensé l'utilisation d'un bit supplémentaire pour coder de nouveaux caractères et étendre plus l'alphabet.

Le tableau B.4 présente les caractéristiques de la factorisation lorsqu'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 512$. On peut voir que les résultats obtenus sur les automates des villes françaises et les polylexicaux anglais ont encore été améliorés. Une autre remarque intéressante concerne l'extension des séries. En effet, sur certains automates la longueur maximale des séries factorisées a augmenté. Ceci peut s'expliquer par l'épuisement des séries de longueur 2 ayant un grand nombre d'occurrences.

Lorsqu'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 1\,024$, les résultats de la factorisation et compression présentés dans le tableau B.5 montrent que l'automate des polylexicaux anglais a encore perdu 12 Ko. L'automate des villes françaises a lui aussi diminué, mais plus faiblement.

L'étape suivante fixe la taille de l'alphabet à $Taille_{alphabet} = 2\,048$. La taille de l'automate des polylexicaux anglais arrête de décroître. En effet, on constate que la factorisation de 1 024 séries supplémentaires (caractères supplémentaires de l'alphabet) par rapport à l'étape précédente, ne permet pas d'éliminer suffisamment de transitions dans l'automate pour réduire sa taille. En revanche, ces factorisations profitent à l'automate des villes françaises qui perd en taille.

Le tableau B.7 révèle que la factorisation de l'automate des villes françaises a traité toutes ses séries, aucune amélioration n'est donc possible (en se basant sur notre fonction gain). Le traitement des automates des villes françaises et DELAF serbe n'a pas atteint 4 096 factorisations, un arrêt est observé respectivement à 3 321 et 3 043.
Cette information indique que le calcul du gain a atteint ses limites pour l'automate du DELAF serbe, à la $3\,044^{ème}$ étape, car il ne décèle plus aucune série intéressante à factoriser, le calcul du gain est négatif ou nul.

La table B.8 indique que le seul automate possédant 8 192 séries factorisées et qui continue à avoir un gain positif est l'automate des polylexicaux anglais. Aucun des autres n'a atteint cette limite. À l'étape suivante ($Taille_{alphabet} = 16\,384$), le nombre maximum de factorisations pour tous les automates est atteint.

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2 178,22	2 031,41	529,44	75,69%	177 465	166 759	2	2
DELAF En	3 081,12	2 843,93	738,96	76,02%	252 664	232 772	2,02	3
DELAF Sr	2 340,47	2 222,85	585,33	74,99%	193 668	184 363	2	2
DELAF De	4 133,67	3 888,09	1 040,56	74,83%	335 284	315 666	2	2
Villes Fr	1 108,25	911,79	240,97	78,26%	95 589	78 899	2,13	4
Poly En	8 963,46	6 899,42	1 919,01	78,59%	717 112	561 373	2,14	6
Catégorie 2								
Web Fr	3 560,97	3 383,04	935,38	73,73%	298 117	283 742	2	2
Web Bg	2 503,96	2 275,89	610,06	75,64%	209 209	192 152	2	2
Web Hu	3 235,67	2 899,57	788,35	75,64%	270 495	248 329	2	2
Web Pt	6 510,09	5 938,83	1 653,47	74,60%	53 8697	501 592	2,44	23

TAB. B.2 – Caractéristiques de la factorisation à $Taille_{alphabet} = 128$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2 178,22	1 975,36	533,32	75,52%	177 465	161 676	2	2
DELAF En	3 081,12	2 728,38	735	76,15%	252 664	222 860	2,01	3
DELAF Sr	2 340,47	2 189,39	598,16	74,44%	193 668	181 334	2	3
DELAF De	4 133,67	3 745,05	1 036,72	74,92%	335 284	303 177	2,01	4
Villes Fr	1 108,25	846,05	231,22	79,14%	95 589	72 694	2,07	4
Poly En	8 963,46	6 518,29	1 869,04	79,15%	717 112	527 793	2,22	6
Catégorie 2								
Web Fr	3 560,97	3 330,83	954,48	73,20%	29 8117	279 113	2	2
Web Bg	2 503,96	2 226,42	618,69	75,29%	20 9209	187 543	2,04	7
Web Hu	3 235,67	2 840,31	800,85	75,25%	27 0495	242 837	2	2
Web Pt	6 510,09	5 803,93	1671,96	74,32%	53 8697	48 8970	2,37	23

TAB. B.3 – Caractéristiques de la factorisation à $Taille_{alphabet} = 256$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2178,22	1943,1	543,36	75,06%	177465	158634	2,01	3
DELAF En	3081,12	2661,16	743,03	75,88%	252664	217054	2,01	3
DELAF Sr	2340,47	2170,26	614,71	73,74%	193668	179484	2,01	4
DELAF De	4133,67	3665,27	1049,77	74,60%	335284	296102	2,51	33
Villes Fr	1108,25	808,15	228,62	79,37%	95589	68996	2,04	4
Poly En	8963,46	6203,58	1773,72	80,21%	717112	500642	2,2	6
Catégorie 2								
Web Fr	3560,97	3300,81	979,52	72,49%	298117	276307	2,04	15
Web Bg	2503,96	2195,51	632,19	74,75%	209209	184546	2,1	17
Web Hu	3235,67	2800,39	818,4	74,71%	270495	239038	2,04	14
Web Pt	6510,09	5716,47	1703,4	73,83%	538697	480677	2,74	32

TAB. B.4 – Caractéristiques de la factorisation à $Taille_{alphabet} = 512$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2178,22	1923,25	556,97	74,43%	177465	156514	2,04	9
DELAF En	3081,12	2617,81	757,01	75,43%	252664	212974	2,09	12
DELAF Sr	2340,47	2157,67	632,96	72,96%	193668	177948	2,1	7
DELAF De	4133,67	3609,77	1069,7	74,12%	335284	291062	2,66	30
Villes Fr	1108,25	779,49	228,58	79,37%	95589	65885	2.36	13
Poly En	8963,46	5960,48	1761,14	80,35%	717112	480049	2,17	6
Catégorie 2								
Web Fr	3560,97	3281,09	1007,3	71,71%	298117	274175	2,17	15
Web Bg	2503,96	2171,66	647,62	74,14%	209209	181960	2,36	17
Web Hu	3235,67	2772,43	839,2	74,06%	270495	236165	2,11	14
Web Pt	6510,09	5652,24	1742,01	73,24%	538697	474455	3,14	32

TAB. B.5 – Caractéristiques de la factorisation à $Taille_{alphabet} = 1024$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2178,22	1911,52	572,74	73,71%	177465	154512	2.21	9
DELAF En	3081,12	2587,93	774,32	74,87%	252664	209339	2,42	12
DELAF Sr	2340,47	2152,76	653,37	72,08%	193668	176578	2,13	7
DELAF De	4133,67	3566,84	1093,05	73,56%	335284	286666	2,56	30
Villes Fr	1108,25	755,89	222,5	79,92%	95589	62441	2,95	13
Poly En	8963,46	5787,01	1762,15	80,34%	717112	463863	2,14	6
Catégorie 2								
Web Fr	3560,97	3266,44	1036,56	70,89%	298117	271885	2,39	15
Web Bg	2503,96	2155,05	664,88	73,45%	209209	179470	2,37	17
Web Hu	3235,67	2753,45	862,48	73,34%	270495	233463	2,32	14
Web Pt	6510,09	5599,83	1783,4	72,61%	538697	468519	3,33	32

TAB. B.6 – Caractéristiques de la factorisation à $Taille_{alphabet} = 2048$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2178,22	1905,77	591,38	72,85%	177465	152464	2,1	9
DELAF En	3081,12	2569,96	795,65	74,18%	252664	205957	2,33	12
DELAF Sr (3043)	2340,47	2151,46	675,08	71,16%	193668	175583	2,08	7
DELAF De	4133,67	3538,18	1120,27	72,90%	335284	282430	2,39	30
Villes Fr (3321)	1108,25	742,56	227,48	79,47%	95589	59895	2,82	13
Poly En	8963,46	5663,06	1779,12	80,15%	717112	451423	2,22	13
Catégorie 2								
Web Fr	3560,97	3262,12	1069,41	69,97%	298117	269765	2,21	15
Web Bg	2503,96	2148,37	685,95	72,61%	209209	177031	2,28	17
Web Hu	3235,67	2743,78	889,36	72,51%	270495	230814	2,26	14
Web Pt	6510,09	5555,04	1828,42	71,91%	538697	462460	3,19	32

TAB. B.7 – Caractéristiques de la factorisation à $Taille_{alphabet} = 4096$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr (4332)	2178,22	1903,77	610,97	71,95%	177465	152228	2,1	9
DELAF En (7285)	3081,12	2564,69	821,54	73,34%	252664	202769	2,18	12
DELAF Sr								
DELAF De (5750)	4133,67	3535,14	1155,7	72,04%	335284	280776	2,27	30
Villes Fr								
Poly En	8963,46	5576,24	1809,83	79,81%	717112	440259	2,42	13
Catégorie 2								
Web Fr (4822)	3560,97	3260,94	1103,25	69,02%	298117	269039	2,18	15
Web Bg (5270)	2503,96	2146,69	708,54	71,70%	209209	175857	2,21	17
Web Hu (5833)	3235,67	2740,8	918,43	71,62%	270495	229077	2,18	14
Web Pt (7883)	6510,09	5531,69	1881,55	71,10%	538697	456847	2,72	32

TAB. B.8 – Caractéristiques de la factorisation à $Taille_{alphabet} = 8192$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr								
DELAF En								
DELAF Sr								
DELAF De								
Villes Fr								
Poly En (15013)	8963,46	5550,79	1860,69	79,24%	717112	430366	2,31	11
Catégorie 2								
Web Fr								
Web Bg								
Web Hu								
Web Pt								

TAB. B.9 – Caractéristiques de la factorisation à $Taille_{alphabet} = 16384$

B.3 Caractéristiques des factorisations d'un automate minimal des mots inversés (Factorisations $FCM_{inverse}$)

La table B.10 présente les caractéristiques de la compression des automates minimaux représentant les mots inversés des dictionnaires lorsque l'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 128$. À titre d'exemple, initialement, la taille de l'automate DELAF français est égale à 2 620 Ko, suite à la factorisation, sa taille diminue à 2 517 Ko, ensuite vient la compression qui réduit l'automate à 651 Ko, ce qui représente un taux de compression de 75%. Ainsi, le nombre de transitions initialement à 212 692 diminue jusqu'à atteindre 205 240. La longueur moyenne des séries factorisées est 2 et la plus longue série factorisée est de longueur 2.
À cette étape, la factorisation est plus efficace, car elle permet d'atteindre l'optimum pour sept des dix automates. Dans ce tableau, on peut aussi remarquer que la majorité des séries qui ont été choisies par la fonction gain sont de longueur 2. Ce choix vient de la forte redondance de ces petites séries.

Dans le tableau B.11 on a fixé la taille maximale de l'alphabet à $Taille_{alphabet} = 256$, on peut voir que le seul résultat qui a été amélioré se produit sur l'automate du web portugais. Pour les autres, le taux de compression a diminué parce que le gain obtenu par la factorisation des séries n'a pas compensé l'utilisation d'un bit supplémentaire pour coder de nouveaux caractères et étendre plus l'alphabet.

Le tableau B.12 présente les caractéristiques de la factorisation lorsqu'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 512$. On peut voir que les résultats obtenus sur les automates des villes françaises et les polylexicaux anglais ont encore été améliorés.

Lorsqu'on fixe la taille maximale de l'alphabet à $Taille_{alphabet} = 1\,024$, les résultats de la factorisation et compression présentés dans le tableau B.13 montrent que les automates des polylexicaux anglais et des villes françaises ont encore perdu en taille pour atteindre leur optimum. En effet, à cette étape on constate que la factorisation de 512 séries supplémentaires permet d'éliminer suffisamment de transitions dans les automates pour réduire leur taille.

La table B.16 indique que le seul automate possédant 8 192 séries factorisées et qui continue à avoir un gain positif est l'automate des polylexicaux anglais. Aucun des autres n'a atteint cette limite (le calcul du gain présente toujours une valeur négative ou nulle). À l'étape suivante ($Taille_{alphabet} = 16384$), le nombre maximum de factorisations pour tous les automates est atteint.

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2620,85	2517,89	651,57	75,14%	212692	205240	2	2
DELAF En	3879,99	3704,98	994,02	74,38%	316237	301540	2,02	3
DELAF Sr	4292,5	4145,42	1123,17	73,83%	346158	340719	2	2
DELAF De	6176,49	5845,06	1579,49	74,43%	496918	479181	2	2
Villes Fr	1109,33	896,44	238,81	78,47%	95472	78190	2,28	5
Poly En	9045,26	6951,95	1928,13	78,68%	725223	564038	2,25	8
Catégorie 2								
Web Fr	3726,56	3570,78	969,79	73,98%	306827	294183	2	2
Web Bg	2777,86	2587,99	677,89	75,60%	228690	213524	2	2
Web Hu	3490,55	3233,16	874,25	74,95%	285820	265195	2	2
Web Pt	6931	6482,55	1816,07	73,80%	566772	531255	2,32	16

TAB. B.10 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 128$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2620,85	2473,08	663,73	74,68%	212692	201236	2,01	3
DELAF En	3879,99	3610,59	1003,28	74,14%	316237	293394	2,01	3
DELAF Sr	4292,5	4123,77	1158,04	73,02%	346158	338672	2	2
DELAF De	6176,49	5727,66	1601,99	74,06%	496918	468532	2,18	31
Villes Fr	1109,33	832,31	228,95	79,36%	95472	71975	2,12	5
Poly En	9045,26	6488,43	1790,31	80,21%	725223	523604	2,29	8
Catégorie 2								
Web Fr	3726,56	3525,93	992,36	73,37%	306827	290196	2	2
Web Bg	2777,86	2540,65	690,22	75,15%	228690	209246	2,02	6
Web Hu	3490,55	3165,26	854,75	75,51%	285820	259190	2	2
Web Pt	6931	6346,05	1774,57	74,40%	566772	518993	2,31	21

TAB. B.11 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 256$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2620,85	2449,17	681,36	74,00%	212692	198974	2,05	8
DELAF En	3879,99	3561,16	1024,66	73,59%	316237	289074	2,05	16
DELAF Sr	4292,5	4112,95	1196,16	72,13%	346158	337517	2,11	6
DELAF De	6176,49	5658,35	1637,93	73,48%	496918	462255	2,45	31
Villes Fr	1109,33	794,02	226,02	79,63%	95472	68188	2,07	6
Poly En	9045,26	6118,92	1742,56	80,74%	725223	491828	2,27	8
Catégorie 2								
Web Fr	3726,56	3500,01	1020,06	72,63%	306827	287740	2,06	18
Web Bg	2777,86	2514,6	708,06	74,51%	228690	206705	2,24	18
Web Hu	3490,55	3122,46	873,89	74,96%	285820	255268	2,06	15
Web Pt	6931	6260,19	1810,92	73,87%	566772	511025	2,81	36

TAB. B.12 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 512$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2620,85	2435,69	701,85	73,22%	212692	197415	2,12	8
DELAF En	3879,99	3530,76	1050,94	72,91%	316237	286096	2,22	16
DELAF Sr	4292,5	4108,62	1236,21	71,20%	346158	336740	2,12	6
DELAF De	6176,49	5612,45	1680,97	72,78%	496918	457953	2,74	31
Villes Fr	1109,33	765,74	217,98	80,35%	95472	65098	2,4	12
Poly En	9045,26	5838,8	1717,13	81,02%	725223	468021	2,2	8
Catégorie 2								
Web Fr	3726,56	3482,17	1050,1	71,82%	306827	285785	2,31	18
Web Bg	2777,86	2495,07	727,62	73,81%	228690	204520	2,38	18
Web Hu	3490,55	3093,99	896,73	74,31%	285820	252396	2,17	15
Web Pt	6931	6197,63	1854,11	73,25%	566772	505041	3,12	36

TAB. B.13 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 1024$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr	2620,85	2430,03	724,54	72,35%	212692	196020	2,13	8
DELAF En	3879,99	3509,95	1079,7	72,17%	316237	283325	2,37	16
DELAF Sr (1651)	4292,5	4107,48	1277,19	70,25%	346158	336113	2,07	6
DELAF De	6176,49	5582,59	1727,92	72,02%	496918	454432	2,56	31
Villes Fr	1109,33	742,93	220,07	80,16%	95472	61752	2,91	12
Poly En	9045,26	5644,12	1709,62	81,10%	725223	449974	2,15	8
Catégorie 2								
Web Fr	3726,56	3469,69	1082,06	70,96%	306827	283849	2,38	18
Web Bg	2777,86	2481,26	748,98	73,04%	228690	202246	2,53	18
Web Hu	3490,55	3074,39	921,86	73,59%	285820	249670	2,33	15
Web Pt	6931	6146,9	1900,08	72,59%	566772	499291	3,32	36

TAB. B.14 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 2048$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr (2982)	2620,85	2428,55	748,64	71,44%	212692	195086	2,09	8
DELAF En	3879,99	3502,02	1112,99	71,31%	316237	280859	2,27	16
DELAF Sr								
DELAF De	6176,49	5567,67	1780,05	71,18%	496918	451416	2,33	31
Villes Fr (3328)	1109,33	729,61	225	79,72%	95472	59192	2,81	12
Poly En	9045,26	5506,66	1720,5	80,98%	725223	436447	2,2	12
Catégorie 2								
Web Fr	3726,56	3466,14	1116,3	70,04%	306827	281801	2,19	17
Web Bg	2777,86	2478,2	774,04	72,14%	228690	200198	2,26	18
Web Hu	3490,55	3065,77	951,05	72,75%	285820	247112	2,26	15
Web Pt	6931	6103,43	1949,81	71,87%	566772	493430	3,18	36

TAB. B.15 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 4096$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr								
DELAF En (4746)	3879,99	3500,85	1148,23	70,41%	316237	280209	2,23	16
DELAF Sr								
DELAF De (5750)	6176,49	5565,66	1836,16	70,27%	496918	450329	2,26	31
Villes Fr								
Poly En	9045,26	5413,94	1747,29	80,68%	725223	424913	2,36	12
Catégorie 2								
Web Fr (4130)	3726,56	3466,11	1151,78	69,09%	306827	281767	2,19	17
Web Bg (2026)	2777,86	2481,29	799,59	71,22%	228690	202268	2.537802	18
Web Hu (5485)	3490,55	3063,41	982,14	71,86%	285820	245725	2,19	15
Web Pt (5342)	6931	6088,59	2008,25	71,03%	566772	490938	3,1	35

TAB. B.16 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 8192$

Automate minimal	Taille initiale (Ko)	Taille après factorisation (Ko)	Taille après compression	Taux compression	Nb transitions avant	Nb transitions après	Longueur moyenne	Longueur maximale
Catégorie 1								
DELAF Fr								
DELAF En								
DELAF Sr								
DELAF De								
Villes Fr								
Poly En	9045,26	5384,23	1795,04	80,15%	725223	414789	2,27	12
Catégorie 2								
Web Fr								
Web Bg								
Web Hu								
Web Pt								

Tab. B.17 – Caractéristiques de la factorisation des automates des mots inversés à $Taille_{alphabet} = 16384$