L'Inférence Grammaticale au pays des Apprentissages Automatiques : Discussions sur la coexistence de deux disciplines

Jean-Christophe Janodet

Habilitation à Diriger des Recherches

présentée par Jean-Christophe Janodet

soutenue publiquement le 3 décembre 2010

relue et corrigée le 3 janvier 2011

L'Inférence Grammaticale au pays des Apprentissages Automatiques : Discussions sur la coexistence de deux disciplines

---

Jury

M. Ricard Gavaldà	Professeur à l'UPC Barcelona	Rapporteur
M. Rémi Gilleron	Professeur à l'Université Lille 3	Rapporteur
M. Colin de la Higuera	Professeur à l'Université de Nantes	Examinateur
M. Pierre Lescanne	Professeur à l'Ecole Nationale Sup. de Lyon	Examinateur
M. Pascal Poncelet	Professeur à l'Université Montpellier 2	Rapporteur
M. Marc Sebban	Professeur à l'Université de Saint-Etienne	Examinateur

A Emmanuelle,
ALEXANDRE J *et*
Codile, le Chamillou.

Remerciements

Mes remerciements vont d'abord aux membres du jury, en particulier Ricard Gavaldà, Rémi Gilleron et Pascal Poncelet qui ont accepté de rapporter sur ce manuscrit, plus long que d'usage, écrit en français. J'ai précieusement noté toutes les remarques qu'ils m'ont faites, souvent en voix *off*, et j'ai beaucoup apprécié d'avoir pu débattre avec eux. Je remercie également Pierre Lescanne, président du jury, que j'ai sollicité à plusieurs reprises par le passé avec des questions de recherche ou de dossiers.

Mes remerciements vont ensuite à Colin de la Higuera, qui m'a formé au métier d'enseignant-chercheur au moins autant qu'il m'a initié à l'Inférence Grammaticale. Il est rare de trouver un binôme avec qui la collaboration soit toujours facile, l'écriture efficace, et les trouvailles étonnantes. Je remercie également Marc Sebban, qui m'a formé en Apprentissage Automatique et qui dirige aujourd'hui l'Informatique stéphanoise.

De façon plus large, je remercie tous les (anciens) doctorants et les étudiants avec qui j'ai eu le plaisir de faire un bout de route : Rémi Eyraud, Frédéric Tantini, Emilie Samuel, Henri-Maxime Suchier, David Combe, Aziz Yahiaoui, Richard Roche, Zheng Chen, Mariette Serrayet, Aurélien Béraud et les autres. Beaucoup des résultats présentés dans ce manuscrit sont issus de nos travaux communs. Qu'ils me pardonnent les débordements enthousiastes dont je suis parfois capable !

Ce travail doit également beaucoup aux chercheurs avec qui j'ai collaboré ou simplement discuté. Ainsi, je voudrais remercier Richard Nock, Leo Becerra-Bonache, Christine Solnon, Guillaume Damiand, Gwenaël Richomme, Fabrice Philippe, Fabien Torre, Géraud Sénizergues, Baptiste Gorin, François Denis, Laurent Miclet, Antoine Cornuéjols et tous mes collègues informaticiens à Saint-Etienne. J'ai également une pensée pour mes collègues mathématiciens et physiciens dont j'apprécie beaucoup la proximité.

Remerciement spécial à Philippe Ezequel pour sa relecture minutieuse de ce manuscrit. Grâce à lui, j'ai appris à ne plus mélanger les "quelques" avec des "quels que", et les "pour tout" avec des "pour tous". De plus, j'ai découvert que les "vaincœurs" n'identifiaient qu'approximativement les "vainqueurs", et ce avec probabilité au moins 1 ...

Enfin, je remercie mon épouse et mes enfants de m'avoir supporté pendant ces dernières semaines ... Comme le grand Winston, je leur avais promis du sang et des larmes : ils n'ont pas été déçus :-)

Table des matières

Chapitre 1

Introduction

J'ai découvert l'Inférence Grammaticale en lisant le Journal Officiel. Il y a des lectures plus divertissantes, mais quand on postulait à la maîtrise de conférences en 2000, Galaxie n'existait pas encore : on téléchargeait un long fichier publié au J.O. par le Ministère, et c'est en parcourant ces lignes, puis en visitant le site web (aujourd'hui défunt) de l'EURISE que j'ai probablement entendu parler pour la première fois de ce domaine de recherche.

Jusqu'alors, et pendant toute la durée de ma thèse, c'était plutôt l'informatique "fondamentale" qui avait été au centre de mes préoccupations. J'avais travaillé sur des langages de programmation logico-fonctionnels, dans le contexte d'une toute petite communauté. Son Graal était de définir des langages universels, dans lesquels tous les paradigmes de programmation (logique, fonctionnel, impératif, à objet, concurrent, ...) auraient co-existé de façon harmonieuse, et tous les styles auraient ainsi pu être pratiqués par les programmeurs.

C'était plus précisément sur la sémantique opérationnelle de ces langages que j'avais travaillé, c'est-à-dire les mécanismes d'exécution des programmes. Ainsi, j'avais étudié et publié sur la réécriture de termes et de graphes, la relation de surréduction et les stratégies permettant d'exécuter un programme de façon optimale. Ma contribution principale avait été de définir une nouvelle classe de graphes, dits *admissibles*, sur laquelle notre sémantique opérationnelle était cohérente et complète. C'était une recherche intéressante, mais le côté très artificiel de nos arguments de vente, confirmé par la pénurie des moyens financiers pendant tout le déroulement de ma thèse m'ont laissé penser, en 2000, qu'une sortie du système universitaire pouvait être opportune[1].

Pourtant, j'avais également des armes pour postuler, en particulier un bagage suffisant pour comprendre et démontrer des théorèmes, des compétences sur les langages formels acquises en thèse que j'ai ensuite étoffées en enseignant la Théorie des Langages, une curiosité scientifique certaine et un goût tout aussi certain pour les os à ronger. Ainsi, ce sont ces arguments-là qui m'ont permis d'entamer une seconde "carrière" ; en 2000, c'étaient des arguments suffisamment convaincants pour qu'un Colin de la Higuera, fondateur et leader de

[1] Les langages universels sont-ils vraiment le futur des langages de programmation ? En facture d'orgues, il existe deux époques : la période baroque qui produisit les instruments sur lesquels jouait Jean-Sébastien Bach, et la période romantique, marquée par Aristide Cavaillé-Coll, qui construisit des instruments pour César Franck (et son merveilleux "Prélude, Fugue et Variations"). Or au début du XXième siècle, on tenta de construire des instruments sur lesquels les organistes auraient pu jouer les deux répertoires. Mais la complexité des problèmes techniques qui en résultait conduisit à des instruments qui étaient aussi inexploitables sur l'un que sur l'autre. On décida donc d'abandonner cette stratégie : aujourd'hui, les organistes choisissent un répertoire et se déplacent pour le jouer sur des orgues adaptées, à la satisfaction du public qui se déplace également.

l'Eurise me donne une chance. J'espère ne pas l'avoir gâchée.

J'ai utilisé mes premières années stéphanoises pour me former, et dans ce cadre, j'ai été fortement influencé par deux mentors. Le premier fut naturellement Colin de la Higuera, pour tous les aspects de l'Inférence Grammaticale. C'est probablement dans ce domaine que j'ai été le plus actif pendant ces 10 dernières années et c'est donc sur l'Inférence Grammaticale que portera l'essentiel de ce manuscrit. Mon second initiateur fut Marc Sebban avec qui je découvris de nombreuses facettes de l'Apprentissage Automatique. Au contraire de l'Inférence Grammaticale, dont la communauté est de taille modeste, ce domaine-là est si vaste qu'il semble impossible à circonscrire. En tout cas, je mentirais si je prétendais ne pas continuer de le découvrir, encore aujourd'hui.

Pendant toutes les années qui ont suivi, j'ai tiré parti des bénéfices de cette période de formation, en collaborant sur de nouveaux travaux, en encadrant des étudiants en Master et en Thèse, en tenant ma place dans les divers projets que nous avons pu monter. Pourtant, cette formation initiale s'est construite sur une ambiguïté fondamentale, sur laquelle je suis revenu maintes et maintes fois pendant ces 10 dernières années, et que j'ai finalement choisi d'utiliser comme fil conducteur de ce manuscrit.

Quelques mots sur l'Inférence Grammaticale

L'inférence grammaticale n'est un domaine de recherche autonome que depuis 1994, année de création de la conférence ICGI, *the International Colloquium in Grammatical Inference*. Mais les premiers travaux qui fondent cette discipline remontent à la fin des années 60 [Gol67], et le domaine est donc actif depuis près de 40 ans. En tant que communauté, elle est composée de chercheurs venant d'horizons divers et travaillant par ailleurs dans des domaines variés : la reconnaissance des formes, la linguistique computationnelle ou la linguistique classique, l'apprentissage automatique, la théorie de l'information, l'algorithmique, la bio-informatique, la théorie des langages, *etc.* La conférence du domaine est ICGI et elle se tient tous les deux ans, mais les workshops et les sessions dédiées à l'Inférence Grammaticale dans d'autres conférences comme ECML, NIPS ou IJCAI sont fréquents et permettent à la communauté de se rencontrer régulièrement. Côté littérature, on trouve des *surveys* ou des chapitres de livres sur la discipline à toutes les époques (par exemple [AS83, Mic90, Sak97, dlH06a]). En la matière, la référence la plus récente et la plus complète est le livre de Colin de la Higuera, centré sur l'Inférence Grammaticale [dlH10], sur lequel je m'appuierai souvent dans ce propos.

Dans un problème typique d'Inférence Grammmaticale, on considère d'abord une classe de *grammaires*. Ce terme est générique : il peut désigner les grammaires usuelles de mots, stochastiques ou non, mais aussi les automates d'arbres, les grammaires de graphes, en fait, tout système formel permettant de générer, reconnaître ou décrire des langages, classiques ou stochastiques. On suppose ensuite disposer d'informations sur l'une de ces grammaires, appelée la *grammaire cible*. Nous savons par hypothèse que cette grammmaire existe mais nous ne la connaissons pas. Quant aux informations, ce sont souvent des exemples de mots (ou d'arbres, ou de graphes) qui sont engendrés par la grammaire cible ; on peut aussi disposer de contre-exemples, ou bien avoir accès à un expert (oracle) répondant à des questions (requêtes). Le problème est alors de savoir si à partir des informations dont on dispose, on peut "apprendre" la grammaire cible. Evidemment, on ne peut y répondre qu'en définissant précisément ce qu'"apprendre" veut dire, donc en se donnant ce qu'on appelle un *paradigme* ou un *cadre d'apprentissage*. Ces derniers sont nombreux, souvent incomparables entre eux, et l'Inférence Grammaticale est riche de résultats positifs ou négatifs permettant d'apprécier

leur pertinence. Nous en verrons plusieurs exemples.

Un problème d'inférence grammaticale consiste donc à déterminer si telle classe de grammaires peut être apprise à partir de tel type d'informations dans tel cadre d'apprentissage. Quant aux applications, l'Inférence Grammaticale fournit des méthodes et des algorithmes qui permettent d'attaquer tout type de problème où une grammaire cible doit être identifiée, à partir de données symboliques. C'est par exemple le cas en *model checking* où on peut chercher à construire un automate de Mealy modélisant le comportement d'un logiciel en le soumettant à des campagnes de tests. Cet automate peut ensuite être utilisé pour vérifier des propriétés sur le logiciel, souvent exprimées dans une logique temporelle usuelle ou de façon équivalente, à l'aide d'un second automate, ce qui permet de tirer partie des nombreux algorithmes fournis par les théoriciens des langages.

Il est bon de noter, dès à présent, que les problèmes de classification n'entrent généralement pas dans le champ d'application de l'Inférence Grammaticale (non stochastique). A l'inverse, ce sont des problèmes centraux en Apprentissage Automatique, et ce dernier domaine a su développer des techniques spécialisées donc performantes, comme les SVM, pour les attaquer frontalement et les résoudre avec succès.

Principales contributions en Inférence Grammaticale

J'ai fait mes premiers pas dans ce domaine en travaillant sur l'*apprentissage des langages de mots infinis* avec Colin de la Higuera en 2000-2001. L'enjeu, franchement théorique, était de répondre à une question que nous avait posé Maurice Nivat, à savoir : quel type de langages de mots infinis peut-on apprendre si l'on ne dispose que de mots de tailles finies ? En pratique, cela aurait pu permettre, par exemple, de deviner la spécification d'un *chip-set* en observant ses réponses à des signaux, mais nous mentirions en disant que ce sont les applications qui ont guidé ce travail. Nous avons obtenu des résultats pour certains types d'ω-langages dits *sûrs*, que nous avons publiés dans la revue *Theoretical Computer Science* [dlHJ04], après avoir été présentés à la conférence ALT'01 [dlHJ01]. Au-delà de la satisfaction d'avoir "trouvé" après avoir "cherché", j'ai découvert à l'issue de ce travail qu'on pouvait voyager (loin) grâce à la Recherche, et rencontrer des chercheurs brillants et souvent célèbres (par exemple, Dana Angluin à Washington, en 2001).

Le second travail, plus conséquent, auquel j'ai contribué, s'est déroulé dans le cadre de la thèse de Rémi Eyraud, que j'ai co-encadrée avec Colin. Cette fois, c'est la communauté internationale en Inférence Grammaticale qui avait posé le problème : en 2002, estimant que l'apprentissage des automates était désormais bien compris, elle avait souhaité encourager les travaux portant sur l'apprentissage de la classe suivante dans la hiérarchie de Chomsky [Cho57], celle des langages hors-contextes. Aussi, elle avait organisé une compétition d'algorithmes, le challenge OMPHALOS, dont les résultats ont été proclamés à Athènes, lors d'ICGI'04 [SCvZ05]. Nous avons participé à cette compétition en nous intéressant à une sous-classe des langages hors-contextes, définis à l'aide de systèmes de réécriture de mots. Aidé par mes acquis en thèse sur les systèmes de réécriture, et par plusieurs discussions informelles avec Géraud Sénizergues, nous avons proposé un algorithme qui, s'il n'a pas gagné la compétition OMPHALOS, était capable d'inférer des langages inapprenables à l'époque. Outre la valorisation qu'en a fait Rémi dans sa thèse, nous avons tiré de ce travail un article paru dans la revue *Machine Learning* [EdlHJ07], qui étend la version d'ICGI'04 [EdlHJ04].

Enfin, le dernier thème conséquent que j'ai abordé en Inférence Grammaticale concerne l'*apprenabilité des langages en présence du bruit d'édition*. Ce type de bruit affecte la suite

des lettres qui composent les exemples d'apprentissage. C'est le bruit qui déforme les mots qu'on tape trop rapidement sur un clavier d'ordi**o**nat**ue**r : des lettres peuvent apparaître, disparaître, être remplacées par d'autres, être inversées, *etc.* Le bruit d'édition est intimement lié à la distance du même nom [Lev65] qu'on retrouve dans de nombreux domaines comme la biologie computationnelle [Gus97, DEKM98], les modèles de langages [ASVB01] et la reconnaissance des formes. Dans un problème de classification de mots, il est possible de traiter ce bruit avec des techniques d'apprentissage automatique, comme nous le verrons plus loin. Cependant, le sujet qui nous intéressait ici était un "vrai" problème d'Inférence Grammaticale, où on cherchait à déterminer quelles classes de langages pouvait être apprises lorsque les informations étaient bruitées, et sous quels paradigmes d'apprentissage on pouvait le faire.

Nous avons abordé cette question de diverses façons. Ainsi, dans [dlHJT06], nous avons utilisé des notions de (pré-)topologie. Mais nos avancées décisives ont été faites lors de la visite de Leonor Becerra-Bonache (Université de Tarragone, aujourd'hui à Saint-Etienne) entre février et décembre 2006 : nous avons démontré que si un algorithme avait accès à un expert capable de *corriger* des exemples soumis par l'apprenant, dans un contexte d'apprentissage dit *actif*, alors certains types de grammaires pouvaient être efficacement appris ; c'est en particulier le cas des boules de mots (pour la distance d'édition), sur lesquelles porteront plusieurs développements dans ce manuscrit. Ces travaux ont été publiés dans *Journal of Machine Learning Research* [BBdlHJT08], après avoir été présenté à ECML'07 [BBdlHJT07].

Par la suite, ce travail s'est poursuivi de maintes façons dans un cadre plus large que l'identification à partir de données bruitées. D'une part, nous avons entrepris l'étude de l'apprenabilité des boules dans d'autres paradigmes [dlHJT08], et d'autres chercheurs (comme Fabien Torre) ont tenté de les utiliser pour résoudre des problèmes de classification de mots [TTT10]. De plus, j'ai entrepris des travaux plus fondamentaux sur la combinatoire de ces boules pendant l'année sabbatique que j'ai passée au LIRMM entre octobre 2009 et juin 2010, avec Gwenaël Richomme et Fabrice Philippe, travaux qui se poursuivent toujours actuellement. Enfin, l'apprentissage à partir de requêtes fut au cœur de la compétition ZULU qui s'est conclue lors d'ICGI'10 à Valencià cet automne [CdlHJ09], après deux ans de préparation et les travaux de recherche en Master 2 de David Combe (Saint-Etienne) et Myrtille Ponge (Nantes). Quant aux requêtes de correction elles-mêmes, elles furent l'objet de plusieurs articles présentés lors d'ICGI'08 à Saint-Malo [Kin08, Tir08].

Excursion aux pays des Apprentissages Automatiques

Trouver une définition consensuelle de ce que recouvre l'Apprentissage Automatique est beaucoup plus difficile que de trouver celle qui caractérise l'Inférence Grammaticale. Je ne pense pas me tromper en disant qu'il existe deux approches :

Une approche discriminative : c'est le point de vue adopté par Christopher Bishop dans son livre [Bis06], qui ne parle que d'Optimisation et de Statistiques sous son titre pourtant fédérateur de *Pattern Recognition and Machine Learning*. Il omet donc volontairement toutes les autres déclinaisons possibles de ce qu'"apprendre automatiquement" pourrait signifier en 2010, comme au cours des 60 années qui ont précédé. Dans ce contexte, mêmes les traditionnels arbres de décision ont disparu de la table des matières. Malheureusement, dans le système français, cette façon de concevoir l'Apprentissage Automatique est plutôt défavorable aux informaticiens. En effet, les mathématiciens et les automaticiens ont autant de légitimité

que nous pour la porter, ce qui repousse l'Apprentissage aux frontières de la discipline Informatique, telle qu'identifiée par le CNU 27 par exemple. Par ailleurs, les étudiants que nous formons dans nos propres Masters, recrutés parce qu'imprégnés d'une culture informatique traditionnelle, hésiteront naturellement à se lancer dans des thèses en Apprentissage, sachant qu'ils n'ont pas les bases suffisantes en Mathématiques pour réussir dans ce domaine. Sur ce sujet, la lecture de [CM08] est indispensable.

Une approche fédérative : c'est l'approche suivie par Antoine Cornuéjols et Laurent Miclet dans leur livre [CM10], où sous le titre d'Apprentissage Artificiel plutôt que d'Apprentissage Automatique, ils font un bilan quasiment exhaustif de toutes les conceptions de l'apprentissage qui sont portées par les communautés qui s'en revendiquent. Dans ce contexte, les arbres de décision, l'Inférence Grammaticale, les modèles graphiques et la Programmation Logique Inductive ont tous une place. Je m'associe évidemment à cette démarche, mais souhaite m'en écarter sur un point : les auteurs fondent l'Apprentissage Artificiel sur une synthèse remarquable des diverses approches théoriques de l'induction[2]. Ils se démarquent ainsi clairement de l'approche de Tom Mitchell dans [Mit97], qui avait probablement renoncé à proposer une théorie unificatrice de l'Apprentissage Artificiel, devant la difficulté d'une telle entreprise. Cependant, l'Inférence Grammaticale (stochastique ou non) n'a pas ces fondements-là, tout comme d'autres thématiques en Apprentissage Artificiel.

Ainsi, je trouve plus simple de considérer que l'Apprentissage Artificiel n'a pas de fondement, sinon épistémologique. C'est un conglomérat de disciplines diverses qui n'ont, comme seul point commun, que le fait de tenter de définir et d'exploiter, chacune à sa manière, ce qu'apprendre veut dire. Ces disciplines sont autonomes mais souvent complémentaires, au sens où elles s'intéressent parfois aux mêmes problèmes ou aux mêmes techniques. L'Inférence Grammaticale est l'une d'entre elles. La Classification Supervisée, présentée ci-après, en est une autre. Et les disciplines couvertes par le livre de Bishop en sont d'autres encore, qui ne peuvent se traduire qu'en problèmes d'Optimisation si tels sont leurs objets. Toutes devraient pouvoir participer à une même conférence qu'on appellerait CAP par exemple, la Conférence en Apprentissage (Artificiel).

Contributions en Classification Supervisée

Comme je l'ai déjà évoqué, ma formation à cette discipline coïncide avec l'arrivée de Marc Sebban à l'EURISE, en 2002. Ensemble, nous nous sommes intéressés à l'apprentissage d'automates à partir de données bruitées, mais cette fois avec l'objectif d'utiliser les algorithmes développés en Inférence Grammaticale (non probabiliste) pour traiter des problèmes en Classification de séquences. Ce problème ne relève plus de l'Inférence Grammaticale *stricto sensu*, dans la mesure où aucune grammaire cible n'est à trouver : on cherche seulement à trouver un bon classifieur, dont l'erreur en généralisation soit la plus petite possible (*i.e.*, dont le pouvoir prédictif soit le plus grand possible). D'ailleurs, les anglo-saxons ne rangent pas ce type de problèmes sous le vocable de *Grammatical Inference* : ils parlent plutôt de *Grammar Induction*.

[2] Je pense en particulier à l'étude et la mise en œuvre des critères inductifs, comme ERM, dans le chapitre 2, ou l'exposé des techniques comme la *cross-validation* visant à estimer la qualité d'une hypothèse dans le chapitre 3.

Pour traiter les données bruitées, nous avons développé deux types de méthodes. Les premières (méthodes *wrapper*) visent à modifier des algorithmes standard en Inférence Grammaticale (comme RPNI ou EDSM) pour qu'ils tiennent compte du bruit pendant la phase d'apprentissage. Ce travaux ont donné lieu à une publication à ICML'03 [JS03] ; ils nous ont également permis de participer, avec Frédéric Tantini pendant son TER, à la compétition "Learning DFA from noisy data", organisée par Simon Lucas lors de la conférence GECCO'04 [JTS04]. Les secondes méthodes (dites *filter*) sont issues du domaine de la Réduction de Données, sur lesquelles Marc avait beaucoup travaillé par le passé. Elles consistent à "nettoyer" les données avant tout processus d'apprentissage, en détectant celles qui sont probablement bruitées. Nous n'avons pas publié ces travaux (sauf dans [SJY02]), mais ils étaient au cœur du DEA d'Aziz Yahiaoui [Yah03], ont servi de base à Pierre Dupont dans [Dup06] et nous ont permis de développer une nouvelle *méthode d'ensemble* [Die00] dont il convient de parler maintenant.

C'est en fait à l'occasion de la visite de Richard Nock à l'EURISE, entre décembre 2003 à février 2004, que nous avons progressé sur cette voie. C'est probablement un des chercheurs les plus brillants que j'ai jamais rencontré. Suivre et comprendre ses développements, et plus encore tenter d'y apporter une touche personnelle, a nécessité un travail considérable (que mon épouse n'a pas toujours approuvé) mais il m'a permis d'acquérir les bases d'un de ses domaines de prédilection, le *boosting*.

Cette technique permet d'optimiser les performances de n'importe quel algorithme d'apprentissage (faible). Nous souhaitions l'utiliser pour améliorer nos algorithmes *wrapper*. Or dans le cadre des données bruitées, les performances de l'algorithme standard (ADABOOST) s'effondrent. Avec Richard, nous avons donc développé une nouvelle procédure, utilisant les services d'un *oracle* capable d'estimer si tel ou tel exemple était bruité ou non. Et pour simuler un tel oracle, nous avons utilisé les méthodes *filter* évoquées précédemment.

Ainsi, nous avons mis au point une nouvelle procédure de boosting, efficace en présence de données bruitées, puis l'avons utilisée en tirant parti de toutes les techniques que nous avions précédemment développées. Ce travail a donné lieu à une publication à ICML'05 [JNSS04] puis un article dans la *Revue d'Intelligence Artificielle* [JNSS05] (notre choix n'a sans doute pas été des plus judicieux). Plus tard, le boosting a été au cœur de la thèse d'Henri-Maxime Suchier que j'ai co-encadrée avec Marc Sebban. Enfin, nous avons entrepris d'autres travaux sur le boosting, cette fois pour traiter des données hétérogènes, qui sont récemment parus dans la revue *Fundamenta Informaticæ* [JSS09].

Vers un plan

En concevant le plan de ce manuscrit, j'ai décidé de me focaliser sur l'Inférence Grammaticale et la Classification Supervisée de séquences (vue comme une discipline de l'Apprentissage Artificiel). Mais en faisant ce choix, j'ai délibérément renoncé à inclure toute une partie de mes travaux récents. En effet, depuis que l'EURISE a disparu, en 2007, et que nous avons rejoint le laboratoire Hubert Curien, nous nous sommes rapprochés de collègues travaillant dans le domaine de l'Image. Ainsi, dans le cadre du projet ANR SATTIC, *String and Trees for Thumbnail Images Classification*, nous avons travaillé ensemble sur des problèmes de modélisation à l'aide de graphes [SdlHJ10], et obtenu des résultats très encourageants sur la résolution des problèmes d'isomorphisme de graphes plans [DdlHJ+09b, DdlHJ+09a]. C'est également sur ce thème qu'Emilie Samuel soutiendra sa thèse début 2011. Ces travaux entrent probablement plus dans le champ de l'Algorithmique, discipline reine en Informatique, plutôt que dans celui

de l'Apprentissage Artificiel. Cela changera bientôt (comme nous le verrons en conclusion), mais c'était une raison suffisante pour ne pas l'évoquer dans ce manuscrit.

Malheureusement, choisir parmi mes travaux ne me donnaient pas pour autant les clés qui me permettraient de les organiser pour les présenter. Or au cours de mes nombreuses lectures préparatoires, je suis retombé sur cette ambiguïté initiale sur laquelle j'avais buté dès mes premières années stéphanoises. Quand on cotoie les communautés en Inférence Grammaticale et en Apprentissage Automatique, qu'on suit leurs conférences et leurs débats, il est manifeste que ces chercheurs ne parlent pas de la même chose quand ils parlent d'Apprentissage. Ainsi, je me suis dit qu'à l'occasion de cette rédaction, je pourrais profiter de mes propres travaux pour tenter d'y voir plus clair, en mettant en évidence des éléments qui pouvaient former le cœur de l'Inférence Grammaticale d'une part, et tenter de positionner ce domaine de recherche par rapport à celui de l'Apprentissage Automatique d'autre part. C'est ainsi qu'en mai 2010, j'ai décidé de faire un plan en deux parties.

Dans la première partie, j'ai voulu expliciter les caractéristiques fondamentales de l'Inférence Grammaticale. Cela ne consiste pas à décrire de façon exhaustive ce que ce domaine couvre (voir [dlH10] pour une vraie vue d'ensemble). En outre, je souhaitais m'appuyer sur mes travaux, et comme je n'ai pas (encore) creusé toutes les questions qui se posent dans ce domaine, j'ai fait des choix (drastiques). Je vais donc traiter mon sujet en deux chapitres, le premier évoquant les objets (grammaires) qu'on manipule (Chapitre 2) et le second, les deux notions d'*apprentissage* les plus fréquentes qu'on utilise dans ce domaine (Chapitre 3).

Puis dans la seconde partie, j'ai voulu m'intéresser au positionnement de l'Inférence Grammaticale par rapport à l'Apprentissage Automatique, et plus précisément la Classification Supervisée de séquences. Pour cela, j'ai choisi de donner, dans le Chapitre 4, des exemples frappants où les disciplines divergent : je montrerai en particulier que certains fondements de l'Apprentissage Automatique n'apportent pas d'éclairage particulièrement intéressant en Inférence Grammaticale (non stochastique), puis je donnerai un exemple de thématique, l'apprentissage actif, sur laquelle les deux disciplines ont des points de vue orthogonaux. Enfin, dans le Chapitre 5, je donnerai un exemple de travail où les deux disciplines ont des objectifs convergents.

Première partie

Une esquisse du cœur de l'Inférence Grammaticale

Préambule

L'objectif de cette première partie est de dégager quelques éléments qui forment le cœur de l'Inférence Grammaticale, prise comme une discipline autonome. Plusieurs livres (ou chapitres de livres, ou *surveys*) sont consacrés à cette discipline [dlH10, Mic90, Sak97, dlH05], et comme l'objectif de cet exposé n'est pas d'écrire un livre, nous avons nécessairement dû faire des choix.

Le choix des mots A l'origine, les problèmes d'Inférence Grammaticale concernaient des langages et des grammaires de *mots*. Ces travaux ont ensuite été étendus pour traiter des langages d'arbres [KBdBB03], de termes [GO93], de mots infinis [SY93], de graphes [LS98], de couples de mots [OGV93] (apprentissage de transducteurs), ou même de mots temporisés [VdWW09]. J'évoquerai rarement ces travaux par la suite : même si je les suis avec intérêt, j'y ai peu contribué, par choix (sauf dans [dlHJ04]).

En effet, j'ai passé l'essentiel de ma thèse à étudier les extensions possibles de résultats connus sur la réécriture de *termes du premier ordre* à la réécriture de *graphes admissibles*, une forme d'arbres infinis rationnels. Pendant ces années, j'ai souvent eu l'impression que nos idées étaient judicieuses, nos algorithmes efficaces, notre thèse convaincante. Et pourtant, nous avions systématiquement des problèmes pour les rendre compréhensibles à l'écrit, et plus encore à l'oral, à cause de la syntaxe très lourde nécessaire pour décrire ces données complexes.

De même en Inférence Grammaticale, travailler avec des données complexes introduit souvent des contraintes techniques spécifiques, lourdes à décrire puis à gérer, et qui nous éloignent, donc, du cœur de la discipline[3].

L'absence de probabilités Un second choix, beaucoup plus discutable pour qui prétend aborder le cœur de la discipline, consiste à ne pas parler d'Inférence Grammaticale *Stochastique*. Dans ce cadre, on ne cherche plus à apprendre des langages et des grammaires, mais des distributions de probabilités sur l'ensemble des mots, qu'on modélise sous la forme d'automates probabilistes [VTdlH$^+$05], d'automates à multiplicités [DE08], de grammaires stochastiques [Wet80], de n-grammes [Goo01], de modèles de Markov cachés [DDE05], *etc.*

Deux questions se posent généralement pour apprendre une telle machine :

1. quelle est la structure du modèle ? Typiquement, quel est le nombre d'états, ou quel est le système de transitions du modèle ?
2. quels sont les paramètres (probabilités élémentaires) du modèle ?

Ces problèmes ont été attaqués de deux manières. La première est effectivement liée au cœur de l'Inference Grammaticale : elle consiste à étendre les paradigmes d'apprentissage et les algorithmes développés dans un cadre non stochastique pour tenir compte de l'ajout de probabilités[4]. Ces extensions ne sont pas triviales : si un échantillon de données permet d'estimer les paramètres d'un modèle, il convient de vérifier qu'avec plus de données, cette estimation converge vers les paramètres réels du modèle cible. En conséquence, l'introduction

[3]Par exemple, l'extension des automates finis à états résiduels (Afer) aux arbres est redoutable, du fait de la non équivalence entre les automates ascendants et descendants [CGL$^+$03].

[4]On pensera par exemple à l'identification à la limite avec probabilité 1 [Ang88, dlHT00], ou à l'algorithme Alergia [CO94] basé sur Rpni [OG92].

de probabilités nécessite de s'attaquer à des problèmes plus spécifiques, que je choisis donc d'écarter.

La seconde approche pour résoudre les deux questions précédentes relève nettement plus de l'Apprentissage Statistique que de l'Inférence Grammaticale : elle vise à développer des techniques pour estimer les paramètres du modèle en utilisant, par exemple, l'algorithme EM ou des techniques de lissage [Goo01]. De même, de nouvelles techniques apparaissent pour déterminer sa structure avec, par exemple, une analyse en composantes principales [BDR09]. Si ce champ d'investigation est particulièrement actif, il me parait plus approprié de l'évoquer dans la seconde partie de ce manuscrit.

Plan de la première partie Pour finir, deux éléments me paraissent importants pour dessiner les contours de l'Inférence Grammaticale classique (*i.e.*, non stochastique). Le premier concerne les objets qu'on apprend, les grammaires de mots (Chapitre 2). Outre les langages traditionnels, j'ai beaucoup travaillé avec des langages définis par une métrique. L'occasion m'est ainsi donnée d'en présenter une synthèse. Le second aspect qui caractérise l'Inférence Grammaticale classique concerne les paradigmes d'apprentissage, spécifiant ce qu'apprendre veut dire. Ayant pratiqué les plus standard dans le contexte des automates et des boules de mots, je tenterai de montrer leurs caractéristiques et leurs limites. En outre, ils me permettront également d'aborder les difficultés qu'on rencontre dans le contexte de l'apprentissage des grammaires hors-contextes.

Chapitre 2

Les objets de l'Inférence Grammaticale

La plupart des travaux faits en Inférence Grammaticale reposent sur la théorie des langages de mots. Il est donc logique d'en faire une brève présentation. D'une part, nous avons besoin de notations que nous réutiliserons dans la suite de ce manuscrit et que nous souhaitons donc fixer dès à présent. D'autre part, nous avons beaucoup travaillé avec des boules de mots. Or comme ces langages ne sont pas standard, il semble utile de décrire quelques-unes de leurs propriétés.

2.1 Mots, langages, grammaires

2.1.1 Eléments de stringologie

Soit Σ un *alphabet*, c'est-à-dire un ensemble fini non vide de symboles appelés *lettres*. Un *mot* $w = a_1 \dots a_n$ est une séquence finie de lettres. On note Σ^* l'ensemble de tous les mots et λ le mot vide. La *concaténation* de deux mots u et v sera notée $u \cdot v$ ou simplement uv. On rappelle que $\langle \Sigma^*, \cdot, \lambda \rangle$ est un monoïde libre.

Etant donné un mot $w = a_1 a_2 \dots a_n$, on désigne par $|w|$ sa longueur et par $|w|_x$ le nombre d'occurrences d'une lettre $x \in \Sigma$ dans w. Les *facteurs* sont des sous-séquences contigües de lettres. On note $w[i..j]$ le facteur $a_i a_{i+1} \dots a_j$ pour tout $1 \leq i \leq j \leq n$; par convention, $w[i..j] = \lambda$ si $i > j$. De plus, on dit que $w[1..j]$ est un *préfixe*, et $w[i..n]$ un *suffixe* de w.

On distingue les facteurs des *sous-mots*, qui sont des sous-séquences potentiellement non contigües de lettres. Formellement, on dit qu'un mot u est un *sous-mot* de w, noté $u \preceq w$, si $u = a_1 \dots a_n$ et qu'il existe $v_0, \dots, v_n \in \Sigma^*$ tels que $w = v_0 a_1 v_1 \dots a_n v_n$. La relation $\preceq$ est un ordre partiel sur Σ^*. L'ensemble des *sous-mots communs* de u et v sera noté $u \wedge v = \{w \in \Sigma^* : w \preceq u \text{ et } w \preceq v\}$. Par exemple, $0011 \wedge 0101 = \{\lambda, 0, 1, 00, 01, 11, 001, 011\}$. Dans la suite, on notera $\ell(u, v) = \max_{w \in u \wedge v} |w|$ la longueur maximum d'un sous-mot commun de u et v, et $\textsc{lcs}(u, v) = \{w \in u \wedge v : |w| = \ell(u, v)\}$ l'ensemble de tous les plus longs sous-mots commun.

De nombreux ordres permettent de comparer des mots entre eux (comme $\preceq$). Parmi les ordres bien fondés, celui qu'on utilise le plus en Inférence Grammaticale est l'ordre *length-lexicographique* : on suppose donné un ordre total $\leq$ sur l'alphabet Σ, et on l'étend à l'ensemble

de tous les mots en posant, pour tout $w_1, w_2 \in \Sigma^*$:

$$w_1 \trianglelefteq w_2 \text{ ssi } \begin{cases} |w_1| < |w_2| \text{ ou} \\ |w_1| = |w_2| \text{ et } \exists u, v_1, v_2 \in \Sigma^*, \exists x_1, x_2 \in \Sigma \text{ tels que} \\ w_1 = ux_1v_1 \text{ et } w_2 = ux_2v_2 \text{ et } x_1 \leq x_2. \end{cases}$$

L'ordre $\trianglelefteq$ est total sur Σ^*, et si $\Sigma = \{0, 1\}$ avec $0 < 1$, alors on a $\lambda \trianglelefteq 0 \trianglelefteq 1 \trianglelefteq 00 \trianglelefteq 01 \trianglelefteq 10 \trianglelefteq 11 \trianglelefteq 000 \trianglelefteq \ldots$ On note $u \triangleleft v$ si $u \trianglelefteq v$ et $u \neq v$. On dit que deux mots $u, v \in \Sigma^*$ sont *consécutifs* si $u \triangleleft v$ et s'il n'existe pas $w \in \Sigma^*$ tel que $u \triangleleft w \triangleleft v$.

2.1.2 Langages formels et grammaires génériques

Toute partie $L \subseteq \Sigma^*$ s'appelle un *langage*. Les plus simples d'entre eux sont probablement les langages finis, dont on notera par $\mathcal{FIN}(\Sigma)$ la classe. Nous décrirons de nombreuses autres classes dans les sections suivantes.

En Inférence Grammaticale, on ne cherche généralement pas à apprendre un langage particulier à partir de données, mais à déterminer si un algorithme est capable d'identifier tout langage d'une classe $\mathcal{L}$ à partir de données. Or comme les langages sont potentiellement infinis, il est nécessaire de disposer d'une seconde classe $\mathcal{G}$ dite de *représentations* ou de *grammaires* permettant de décrire chaque langage de $\mathcal{L}$ de façon *finie*. Le terme de grammaire est ici utilisé de façon générique : il ne fait pas nécessairement référence aux grammaires de Chomsky, même si ces dernières servent souvent de représentations.

Formellement, les classes $\mathcal{L}$ et $\mathcal{G}$ seront supposées liées par une fonction dite de *nommage* $\mathbb{L} : \mathcal{G} \to \mathcal{L}$ qui est totale ($\forall G \in \mathcal{G}, \mathbb{L}(G) \in \mathcal{L}$) et surjective ($\forall L \in \mathcal{L}, \exists G \in \mathcal{G} : \mathbb{L}(G) = L$).

Ce seront les grammaires qui seront les résultats des algorithmes d'identification et non les langages eux-mêmes. Aussi, lorsque nous décrirons les classes de grammaires utilisées dans ce manuscrit, nous donnerons également des informations sur trois de leurs caractères, qui interviennent lors la conception et de l'analyse des algorithmes.

Le premier concerne la taille d'une grammaire $G \in \mathcal{G}$ qu'on notera par $\|G\|$. On supposera toujours que cette valeur est polynomialement liée aux nombres de bits nécessaires pour encoder G, quelle que soit sa nature. Dans le cas des langages finis, on peut représenter les langages par eux-mêmes, et dans ce cas, $\|L\|$ désignera la somme des longueurs des mots de L (alors que $|L|$ ou $\#(L)$ désigneront la cardinalité de L).

Le second caractère concerne le problème du mot. On supposera toujours disposer d'un algorithme d'analyse syntaxique (*parser*) qui, étant donné un mot $w \in \Sigma^*$ et une grammaire $G \in \mathcal{G}$, décide du problème "$w \in \mathbb{L}(G)$?". Dans l'affirmative, on dira que G *reconnaît* w. Un parser fait un calcul (comme de construire un arbre de dérivation) pour déterminer si un mot appartient au langage reconnu par une grammaire. Comme la plupart des paradigmes d'apprentissage que nous envisagerons seront soumis à des contraintes d'efficacité, nous exigerons en général que le parser considéré soit un algorithme polynomial en $\|G\|$ et $|w|$.

Enfin, le dernier caractère concerne la décidabilité du problème de l'équivalence de grammaires. On dira que deux grammaires G_1 et G_2 sont *équivalentes* si $\mathbb{L}(G_1) = \mathbb{L}(G_2)$.

2.2 Les langages standard

Comme dans la plupart des branches de l'Informatique, les langages qu'on considère traditionnellement en Inférence Grammaticale sont ceux de la *hiérarchie de Chomsky* [Cho57]. Dès

sa conception, cette hiérarchie visait à classifier les langages selon la forme la plus simple des grammaires (de Chomsky) qui les engendrent. La classe $\mathcal{FIN}(\Sigma)$ des langages finis forment la couche 4 de la hiérarchie, et nous allons décrire les couches 3 et 2 dans cette section, de manière à fixer nos notations.

Notons cependant que cette hiérarchie est largement remise en cause aujourd'hui, en particulier par les linguistes, convaincus que les langues naturelles sont composées d'éléments appartenant à plusieurs des couches et sont donc orthogonales à la description initiale de Chomsky [AvZ04]. En Inférence Grammaticale, elle s'avère inadaptée lorsqu'il s'agit par exemple de déterminer les langages identifiables à partir de données bruitées. De plus, elle n'est pas suffisamment discriminante, puisque qu'elle regroupe souvent dans une même couche, des langages de nature profondément différente (par exemple, dans la couche 2, des langages déterministes, des langages non linéaires, des langages ambigus). Nous en verrons un aperçu lorsque nous présenterons les problèmes liés à l'apprenabilité des grammaires hors-contextes.

2.2.1 Langages rationnels et automates

La classe des langages rationnels sera notée $\mathcal{RAT}(\Sigma)$. Ils forment la couche 3 de la hiérarchie de Chomsky. Ce ne sont pas les langages les plus simples que l'on puisse imaginer[1], mais ceux qui ont été le plus étudiés en Inférence Grammaticale. Nous les utiliserons d'ailleurs pour illustrer les définitions des cadres d'apprentissage standard. Il existe de nombreuses manières équivalentes de les introduire (en utilisant le théorème de Kleene). La plus "économe" consiste à passer par les *automates finis déterministes* (AFD) dont la classe sera notée $\mathcal{AFD}(\Sigma)$.

Un *automate fini déterministe* est un quintuplet $A = \langle Q, \Sigma, i, F, \delta \rangle$ où Q est un ensemble fini d'états, $i \in Q$ est un état initial, $F \subseteq Q$ est un ensemble d'états finaux et $\delta : Q \times \Sigma \to Q$ est une fonction de transition[2]. Nous étendons δ à l'ensemble de tous les mots de Σ^* et définissons ainsi une fonction que nous notons de nouveau δ en posant $\delta(q, \lambda) = q$ et $\delta(q, wa) = \delta(\delta(q, w), a)$ pour tout $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$. Tout automate peut être complété avec un état puit, de sorte que δ soit une fonction totale. Quant à la taille $\|A\|$ d'un AFD A, c'est le nombre d'états. En effet, si l'alphabet est fixé, cette valeur est polynomialement liée au nombre de bits nécessaires pour encoder A (voir [Den01] pour une définition rigoureuse d'un encodage possible).

On définit le langage reconnu par A en posant $\mathbb{L}(A) = \{w \in \Sigma^* : \delta(i, w) \in F\}$ et on dit qu'un langage L est *rationnel* s'il existe un AFD A tel que $\mathbb{L}(A) = L$. Dans nos exemples, nous utiliserons aussi les expressions régulières pour décrire ces langages.

On dit qu'un langage K est un *résiduel* de L s'il existe $u \in \Sigma^*$ tel que $K = u^{-1}L = \{w \in \Sigma^* : uw \in L\}$. Par le théorème de Myhill-Nerode [Ner58], on sait que L est rationnel si et seulement si L a un nombre fini de résiduels. En effet, si L est reconnu par un AFD (complet) $A = \langle Q, \Sigma, i, F, \delta \rangle$, alors pour tout $w \in \Sigma^*$, l'AFD $A_w = \langle Q, \Sigma, \delta(i, w), F, \delta \rangle$ reconnait le résiduel $w^{-1}L$. Comme A a un nombre fini d'états, le nombre de résiduels de L est fini.

Réciproquement, supposons que L ait un nombre fini de résiduels $L_1, L_2, \dots L_n$. Alors on définit $R = \langle Q, \Sigma, i, F, \delta \rangle$ tel que $Q = \{L_1, L_2, \dots L_n\}$, $i = \lambda^{-1}L = L$, $F = \{L_i : \lambda \in L_i\}$ et pour tout $a \in \Sigma$, $\delta(L_i, a) = L_j$ s'il existe $w \in \Sigma^*$ tel que $L_i = w^{-1}L$ et $L_j = (wa)^{-1}L$. On peut montrer que R est un AFD complet qui reconnait L. R est appelé l'*automate minimal* du langage L. Enfin, rappelons qu'aux noms des états près, cet automate minimal est unique et

[1] comme en témoigne l'imposant livre de J. Sakarowitch [Sak03].

[2] Nous ne considérerons que des automates déterministes par la suite.

peut être calculé en temps $\mathcal{O}(n \log n)$ à partir de n'importe quel AFD de n états qui reconnait L [HMU01].

En pratique, utiliser des AFD s'avère relativement simple. En effet, concernant le *parsing* d'un mot w, il suffit de vérifier que $\delta(i, w) \in F$ pour déterminer si $w \in \mathbb{L}(A)$, ce qui se fait en temps $\mathcal{O}(|w|)$. Par ailleurs, pour déterminer si deux AFD A_1 et A_2 sont équivalents, il suffit de les minimiser puis les comparer, ce qu'on peut réaliser en temps $\mathcal{O}(n_1 \log n_1 + n_2 \log n2)$.

2.2.2 Langages hors-contextes et grammaires

Pour décrire les langages de la couche 2 de la hiérarchie de Chomsky, les *langages hors-contextes*, il est nécessaire d'introduire la notion de *grammaire hors-contexte* (GHC) [ABB97]. Une telle grammaire est un quadruplet $G = \langle \Sigma, V, P, S \rangle$ où Σ est un alphabet de *symboles terminaux*, V est un alphabet de *variables* ou de *symboles non-terminaux*, $P \subseteq V \times (\Sigma \cup V)^*$ est un ensemble fini de *règles de production*, et $S \in V$ est appelé l'*axiome* (ou le symbole start). La taille $\|G\|$ de la grammaire sera donnée par $\sum_{(A \to u) \in P} |u|$.

Nous écrirons $uTv \Rightarrow uwv$ quand $(T, w) \in P$ et $u, v \in (\Sigma \cup V)^*$. $\Rightarrow^*$ désigne la fermeture réflexive et transitive de $\Rightarrow$. On écrira $u_0 \overset{k}{\Rightarrow} u_k$ s'il existe $u_0, \ldots, u_k \in \Sigma^*$ tels que pour tout $i \in 0, 1, \ldots (k-1)$, $u_i \Rightarrow u_{i+1}$. On note $\mathbb{L}(G)$ le langage $\{w \in \Sigma^* : S \Rightarrow^* w\}$, et un langage L est dit *hors-contexte* s'il existe une grammaire hors-contexte G telle que $\mathbb{L}(G) = L$. Enfin, on notera $\mathcal{LHC}(\Sigma)$ la classe des langages hors-contextes et $\mathcal{GHC}(\Sigma)$ celle des grammaires hors-contextes.

En pratique, et malgré leur importance dans des domaines comme la compilation, les GHC sont d'un emploi beaucoup plus complexe que les AFD. Concernant le parsing d'un mot w par une grammaire G, l'algorithme le plus connu, CYK, nécessite de mettre G sous une forme normale particulière (dite de Chomsky) avant l'analyse proprement dite, ce qui peut augmenter considérablement la complexité du parser. Malgré tout, les auteurs de [LL09] montrent comment réaliser l'analyse d'un mot en temps $\mathcal{O}(|G| \cdot |w|^3)$ à partir d'une GHC G quelconque, cette complexité incluant donc la mise sous forme normale de la grammaire. Il n'en reste pas moins que cette complexité est cubique en $|w|$, ce qui explique pourquoi de nombreux autres algorithmes beaucoup plus efficaces ont été développés pour des types particuliers de GHC (comme les grammaires LR). Enfin, concernant l'équivalence de GHC, le problème est "difficile à résoudre" puisqu'il est tout bonnement indécidable.

2.2.3 Autres types de langages usuels

De nombreuses autres classes de langages ont été étudiées en Inférence Grammaticale. Certaines sont en-deça de la couche 2 (comme la classe des langages sensibles au contexte, ou même celle des langages récursivement énumérables, étudiés en Inférence Inductive [JORS99]). D'autres sont obtenues à partir des couches 2, 3, 4 de la hiérarchie de Chomsky en imposant des contraintes supplémentaires sur les grammaires (conduisant par exemple aux automates k-réversibles [Ang82], ou aux grammaires linéaires déterministes [dlHO02a]).

D'autres enfin sont définies par des mécanismes autres que des grammaires, et n'entrent donc pas facilement dans la hiérarchie. C'est par exemple le cas des langages à base de motifs (*pattern languages*) [Ang79], des langages générés par des L-systèmes [Yok92], des langages *mildly-context-sensitive* [OABBA06] ou des langages congruentiels [BO93].

Outre leurs applications, ces derniers langages sont étudiés en Inférence Grammaticale soit parce qu'ils permettent d'établir de nouveaux résultats d'apprenabilité, soit parce que

leur étude met en évidence les faiblesses possibles des paradigmes d'apprentissage.

2.3 Les langages métriques

Cette famille de langages n'est pas standard. Nous avons été amené à la définir pour résoudre des problèmes d'identification à partir de données bruitées que nous présenterons plus loin. L'idée est relativement simple : si on munit Σ^* d'une métrique[3] $d()$, alors on dispose de tout un arsenal de langages définis par leurs propriétés topologiques. Typiquement, la boule de centre $o \in \Sigma^*$ et de rayon $r \geq 0$ est le langage des mots à distance plus petite que r de o : $\{w \in \Sigma^* : d(o, w) \leq r\}$. De même, la plupart des coniques (ellipses, hyperboles) admettent des définitions géométriques uniquement basées sur une distance, ce qui permet de les considérer sur l'espace des mots.

L'idée de faire de la géométrie avec des mots est séduisante. Mais notre expérience en la matière montre que les analogies avec la géométrie usuelle du plan tournent rapidement court : Σ^* n'est pas muni d'une structure d'espace affine. Aussi, ne disposant pas de vecteurs, nous aurons beaucoup de mal à définir la notion de droite par exemple, ou simplement à raisonner sur un dessin. Pire, l'intuition de la géométrie usuelle nous a souvent conduit sur des fausses pistes ... il convient de s'en méfier.

Parmi toutes les métriques qu'on peut envisager sur Σ^*, les distances d'édition jouent un rôle de premier plan. En effet, elles sont au cœur des problèmes de recherche approximative de motifs (*approximate string matching*) [Nav01] avec des applications nombreuses en bio-informatique [Gus97, DEKM98], dans les correcteurs orthographiques [SM02], la recherche de musique [LU00], *etc.* En apprentissage automatique, on les utilise pour modéliser le bruit dans des données textuelles [SS92, dlHJT06], ou comme paramètre pour faire de l'apprentissage par plus proches voisins [CNBYM01] ou de l'apprentissage par analogie [DM04].

2.3.1 Distances d'édition

Soit Σ un alphabet. On définit l'ensemble des *opérations d'édition* $Z = Z_d \cup Z_i \cup Z_s \cup \{(\lambda, \lambda)\}$ où $Z_d = \{(a, \lambda) : a \in \Sigma\}$ est l'ensemble des *délétions*, $Z_i = \{(\lambda, b) : b \in \Sigma\}$ l'ensemble des *insertions* et $Z_s = \Sigma \times \Sigma$ l'ensemble des *substitutions*. Nous ajoutons (λ, λ) à l'ensemble Z pour homogénéiser notre présentation ; cette opération ne joue généralement aucun rôle, sauf dans le cadre des distances d'édition stochastiques (où son coût ne peut pas être nul [RY98]).

De nombreuses autres opérations ont été proposées dans la littérature, selon les domaines d'application où cette distance était étudiée, par exemple l'*échange* de deux lettres adjacentes, ou des opérations très spécifiques comme le remplacement du facteur *ph* par la lettre *f* dans une application de traitement de la langue [BM00]. Dans la suite, nous n'utiliserons que les opérations standard.

On considère maintenant une fonction de *coût* $\mathcal{C} : Z \to \mathbb{R}^+$ sur les opérations d'édition avec les deux hypothèses suivantes :

1. Sauf énoncé contraire, on supposera que $\mathcal{C}(z)$ est un entier, pour tout $z \in Z$. En effet, le cas où les coûts sont rationnels peut toujours se ramener à des coûts entiers (à un facteur

[3] Dans tout ce manuscrit, j'utiliserai le mot *distance* pour désigner une fonction positive à deux arguments, et lorsqu'il sera nécessaire de préciser qu'une telle distance vérifie en plus les axiomes usuels des mathématiciens (symétrie, inégalité triangulaire, distance nulle d'un élément à lui-même), alors je parlerai de métrique.

multiplicatif près), et le cas où les coûts sont irrationnels ne permet généralement plus de calculer la distance d'édition de façon exacte ;

2. On supposera que $\mathcal{C}$ définit une métrique sur l'ensemble $\Sigma \cup \{\lambda\}$. Grâce à cette contrainte, les multiples définitions possibles de la distance d'édition sont toutes équivalentes. De plus, cette contrainte est importante pour que la distance soit efficacement calculable. Par la suite, on aura donc $\mathcal{C}(a,a) = 0$ et $\mathcal{C}(a,b) = \mathcal{C}(b,a)$ pour tout $a, b \in \Sigma \cup \{\lambda\}$; en d'autres termes, la matrice des coûts sera symétrique, positive, à diagonale nulle.

Alignements et distances d'édition

Un *alignement* (ou *script d'édition*) de deux mots u et v est une séquence d'opérations d'édition $S = (a_1, b_1)(a_2, b_2)\dots(a_s, b_s)$ telle que $u = a_1 a_2 \dots a_s$ et $v = b_1 b_2 \dots b_s$. Par exemple, $(1,0)(0,1)$ et $(1,\lambda)(0,0)(\lambda,1)$ sont deux alignements des mots 10 et 01. A chaque alignement S de u et v, on associe son coût par $\mathcal{C}$ en posant $\mathcal{C}(S) = \sum_{i=1}^{s} \mathcal{C}(a_i, b_i)$.

La *distance d'édition* $d_{\mathcal{C}}(u,v)$ des mots u et v est alors définie comme le coût minimum par $\mathcal{C}$ de tout alignement de u et v :

$$d_{\mathcal{C}}(u,v) = \min\left\{\mathcal{C}(S) : S = (a_1,b_1)(a_2,b_2)\dots(a_s,b_s), u = a_1a_2\dots a_s, v = b_1b_2\dots b_s\right\}.$$

De plus, un alignement sera dit *optimal* si son coût vaut exactement $d_{\mathcal{C}}(u,v)$.

Suivant [CHL07], si $\mathcal{C}$ est une métrique sur l'ensemble $\Sigma \cup \{\lambda\}$, alors (1) $d_{\mathcal{C}}$ est une métrique sur Σ^* et (2) $d_{\mathcal{C}}(u,v)$ est calculable en temps $\mathcal{O}(|u| \cdot |v|)$ par programmation dynamique avec l'Algorithme 1. Ce dernier se trouve dans tous les manuels d'enseignement qui traitent de programmation dynamique comme [KT05, CLR92]. Cependant, de nombreuses optimisations de ce calcul ont été proposées dans la littérature (voir [Gus97] pour plus de détails). De façon étonnante, on cite souvent un article de Wagner & Fisher publié en 1974 comme référence pour l'algorithme initial. Pourtant, c'est un algorithme qui a été publié une bonne dizaine de fois dans les années 70 (voir [Kru83] pour une liste exhaustive de références), mais l'Histoire n'en a retenu qu'une seule...

Algorithm 1: Calcul usuel de la distance d'édition.

Input: Deux mots $u = a_1 a_2 \dots a_n$ et $v = b_1 b_2 \dots b_m$
Output: Une matrice $M[0..n, 0..m]$ telle que $d_{\mathcal{C}}(u,v) = M[n,m]$

```
1 M[0,0] ← 0;
2 for i = 1 to n do M[i,0] ← M[i−1,0] + C(a_i, λ);
3 for j = 1 to m do M[0,j] ← M[0,j−1] + C(λ, b_j);
4 for i = 1 to n do
5     for j = 1 to m do
6         M[i,j] ←
          min{M[i−1,j] + C(a_i,λ), M[i,j−1] + C(λ,b_j), M[i−1,j−1] + C(a_i,b_j)};
7 return M
```

Distances d'édition usuelles

Plusieurs distances célèbres sur les mots sont des distances d'édition pour des fonctions de coût données. La première d'entre elles, la *distance de Levenshtein* [Lev65], notée $d_E(w,w')$,

est obtenue avec le coût 1 pour toutes les opérations d'édition[4]. Par exemple, $d_E(0100, 001) = 2$ en utilisant $(0,0)(1,\lambda)(0,0)(0,1)$ comme alignement optimal. C'est cette distance que nous utiliserons la plupart du temps dans la suite.

La *distance de Hamming* $d_H(w, w')$ est la distance où seules les substitutions sont autorisées, c'est-à-dire, $\mathcal{C}(a,\lambda) = \mathcal{C}(\lambda, a) = +\infty$ et $\mathcal{C}(a,b) = 1$ pour tout $a, b \in \Sigma, a \neq b$. Notons que le calcul de $d_H(w, w')$ peut se faire en utilisant l'équation suivante :

$$d_H(a_1a_2\dots a_k, b_1b_2\dots b_l) = \begin{cases} \#\left(\{i : a_i \neq b_i, 1 \leq i \leq k\}\right) \text{ si } k = l, \\ +\infty \text{ sinon.} \end{cases}$$

Par exemple, $d_H(0100, 001) = +\infty$ et $d_H(0100, 0010) = 2$ en utilisant l'alignement (unique, donc optimal) $(0,0)(1,0)(0,1)(0,0)$.

Enfin, la *distance* Lcs $d_\wedge(w, w')$ est la distance d'édition où seules les insertions et les délétions sont autorisées[5]. De façon équivalente, $d_\wedge$ est la distance d'édition où le coût des insertions et des délétions vaut 1, et celui des substitutions vaut 2. On peut aussi montrer que $d_\wedge(w, w') = |w| + |w'| - 2\ell(w, w')$. Par exemple, $d_\wedge(0100, 001) = 3$ en utilisant $(0,0)(1,\lambda)(0,0)(0,\lambda)(\lambda,1)$ ou $(\lambda,0)(0,0)(1,1)(0,\lambda)(0,\lambda)$ comme alignements optimaux.

Parmi ces trois distances, celle de Levenshtein combinant des insertions / délétions et des substitutions est la plus difficile à manipuler. Cependant, ces distances sont liées par des relations assez standard :

Proposition 1 *Pour tout $u, v \in \Sigma^*$,*

1. $d_E(u,v) \leq d_H(u,v)$,
2. $\max(|u|,|v|) - \ell(u,v) \leq d_E(u,v) \leq \max(|u|,|v|)$,
3. $||u| - |v|| \leq d_E(u,v) \leq d_\wedge(u,v)$,
4. $d_E(u,v) = d_\wedge(u,v) = |u| - |v|$ *si et seulement si* $v \preceq u$.

La propriété 2 est prouvée dans [NR05]. Les autres viennent du fait qu'une distance d'édition où on interdit soit les substitutions, soit les insertions / délétions est forcément plus grande que la distance de Levenshtein (qui les autorise toutes).

2.3.2 Boules de mots

Considérons n'importe quelle distance d'édition $d_\mathcal{C}$ sur Σ^*.

On définit la *$d_\mathcal{C}$-boule de centre $o \in \Sigma^*$ et de rayon $r \in \mathbb{N}$* en posant $B_r(o) = \{w \in \Sigma^* : d_\mathcal{C}(o,w) \leq r\}$. Par exemple, si $\Sigma = \{0,1\}$ et que la distance est celle de Levenshtein, alors $B_1(10) = \{0, 1, 00, 10, 11, 010, 100, 101, 110\}$ et $B_r(\lambda) = \Sigma^{\leq r}$ pour tout $r \geq 0$.

On notera $\mathcal{BALL}^\mathcal{C}(\Sigma)$ la classe des boules définies pour la distance $d_\mathcal{C}$. On notera simplement $\mathcal{BALL}(\Sigma)$ la classe $\mathcal{BALL}^E(\Sigma)$ des boules de Levenshtein : c'est la première classe de boules qu'on considérera par la suite.

Représentation des boules

La "grammaire" que nous utiliserons comme représentation de la boule $B_r(o)$ sera le couple (o, r), de taille $|o| + \log r$. Cette représentation a deux avantages. D'une part, on peut

[4] sauf la substitution d'une lettre par elle-même, dont le coût est nécessairement nul.

[5] Lcs est une abbréviation de Longest Common Substring : je ne connais pas de nom standard pour $d_\wedge$.

déterminer en temps polynomial si un mot w appartient à la boule $B_r(o)$: il suffit (1) de calculer $d_C(o, w)$ et (2) de vérifier si cette distance est $\leq r$, ce qu'on peut faire en temps $\mathcal{O}(|o| \cdot |w| + \log r)$.

D'autre part, dans le cas de la distance de Levenshtein, et dès que l'alphabet a au moins deux lettres, on peut montrer qu'une boule a une représentation unique (donc canonique) :

Théorème 1 ([BBdlHJT08]) *Pour la distance de Levenshtein, si* $|\Sigma| \geq 2$ *et* $B_{r_1}(o_1) = B_{r_2}(o_2)$*, alors* $o_1 = o_2$ *et* $r_1 = r_2$*.*

Ce théorème est faux si l'alphabet n'a qu'une lettre $\{0\}$, puisqu'alors $B_2(0) = B_3(\lambda) = \{\lambda, 0, 00, 000\}$ par exemple. Par ailleurs, ce théorème reste vrai pour la distance Lcs, mais il est faux pour la distance de Hamming (puisque pour tout $r \geq 3$, $B_r(000) = \Sigma^3$).

Classe des bonnes boules

A ce stade, il est important de noter que toutes les boules ne seront pas intéressantes en Inférence Grammaticale pour des raisons de complexité. Par exemple, les plus longs mots contenus dans une boule sont de taille $|o| + r$, et donc exponentielle en $\log r$; on les obtient en faisant r insertions de lettres dans o. Ceci n'est pas gênant si r n'est pas trop grand devant $|o|$, mais si $r > 2^{|o|}$, alors il est clair qu'à aucun moment, on ne pourra développer d'algorithmes d'identification utilisant des mots en dehors de la boule.

Nous dirons donc qu'une boule $B_r(o)$ est *bonne* si $r \leq |o|$, et nous désignerons par $\mathcal{GOODBALL}(\Sigma)$ la classe des bonnes boules (pour la distance de Levenshtein). Cette seconde classe de boules sera la plus importante que nous utiliserons par la suite.

Petits retours (de manivelle) en Géométrie

Pour finir, revenons sur les propriétés "géométriques" contre-intuitives des boules de Levenshtein. Comme l'a montré Frédéric Tantini dans sa thèse [Tan09],

- les boules ne sont pas homogènes : dans une boule $B_r(o)$ sur un alphabet de 2 lettres, la moitié des mots sont de longueur maximum $|o| + r$, donc situés sur la frontière "supérieure" de la boule ; en comparaison, le nombre de mots de longueur minimale est négligeable ;
- les boules ont des volumes incomparables : pour un même rayon, et des centres contenant les mêmes nombres d'occurrences de lettres, leur volume peut aller du simple au double, par exemple, $\#(B_2(00001111)) = 172$ et $\#(B_2(01010101)) = 254$ sur l'alphabet $\{0, 1\}$;
- l'inclusion de boules n'est pas monotone : une boule de grand rayon peut très bien être incluse dans une boule de petit rayon, par exemple, $B_5(01) \subset B_4(0101)$.

Un dernier problème, pour la route

Une dernière propriété qui n'a pas l'air vérifiée concerne la convexité. Intuitivement, une boule de mots devrait avoir une frontière relativement régulière, et on devrait pouvoir profiter de cette propriété pour identifier les boules. Mais comment la définir ?

L'idée la plus simple consiste à introduire l'*intervalle* (ou le *segment*) entre deux mots : on pose $[\![u, v]\!] = \{w : d_E(u, w) + d_E(w, v) = d_E(u, v)\}$. On peut alors stipuler qu'un langage L est *convexe* si pour tout $u, v \in \Sigma^*$, $(u, v \in L \Longrightarrow [\![u, v]\!] \subseteq L)$. On notera qu'une notion similaire existe en théorie des graphes : la notion d'intervalle entre deux sommets u, v peut se définir comme l'ensemble de tous les sommets le long des plus courts chemins entre u

et v [FJ86]. Malheureusement, les boules ne sont pas convexes en ce sens. En effet, soient $o = 01, u = 011, v = 101, w = 1011$. Comme $d_E(o,u) = d_E(o,v) = 1$, on a $u, v \in B_1(o)$. De plus, $w \in [\![u,v]\!]$ car $d_E(u,w) = d_E(w,v) = 1$ et $d_E(u,v) = 2 = d_E(u,w) + d_E(w,v)$. Malheureusement, $d_E(o,w) = 2$, donc $w \notin B_1(o)$, et donc $[\![u,v]\!] \not\subseteq B_1(o)$.

Nous avons alors cherché à affaiblir la notion d'intervalle. On rappelle que $\preceq$ denote la relation de sous-mots. On définit l'*intervalle faible* $[u,v] = \{w : u \preceq w \preceq v\}$ si $u \preceq v$, et $\emptyset$ sinon. Puis on stipule qu'un langage L est *convexe* si pour tout $u, v \in \Sigma^*$, $u, v \in L \Longrightarrow [u,v] \subseteq L$. Mais de nouveau, les boules ne sont pas convexes en ce sens. En effet, soient $o = 010, x = 001, z = 1001, y = 01001$. On a $x \preceq z \preceq y$. De plus, comme $d_E(o,x) = d_E(o,y) = 2$, on a $x, y \in B_2(o)$. Pourtant, $d_E(o,z) = 3$, donc $z \notin B_2(o)$, donc $[x,y] \not\subseteq B_2(o)$.

Enfin, nous avons carrément tenté d'éliminer la notion d'intervalle en stipulant qu'un langage L devait être convexe si pour tout $u, v \in L$, il existait au moins une dérivation $u = u_0 \to u_1 \to \ldots \to u_n = v$ telle que (1) $n = d_E(u,v)$, (2) $d_E(u_i, u_{i+1}) = 1$ pour $0 \leq i < n$ et (3) $u_i \in L$ pour $0 \leq i \leq n$. Et malheureusement, nous avons encore trouvé des exemples de boules qui n'étaient pas convexes en ce sens. Soient $o = 0011, u = 0111000, v = 1110001$. On a $u, v \in B_4(o)$. De plus, $d_E(u,v) = 2$ et il n'existe que deux dérivations de longueur minimum : $u \to 01110001 \to v$ et $u \to 111000 \to v$. Or aucun des deux mots centraux n'est dans $B_4(o)$, puisque leur distance à o vaut 5.

Nous terminons donc sur un constat d'échec : soit la bonne définition de convexité reste à trouver (nous aurions manqué d'imagination ?), soit il faut se résoudre à admettre que les boules de mots ne sont pas convexes, qu'elles ont une frontière de dentelles, qu'elles évoquent plus des étoiles de mer que des disques du plan.

2.3.3 Représentations des boules

Nous avons choisi de représenter la boule $B_r(o)$ par le couple (o,r). Néanmoins, d'autres représentations pourraient être envisagées pour les boules. En effet, comme ce sont des langages finis, pourquoi ne pas encoder une boule par elle-même, *i.e.*, par l'ensemble des mots qu'elle contient ? En fait, cette représentation n'est pas *raisonnable* [GJ79] car sa taille n'est pas polynomiale en $|o| + \log r$. En effet, en supposant que $\Sigma^{-k} = \emptyset$ pour tout $k > 1$, on a $B_r(o) \subseteq \bigcup_{-r \leq k \leq r} \Sigma^{|o|+k}$, et donc :

$$\#\left(B_r(o)\right) \leq \sum_{-r \leq k \leq r} (\#\Sigma)^{|o|+k} = \mathcal{O}\left((\#\Sigma)^{|o|+r}\right).$$

Cette borne est précise dans le cas de la boule $B_r(\lambda)$. Par ailleurs, il est facile de vérifier expérimentalement que toute énumération exhaustive des mots d'une boule est impossible dès que le rayon est suffisamment grand.

De même, les automates sont une autre représentation possible pour les boules. Par exemple, il n'est pas difficile de voir que toute boule de Levenshtein $B_r(o)$ est reconnu par un *automate non déterministe avec λ-transitions* qui a $\mathcal{O}\left(|o| \cdot r\right)$ états (voir Figure 2.1). Pourtant, l'utilisation d'automates non déterministes n'est pas très efficace pour analyser des mots. De plus, ces automates ne sont généralement pas identifiables en Inférence Grammaticale [dlH97].

Concernant les représentations avec des automates déterministes, le problème n'est malheureusement pas complètement résolu : à notre connaissance, il n'existe aucune construction générale directe de l'automate minimal $M_r(o)$ d'une $d_{\mathcal{C}}$-boule $B_r(o)$. Mais nous avons travaillé sur des constructions pour des distances particulières.

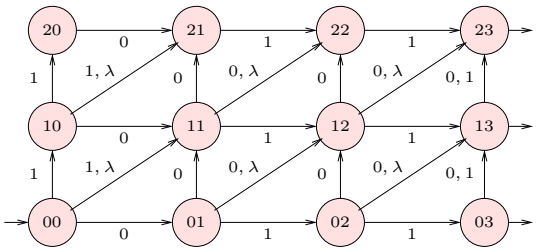


FIG. 2.1 – Un automate non déterministe avec λ-transitions reconnaissant la boule de Levenshtein $B_2(011)$, sur l'alphabet $\Sigma = \{0, 1\}$.

Cas des boules de Hamming

Considérons la boule de Hamming $B_r(o)$ centrée sur $o = c_1 c_2 \dots c_n$. Nous supposons que $|\Sigma| \geq 2$ (car le problème est trivial avec une seule lettre). On peut tout de suite remarquer que si $r \geq |o|$, alors $B_r(o) = \Sigma^{|o|}$, donc l'automate minimal $M_r(o)$ correspondant est une ligne de $|o| + 1$ états. Nous considérons donc le cas $r \leq |o|$. Alors, on obtient l'automate minimal (incomplet) $M_r(o) = \langle Q, \Sigma, i, F, \delta \rangle$ en posant $Q = \{(i, j) : 0 \leq i \leq r, i \leq j \leq i + |o| - r\}$, $i = (0, 0)$, $F = \{(r, |o|)\}$ et la fonction de transition δ comme suit :

- $\delta((i, j), c_{j+1}) = (i, j + 1)$, pour tout $0 \leq i \leq r, i \leq j \leq i + |o| - r - 1$,
- $\delta((i, j), x) = (i+1, j+1)$, pour tout $x \in \Sigma \setminus \{c_{j+1}\}, 0 \leq i \leq r-1, i \leq j \leq i+|o|-r-1$, et
- $\delta((i, i + |o| - r), x) = (i + 1, i + |o| - r + 1)$, pour tout $x \in \Sigma, 0 \leq i \leq r - 1$.

Par exemple, l'automate $M_2(01100)$ pour $\Sigma = \{0, 1, 2\}$ est représenté dans la Figure 2.2.

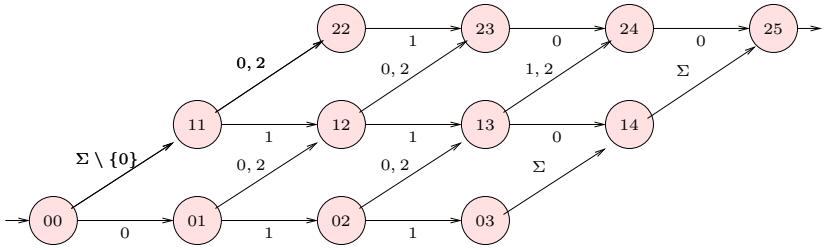


FIG. 2.2 – L'automate minimal $M_2(01100)$ de la boule de Hamming $B_2(01100)$.

Intuitivement, l'état (i, j) stocke les informations du passé : la composante i est le nombre de substitutions qui ont affecté les premières lettres du centre, et la composante j est l'indice de la dernière lettre qui a été traitée dans o. Ainsi, nous obtenons :

Théorème 2 *Si $|\Sigma| \geq 2$, alors $M_r(o)$ est l'automate minimal (incomplet) de la boule de Hamming $B_r(o)$. Il a $(r + 1)(|o| - r + 1)$ états si $r \leq |o|$, et $|o| + 1$ états sinon.*

Cas des boules LCS

Dans le cas des boules de Levenshtein, plusieurs algorithmes construisant des AFD ont été proposés dans la littérature, en particulier celui d'Ukkonen [Ukk85] ou celui de Schultz et Mihov [SM02]. Malheureusement, les AFD qu'ils retournent ne sont pas minimaux, comme nous

le verrons dans le paragraphe suivant. Or dans le cadre d'un travail commun avec Gwenaël Richomme et Fabrice Philippe au LIRMM, nous avons généralisé l'algorithme d'Ukkonen à toute distance d'édition, ce qui nous a ensuite permis d'étudier plus finement les propriétés des AFD produits. Et il s'avère que dans le cas de la distance LCS, ces AFD sont bien minimaux.

Commençons par les définir formellement :

Définition 1 *Considérons une boule* LCS $B_r(o)$ *avec* $o = c_1c_2 \dots c_n$ *et* $r \geq 0$.

- *On associe à chaque mot* $p \in \Sigma^*$, *un tuple* $t_p = (x_0, x_1, \dots, x_n)$ *avec*

$$x_i = \min\{d_\wedge(p, o[1..i]), r+1\}$$

  pour tout $0 \leq i \leq n$. *On note* $\mathbb{T} = \{t_p : p \in \Sigma^*\}$.
- *On définit l'automate* $A_r(o) = \langle Q, \Sigma, i, F, \delta\rangle$ *tel que (1)* $Q = \mathbb{T}$, *(2)* $i = t_\lambda$, *(3)* $F = \{t_p = (x_0, x_1, \dots, x_n) \in \mathbb{T} : x_n \leq r\}$ *et (4)* $\delta(t_p, a) = t_{pa}$ *pour tout* $t_p \in \mathbb{T}, a \in \Sigma$.

Pour un mot $p \in \Sigma^*$ donné, les composantes de t_p sont les distances LCS de p aux préfixes de o, tronquées à $r+1$. Par exemple, considérons la boule LCS $B_2(0011)$. Alors $t_{10} = (x_0, x_1, x_2, x_3, x_4)$ avec $x_0 = \min\{d_\wedge(10, \lambda), 3\} = 2$, $x_1 = \min\{d_\wedge(10, 0), 3\} = 1$, $x_2 = \min\{d_\wedge(10, 00), 3\} = 2$, $x_3 = \min\{d_\wedge(10, 001), 3\} = 3$, $x_4 = \min\{d_\wedge(10, 0011), 3\} = \min\{4, 3\} = 3$. Aussi, $t_{10} = (2, 1, 2, 3, 3)$. De même, $t_\lambda = (3, 3, 2, 1, 0)$ et $t_{0101} = (3, 3, 2, 1, 2)$.

Maintenant, on a le résultat suivant :

Théorème 3 *L'automate* $A_r(o)$ *est un* AFD *qui reconnaît la boule* LCS $B_r(o)$.

Preuve: D'abord, $A_r(o)$ est un automate fini car $|\mathbb{T}|$ est borné par $(r+2)^{|o|+1}$. Ensuite, $A_r(o)$ reconnaît la boule $B_r(o)$ car pour tout $w \in \Sigma^*$, $\delta(i, w) \in F \Leftrightarrow t_w \in F \Leftrightarrow x_n = d_\wedge(w, o) \leq r \Leftrightarrow w \in B_r(o)$. Enfin, pour établir que $A_r(o)$ est déterministe, il faut montrer que si $t_p = t_q$, alors $t_{pa} = t_{qa}$ pour tout $p, q \in \Sigma^*, a \in \Sigma$; en d'autres termes, il faut montrer que δ est bien une fonction. Pour cela, posons que $t_p = (x_0, x_1, \dots x_n)$ avec $x_i = \min\{u_i, r+1\}$ et $u_i = d_\wedge(p, o[1..i])$ pour $0 \leq i \leq n$. De même, supposons que $t_{pa} = (y_0, y_1, \dots y_n)$ avec $y_i = \min\{v_i, r+1\}$ avec $v_i = d_\wedge(pa, o[1..i])$.

Examinons de plus près les tuples non tronqués $U_p = (u_0, u_1, \dots, u_n)$ et $U_{pa} = (v_0, v_1, \dots, v_n)$. Le tuple $U_p = (u_0, u_1, \dots, u_n)$ est en fait la dernière ligne de la table qu'on construirait pour calculer $d_\wedge(p, o)$ en utilisant l'algorithme de programmation dynamique standard. Du coup, le calcul de U_{pa} se fait en ajoutant virtuellement une ligne dans cette table et en suivant le schéma de programmation dynamique : en notant $\mathcal{C}$ la fonction de coût pour la distance LCS, on a (1) $v_0 = u_0 + \mathcal{C}(a, \lambda)$ et (2) $v_{i+1} = \min\{v_i + \mathcal{C}(\lambda, c_{i+1}), u_{i+1} + \mathcal{C}(a, \lambda), u_i + \mathcal{C}(a, c_{i+1})\}$ pour $0 \leq i \leq (n-1)$.

Revenons maintenant aux tuples t_p et t_{pa} dont les valeurs sont tronquées à $r+1$. Comme la troncature se fait avec un min, et que c'est ce même min qui est utilisé dans les équations de récurrence, on a clairement (1) $y_0 = \min\{x_0 + \mathcal{C}(a, \lambda), r+1\}$ et (2) $y_{i+1} = \min\{x_i + \mathcal{C}(\lambda, c_{i+1}), x_{i+1} + \mathcal{C}(a, \lambda), x_i + \mathcal{C}(a, c_{i+1}), r+1\}$ pour tout $0 \leq i \leq (n-1)$. On constate que les composantes de t_{pa} ne dépendent que des composantes de t_p, des lettres de o et de la lettre a. Donc si on considère deux mots $p, q \in \Sigma^*$ tels que $t_p = t_q$, alors on a bien $t_{pa} = t_{qa}$ compte tenu des équations précédentes, et donc δ est bien une fonction. □

En outre, nous avons montré le résultat suivant sur la minimalité de $A_r(o)$. Comme nous ne l'avons pas encore publié, nous reportons sa preuve, relativement technique, en annexe (cf. Section A.1) :

Théorème 4 *Soit Σ un alphabet contenant au moins deux lettres, dont une qu'on distingue et qu'on note $\diamondsuit$. On considère la boule* Lcs $B_r(o)$ *avec $o \in (\Sigma \setminus \{\diamondsuit\})^*$ et $r \geq 0$. Alors l'automate $A_r(o)$ est minimal.*

Enfin, on peut calculer $A_r(o)$ de façon efficace (si sa taille est raisonnable) :

Théorème 5 *Supposons que $A_r(o)$ ait N états (et donc $N \cdot |\Sigma|$ transitions). Alors l'*Afd $A_r(o)$ *peut être calculé en temps $\mathcal{O}(N \cdot |\Sigma| \cdot |o|)$.*

En conséquence, s'il existait un polynôme $p()$ tel que $\|A_r(o)\| = \mathcal{O}(p(|o|, \log r))$, alors on pourrait calculer $A_r(o)$ en temps polynomial. Malheureusement, nous verrons que ce n'est pas le cas, donc que $A_r(o)$ n'est pas une représentation raisonnable de la boule Lcs $B_r(o)$.

Pour montrer ce théorème, on utilise l'Algorithme 2 initialement décrit par Ukkonen dans [Ukk85] pour le cas de la distance de Levenshtein. Il fonctionne en découvrant incrémentalement les états et les transitions à partir de l'état initial.

Algorithm 2: Algorithme d'Ukkonen [Ukk85].

Input: Une boule Lcs $B_r(o)$ sur l'alphabet Σ avec $o = c_1 c_2 \dots c_n$ et $r \geq 0$, la fonction de coût $\mathcal{C}$ de la distance Lcs
Output: L'Afd $A_r(o) = \langle Q, \Sigma, i, F, \delta \rangle$

```
1  i ← (x_0, x_1, ..., x_n) avec x_0 = 0 et x_{k+1} = min{x_k + C(λ, c_{k+1}), r + 1};
2  Q ← {i};
3  F ← if (x_n ≤ r) then {i} else ∅ ;
4  soit S une pile pile ; empiler i dans S;
5  while S n'est pas vide do
6      t ← dépiler S;
7      supposons que t = (x_0, x_1, ..., x_n);
8      foreach a ∈ Σ do
9          soit t' = (y_0, y_1, ... y_n) avec y_0 = min{x_0 + C(a, λ), r + 1} et
           y_{k+1} = min{x_k + C(λ, c_{k+1}), x_{k+1} + C(a, λ), x_k + C(a, c_{k+1}), r + 1};
10         δ(t, a) ← t';
11         if t' ∉ Q then
12             Q ← Q ∪ {t'} ;
13             if (y_n ≤ r) then F ← F ∪ {t'} ;
14             empiler t' dans S
```

Preuve: *(du Théorème 5)* La correction de l'algorithme, en particulier le calcul des transitions, vient de la preuve du Théorème 3. Concernant la complexité, on peut noter que le calcul de l'état initial i et de chaque transition $t' = \delta(t, a)$ se fait en temps $\mathcal{O}(|o|)$. De plus, vérifier si un état t' a déjà été généré ou non, peut se faire en temps $\mathcal{O}(|o|)$ avec la bonne structure de données : on peut voir t' comme un mot de longueur $|o| + 1$ écrit sur l'alphabet $\{0, 1, \dots, r + 1\}$, donc il suffit d'utiliser un arbre reconnaisseur de préfixes (Pta) comme structure de données pour l'ensemble Q pour trancher la question de l'appartenance de t' à Q en temps linéaire. A chaque itération, l'algorithme crée une nouvelle transition, et donc il y a nécessairement $N \cdot |\Sigma|$ itérations. Et comme les états sont générés et comparés en temps $\mathcal{O}(|o|)$, la complexité totale est en temps $\mathcal{O}(N \cdot |\Sigma| \cdot |o|)$. □

Bien entendu, le théorème précédent est insatisfaisant puisque la complexité de l'algorithme est fonction de la taille de la sortie et non de l'entrée. Mais d'une part, c'est un résultat à défaut de mieux, et d'autre part, c'est un algorithme qui a un intérêt pratique : il permet de générer les automates minimaux de "grosses" boules Lcs, puis d'étudier leur taille (nombre d'états) en fonction de celle du centre et du rayon. Parmi nos expérimentations, nous en reprenons deux qui nous semblent plus intéressantes.

Dans un premier temps, nous avons voulu expliciter le nombre moyen des états de l'automate minimal d'une boule Lcs $B_r(o)$ lorsque $|o| = 2r$. Nous avons choisi ce rapport entre la longueur du centre et le rayon, car expérimentalement, c'est celui qui produit les Afd les plus grands. L'alphabet est fixé à $\Sigma = \{1, 2, \diamondsuit\}$ mais les centres sont sur l'alphabet $\{1, 2\}$ pour être dans les conditions de validité du Théorème 4. Puis nous avons construit la Table 2.1 selon deux modes. Pour des rayons faibles, nous avons estimé $\|A_r(o)\|$ en considérant toutes les boules avec des centres de longueur $2r$. Or au-delà de $r = 9$, le nombre d'automates minimaux est trop important pour qu'ils puissent tous être générés ; nous avons alors tiré au hasard un certain pourcentage (12.5%) des centres de longueurs $2r$ pour faire notre estimation.

Rayon r	Taille du centre $\|o\|$	Nombre moyen d'états de $A_r(o)$		Notes
		Moyenne	Ecart type	
0	0	2.0	0.0	
1	2	7.5	0.5	
2	4	21.0	2.2	
3	6	50.8	6.6	
4	8	114.1	17.1	
5	10	246.1	41.9	
6	12	517.9	98.3	
7	14	1073.3	224.9	
8	16	2199.6	504.5	
9	18	4470.0	1113.4	(*estimé sur*
10	20	9022.1	2525.5	*12.5% des boules*)

Tab. 2.1 – Taille moyenne de l'automate minimal de la boule Lcs $B_r(o)$ en fonction de r (avec $|o| = 2r$) .

Dans une seconde expérimentation, nous avons cherché, pour chaque rayon r quelles pouvaient être les boules Lcs $B_r(o)$ (avec $|o| = 2r$) ayant les plus grands automates minimaux (voir Table 2.2). Nous avons initialement recensé ces centres dans l'espoir de trouver une relation entre tous ces "pires cas" que nous aurions ensuite pu exploiter pour établir une borne sur le nombre d'états des automates minimaux. Ce travail est toujours en cours.

Dans le premier et le second tableau, on observe immédiatement que la taille de l'automate minimal de $B_r(o)$ peut doubler chaque fois que r augmente de 1. On observe également qu'à rayon fixé, il existe de grandes différences entre les pires cas (plus de 20000 états pour $r = 10$), le cas moyen (quelques 10000 états pour $r = 10$) et les cas simples : l'automate minimal de la boule de centre $1^{20} = \underbrace{11\ldots1}_{20}$ et de rayon 10 n'a que 232 états. Derrière ces chiffres se cache toute la difficulté de cette étude : il est nécessaire de considérer des centres très particuliers pour mettre en évidence des automates minimaux qui explosent en nombre d'états. Ainsi, trouver une famille de centres pour laquelle la taille des automates minimaux

r	$\|o\|$	$\|\|A_r(o)\|\|$ max.	Centre o correspondant
0	0	2	λ
1	2	8	12
2	4	23	1212, 1221
3	6	60	122112, 112212
4	8	143	12211212
5	10	325	1122211212
6	12	744	112222112121
7	14	1683	11222211122121
8	16	4027	1122222111122121
9	18	9134	112222221111122121
10	20	20515	11222222111111221212
11	22	45500	1112222222111111221212
12	24	100013	111222222222111111221212
13	26	221043	11122222222211111122211212
14	28	482093	1112222222221111111122211212
15	30	1060845	111222222222211111111122211212

TAB. 2.2 – Exemples d'automates $A_r(o)$ de grande taille en fonction de r et de o (avec $|o| = 2r$). .

serait prouvablement exponentielle (*i.e.*, de l'ordre de $2^r|o|$) reste un problème ouvert.

En tous cas, nous pouvons désormais conclure sur la possibilité d'utiliser les automates minimaux des boules Lcs comme représentant de ces boules : il est manifeste sur ces exemples que leur taille n'est pas polynomialement liée à $|o| + \log r$, donc que ce ne sont pas des représentations raisonnables.

Cas des boules de Levenshtein

Plusieurs des constructions et des résultats précédents se généralisent aux boules de Levenshtein, et aux $d_{\mathcal{C}}$-boules quelconques. Ainsi, il suffit de remplacer la distance $d_\wedge$ par la distance $d_{\mathcal{C}}$ dans la Définition 1 pour obtenir une nouvelle construction de l'automate $A_r(o)$. On peut facilement montrer que ce dernier reconnait la $d_{\mathcal{C}}$-boule $B_r(o)$ (en généralisant le Théorème 3), puis modifier l'algorithme d'Ukkonen pour le construire "efficacement" (Théorème 5).

Par contre, $A_r(o)$ n'est généralement plus minimal, *i.e.*, le Théorème 4 ne se généralise pas. En effet, considérons de nouveau la boule de Hamming $B_2(01100)$ sur l'alphabet $\Sigma = \{0, 1, 2\}$ dont nous avons représenté l'automate minimal dans la Figure 2.2. L'automate $A_2(01100)$ qu'on obtient avec l'algorithme d'Ukkonen est celui de la Figure 2.3. Si on avait travaillé avec une boule de Levenshtein plutôt qu'une boule de Hamming, on aurait observé le même phénomène de non minimalité. En fait, on observe ce phénomène dès que des substitutions interviennent.

Enfin, concernant les nombres d'états de $A_r(o)$ pour la distance de Levenshtein, des expérimentations montrent qu'il est toujours plus grand que celui qu'on obtient avec la distance Lcs. Il en est de même pour l'automate minimal correspondant (qu'on peut obtenir efficacement à l'aide d'un algorithme de minimisation d'automate acyclique comme celui de [Wat03]). Plus précisément, on observe que le nombre d'états peut être de l'ordre de $3^r.|o|$

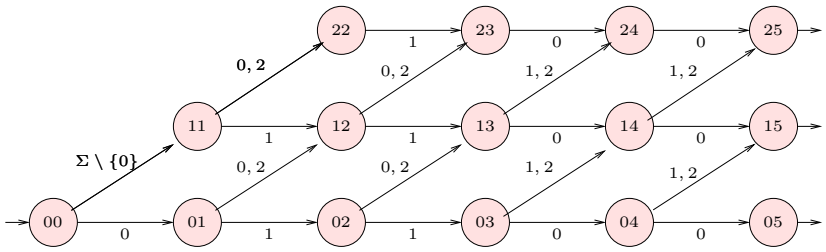


FIG. 2.3 – L'automate $A_2(01100)$ pour la distance de Hamming. Un état (i, j) correspond à un tuple $t_p = (x_0, x_1, \ldots, x_5)$ pour lequel $x_j = i$ et $x_{k \neq j} = 3$.

mais aucune preuve n'existe. Donc de nouveau, la taille de l'automate minimal d'une boule de Levenshtein n'est pas polynomialement lié à $|o| + \log r$, et un AFD ne peut pas servir de représentation raisonnable.

Une autre idée ?

Nous avons vu que pour la plupart des distances d'édition, les AFD ne pouvaient pas servir de représentation pour les boules de mots : la taille de l'automate minimal d'une boule $B_r(o)$ n'est généralement pas polynomiale en $|o| + \log r$. A l'inverse, avec $|o| + r$ états, les automates non-déterministes sont des bons candidats, mais présentent des inconvénients en Inférence Grammaticale.

Or dans [DLT02], les auteurs ont étudié un autre type d'automates non déterministes, les AFER, qui intuitivement, ont de bonnes propriétés vis à vis des boules de mots (compte tenue de la forme des automates non déterministes avec λ-transitions qui reconnaissent ces boules). Ces AFER ont des propriétés intéressantes : ils admettent des formes canoniques minimales (en nombre d'états). Dans le cas général, ils ne sont pas identifiables[6] [DLT04], mais ils pourraient l'être lorsqu'ils sont utilisés pour représenter des boules de mots.

C'est donc une piste qui mériterait d'être explorée.

[6]Plus précisément, les AFER ne sont pas *polynomialement caractérisables* (au sens de [dlH97]), et n'admettent donc pas tous des ensembles caractéristiques polynomiaux

Chapitre 3

La notion d'apprentissage en Inférence Grammaticale

En Inférence Grammaticale, un algorithme a la capacité d'*apprendre* si, sous certaines conditions, il est en mesure d'*identifier chaque grammaire d'une classe* $\mathcal{G}$ à partir de données. On ne s'intéressera donc jamais à une grammaire particulière, mais à l'ensemble de toutes les grammaires d'une classe, chacune d'entre elles étant susceptible d'être prise pour *cible* de l'apprentissage.

Il existe principalement deux notions d'identification. En identification *exacte*, un algorithme doit retrouver exactement toute grammaire cible à l'issue de l'apprentissage. Due à Gold, c'est historiquement la première notion d'identification qui ait existé. Au contraire, en identification *approximative*, un algorithme doit "seulement" retrouver une bonne approximation pour toute grammaire cible, lorsqu'il a suffisamment d'informations pour apprendre. C'est le cadre de la PAC-apprenabilité de Valiant, adapté à l'apprentissage à partir de mots.

Enfin, pour chaque type d'identification, des hypothèses sont formulées sur la manière dont sont générées ou récupérées les données. Nous traiterons ici le cas où les données sont *subies*, au sens où un algorithme reçoit ces données de l'environnement extérieur sans qu'il puisse interragir avec lui. Nous aborderons le cas où les données sont *choisies* par l'algorithme dans le Chapitre 4, où nous présenterons le paradigme d'identification à partir de requêtes d'Angluin.

Il est important de noter dès à présent que dans un problème d'Inférence Grammaticale, les données ne pré-existent pas au choix de la classe de grammaire étudiée ; elles sont sensées caractériser une grammaire cible que l'algorithme doit identifier. C'est donc un problème assez différent de ceux qu'on traite habituellement en Classification Supervisée où, après avoir observé des données, on se donne une famille d'hypothèses choisie pour ses bonnes propriétés, puis on calcule celle d'entre elles qui se comporte le mieux sur les données (par exemple, celle qui minimise l'erreur en généralisation). Nous y reviendrons au chapitre suivant.

Ici, nous présenterons successivement les cadres de l'identification exacte et de l'identification approximative, en illustrant ces paradigmes avec des exemples sur les automates et les boules de mots. Ceux-ci proviennent soit de la littérature, soit de nos propres travaux sur le sujet [dlHJ04, dlHJT08, Tan09]. Nous verrons dans un troisième temps le cas des grammaires hors-contextes, sur lequel nous avons travaillé avec Rémi Eyraud pendant sa thèse [dlHJ04, Eyr06].

3.1 Identification exacte

L'*identification à la limite*, proposée par Gold en 1967 est un des plus anciens cadres visant à définir formellement ce qu'*apprendre* signifie [Gol67]. Il fut le seul cadre de référence pendant près de 15 ans, jusqu'à l'émergence du cadre PAC de Valiant [Val84], puis le rapprochement des chercheurs en Intelligence Artificielle et en Statistique sous l'influence de Vapnik [BEHW89]. Une vue d'ensemble des résultats obtenus pendant ces premières années est présentée dans [AS83]. Notons que Gold avait pour objectif explicite d'apprendre automatiquement la langue naturelle et qu'il plaçait cette tâche dans le domaine de l'Intelligence Artificielle.

3.1.1 Identification à la limite

Dans le paradigme standard d'identification à la limite de Gold, un apprenant reçoit une séquence infinie d'informations (présentation) qui doivent l'aider à trouver la représentation $G \in \mathcal{G}$ d'un langage cible inconnu $L \in \mathcal{L}$. Mathématiquement, chaque présentation est une fonction $f : \mathbb{N} \to \mathbb{X}$ vérifiant certaines propriétés, où $\mathbb{N}$ désigne l'ensemble des entiers naturels et $\mathbb{X}$ est un ensemble dépendant du type de présentation considéré. On note $\mathcal{P}$ l'ensemble de toutes les présentations.

Pour pouvoir apprendre les langages de la classe $\mathcal{L}$ à l'aide des présentations de $\mathcal{P}$, il est nécessaire de supposer (1) qu'à chaque langage $L \in \mathcal{L}$ est associé un sous-ensemble non vide $\mathcal{P}_L$ de présentations, et (2) que l'ensemble $\{\mathcal{P}_L : L \in \mathcal{L}\}$ forme une partition de $\mathcal{P}$. En effet, si le point (1) n'est pas vérifié, alors certains langages de la classe $\mathcal{L}$ ne sont pas décrits par des présentations, et si le point (2) ne l'est pas, alors soit il existe des présentations ne correspondant à aucun langage, soit deux langages distincts sont décrits par une même présentation, et sont donc indifférentiables à l'aide des informations de la présentation ; ce sont évidemment des conditions suffisantes pour que la classe $\mathcal{L}$ ne soit pas apprenable avec les présentations de $\mathcal{P}$.

Parmi tous les types de présentations, deux sont particulièrement utilisés :

$\mathcal{P}$=TEXTE Dans ce cas, toute présentation est une fonction $f : \mathbb{N} \to \Sigma^*$; de plus, pour chaque langage $L \in \mathcal{L}$ et chaque présentation $f \in \mathcal{P}_L$, tous les mots de L doivent apparaître au moins une fois dans f : $f(\mathbb{N}) = L$.

$\mathcal{P}$=INFORMANT Dans ce cas, toute présentation est une fonction $f : \mathbb{N} \to \Sigma^* \times \{+, -\}$; de plus, pour chaque langage $L \in \mathcal{L}$ et chaque présentation $f \in \mathcal{P}_L$, tous les mots de L doivent apparaître avec l'étiquette $+$, et tous les mots n'appartenant pas à L, avec l'étiquette $-$: $f(\mathbb{N}) = (L \times \{+\}) \cup ((\Sigma^* \setminus L) \times \{-\})$. Tout élément d'un informant sera donc un couple (w, ℓ) où $(w \in L \Rightarrow \ell = +)$ et $(w \notin L \Rightarrow \ell = -)$; on parlera d'*exemple positif* (ou simplement d'exemple) dans le premier cas, et d'*exemple négatif* (ou de contre-exemple) dans le second.

Pour toute présentation $f \in \mathcal{P}$, $f(n)$ désigne l'élément d'indice n dans f, et f_m la *séquence* $f(0)f(1)\dots f(m)$ des $m+1$ premiers éléments de f. Clairement, il peut exister des doublons dans la séquence f_m, c'est-à-dire deux indices i et j tels que $f(i) = f(j)$; aussi, nous noterons $\widehat{f_m}$ l'*ensemble* constitué par les $m+1$ premiers éléments de f (et on aura donc $\#(\widehat{f_m}) \leq m$). Enfin, nous appelons $\mathrm{seq}(\mathcal{P})$ l'ensemble de tous les débuts possibles de présentations : $\mathrm{seq}(\mathcal{P}) = \{f_m : f \in \mathcal{P}, m \in \mathbb{N}\}$.

Dans la suite, nous évoquerons souvent le fait qu'une grammaire $G \in \mathcal{G}$ soit *consistante* ou non avec une donnée $x \in \mathbb{X}$, ce que nous noterons $G \vdash x$. Lorsque nous utilisons des TEXTES

comme type de présentations, cela signifiera que x est un mot de Σ^* qui est reconnu par G. Et lorsque nous travaillons avec des INFORMANTS, cela signifiera que x est un couple (w, ℓ) où w est un mot de Σ^* qui est reconnu par G si $\ell = +$, et qui n'est pas reconnu par G si $\ell = -$. Nous utiliserons la même terminologie pour les ensembles $E \subseteq \mathbb{X}$ de données.

Algorithme d'identification exacte

On suppose désormais fixés la classe $\mathcal{L}$ de langages, la classe $\mathcal{G}$ des grammaires, et l'ensemble $\mathcal{P}$ des présentations. Dans le cadre de Gold, un apprenant est un algorithme déterministe qui prend des débuts de présentations en entrée, et qui retourne des grammaires en sortie, donc une fonction calculable $\mathfrak{A} : \text{seq}(\mathcal{P}) \to \mathcal{G}$. Comme l'apprenant retourne des grammaires et que les grammaires déterminent les langages, on omettra la plupart du temps les langages eux-mêmes. Ainsi, on définit l'identification à la limite de la façon suivante :

Définition 2 (Identification à la limite) *Soient $\mathcal{G}$ une classe de grammaires et $\mathcal{P}$ une classe de présentations.*

Un algorithme $\mathfrak{A}$ est un algorithme d'identification à la limite pour $\mathcal{G}$ à partir de $\mathcal{P}$ *si $\mathfrak{A}$ prend en entrée des débuts de présentations de seq($\mathcal{P}$) et retourne des grammaires $H \in \mathcal{G}$.*

On dit que $\mathcal{G}$ est identifiable à la limite *à partir de $\mathcal{P}$ s'il existe un algorithme d'identification à la limite $\mathfrak{A}$ tel que pour toute grammaire $G \in \mathcal{G}$ et toute présentation $f \in \mathcal{P}_{\mathbb{L}(G)}$, il existe un rang $n \geq 0$ tel que pour tout $m \geq n$, $\mathfrak{A}(f_m) = \mathfrak{A}(f_n)$ et $\mathbb{L}(\mathfrak{A}(f_n)) = \mathbb{L}(G)$.*

De nombreux résultats d'apprenabilité et de non apprenabilité existent dans le cadre de Gold. Ainsi, la classe des AFD n'est pas identifiable à la limite à partir de TEXTES[1], mais elle l'est à partir d'INFORMANTS [Gol67]. D'ailleurs, Gold a montré que toute classe récursivement énumérable de langages récursifs était identifiable à la limite à partir d'INFORMANTS : il suffit d'énumérer les grammaires et trouver la première qui couvre tous les exemples positifs et rejettent tous les exemples négatifs ; si à un rang donné de l'informant, la grammaire n'est pas la grammaire cible, alors on trouvera dans le futur un exemple positif non couvert, ou un exemple négatif couvert, ce qui nous permettra de relancer l'énumération des grammaires jusqu'à trouver une grammaire plus adéquate ; à un moment de l'énumération, on trouvera la grammaire cible, et l'algorithme couvergera. Clairement, cet algorithme n'a aucun intérêt pratique puisqu'il repose sur un processus d'énumération de grammaires : aucune contrainte d'efficacité n'est imposée sur le paradigme, et cela conduit manifestement à un résultat d'apprenabilité peu pertinent.

Contraintes de polynomialité

L'"échec" de cette première tentative de définition de l'apprenabilité n'en était sans doute pas un dans le contexte de l'époque (1967). En effet, le cadre de Gold relève probablement plus du domaine de la Calculabilité que de celui de l'Algorithmique. Or considérons l'état de l'art en Algorithmique en 1967 : selon [GJ79], la découverte du premier problème $\mathcal{NP}$-complet ne date que de 1971 [Coo71] et les premières réductions entre problèmes visant à recencer les problèmes $\mathcal{NP}$-complets, de 1972 [Kar72]. De fait, les tentatives d'intégration de contraintes de polynomialité dans le paradigme d'identification à la limite seront entreprises à la la fin des années 70 (par exemple dans [Gol78]). Nous rappelons maintenant les plus célèbres, en nous reposant principalement sur [Pit89].

[1] car c'est une classe *super-finie* de langages [Gol67].

Tout d'abord, il est raisonnable de penser que la polynomialité doit concerner la quantité de temps dont l'algorithme dispose pour apprendre :

Définition 3 (Temps de mise à jour) *On dit qu'un algorithme* $\mathfrak{A}$ *a un* temps de mise à jour polynomial *s'il existe un polynôme* $p()$ *tel que, pour chaque présentation* $f \in \mathcal{P}$ *et chaque entier* $n \geq 0$*, construire* $\mathfrak{A}(f_n)$ *nécessite un temps en* $\mathcal{O}(p(\|f_n\|))$.

Cependant, exiger un temps de mise à jour polynomial n'est pas suffisant : un apprenant pourrait recevoir un nombre exponentiel d'exemples sans rien faire d'autre qu'attendre, puis utiliser la grande quantité de temps qu'il a accumulée pour résoudre un problème $\mathcal{NP}$-difficile [Pit89].

Aussi, d'autres contraintes sont nécessaires. La première vise à limiter le nombre de changement d'hypothèses (*Mind Changes*) que l'algorithme peut faire pendant l'apprentissage [AS83] :

Définition 4 (Changement d'avis) *Etant donnés un algorithme* $\mathfrak{A}$ *et une présentation* $f \in \mathcal{P}$*, on dit que* $\mathfrak{A}$ change d'avis au rang n *si* $\mathfrak{A}(f_n) \neq \mathfrak{A}(f_{n-1})$*. On dit que* $\mathfrak{A}$ *est* conservatif *s'il ne change jamais d'avis lorsque son hypothèse courante est consistante avec le nouvel élément présenté.*

Un algorithme $\mathfrak{A}$ *identifie une classe* $\mathcal{G}$ *à la limite* en temps MC-polynomial *si (1)* $\mathfrak{A}$ *identifie* $\mathcal{G}$ *à la limite, (2)* $\mathfrak{A}$ *a un temps de mise à jour polynomial et (3)* $\mathfrak{A}$ *fait un nombre polynomial de changements d'avis.*

Concernant le point (3), soit $\#\textsc{Mc}(f) = |\{k \geq 0 : \mathfrak{A}(f_k) \neq \mathfrak{A}(f_{k+1})\}|$ *; on veut qu'il existe un polynôme* $p()$ *tel que, pour chaque grammaire* G *et chaque présentation* f *de* $\mathbb{L}(G)$*,* $\#\textsc{Mc}(f) \leq p(\|G\|)$.

La seconde concerne le nombre d'erreurs implicites de prédiction (*Implicit Prediction Error*) que l'algorithme est autorisé à commettre pendant l'apprentissage, c'est-à-dire le nombre de fois où il produit une hypothèse qui n'est pas consistante avec le nouvel élément présenté [Pit89] :

Définition 5 (Erreur implicite de prédiction) *Etant donnés un algorithme d'apprentissage* $\mathfrak{A}$ *et une présentation* $f \in \mathcal{P}$*, on dit que* $\mathfrak{A}$ *commet* une erreur implicite de prédiction au rang n *si* $\mathfrak{A}(f_{n-1}) \not\vdash f(n)$*. On dit que* $\mathfrak{A}$ *est* consistant *s'il change d'avis chaque fois qu'il commet une erreur de prédiction.*

Un algorithme $\mathfrak{A}$ *identifie une classe* $\mathcal{G}$ *à la limite* en temps IPE-polynomial *si (1)* $\mathfrak{A}$ *identifie* $\mathcal{G}$ *à la limite, (2)* $\mathfrak{A}$ *a un temps de mise à jour polynomial et (3)* $\mathfrak{A}$ *fait un nombre polynomial d'erreurs implicites de prédiction.*

Concernant le point (3), soit $\#\textsc{Ipe}(f) = |\{k \geq 0 : \mathfrak{A}(f_k) \not\vdash f(k+1)\}|$ *; on veut qu'il existe un polynôme* $p()$ *tel que, pour chaque grammaire* G *et chaque présentation* f *de* $\mathbb{L}(G)$*,* $\#\textsc{Ipe}(f) \leq p(\|G\|)$.

Malgré leur différence, les deux mesures de polynomialité précédentes ne sont pas complètement indépendentes :

Lemme 1 *Si* $\mathcal{G}$ *est identifiable en temps MC-polynomial avec un algorithme consistant, alors* $\mathcal{G}$ *est identifiable en temps IPE-polynomial. De même, si* $\mathcal{G}$ *est identifiable en temps IPE-polynomial avec un algorithme conservatif, alors* $\mathcal{G}$ *est identifiable en temps MC-polynomial.*

Preuve: Si un algorithme $\mathfrak{A}$, identifiant $\mathcal{G}$ en temps MC-polynomial, est consistant, alors $\#\textsc{Ipe}(f) \leq \#\textsc{Mc}(f)$, car $\mathfrak{A}$ change d'avis chaque fois qu'une erreur implicite de prédition

est détectée. Donc $\mathfrak{A}$ identifie $\mathcal{G}$ en temps IPE-polynomial. De même, si $\mathfrak{A}$ identifie $\mathcal{G}$ en temps IPE-polynomial et qu'il est conservatif, alors $\#\text{Mc}(f) \leq \#\text{Ipe}(f)$, car $\mathfrak{A}$ ne change pas d'avis tant qu'une erreur (explicite) de prédition n'est pas détectée. □

Enfin, une dernière contrainte concerne la quantité minimum de données qu'un algorithme doit recevoir pour apprendre :

Définition 6 (Ensemble caractéristique) *Etant donnés un algorithme d'apprentissage $\mathfrak{A}$ et une grammaire $G \in \mathcal{G}$, on dit qu'un ensemble fini $C \subseteq \mathbb{X}$ est un* ensemble caractéristique *de G pour $\mathfrak{A}$ si pour toute présentation $f \in \mathcal{P}_{\mathbb{L}(G)}$ et pour tout $n \geq 0$, si $C \subseteq \widehat{f_n}$, alors $\mathbb{L}(\mathfrak{A}(f_n)) = \mathbb{L}(G)$.*

On dit que $\mathfrak{A}$ identifie une classe $\mathcal{G}$ à la limite en temps CS-polynomial *si (1) $\mathfrak{A}$ identifie $\mathcal{G}$ à la limite, (2) $\mathfrak{A}$ a un temps de mise à jour polynomial et (3) il existe un polynôme $p()$ tel que toute grammaire $G \in \mathcal{G}$ admette un ensemble caractéristique $C \subseteq \mathbb{X}$ de taille $\|C\| \leq p(\|G\|)$.*

Notons que des trois contraintes, la troisième est sans doute la plus importante. En effet, le nombre de changements d'hypothèses ou le nombre d'erreurs implicites n'ont de sens que si l'on est effectivement en présence de présentations, donc que les données arrivent sous la forme d'un flux. A l'inverse, la troisième porte sur des ensembles caractéristiques d'informations permettant d'identifier des grammaires. Ainsi, elle continue d'avoir un sens si nous n'avions pas affaire à un flux de données, mais à un *échantillon* (*i.e.*, un ensemble) de données.

Dans ce dernier cadre, des définitions plus contraignantes existent. On peut par exemple exiger qu'en l'absence d'ensemble caractéristique, les réponses de l'algorithme soient consistantes avec les données d'apprentissage [dlH97] ou que, sachant la cible, le calcul de l'ensemble caractéristique soit faisable en temps polynomial [GM96].

Panorama des résultats d'identification pour les boules et les Afd

Dans [dlHJT08, Tan09], nous avons fait une étude systématique de l'apprenabilité des bonnes boules, des Afd et des boules quelconques selon les critères d'identification polynomiale exposés précédemment, et nous avons obtenu la table suivante :

Critère	$\mathcal{GOODBALL}(\Sigma)$	$\mathcal{AFD}(\Sigma)$	$\mathcal{BALL}(\Sigma)$
MC Texte	Oui	Non	Non
CS Texte	Oui	Non	Non
IPE Texte	Oui	Non	Non
MC Informant	Oui	Oui [dlHJT08, Theo.9]	Oui
CS Informant	Oui	Oui [Gol78, OG92]	Non
IPE Informant	Non	Non [Pit89, Corol.6]	Non

Certains de ces résultats sont connus de longue date (en particulier ceux qui concernent les Afd), et d'autres sont immédiats : un résultat négatif pour les bonnes boules est forcément négatif pour les boules quelconques ; de même, un résultat positif pour les boules quelconques est nécessairement positif pour les bonnes boules.

Concernant l'apprenabilité à partir de Textes, on remarque que les résultats sont indépendants du critère de polynomialité choisi. Plus précisément, ce sont les classes de langages que nous considérons qui ne permettent pas de discriminer les différentes contraintes

de polynomialité dans le cas de TEXTES[2]. Le fait que les AFD ne soient pas identifiables en temps X-polynomial à partir de TEXTES vient du fait qu'ils ne sont pas identifiables à la limite à partir de TEXTES [Gol67], comme nous l'avons déjà évoqués. Concernant les résultats obtenus pour les boules quelconques, leurs preuves ont été élaborées pendant la rédaction de [Tan09] ; nous les reprenons dans l'Annexe A.2. Enfin, concernant les résultats positifs d'apprenabilité pour les bonnes boules, nous les avons obtenus dans [dlHJT08] en introduisant la notion de *témoin de minimalité*. Comme c'est une technique puissante, nous lui consacrerons la Section 3.1.2.

Au contraire de ce qu'on observe avec les TEXTES, utiliser des INFORMANTS permet de discriminer les différentes contraintes de polynomialité : il apparaît qu'identifier en temps MC-polynomial est plus facile qu'identifier en temps CS-polynomial, ce qui est plus facile qu'identifier en temps IPE-polynomial. Bien sûr, on ne peut pas tirer de règle générale à partir de cas particuliers, mais les techniques de preuves utilisées permettent de comprendre les raisons de ce constat.

Concernant les résultats, tous positifs, d'apprenabilité en temps MC-polynomial, ils résultent du fait que pour chaque grammaire (minimale) G que nous considérons, il existe des ensembles de données E qui (1) caractérisent parfaitement G et (2) sont repérables au sein d'un échantillon d'apprentissage. Aussi, un algorithme trivial qui se contente d'attendre de telles données sans changer d'avis identifie bien la grammaire cible en temps MC-polynomial, avec 1 seul changement d'avis (voir l'exemple dans la note de bas de page, voir également l'Annexe A.2 pour le cas des boules et la Section 3.1.3 pour le cas plus élaboré des AFD). Mais bien entendu, la taille de E peut être exponentielle en celle de G. Comme ce critère n'est pas pris en compte pour l'identification en temps MC-polynomial (il l'est pour l'identification en temps CS-polynomial), cela conduit à des résultats d'identification très faibles.

A l'inverse, tous nos résultats d'identification en temps IPE-polynomial sont négatifs. Pour les montrer, on utilise une technique introduite par Pitt dans [Pit89] : si l'une de nos classes était identifiable en temps IPE-polynomial à partir d'INFORMANTS, alors elle le serait avec des requêtes d'équivalence uniquement, et nous verrons que ce n'est pas le cas (en Section 4.2.2). En fait, comme les classes identifiables avec des requêtes d'équivalence uniquement sont plutôt rares, l'identification en temps IPE-polynomial à partir d'INFORMANTS est un cadre très (voire trop) exigeant. D'ailleurs, des définitions plus souples d'identification en temps IPE-polynomial ont été étudiées dans la littérature (par exemple, [Yok95]).

Cette étude exhaustive permet également d'observer un phénomène intéressant pour les bonnes boules, dans le cadre de l'identification en temps IPE-polynomial : celle-ci est possible à partir de TEXTES mais pas à partir d'INFORMANTS. C'est donc un cas "étrange" où on peut apprendre à partir de données positives seulement, mais pas à partir de données positives et négatives. D'un certain point de vue, il faut donc se méfier de l'intuition selon laquelle, plus on a d'information sur une cible, plus il est facile de l'identifier. D'un autre point de vue, on peut s'interroger sur la portée des subtilités d'un paradigme qui induit ce type de constatations.

Au final, les résultats d'identification les plus raisonnables sont manifestement obtenus sous la contrainte d'identification en temps CS-polynomial. Ce n'est pas un constat très

[2]Si nous avions choisi la classe $\mathcal{G}_n$ des langages $L_w = \Sigma^n \setminus \{w\}$ de tous les mots de longueur n sauf un, w, alors nous pourrions l'identifier à partir de TEXTES en temps MC-polynomial (avec un algorithme trivial attendant d'avoir vu tous les mots de longueur n sauf 1, et changeant alors d'avis une seule fois pour retourner la bonne hypothèse), mais pas en temps IPE-polynomial (car cette classe n'est pas identifiable avec des requêtes d'équivalence uniquement, voir Section 4.2.2).

original, mais comme initialement, il a été formulé pour les AFD et que nous avons besoin des outils mis en place dans ce cadre pour la suite de ce manuscrit (en particulier, l'algorithme RPNI [OG92]), nous y reviendrons dans la Section 3.1.3.

3.1.2 Identification des bonnes boules à partir de TEXTES

Les bonnes boules sont identifiables à partir de TEXTES, quel que soit le critère de polynomialité choisi. Pour démontrer ce résultat, nous avons introduit, dans [dlHJT08], la notion de *témoin de minimalité* pour un ensemble fini $E = \{x_1, x_2, \ldots, x_n\}$ de mots. Un tel témoin est un quintuplet (u, v, w, o, r) tel que

1. $u, v \in E$ soient deux mots de longueur maximum dans E,
2. $w \in E$ soit de longueur minimum dans E,
3. u et v admettent o comme *unique* plus long sous-mot commun,
4. r soit un entier tel que $|u| - |w| = 2r$ et $|u| - |o| = r$ et
5. $E \subseteq B_r(o)$.

Toute bonne boule admet des témoins de minimalité. En particulier, si 0 et 1 sont des lettres, alors le tuple $(0^r o, 1^r o, w, o, r)$ avec $w \in B_r(o)$ de longueur $|o| - r$ est un témoin pour $B_r(o)$. De plus, on a le résultat suivant :

Lemme 2 *Soit $E = \{x_1, x_2, \ldots, x_n\}$ un ensemble de mots.*

1. *On peut déterminer si E admet un témoin de minimalité en temps polynomial, avec un algorithme qui exhibe un tel témoin quand il existe ;*
2. *Si (u, v, w, o, r) est un témoin de minimalité pour E, alors pour toute boule $B_{r'}(o')$ contenant E, soit $r' > r$, soit ($r' = r$ et $o' = o$) ; autrement dit, $B_r(o)$ est l'unique boule de rayon minimum qui contient E.*

Preuve: Concernant le premier point, on itère sur tous les triplets (u, v, w) de mots satisfaisant les conditions (1) et (2), puis on calcule r en vérifiant que $|u| - |w|$ est pair (condition (4)). La condition (3) est plus délicate : pour deux mots u et v de même longueur n, on peut avoir $\#(\text{LCS}(u, v)) > 1.442^n$ [Gre03], ce qui condamme toute énumération de $\text{LCS}(u, v)$ en temps polynomial. Cependant, il existe une structure de données, le LCS-graphe [Ric00], permettant de stocker $\text{LCS}(u, v)$; on peut construire ce graphe en temps et en espace $\mathcal{O}(|u| \cdot |v|)$ [Gre02], puis l'utiliser pour vérifier que $\text{LCS}(u, v)$ est bien un singleton, ce qui permet ensuite de trouver o. Il suffit enfin de vérifier les conditions (4) et (5).

Concernant le second point, supposons que $E \subseteq B_{r'}(o')$. On a $d(u, w) \geq |u| - |w| = 2r$, et comme $\{u, w\} \subseteq E \subseteq B_{r'}(o')$, on a aussi $d(u, w) \leq 2r'$, donc $r \leq r'$. Supposons que $r' = r$. On a $d(u, w) = 2r \leq d(u, o') + d(o', w)$ et comme $u, w \in B_r(o')$, on a aussi $d(u, o') \leq r$ et $d(o', w) \leq r$, donc $d(u, o') = d(o', w) = r$. On en déduit que $|o'| = |o|$. En effet, comme $|u| - |o'| \leq d(u, o') = r$, on a $|o'| \geq |u| - r = |o|$, et comme $|o'| - |w| \leq d(o', w) = r$, on a $|o'| \leq |w| + r = |o|$. Enfin, si $o' \neq o$, comme $\text{LCS}(u, v) = \{o\}$ et $|o| = |o'|$, on a $o' \notin \text{LCS}(u, v)$, c'est-à-dire soit $o' \not\preceq u$, soit $o' \not\preceq v$. Mais dans ce cas soit $d(o', u) > |u| - |o'| = r$, soit $d(o', v) > r$, ce qui est impossible puisque $u, w \in B_r(o')$. Donc si $r' = r$ alors $o' = o$. □

Exploitation des témoins de minimalité

Disposer de témoins de minimalité est intéressant pour identifier les bonnes boules en temps MC-polynomial, comme le montre l'Algorithme 3. En effet, si un témoin (u, v, w, o, r) est détecté dans un TEXTE, alors soit l'algorithme converge vers $B_r(o)$, soit on est certain que la boule cible a un rayon $r' > r$, ce qui permet d'attendre, sans changer d'avis, un nouveau témoin pour une boule, qui sera forcément de plus grand rayon. Pendant cette période, les boules renvoyées par l'algorithme ne sont généralement pas consistantes avec les données. Malgré tout, pour une boule cible $B_r(o)$, on dispose d'un algorithme faisant au plus r changements d'avis, soit un nombre polynomial (puisque $r \leq |o|$ pour les bonnes boules), avant de converger. Notons de plus que si (u, v, w, o, r) est un témoin pour la boule cible, alors l'ensemble $\{u, v, w\}$ joue le rôle d'un ensemble caractéristique pour cet algorithme, et donc il identifie aussi les bonnes boules en temps CS-polynomial.

Algorithm 3: IDFGOODBALLSFROMTEXT(f), version simplifiée

Input: Un texte $f = x_1x_2 \dots x_n$
Output: Une bonne boule (x, s)

```
if |f| = 1 then
    return (x_1, |x_1|) ;
else if f̂ admet un témoin de minimalité (u, v, w, o, r) then
    return (o, r) ;
else
    return IDFGOODBALLSFROMTEXT(x_1 x_2 ... x_{n-1}) ;
```

L'Algorithme 4 suivant est plus sophistiqué : il identifie lui aussi $\mathcal{GOODBALL}(\Sigma)$ en temps MC et CS-polynomial, mais il présente l'avantage d'être consistant, et permet donc d'identifier $\mathcal{GOODBALL}(\Sigma)$ en temps IPE-polynomial (grâce au Lemme 1). Plus précisément, il retourne deux types de boules : soit des boules valides qui proviennent de témoins de minimalité, soit des boules *refuges* dont le rayon est suffisamment grand pour contenir toutes les données du texte. Pour cela, on choisit un plus long mot c dans le texte comme centre[3], et on prend $|c|$ comme rayon de la boule refuge. En analysant les différentes situations où un changement d'avis peut se produire au cours de l'identification d'une boule $B_r(o)$, on peut montrer que leur nombre est majoré par $4r$ [dlHJT08, Theo.6].

Généralisation de le technique

La technique précédente est intéressante, car on peut facilement la réutiliser pour d'autres classes de grammaires. Aucune définition générale de la notion de témoin de minimalité n'existe actuellement, mais nous savons décrire ses caractéristiques. Ainsi, pour une grammaire cible (minimale) G, un tel témoin est un ensemble de données $E \subseteq \mathbb{X}$ tel que G soit consistant avec E, et pour toute grammaire H consistante avec E, soit $H = G$, soit $\|H\| > \|G\|$; en outre, il est nécessaire qu'on puisse décider en temps polynomial si un ensemble $E \subseteq \mathbb{X}$ est un témoin de minimalité pour une certaine grammaire, et dans ce cas, qu'on sache calculer cette grammaire à partir de E en temps polynomial.

[3]On choisit ici le plus grand au sens *length-lexicographique*, mais un autre conviendrait parfaitement.

Algorithm 4: IdfGoodBallsFromText(f), version complète

Input: Un texte $f = x_1 x_2 \dots x_n$
Output: Une bonne boule (x, s)

```
if |f| = 1 then
    return (x_1, |x_1|) ;
else if f̂ admet un témoin de minimalité (u, v, w, o, r) then
    return (o, r) ;
else
    (o', r') ← IdfGoodBallsFromText(x_1 x_2 ... x_{n-1}) ;
    if x_n ∈ B_{r'}(o') then
        return (o', r') ;
    else
        c ← max_⪯{f̂} ;
        return (c, |c|) ;
```

Si des témoins existent pour chaque grammaire (minimale) de la classe $\mathcal{G}$, alors cette classe est identifiable en temps MC-polynomial : l'algorithme d'identification se contente de vérifier si les données qu'il a reçu forment un témoin de minimalité, et dans ce cas uniquement, change d'avis au profit d'une grammaire forcément plus proche, en taille, de la grammaire cible. Par ailleurs, s'il existe un polynôme $p()$ tel que pour chaque grammaire cible G, il existe au moins un témoin de minimalité E de taille $\mathcal{O}(p(\|G\|)$, alors l'algorithme précédent identifie $\mathcal{G}$ en temps CS-polynomial. Si enfin, l'algorithme précédent est consistant (en changeant par exemple d'avis au profit de grammaires refuges), alors il identifie $\mathcal{G}$ en temps IPE-polynomial.

Au-delà du cas des bonnes boules et des Afd (que nous verrons dans la section suivante), prenons l'exemple de l'identification à partir de Textes de la classe $\mathcal{IDEAUX}(\Sigma)$, contenant les langages $\widehat{w} = \{u \in \Sigma^* : u \preceq w\}$ des sous-mots de $w \in \Sigma^*$. Un ensemble E de mots sera un *témoin de minimalité* s'il contient un mot $m \in E$ tel que tout autre mot de E soit sous-mot de m. Dans ce cas, on peut considérer l'Algorithme 5. Celui-ci ne change d'avis qu'au profit d'une hypothèse valide quand un témoin de minimalité est trouvé, ou d'une hypothèse refuge quand ce témoin est absent ; pour la construire, il suffit de remarquer que tout mot de longueur inférieure ou égale à p sur un alphabet $\Sigma = \{a_1, a_2, \dots, a_k\}$ est forcément un sous-mot de $(a_1 a_2 \dots a_k)^p$. Donc pour une "grammaire" cible w, cet algorithme fait moins de $2|w|$ changements d'avis, et comme il est consistant, on en déduit qu'$\mathcal{IDEAUX}(\Sigma)$ est identifiable à partir de Textes en temps MC, CS et IPE-polynomial.

De même, considérons la classe $\mathcal{AFER}(\Sigma)$ des *automates finis à états résiduels* [DLT02] et le problème de son identification à partir d'Informants. Pour cette classe d'automates non déterministes, il existe des automates *canoniques* (qui sont minimaux en nombre d'états, et maximaux en nombre de transitions), et des témoins de minimalité similaires à ceux que nous donnerons pour les Afd minimaux. Malheureusement, ils peuvent être exponentiels en la taille de l'Afer cible (car pour certains Afer canonique cible, tout ensemble caractéristique est nécessairement de taille exponentielle [DLT04]). Du coup, on peut les identifier en temps MC-polynomial mais pas en temps CS-polynomial. En outre, comme tout Afd est un Afer particulier et que les Afd ne sont pas identifiables en temps IPE-polynomial, les Afer ne sont pas non plus identifiables en temps IPE-polynomial.

Algorithm 5: IdfIdealsFromText(f)

Input: Un texte $f = x_1 x_2 \dots x_n$
Output: La représentation w de l'idéal $\widehat{w}$
// On suppose que l'alphabet est $\Sigma = \{a_1, a_2, \dots, a_k\}$
if $\widehat{f}$ admet un témoin de minimalité m **then**
  return m ; // idéal valide
else
  $p \leftarrow \max\{|x| : x \in \widehat{f}\}$;
  return $(a_1 a_2 \dots a_k)^p$; // idéal refuge

3.1.3 Identification des Afd à partir d'Informants

Comme nous l'avons déjà dit, les résultats d'identification de la classe $\mathcal{AFD}(\Sigma)$ à partir d'Informants sont connus depuis longtemps. Nous n'en avons évidemment pas la paternité[4], mais nous souhaitons les reprendre pour deux raisons. D'une part, l'identification en temps MC-polynomial des Afd peut se démontrer à l'aide de témoins de minimalité, ce qui nous permettra d'illustrer, une nouvelle fois, l'intérêt de cette notion. D'autre part, ces résultats reposent en partie sur l'algorithme Rpni [OG92]. Or comme nous rencontrerons de nouveau cet algorithme dans le chapitre suivant, il est utile de le présenter dès à présent.

Identification des Afd en temps MC-polynomial

Considérons un automate minimal complet $A = \langle Q, \Sigma, i, F, \delta \rangle$; pour tout état $q \in Q$, on définit le mot $w_q = \min_{\trianglelefteq}\{w \in \Sigma^* : \delta(i, w) = q\}$. Puis on appelle *témoin de minimalité* de A tout ensemble $E \subseteq \Sigma^* \times \{+, -\}$ vérifiant les conditions suivantes :

1. Tous les états de A sont visités, qu'ils soient finaux ou non : pour tout état $p \in Q$, si $p \in F$, alors $(w_p, +) \in E$ et si $p \notin F$, alors $(w_p, -) \in E$;
2. Toutes les transitions sont exercées : Pour tout état $p \in Q$ et toute lettre $a \in \Sigma$, si $\delta(p, a) \in F$, alors $(w_p a, +) \in E$, et si $\delta(p, a) \notin F$, alors $(w_p a, -) \in E$;
3. Tout état qui est l'extrémité d'une transition est distinguable d'un autre état : pour tous états $p, q \in Q$ et toute lettre $a \in \Sigma$, si $\delta(p, a) \neq q$, alors il existe $u \in \Sigma^*$ tel que soit $\{(w_p au, +), (w_q u, -)\} \subseteq E$, soit $\{(w_p au, -), (w_q u, +)\} \subseteq E$.

On peut montrer qu'un tel ensemble E est *caractéristique* pour l'algorithme Rpni[5]. Donc si un échantillon d'apprentissage contient E, alors le résultat de Rpni sur cet échantillon sera l'automate minimal A. Toutefois, un lecteur avisé notera que la définition du témoin E est plus contraignante que celles des ensembles caractéristiques pour Rpni qu'on trouve dans la littérature (par exemple, [DM98]). Par exemple, si un état p de A n'est pas final, on exige malgré tout que $(w_p, -) \in E$; or cette information n'est pas utile pour que Rpni converge, car tout état indéterminé est considéré comme non acceptant.

[4] sauf, peut-être, celui sur l'identification des Afd en temps MC-polynomial : nous n'avons pas retrouvé de trace de ce résultat dans la littérature.

[5] En fait, comme Rpni apprend sur des échantillons d'apprentissage plutôt que sur des informants, cette notion d'ensemble caractéristique n'est pas tout à fait la même que celle de notre Définition 6, mais les différences sont minimes.

En fait, notre définition permet d'établir une propriété plus forte que la convergence de Rpni : si un Afd B est consistant avec l'ensemble E, alors soit $B = A$, soit B a au moins un état de plus que A [dlHJT08, Theo.9]. Ainsi, notre témoin est bien un témoin de minimalité au sens de la section précédente, qu'on va maintenant pouvoir utiliser pour montrer qu'$\mathcal{AFD}(\Sigma)$ est identifiable en temps MC-polynomial.

Pour cela, considérons l'Algorithme 6. Cet algorithme, basé sur Rpni, identifie clairement $\mathcal{AFD}(\Sigma)$ en temps CS-polynomial : lorsque le témoin de minimalité de l'Afd cible apparait dans les données, l'algorithme converge. Toutefois, il vérifie à chaque étape si l'Informant dont il dispose est un témoin de minimalité de l'automate courant, et ne change d'avis que dans ce cas-là. C'est donc un algorithme non consistant, mais par la propriété de minimalité, s'il n'a pas convergé, alors nous savons que tout Afd compatible avec les données, dont l'Afd cible, aura strictement plus d'états que l'Afd courant. Ainsi, le nombre de changements d'avis de l'algorithme est au plus linéaire en la taille de l'automate cible, et donc cet algorithme identifie bien $\mathcal{AFD}(\Sigma)$ en temps MC-polynomial (en plus de l'identifier en temps CS-polynomial).

Algorithm 6: IdfDfaFromInformant(f)

Input: Un informant $f = (x_1, \ell_1)(x_2, \ell_2)\dots(x_n, \ell_n)$
Output: Un automate de $\mathcal{AFD}(\Sigma)$

```
1 if |f| = 0 then
2 |  return ⟨{0}, Σ, 0, ∅, ∅⟩ ;
3 else
4 |  E+ ← {x_i : (x_i, +) ∈ f̂} ; E− ← {x_i : (x_i, −) ∈ f̂} ;
5 |  A ← Rpni(E+, E−) ;
6 |  if f̂ est un témoin de minimalité pour A then
7 |  |  return A ;
  |  |  // si A n'est pas la cible, alors tout Afd compatible avec les
  |  |     données, dont l'Afd cible, a strictement plus d'états que A
8 |  else
9 |  |  return IdfDfaFromInformant((x_1, ℓ_1)(x_2, ℓ_2)...(x_{n−1}, ℓ_{n−1})) ;
  |  |  // on ne change pas d'avis
```

Principe de l'algorithme Rpni

Rappelons maintenant les principes de fonctionnement de Rpni [OG92]. Nous ne les exposons pas uniquement par soucis d'exhaustivité, mais parce que nous retrouverons cet algorithme dans le chapitre suivant. En outre, il appartient désormais au folklore de l'Inférence Grammaticale ; la version que nous en donnons est proche de celle de [DM98].

En entrée, Rpni prend un ensemble E_+ de mots reconnus par un Afd cible (exemples) et un ensemble E_- de mots que cet Afd rejette (contre-exemples). Après la phase d'inférence, Rpni retourne un Afd qui généralise les données de E_+ tout en rejetant les données de E_-. De plus, si les données contiennent un ensemble caractéristique de l'Afd cible, alors on peut montrer que Rpni trouve, en temps polynomial, cet Afd. A titre d'exemple, nous prendrons $E_+ = \{\lambda, 1, 01, 011, 101\}$ et $E_- = \{00, 10\}$.

La première tâche entreprise par Rpni est la construction du Pta (*prefix tree acceptor*) des mots de E_+. Ce Pta est le plus grand Afd reconnaissant uniquement les mots de E_+.

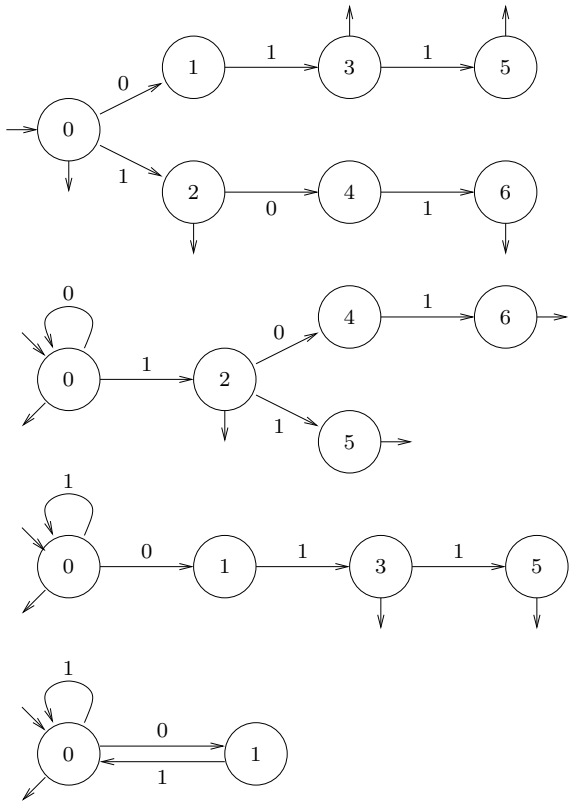


FIG. 3.1 – Le premier AFD est le PTA de $\{\lambda, 1, 01, 011, 101\}$; le second résulte de la fusion des états 1 et 0 dans le PTA, et le troisième, de la fusion des états 2 and 0. Le dernier résulte, lui, de la fusion des états 3 et 0 à partir du troisième.

Il a la forme d'un arbre et ses états sont numérotés selon l'ordre *length-lexicographique* $\trianglelefteq$ sur les préfixes de E_+. Pour le définir formellement, posons $S = \text{pref}(E_+)$ et supposons que S soit un ensemble de N mots. Notons $\underline{w}$ le numéro d'apparition (entre 0 et $N-1$) de w dans S quand on parcourt S selon $\trianglelefteq$. On appelle PTA de E_+ l'AFD $A = \langle Q, \Sigma, i, F, \delta \rangle$ tel que $Q = \{\underline{w} : w \in S\} = \{0, 1, \ldots, N-1\}$, $i = \underline{\lambda} = 0$, $F = \{\underline{w} : w \in E_+\}$ et $\delta = \{(\underline{w}, a) \mapsto \underline{wa} : w \in S, a \in \Sigma, wa \in S\}$. Dans notre exemple, le PTA des mots de E_+ est le premier AFD de la Figure 3.1.

Une fois le PTA construit, RPNI parcourt les états dans l'ordre de la numérotation. Pour traiter le $i$ème état, RPNI essaie de le fusionner avec les états $0, 1, \ldots i-1$ (dans cet ordre). Fusionner deux états consiste à les regrouper en un seul dont le numéro est le minimum entre ceux des deux états fusionnés. Cet état est final si l'un des deux états l'était. Quant aux transitions sortant des deux états fusionnés, elles sont elles-mêmes fusionnées lorsqu'elles sont étiquetées par la même lettre, et dans ce cas, les deux états qu'elles pointent sont fusionnés, de manière récursive. Ainsi, dans la Figure 3.1, RPNI tente la fusion des états 1 et 0 du PTA. Ceci crée une boucle étiquetée par 0 sur l'état 0. Comme 1 et 0 ont tous deux des transitions sortantes d'étiquette 1, les états 3 et 2 qu'elles pointent sont eux-mêmes fusionnés, ce qui conduit au second AFD de la Figure.

Une fusion *réussit* si aucun exemple de E_- n'est accepté par l'AFD résultant de cette

fusion (*i.e.*, l'AFD est consistant avec les données) ; elle *échoue* sinon. Dans notre exemple, la fusion de 1 et 0 échoue puisque $00 \in E_-$ est accepté. Donc RPNI abandonne cette fusion et tente la fusion des états 2 et 0, ce qui conduit au troisième AFD dans la Figure 3.1. Celui-ci n'accepte aucun exemple de E_- donc la fusion réussit. RPNI repart donc de cet AFD et tente, avec succès, la fusion des états 3 et 0, ce qui conduit au dernier AFD représenté. Comme plus aucune fusion n'est possible, cet AFD est le résultat final de RPNI sur ces données.

Pour une description plus formelle de RPNI, considérons un AFD $A = \langle Q, \Sigma, i, F, \delta \rangle$. Une *partition* des états de A est un ensemble $\pi = \{B_1, B_2, \ldots, B_k\}$ composé de sous-ensembles de Q tels que (1) $B_i \cap B_j = \emptyset$ pour tout $i \neq j$ et (2) $\bigcup_{i=1}^{k} B_i = Q$; on note alors $\pi(q)$ l'unique bloc B_i tel que $q \in B_i$.

On dit que π est une partition *déterminisante* si π est compatible avec la fonction de transitions de A, c'est-à-dire, pour tous états $p_1, p_2, q_1, q_2 \in Q$ et toute lettre $a \in \Sigma$, si $\pi(p_1) = \pi(p_2)$ et $\delta(p_1, a) = q_1$ et $\delta(p_2, a) = q_2$, alors $\pi(q_1) = \pi(q_2)$. Dans ce cas, on peut définir l'AFD *quotient de A par π* en posant $A/\pi = \langle Q/\pi, \Sigma, \pi(i), F/\pi, \delta/\pi \rangle$ avec $Q/\pi = \{\pi(q) : q \in Q\}$, $F/\pi = \{\pi(q) : q \in F\}$ et $\delta/\pi = \{(\pi(p), a) \mapsto \pi(q) : p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

Enfin, étant donnés une partition déterminisante π et deux états p, q, fusionner p et q revient à calculer la plus petite partition déterminisante π' telle que (1) $\pi'(r) \supseteq \pi(r)$ pour tout état $r \in Q$, et (2) $\pi'(p) = \pi'(q)$; c'est donc la fermeture de $\pi \setminus \{\pi(p), \pi(q)\} \cup \{\pi(p) \cup \pi(q)\}$ par la fonction de transition δ. On peut faire ce calcul efficacement en utilisant des UNION_FIND comme structure de données pour représenter les partitions [CLR92].

Nous donnons maintenant un pseudo-code de RPNI dans l'Algorithme 7. A chaque étape, il maintient une partition déterminisante π des états du PTA A de E_+. L'AFD courant, A/π, reconnaît tous les mots de E_+ et rejette tous les mots de E_-. L'ordre de parcours des états est maintenu grâce à la numérotation des états du PTA. Après avoir fusionné deux états, on vérifie que l'AFD A/π' n'accepte aucun mot de E_- avec la fonction COMPATIBLE. Par construction, A/π' accepte nécessairement tous les mots de E_+, ce qui préserve l'invariant de la boucle principale. Enfin, les conditions techniques $p = \min\{\pi(p)\}$ et $q = \min\{\pi(q)\}$ nous évitent de considérer des états qui ont déjà été fusionnés (avec des états d'indices qui sont forcément plus petits), donc de refaire inutilement du travail.

Algorithm 7: RPNI(E_+, E_-)

Input: Deux ensembles $E_+, E_- \subseteq \Sigma^*$ tels que $E_+ \cap E_- = \emptyset$
Output: Un AFD A tel que $E_+ \subseteq \mathbb{L}(A)$ et $\mathbb{L}(A) \cap E_- = \emptyset$

```
1 A ← Construire_Pta(E+) ; // on suppose que A = ⟨Q, Σ, i, F, δ⟩
2 π ← {{q} : q ∈ Q};
3 for p = 1 to |Q| − 1 do
4     for q = 0 to p − 1 do
5         if p = min{π(p)} et q = min{π(q)} then
6             π' ← Calcul_fusion(π, p, q, δ);
7             if Compatible(A/π', E−) then π ← π' ;
8 return(A/π)
```

Deux décennies d'Inférence Grammaticale avec RPNI

Nous souhaitons, pour finir, donner une idée de la portée de l'algorithme RPNI en Inférence Grammaticale. Son succès s'explique peut-être parce que ses créateurs ont su faire la synthèse entre des idées qui avaient été développées pendant les 20 années précédentes, comme l'idée d'exploiter des contre-exemples pour rejeter la fusion de deux états [TB73], ou le passage par un ensemble caractéristique [Gol78], et des idées nouvelles, comme le fait d'apprendre en généralisant un PTA, ou de fixer un ordre dans les fusions. Par la suite, de nombreux auteurs ont étudié l'influence des différentes options d'élaboration de l'algorithme.

Tout d'abord, il existe divers choix d'implémentation. Certaines versions de l'algorithme utilisent un PTA généralisé [AS94], construit à partir des données positives et négatives, et d'autres n'utilisent pas de PTA, mais introduisent incrémentalement des états qui auraient été présents si le PTA avait été construit [dlHOV96]. En outre, les structures de données utilisables sont nombreuses. Ainsi, l'étude de la complexité de l'algorithme reste un sujet de recherche (voir [dlH10, p.268]).

Ensuite, plusieurs travaux ont porté sur l'ordre dans lequel les fusions devaient être faites. Il y avait une motivation pratique dans cette question. En effet, l'ordre des fusions influence la vitesse de convergence de l'algorithme. Ainsi, plusieurs heuristiques ont été proposées, en particulier EDSM, exploitée dans l'algorithme BLUEFRINGE, vainqueur de la compétition ABBADINGO [LPP98, LPC98]. Mais dans le même temps, des travaux plus théoriques ont montré que toute technique de choix des fusions fondée sur des critères heuristiques ne permettait plus d'assurer l'existence d'ensembles caractéristiques polynomiaux [dlHOV96], ce qui n'a évidemment pas manqué de susciter un débat entre les partisans d'une Inférence Grammaticale visant des succès sur des problèmes réels, et ceux plus orthodoxes craignant (peut-être) le développement foisonnant de nouvelles heuristiques.

Le critère d'acceptation des fusions a également suscité plusieurs travaux. Ainsi, dans [JS03, JTS04], nous avons adopté des critères statistiques permettant de travailler à partir de données bruitées ; nous y reviendrons dans le chapitre suivant. En outre, plusieurs techniques comme le typage [CFKdlH04, BdlH01, OV96] visent à se prémunir contre des fusions inadéquates, en tenant compte d'informations supplémentaires sur le domaine d'application.

Au-delà, RPNI a été adapté pour apprendre des automates non déterministes [CF03], des automates stochastiques avec des algorithmes comme ALERGIA [CO94] ou MDI [TDdlH00], des automates d'arbres [GO93] ou des transducteurs de mots avec l'algorithme OSTIA [OGV93], *etc.* Dans [dlHJ04], nous l'avons exploité pour identifier des automates de Büchi reconnaisant des ω-langages sûrs. Plus récemment (mais c'est un défi que s'est donné la communauté depuis longtemps), un effort d'homogénéisation de tous ces algorithmes a été entrepris, visant à mettre en évidence leurs points communs et leurs différences [dlH10].

Enfin, il existe un lien entre RPNI et de nombreux algorithmes d'identification de grammaires autres que des automates. Pour le mettre en évidence, il faut se placer au niveau des langages plutôt que des automates : étant donné un langage $L \subseteq \Sigma^*$, on définit la *congruence à droite induite par* L sur Σ^* en posant $u \sim_L v$ si pour tout $w \in \Sigma^*$, $(uw \in L \Leftrightarrow vw \in L)$. La relation $\sim_L$ est une relation d'équivalence sur Σ^*. De plus, on peut montrer qu'un langage L est rationnel si et seulement si $\sim_L$ est d'index fini, *i.e.*, si le nombre de classes de congruence de $\sim_L$ est fini. Pour cela, on montre que pour tout AFD A reconnaissant L, il existe une surjection[6] des classes de congruence de $\sim_L$ dans l'ensemble des états de A.

[6] C'est une bijection lorsque A est un automate minimal.

Maintenant, de façon abstraite, RPNI dispose de données positives desquelles il extrait une première relation de congruence (états du PTA), plus fine que celle du langage cible L (sur les données positives). Il cherche ensuite à retrouver les classes de congruence de L (états de l'AFD cible) en fusionnant celles dont il dispose (états de l'AFD courant), guidé par les données négatives.

De même, de nombreux algorithmes d'identification de grammaires cherchent à exploiter, eux-aussi, une congruence d'index fini. On pensera par exemple aux travaux sur l'apprentissage de grammaires très simples [Yok03], des grammaires linéaires déterministes [dlHO02a], des grammaires substituables [CE05] ou des langages congruentiels [EdlHJ07]. On pensera également à l'algorithme DELETE2 [DLT04] qui vise à apprendre des AFER, des automates non déterministes dont les états sont malgré tout liés aux classes de la congruence à droite $\sim_L$. On pensera enfin à toutes les extensions directes de RPNI aux arbres et autres, que nous avons déjà évoquées.

En parcourant la littérature, j'ai souvent eu l'impression que l'exploitation ou non d'une congruence d'index finie marquait la limite entre des travaux aboutissant à des théorèmes d'identification et des travaux plus exploratoires (par exemple, [NM05]). En somme, l'existence de cette congruence favorise l'émergence de résultats théoriques solides.

3.2 Identification approximative

Le paradigme d'apprentissage PAC (pour Probablement Approximativement Correct) [Val84] a beaucoup impacté l'Inférence Grammaticale quand il est apparu, et plus encore l'Apprentissage Automatique. L'objectif de Valiant était de donner des bases solides à l'Apprentissage, en réaction à toutes les méthodes visant à mimer le comportement humain qu'avait proposées les chercheurs en Intelligence Artificielle (voir [MCM83] pour une synthèse des résultats de l'époque). Pour ce faire, Valiant se focalise sur la logique booléenne, vue comme un système de description "universel" de concepts, et propose un cadre d'apprentissage probabiliste sujet à des contraintes d'efficacité (complexité polynomiale).

Par la suite, de nombreuses variantes de ce paradigme ont existé, mais elles sont généralement équivalentes sous des hypothèses relativement faibles [HKLW91]. Nous choisissons donc un cadre possible, le cadre **one-oracle**$(1/\delta, rand, s\_known, always\_halts)$ selon la terminologie de [HKLW91], adapté aux données manipulées en Inférence Grammaticale (des mots).

3.2.1 L'apprentissage PAC

Comme dans le cadre de l'Identification à la limite, nous considérons une classe de grammaires $\mathcal{G}$, mais plutôt qu'une classe $\mathcal{P}$ de présentations, nous notons $\mathcal{D}$ la classe de toutes les distributions de probabilités sur Σ^* et nous supposons qu'il existe un oracle EX tel que pour toute grammaire $G \in \mathcal{G}$ et toute distribution $\mu \in \mathcal{D}$, $\text{EX}(G, \mu)$ retourne en temps $\mathcal{O}(1)$ un exemple (w, ℓ) où (1) w est tiré aléatoirement et indépendamment selon μ et (2) $(w \in \mathbb{L}(G) \Rightarrow \ell = +)$ et (3) $(w \notin \mathbb{L}(G) \Rightarrow \ell = -)$.

Cette hypothèse sur la façon dont sont générées les données a plusieurs avantages. D'une part, elle est rigoureuse et permet de travailler dans un cadre probabiliste solide. D'autre part, elle évite de supposer que tout exemple (ou contre-exemple) apparaît nécessairement dans les données pour qui sait attendre, comme c'est le cas pour les TEXTES ou les INFORMANTS : il

est possible que la probabilité d'apparition d'un mot soit nulle pour telle ou telle distribution de $\mathcal{D}$.

Enfin, le paradigme PAC ne vise pas à identifier une cible de façon exacte, mais à l'approximer de façon correcte, donc à minimiser l'erreur entre la cible et l'hypothèse apprise. Naturellement, cette erreur se mesure en fonction de la distribution : c'est la probabilité qu'un exemple tiré selon μ soit mal étiqueté par l'hypothèse apprise :

Définition 7 (Hypothèse ϵ-bonne) *Soit $G \in \mathcal{G}$ une grammaire cible, $\mu \in \mathcal{D}$ une distribution sur Σ^* et $\epsilon > 0$ un* paramètre d'erreur*. On dit qu'une grammaire H est une* hypothèse ϵ-bonne *par rapport à G et μ si $P_{x\sim\mu}(x \in \mathbb{L}(G) \oplus \mathbb{L}(H)) < \epsilon$.*

Cependant, comme les exemples sont tirés selon μ, on ne peut pas se prémunir d'un tirage extrêmement défavorable, donc imposer qu'un algorithme retourne, dans tous les cas, des hypothèses ϵ-bonnes : aucune classe de grammaires ne serait apprenable sous des conditions aussi sévères. Aussi, on introduit un second paramètre de *confiance* δ, et on exige l'obtention d'une hypothèse ϵ-bonne avec probabilité au moins $1 - \delta$.

Définition 8 (PAC-apprenabilité polynomiale) *Soit $\mathcal{G}$ une classe de grammaires.*

Un algorithme $\mathfrak{A}$ est un algorithme d'apprentissage PAC pour $\mathcal{G}$ *si $\mathfrak{A}$ prend en entrée le paramètre d'erreur $\epsilon > 0$, le paramètre de confiance $\delta > 0$, et un entier $n \geq 0$, et pour toute grammaire $G \in \mathcal{G}$ de taille $\|G\| \leq n$ et toute distribution $\mu \in \mathcal{D}$ sur Σ^*, ayant accès à l'oracle $\text{EX}(G, \mu)$, $\mathfrak{A}$ retourne une grammaire $H \in \mathcal{G}$ telle qu'avec probabilité au moins $1 - \delta$, H soit une hypothèse ϵ-bonne par rapport à G et μ.*

On dit que $\mathcal{G}$ est polynomialement PAC-apprenable *s'il existe un algorithme d'apprentissage PAC pour $\mathcal{G}$ qui s'exécute en temps polynomial en $1/\epsilon$, $1/\delta$, n et la longueur m du plus long exemple fournit par l'oracle pendant l'exécution.*

Outre les paramètres ϵ et δ, cette définition fait intervenir la longueur m du plus long exemple fournit par l'oracle pendant l'exécution, et une borne n sur la taille de la grammaire cible. Ce sont deux paramètres qui méritent d'être discutés.

Influence de la longueur des exemples

Dans la Définition 8, la taille m du plus long exemple fourni par l'oracle pendant l'exécution entre en jeu dans le calcul de la complexité. En effet, comme on tire des mots selon μ, il y a toujours une chance pour que le mot que l'oracle retourne soit trop long pour qu'on puisse le traiter en temps polynomial (en $1/\epsilon$, $1/\delta$ et n). C'est donc une manière d'éviter ce problème "intrinsèque" à l'utilisation de distributions sur des mots. De même, la taille du plus long contre-exemple fournit par un Oracle en réponse à une requête d'équivalence EQ entre en jeu dans le calcul de la complexité d'un algorithme d'apprentissage actif [Ang87a].

Une autre façon de procéder [War89, KV89, KV94] consiste à supposer qu'il existe une borne m sur la longueur des mots que l'oracle doit fournir, que ce paramètre est connu de l'algorithme d'apprentissage PAC et qu'il intervient dans le calcul de la complexité en temps. Cette hypothèse est "plausible" dans la mesure où on peut estimer m aussi finement qu'on le souhaite à partir d'un échantillon de mots tirés selon μ :

Lemme 3 ([dlHJT08, dlH10]) *Soit μ une distribution sur Σ^*. Alors pour tous $\epsilon, \delta > 0$, avec probabilité $> 1 - \delta$, si on tire un échantillon E d'au moins $(1/\epsilon)\ln(1/\delta)$ mots selon μ, alors la probabilité qu'un nouveau mot w tiré selon μ soit plus long que tous les mots de E est inférieure à ϵ. De façon plus formelle, si $\ell_E = \max\{|y| : y \in E\}$, alors $P_{x\sim\mu}(|x| > \ell_E) < \epsilon$.*

Preuve: Soit ℓ le plus petit entier tel que $P_{x\sim\mu}(|x| > \ell) < \epsilon$. Pour estimer ℓ, il suffit de choisir un échantillon E qui soit suffisamment grand pour être pratiquement sûr (avec probabilité $> 1-\delta$) qu'il contient un mot de longueur $\geq \ell$. Clairement, la probabilité de tirer n mots dans E de longueur $< \ell$ est $\leq (1-\epsilon)^n$. Donc la probabilité d'obtenir au moins un mot de longueur $\geq \ell$ après n tirages est $> 1-(1-\epsilon)^n$. Pour construire E, il suffit donc de choisir n de sorte que $1-(1-\epsilon)^n > 1-\delta$, c'est-à-dire, $(1-\epsilon)^n < \delta$. Comme $(1-\epsilon)^n \leq e^{-n\epsilon}$, il suffit de choisir $n \geq (1/\epsilon)\ln(1/\delta)$ pour que la propriété soit vraie. □

Nécessité d'une borne sur la taille de la cible

Le second point qu'il faut aborder est la présence d'une borne $n \geq 0$ sur la taille de la grammaire cible qui est connue de l'algorithme : s'il est naturel que cette taille intervienne dans le calcul de la complexité, on peut se poser la question de savoir pourquoi l'algorithme devrait la connaître. En fait, on peut lever cette hypothèse et supposer que l'algorithme ne connait pas n [HKLW91], mais pour montrer que les deux définitions sont équivalentes, il est alors nécessaire de relâcher le critère d'arrêt de l'algorithme, et d'utiliser un critère d'arrêt probabiliste :

Définition 9 (Pac-apprenabilité "en aveugle") *Un algorithme $\mathfrak{A}$ est un* algorithme d'apprentissage Pac "en aveugle" pour $\mathcal{G}$ *si $\mathfrak{A}$ prend en entrée le paramètre d'erreur $\epsilon > 0$ et le paramètre de confiance $\delta > 0$, et s'il existe un polynôme $p()$ tel que pour toute grammaire $G \in \mathcal{G}$ et toute distribution $\mu \in \mathcal{D}$ sur Σ^*, ayant accès à l'oracle $\mathrm{Ex}(G,\mu)$, avec probabilité au moins $1-\delta$,*

- *$\mathfrak{A}$ retourne une hypothèse $H \in \mathcal{G}$ qui est ϵ-bonne par rapport à G et μ, et*
- *le temps d'exécution de $\mathfrak{A}$ est borné par $p(1/\epsilon, 1/\delta, \|G\|, m)$ où m est la longueur du plus long exemple fournit par l'oracle pendant l'exécution.*

Autrement dit, avec probabilité $< \delta$, il est possible que l'algorithme s'exécute en un temps exponentiel, ou même qu'il ne s'arrête pas.

Pour montrer que la Pac-apprenabilité avec connaissance d'une borne sur la taille de la cible implique celle sans cette connaissance [HKLW91, Lemma 3.10], on montre qu'un apprenant peut déterminer, grâce à l'Oracle, si son hypothèse courante est ϵ-bonne ou non, mais qu'il ne peut pas le faire avec certitude :

Lemme 4 ([HKLW91, Lemma 3.9]) *Soient $G \in \mathcal{G}$ une grammaire et $\mu \in \mathcal{D}$ une distribution. Soient $\epsilon, \delta > 0$ deux paramètres, $i \geq 0$ un entier, et $H \in \mathcal{G}$ une hypothèse qu'on teste selon la procédure suivante :*

1. *Tirer un échantillon E de $m = (32/\epsilon)((i\ln 2) + \ln(2/\delta))$ exemples selon $\mathrm{Ex}(G,\mu)$;*
2. *Calculer le pourcentage e d'exemples de E mal classés par H (erreur empirique) ;*
3. *Si $e < (3/4)\epsilon$, alors accepter H, sinon rejeter H.*

On a les propriétés suivantes :

- *si l'erreur réelle de H est $\geq \epsilon$, alors la probabilité que le test précédent accepte H à tort est $< \delta/2^{i+1}$.*
- *si l'erreur réelle de H est $< \epsilon/2$, alors la probabilité que le test précédent rejette H à tort est $< \delta/2^{i+1}$.*

Le lemme précédent porte en son sein une explication du problème de l'arrêt probabiliste : comme l'apprenant va tester ses hypothèses avec des échantillons tirés selon μ, il est tout à fait possible que des tirages particulièrement défavorables le placent systématiquement dans une situation où il rejette à tort des hypothèses qui sont pourtant ϵ-bonnes. Dans ce cas, le temps d'exécution de l'algorithme d'apprentissage peut être arbitrairement grand.

A l'inverse, si l'algorithme connait une borne sur la taille de la cible, alors il peut l'utiliser pour s'arrêter après un temps d'exécution polynomial même s'il n'a pas approximé correctement la cible (comme le montre la preuve de la réciproque de l'équivalence [HKLW91, Lemma 3.8]).

3.2.2 Un cadre trop difficile pour l'Inférence Grammaticale

Passons maintenant aux résultats de PAC-apprenabilité en Inférence Grammaticale. Ceux-ci sont généralement négatifs, pour diverses raisons que nous aborderons par la suite. Commençons par le cas des bonnes boules. Les techniques typiques permettant de prouver l'impossibilité de la PAC-apprenabilité reposent souvent sur des hypothèses de complexité [PV88]. Rappelons que $\mathcal{RP}$ (pour *Randomised Polynomial Time*) est la classe de complexité des problèmes de décisions pour lesquels une machine de Turing probabiliste existe, qui (1) fonctionne en temps polynomial en fonction de la taille de l'entrée, (2) répond toujours NON sur une instance négative mais (3) ne répond OUI qu'avec une probabilité $> 1/2$ sur une instance positive (et NON sinon). L'algorithme est randomisé : il peut utiliser un générateur aléatoire pendant son exécution. Clairement, l'algorithme ne fait pas d'erreur sur une instance négative. Et sur les instances positives, comme l'erreur est $\leq 1/2$, on peut la rendre aussi petite qu'on veut en répétant l'exécution de l'algorithme autant de fois que nécessaire : après k exécutions, la probabilité de n'avoir vu que des réponses NON alors que la réponse est OUI sera $\leq 1/2^k$. Nous utiliserons l'hypothèse habituelle selon laquelle $\mathcal{RP} \neq \mathcal{NP}$ [Pap94].

Nous allons maintenant montrer que les bonnes boules ne sont pas polynomialement PAC-apprenable. Pour cela, nous allons montrer que le problème associé de consistance est $\mathcal{NP}$-complet, en construisant une réduction du problème $\mathcal{NP}$-complet **Plus Longue Sous-séquence Commune**. Si le problème de consistance est $\mathcal{NP}$-complet, et s'il existait un algorithme pouvant PAC-apprendre les bonnes boules, alors on pourrait l'utiliser pour montrer qu'un problème $\mathcal{NP}$-complet est aussi dans $\mathcal{RP}$.

Lemme 5 ([dlHJT08]) *Les trois problèmes suivants sont $\mathcal{NP}$-complets :*

1. ***Plus Grande Sous-séquence Commune*** *(LCS) : Etant donnés n mots $x_1, \ldots, x_n$ et un entier k, existe-t'il un mot w de longueur $\geq k$ qui soit un sous-mot de chaque x_i ?*
2. ***Plus Grande Sous-séquence Commune d'une Longueur Donnée*** *(LCSSGL) : Etant donnés n mots $x_1, \ldots, x_n$ de longueur $2k$, existe-t'il un mot w de longueur k qui soit un sous-mot de chaque x_i ?*
3. ***Boule Consistante*** *(CB) : Etant donnés deux ensembles de mots X_+ et X_-, existe-t'il une bonne boule contenant tous les mots de X_+ et aucun de X_- ?*

Preuve: (1) Voir [Mai77]. (2) Voir [dlHC00, Problem LCS0, page 42]. (3) Etant donnés deux ensembles X_+ et X_- et une boule $B_r(o)$, on peut vérifier en temps polynomial que $X_+ \subseteq B_r(o)$ et $X_- \cap B_r(o) = \emptyset$: il suffit de calculer la distance de chaque mot au centre puis de comparer au rayon. Donc le problème CB est dans $\mathcal{NP}$. Pour montrer qu'il est $\mathcal{NP}$-complet, on construit une réduction du problème précédent. On considère donc n mots

$x_1, \ldots, x_n$ de longueur $2k$. On pose $X_+ = \{\lambda, x_1, \ldots, x_n\}$. Pour construire X_-, on prend chaque mot x_i et on insère chaque lettre possible à chaque position possible dans chaque mot ; on construit ainsi $n(2k+1)|\Sigma|$ mots de taille $2k+1$. Il s'avère qu'une bonne boule qui doit contenir tous les mots de X_+ et aucun mot de X_- a nécessairement un centre de longueur k et un rayon de k. Le centre est donc une sous-séquence commune de longueur k des n mots de longueur $2k$. Réciproquement, si une boule est construite en utilisant comme centre une sous-séquence commune de longueur k des mots de $X_+ \setminus \{\lambda\}$, alors elle est de rayon k, et contient alors tous les mots de X_+ et aucun mot de X_-. □

Théorème 6 ([dlHJT08]) *$\mathcal{GOODBALL}(\Sigma)$ n'est pas polynomialement* PAC*-apprenable, sauf si $\mathcal{RP} = \mathcal{NP}$.*

Preuve: Supposons que $\mathcal{GOODBALL}(\Sigma)$ soit polynomialement PAC-apprenable avec un algorithme $\mathfrak{A}$ et prenons une instance $\langle X_+, X_-\rangle$ du problème CB. On définit $h = |X_+| + |X_-|$ et la distribution suivante sur Σ^* : $\mu(x) = 1/h$ si $x \in X_+ \cup X_-$, 0 sinon. On pose ensuite $\epsilon = 1/(h+1)$, $\delta < 1/2$ et $n = \max\{|w| : w \in X_+\}$. Soit $B_r(o)$ la bonne boule que retourne $\mathfrak{A}(\epsilon, \delta, n)$ et testons si ($X_+ \subseteq B_r(o)$ et $X_- \cap B_r(o) = \emptyset$). S'il n'existe pas de boule consistante, alors $B_r(o)$ n'est pas consistante avec les données, donc le test est faux. Supposons maintenant qu'il existe une boule consistante, et considérons la boule apprise $B_r(o)$. Avec probabilité au moins $1 - \delta > 1/2$, $B_r(o)$ est une hypothèse ϵ-bonne ; or les mots de probabilités non nulles ont tous une probabilité $> \epsilon$, donc l'erreur de $B_r(o)$ est nulle ; par conséquent, avec probabilité $> 1/2$, le test est vrai. Cette procédure s'exécute en temps polynomial en $1/\epsilon, 1/\delta$ et n. Donc si les bonnes boules étaient polynomialement PAC-apprenables, il existerait un algorithme randomisé permettant de résoudre un problème $\mathcal{NP}$-complet. □

Concernant la PAC-apprenabilité des AFD, de nombreuses études ont été menées [Pit89, KV94]. D'abord, notons que le problème de consistance associé, c'est-à-dire celui de construire un AFD qui soit consistant avec les données, n'est pas difficile : il suffit de construire le PTA de l'ensemble des exemples positifs. Du coup, la méthode que nous avons utilisée pour les bonnes boules, se basant sur l'hypothèse que $\mathcal{RP} \neq \mathcal{NP}$, n'est pas applicable aux AFD. Une autre technique a suscité beaucoup d'espoir. Elle reposait sur un résultat du "gang des 4" montrant que s'il existait un algorithme du type "Rasoir d'Occam" permettant d'approximer polynomialement un automate minimal à partir d'exemples et de contre-exemples, alors on pourrait PAC-apprendre les AFD [BEHW87]. Or cette approximation est impossible [PW93], et cette voie est donc sans issue. Finalement, la non-PAC-apprenabilité des AFD fait aujourd'hui consensus, mais sous des hypothèses plus fortes que le fait que $\mathcal{RP} \neq \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$, en utilisant la difficulté de problèmes usuels en cryptographie :

Théorème 7 ([KV89]) *Si $\mathcal{AFD}(\Sigma)$ était polynomialement* PAC*-apprenable, alors on pourrait inverser la fonction de cryptage de* RSA *à l'aide d'un algorithme probabiliste polynomial.*

En fait, les travaux de Kearns et Valiant amènent à des résultats plus forts : ils établissent que la classe des langages rationnels est *inhéremment non prédictible*, ce qui signifie que quel que soit la classe $\mathcal{G}$ de grammaires (de tailles raisonnables) qu'on choisit pour représenter les langages rationnels, celle-ci ne peut pas être polynomialement PAC-apprenable sous des hypothèses cryptographiques usuelles.

3.2.3 Un cadre prometteur plus "simple"

D'une certaine manière, ces résultats négatifs ont beaucoup freiné l'Inférence Grammaticale. En effet, si l'on ne peut pas construire de bonnes approximations d'un AFD cible, alors cela condamne peu ou prou le fait d'utiliser des techniques d'inférence grammaticale pour traiter des données réelles dans des problèmes de classification.

Cependant, une des critiques les plus récurrentes de paradigme PAC vient de ce qu'il nécessite de travailler avec n'importe quelle distribution μ. Et toute une communauté de chercheurs a cherché à restreindre cette condition tout en gardant des distributions qui soient suffisamment générales pour que la plupart des distributions soient toujours disponibles. Une manière théorique mais porteuse de restreindre la classe de distributions est de ne considérer que des distributions admissibles [Den01], définies avec la complexité de Kolmogorov. Pour chaque grammaire $G \in \mathcal{G}$, on définit la distribution admissible $\mathbf{m}_G$ par $\mathbf{m}_G(w) = \frac{2^{-K(w|G)}}{\alpha_G}$ où $K(w|G)$ est la taille du plus petit programme p prenant G en entrée et calculant le mot w sur une machine de Turing qui est préfixe, universelle et additivement optimale, et α_G est un terme de normalisation.

On peut alors définir la notion d'apprentissage PAC-simple [DdG96, LV91], en restreignant la classe des distributions à celles des distributions admissibles pour les grammaires cibles. Ainsi, plusieurs classes de grammaires dont $\mathcal{AFD}(\Sigma)$ ont été prouvées polynomialement PAC-simple apprenables [PH97]. Cependant, le fait d'utiliser des exemples simples permet de faire mieux, avec un paradigme d'apprentissage qui retourne des hypothèses parfaites plutôt que des hypothèses approximatives :

Définition 10 ([Den01]) *Soit $\mathcal{G}$ une classe de grammaires.*

Un algorithme $\mathfrak{A}$ est un algorithme d'apprentissage PEC simple *pour $\mathcal{G}$ si $\mathfrak{A}$ prend en entrée le paramètre de confiance $\delta > 0$ et un entier n, et pour toute grammaire $G \in \mathcal{G}$ tel que $\|G\| \leq n$, ayant accès à $\mathrm{EX}(G, \mu_G)$, $\mathfrak{A}$ retourne une grammaire $H \in \mathcal{G}$ telle qu'avec probabilité au moins $1 - \delta$, $\mathbb{L}(H) = \mathbb{L}(G)$.*

On dit que $\mathcal{G}$ est polynomialement PEC*-simple apprenable s'il existe un algorithme d'apprentissage* PEC *simple pour $\mathcal{G}$ qui soit polynomial en $1/\delta$ et n.*

Théorème 8 *$\mathcal{GOODBALL}(\Sigma)$ et $\mathcal{AFD}(\Sigma)$ sont polynomialement* PEC*-simple apprenables.*

Preuve: On utilise la Proposition 1 de [Den01] stipulant que si une classe de grammaires est polynomialement apprenable dans le modèle de Goldman et Mathias [GM96], alors elle est polynomialement PEC-simple apprenable. C'est cette technique que l'auteur utilise pour montrer le résultat concernant $\mathcal{AFD}(\Sigma)$. Concernant $\mathcal{GOODBALL}(\Sigma)$, considérons n'importe quelle bonne boule $B_r(o)$ sur un alphabet d'au moins deux lettres qu'on appelle 0 et 1. Le Professeur construit l'ensemble $S = \{(0^r o, +), (1^r o, +)\}$; ces deux mots sont dans $B_r(o)$ car ils sont obtenus par insertion de r occurrences d'une même lettre devant o ; de plus, ce sont des mots de longueur maximum dans la boule, et on a $|o| + r \leq 2|o| = \mathcal{O}(|o| + \log r)$ (puisque $r \leq |o|$). L'Adversaire complète S. Enfin l'Apprenant récupère S, sélectionne les plus longs mots, retrouve deux mots de la forme $x^k u$ et $y^k u$ qu'il aligne pour déduire $o = u$ et $r = k$. $\mathcal{GOODBALL}(\Sigma)$ est donc bien polynomialement apprenable dans le modèle de Goldman et Mathias, ce qui permet de conclure. □

Enfin, cerise sur le gateau, on peut même PEC-apprendre les AFD avec des exemples positifs seulement :

Théorème 9 ([Den01]) *$\mathcal{AFD}(\Sigma)$ est polynomialement* PEC*-simple apprenable à partir d'exemples positifs seulement.*

Malheureusement, aucun algorithme effectif n'existe pour faire cet apprentissage. Plus précisément, F. Denis a proposé un algorithme dans son article, mais il est paramétré par de nombreuses constantes qui dépendent de la machine de Turing qu'on considère, et qu'on ne peut donc absolument pas deviner, estimer, approcher. C'est donc un résultat théorique qui montre que c'est possible, mais qui n'est pas implémentable.

Concernant les bonnes boules, nous pensons également que :

Conjecture 1 *La classe $\mathcal{GOODBALL}(\Sigma)$ est polynomialement* PEC*-simple apprenable à partir d'exemples positifs seulement.*

En effet, intuitivement, étant donné une boule cible $B_r(o)$ sur un alphabet à 3 lettres, il suffit de connaître 3 mots de longueur maximum dans la boule pour retrouver o et r [BBdlHJT08, Section 5.2.4, pages 1865-1866]. Or nous avons vu que la moitié des mots d'une boule sont de longueur maximum. De plus, ces mots s'obtiennent simplement par l'insertion de r lettres dans le centre o ; aussi, la complexité de Kolmogorov de tels mots sachant la cible est relativement faible, donc nous pensons (sans preuve formelle) que ces exemples sont simples.

Pour conclure, je ne suis en fait pas sûr que la voie des exemples simples soit tout à fait la bonne, si elle conduit à des algorithmes qui ne sont pas implémentables. Mais par contre, l'idée qu'on restreigne la classe des distributions à des distributions rationnelles [DE08], ou des distributions engendrées par des automates stochastiques déterministes [VTdlH$^+$05], permettrait sans doute d'avancer : cette hypothèse parmi d'autres (comme la μ-distinguabilité [CT04]) éliminerait les distributions très étranges qui empêchent la PAC-apprenabilité, comme le montrent les travaux sur les distributions admissibles. Il reste à voir comment le fait de poser une telle hypothèse peut ensuite être exploité pour obtenir un résultat de PEC ou de PAC-apprenabilité.

3.3 Le cas des grammaires hors-contextes

Depuis quelques années, l'identification (exacte ou approximative) des GHC a suscité un regain d'intérêt. Les premiers travaux sur le sujet ont été faits au Japon à la fin des années 80 [Yok89], mais la difficulté de la tâche, et les succès des années 90 sur l'apprentissage des AFD ont détourné l'attention des chercheurs sur des problèmes plus "simples". Ce n'est que récemment, à la suite de la compétition OMPHALOS [SCvZ05], que le sujet est de nouveau sur le devant de la scène dans la communauté Inférence Grammaticale.

Ce problème est intéressant pour au moins deux raisons. Tout d'abord, dans plusieurs applications, les langages rationnels ne suffisent pas à modéliser correctement les concepts. C'est évidemment le cas des langues naturelles, écrites ou parlées qui se décrivent principalement avec des structures hors-contextes (et quelques structures sous-contextes) [BBDT06]. C'est aussi le cas en bio-informatique, pour ce qui est de la structure secondaire de l'ARN de transfert [SBH$^+$94]. En outre, les langages de balises comme XML sont intrinsèquement hors-contextes (langages de parenthèses) [Fer01, Chi01]. On retrouve enfin des structures hors-contextes en musique, en compression de textes, *etc.*

Par ailleurs, la classe des grammaires hors-contextes est suffisamment complexe pour mettre à jour toutes les failles des modèles d'apprentissage usuels. En effet, historiquement, ces paradigmes ont été mis au point sur les langages rationnels, parfois comme extension de

travaux sur l'apprentissage des fonctions booléennes (pour ce qui est du cadre PAC [Val84]), parfois pour éviter d'obtenir des résultats qui seraient contre-intuitifs (comme la possibilité d'apprendre efficacement tout automate non déterministe [dlH97]). En conséquence, toute classe de langages non rationnels, comme celle des boules de mots, celle des *pattern languages* [Ang79], ou celle des grammaires hors-contextes, est susceptible d'ébranler un paradigme apparemment solide.

Et de ce point de vue, les grammaires hors-contextes sont redoutables.

3.3.1 Difficulté de la tâche

Plusieurs analyses des caractéristiques qui compliquent le problème de l'identification de $\mathcal{GHC}(\Sigma)$ ont été faites. On consultera en particulier celle présentée par Rémi Eyraud dans sa thèse [Eyr06] et celle de Colin de la Higuera dans son livre [dlH10].

Sur l'équivalence des grammaires hors-contextes

L'équivalence de deux GHC est un problème indécidable. Ce constat implique "immédiatement" que la classe $\mathcal{GHC}(\Sigma)$ n'admette pas d'ensembles caractéristiques polynomiaux, donc qu'elle ne soit pas identifiable en temps CS-polynomial à partir d'INFORMANTS. Exprimé de façon plus technique :

Théorème 10 ([dlH97]) *Si $\mathcal{GHC}(\Sigma)$ était polynomialement caractérisable, alors l'équivalence des* GHC *serait décidable*[7].

Nous savons par ailleurs que l'identification à partir de TEXTES n'est pas non plus possible, car la classe $\mathcal{AFD}(\Sigma)$ n'est pas identifiable à partir de TEXTES, et les AFD sont des représentations polynomialement équivalentes aux grammaires régulières. Et cet argument s'applique également à la non-PAC-apprenabilité de $\mathcal{GHC}(\Sigma)$. En somme, nous savons dès à présent que l'intégralité de la classes des grammaires hors-contextes ne peut pas être identifiée dans les cadres que nous avons développés précédemment.

Pour contourner cette difficulté, il existe deux pistes. La première consiste à supposer que les données d'apprentissage ne sont pas des exemples (et éventuellement des contre-exemples), mais des informations plus riches, comme des *squelettes*, c'est-à-dire des arbres de dérivation dont seules les feuilles sont étiquetées. Et effectivement, on a :

Théorème 11 ([Sak90]) *La classe $\mathcal{GHC}(\Sigma)$ est identifiable à la limite à partir de* SQUELETTES *en temps CS-polynomial.*

Cependant, trouver des données sous la forme des squelettes dans la nature n'est pas très fréquent, ce qui limite l'intérêt pratique de ce résultat. Aussi, la seconde piste consiste à renoncer à l'identification de l'intégralité de la classe $\mathcal{GHC}(\Sigma)$, en considérant donc des sous-classes de GHC avec des propriétés intéressantes.

[7] De façon encore plus technique, on peut également noter que dans le modèle de Goldman et Mathias [GM96], la fonction qui permettrait à un Professeur d'extraire un *teaching set* à partir d'une GHC n'est pas calculable. Sinon, il suffirait d'utiliser cet algorithme pour extraire des *teaching sets*, puis décider de l'équivalence de deux GHC en testant la consistance de ces ensembles avec les grammaires.

Sur le déterminisme des langages hors-contextes

Il y a des raisons intuitives à vouloir travailler avec des représentations déterministes : la classe $\mathcal{AFD}(\Sigma)$ est identifiable en temps CS-polynomial à partir d'INFORMANTS alors que celle des automates finis non déterministes ne l'est pas [dlH97]. Pour les langages hors-contextes, le déterminisme est une propriété des automates à pile qui les reconnaissent. Par extension, on dit qu'une grammaire est déterministe si elle engendre un langage qui peut être reconnu par un automate à pile déterministe.

Mais les choses sont évidemment plus compliquées que pour les langages rationnels. D'abord, tous les langages hors-contextes ne sont pas déterministes. C'est le cas des langages intrinsèquement ambigus comme $\{0^n1^n2^m : n, m \geq 1\} \cup \{0^m1^n2^n : n, m \geq 1\}$, ou d'un langage non ambigu mais non déterministe comme $\{0^n1^n : n \geq 1\} \cup \{0^n1^{2n} : n \geq 1\}$. De plus, le problème "Etant donnée une GHC quelconque G, le langage $\mathbb{L}(G)$ est-il déterministe ?" est indécidable [Har78, Theorem 8.5.3].

Mais par contre, on connait des classes de grammaires qui, structurellement, n'engendrent que des langages déterministes. De plus, l'équivalence des automates à pile déterministes est décidable [Sén02], ce qui permet, au moins potentiellement, de contourner les problèmes du paragraphe précédent. Toutefois, cet argument tourne court en face de difficultés techniques extrêmes : aucun algorithme efficace n'existe à l'heure actuelle pour tester cette équivalence. De plus, on ne sait pas s'il existe des formes canoniques calculables (automates minimaux) pour les langages déterministes. Or c'est handicapant car c'est généralement ces formes canoniques qui sont obtenues par les algorithmes d'identification.

Enfin, on sait d'ores et déjà que l'identification en temps CS-polynomial de l'intégralité de cette classe de grammaires est impossible, car parmi eux, il continue d'exister des grammaires avec de mauvaises propriétés.

Sur l'expansivité des grammaires hors-contextes

Dans le cadre de l'identification en temps CS-polynomial, on exige que la taille des ensembles caractéristiques soient polynomialement liée à celle des grammaires cibles. Or il existe des grammaires très simples dont le plus petit mot engendré est de taille exponentielle en la taille de la grammaire : on dit qu'une telle grammaire est *expansive* [dlH06a]. Un exemple classique est celui de la grammaire $\langle\{0\}, \{N_0, N_1 \dots N_k\}, P, N_0\rangle$ où pour tout $i < k$, $(N_i \to N_{i+1}N_{i+1}) \in P$, et $(N_k \to 0) \in P$. Le plus petit mot qu'engendre cette grammaire est 0^{2^k}. Donc de fait, aucun ensemble caractéristique polynomial ne peut exister pour cette grammaire. Pourtant le langage reconnu est fini (et déterministe).

Une manière de s'en sortir est de restreindre encore la classe des grammaires déterministes, en imposant un critère de linéarité sur les règles : on dit qu'une grammaire est *linéaire* si chaque partie droite de règle contient au plus un non-terminal. Dans ce cas, les grammaires ne peuvent plus être expansives. Mais seule, la propriété de linéarité n'est pas suffisante pour apprendre : comme l'équivalence de deux grammaires linéaires est indécidable, cette classe n'est pas identifiable en temps CS-polynomial à partir d'INFORMANTS [dlH97]. En revanche, en combinant linéarité et déterminisme, on trouve (enfin) des résultats positifs dans la littérature, par exemple :

Théorème 12

– *La classe de* very simple grammars *est identifiable en temps IPE-polynomial à partir de* TEXTES *[Yok03] ;*

– *La classe des* deterministic linear grammars *est identifiable en temps CS-polynomial à partir d'*Informants *[dlHO02a].*

Malgré les résultats précédents, disposer de grammaires expansives nous conduit forcément à nous poser des questions sur le choix de la représentation des langages hors-contextes sous la forme de grammaires. Ne vivons nous pas simplement les mêmes déboires que si nous cherchions à apprendre des langages rationnels en utilisant des expressions régulières plutôt que des Afd ? En guise d'exemple, l'automate minimal qui reconnait le langage dénoté par l'expression régulière $(0 + 1)^*0(0 + 1)^n$ a 2^{n+1} états, et son ensemble caractéristique par rapport à Rpni est lui-même de taille exponentielle en n. Pourtant, on sait identifier les Afd en temps CS-polynomial à partir d'Informants, mais on ne sait pas identifier les expressions régulières[8]. Autrement dit, dans le cadre des Ghc, on peut penser qu'en changeant de représentation, on devrait pouvoir écarter ce problème d'expansivité.

Sur les problèmes de reconnaissance

Une autre grande différence entre les Afd et les Ghc tient au fait que les premiers sont des systèmes reconnaisseurs de langages, alors que les seconds sont des systèmes générateurs. En conséquence, il est nécessaire d'utiliser des algorithmes annexes plus ou moins sophistiqués pour déterminer si un mot est reconnu par une Ghc ou non, consistant d'abord à mettre cette grammaire sous une forme normale particulière avant de réaliser l'analyse proprement dite, ce qu'on peut faire en temps $\mathcal{O}(|G| \cdot |w|^3)$ à partir d'une grammaire G quelconque [LL09]. A l'inverse, si nous savions travailler avec des automates à pile déterministes, l'analyse des mots pourraient se faire en temps linéaire [Har78]. De même, avec les classes de grammaires linéaires et déterministes pour lesquelles des résultats positifs d'identification ont été obtenus, ou bien celles pour lesquelles il est "facile" d'obtenir des résultats d'identification (comme les grammaires fortement non-ambigües [dlHO02b]), ce problème d'analyse est rendu trivial grâce aux contraintes imposées sur les règles de grammaires.

En tout cas, à partir du moment où il faut faire des transformations de grammaires pour faire de l'analyse syntaxique, on perd le lien direct entre ces grammaires et les données d'apprentissage, ce qui rend difficile tout bilan de l'analyse des données, donc le fait même d'être guidé par les données en cours d'apprentissage. Pour compliquer encore notre tâche, une grammaire est un ensemble souvent indivisible de règles, où le simple fait d'enlever ou d'ajouter une règle "détruit" la grammaire (*i.e.*, aucune bribe du langage qui était reconnu n'est préservée). Dans ces conditions, comment réaliser et exploiter des analyses syntaxiques en cours d'apprentissage, avec des grammaires plus ou moins quelconques, qui ne sont pas sous une forme normale particulière, où certaines règles n'ont pas encore été découvertes et d'autres n'ont pas encore été réfutées ?

De nouveau, ces éléments plaident en faveur d'un changement de représentations, où le lien entre les données et les représentations soient plus immédiats, ce que nous avons fait dans [EdlHJ04], en choisissant des systèmes de réécriture de mots.

3.3.2 Caractéristiques des systèmes de réécriture de mots

Les systèmes de réécriture ont été au cœur de mon travail de thèse [Jan00]. C'est donc avec un certain enthousiame que j'ai abordé cette étude avec Rémi Eyraud et Colin de la Higuera, cette fois pour proposer une solution alternative à l'identification des Ghc.

[8] On sait le faire pour des sous-classes particulières [Fer05].

Réécrire un mot consiste à remplacer, autant que faire se peut, certains facteurs par d'autres en suivant des règles dite de réécriture. Une telle règle est un couple de mots $(l, r) \in (\Sigma^*)^2$, noté $l \rightarrowtail r$, et on dit qu'un mot w_1 se *réécrit* ou se *réduit* en w_2, noté $w_1 \rightarrowtail w_2$, s'il existe $u, v \in \Sigma^*$ tel que $w_1 = ulv$ et $w_2 = urv$. On note $\rightarrowtail^*$ la fermeture réflexive et transitive de $\rightarrowtail$ et on parle de *dérivation de réécriture* quand $w_1 \rightarrowtail^* w_2$. Par exemple, si on considère l'unique règle $01 \rightarrowtail \lambda$, alors le mot 01001101 peut se réécrire de la façon suivante :

$$0100110\underline{1} \rightarrowtail 010\underline{01}1 \rightarrowtail \underline{01}01 \rightarrowtail 01 \rightarrowtail \lambda.$$

D'autres dérivations sont possibles, car plusieurs facteurs peuvent être réécrits à chaque étape, mais il est facile de voir que toutes les dérivations du mot 01001101 conduisent au mot λ si on les pousse suffisamment loin. Plus généralement, on voit que tout mot du langage de Dick (langage de parenthèses) sur l'alphabet $\{0, 1\}$ se réécrit en λ en utilisant cette unique règle de réécriture, et que cette propriété n'est vraie que pour les mots de ce langage. Autrement dit, la donnée (1) du système de réécriture de mots (SRM) formé d'une seule règle $R = \{01 \rightarrowtail \lambda\}$ et (2) de l'ensemble de mots irréductibles formé d'un seul mot $I = \{\lambda\}$ est une représentation possible du langage de Dick, alternative à la grammaire $\langle\{0, 1\}, \{S\}, \{S \rightarrow 0S1S\}, S\rangle$. De même, le lecteur peut facilement vérifier que le langage $\{0^n1^n : n \geq 0\}$ se décrit à l'aide du système $\langle\{0011 \rightarrow 01\}, \{01, \lambda\}\rangle$, ou que $\mathbb{L}(\langle\{01 \vdash \lambda, 10 \vdash \lambda\}, \{\lambda\}\rangle) = \{w \in \Sigma^* : |w|_0 = |w|_1\}$.

On dit qu'un langage est *congruentiel* [MNO88] s'il existe un système $\langle R, I\rangle$ pour le décrire. La terminologie est naturellement proche des remarques que nous faisions en conclusion de la Section 3.1.3 : la fermeture réflexive, symmétrique et transitive de $\rightarrowtail$ est une relation d'équivalence sur Σ^* ; les mots d'un langage congruentiel sont ceux qui sont équivalents à un des mots de l'ensemble I, et comme cet ensemble est fini, la congruence est "partiellement" d'index fini (*i.e.*, elle l'est sur les mots du langage).

Un palliatif aux problèmes de reconnaissance et d'expansivité

Utiliser des SRM est intéressant car ils partagent avec les AFD le fait d'être reconnaisseur et non générateur. Toutefois cette propriété n'est pas vraie pour tous les SRM : il est tout à fait possible qu'un SRM engendre des dérivations infinies (par exemple avec la règle $0 \rightarrowtail 00$). D'ailleurs, on peut montrer qu'un SRM composé d'*une seule* règle de réécriture à la puissance expressive d'une machine de Turing [Dau89]. Comme cette propriété de *terminaison* (ou de *normalisation forte*) est indécidable en général, il est impératif d'imposer des contraintes sur les règles de réécriture. Typiquement, on exige qu'il existe un ordre (strict) bien fondé $\ggg$ sur Σ^* tel que pour toute règle $l \rightarrowtail r$ et pour tout mot $u, v \in \Sigma^*$, on ait $ulv \ggg urv$.

Malheureusement, même si toutes les dérivations terminent, il est encore tout à fait possible qu'elles ne soient pas de longueurs polynomiales en la taille du mot initial. Dans ce cas, l'analyse syntaxique d'un mot par réécriture n'est pas réalisable. Considérons par exemple le SRM suivant sur l'alphabet $\Sigma = \{\$, £, 0, 1, c, d\}$:

$$R = \left\{ \begin{array}{llllll} 1£ & \rightarrowtail & 0£, & 0£ & \rightarrowtail & c1d£, \\ 0c & \rightarrowtail & c1, & 1c & \rightarrowtail & 0d, \\ d1 & \rightarrowtail & 1d, & dd & \rightarrowtail & \lambda \end{array} \right\}$$

Toutes les dérivations induites par R sont finies[9]. Pourtant, si on utilise R pour réécrire le mot $\$\underbrace{11\ldots1}_{n}£ = \$1^n£$ en $\$0^n£$, alors tous les codages sur n bits des entiers entre $2^n - 1$

[9]En effet, en supposant que $d > 1 > 0 > c > £ > \$$, la partie gauche d'une règle est toujours lexicographiquement plus grande que la partie droite, donc ce SRM est fortement normalisant [DJ90].

et 0 vont être rencontrés au moins une fois, donc la longueur de la dérivation correspondante est exponentielle. Par exemple, pour $n = 4$, on aura :

$$
\begin{array}{ll}
 & \$1111£ \\
\rightarrowtail & \$1110£ \\
\rightarrowtail & \$11\underline{1c}1d£ \rightarrowtail \$110\underline{d1}d£ \rightarrowtail \$1101\underline{dd}£ \rightarrowtail \$1101£ \\
\rightarrowtail & \$1100£ \\
\rightarrowtail & \$11\underline{0c}1d£ \rightarrowtail \$1\underline{1c}11d£ \rightarrowtail \$10\underline{d1}1d£ \rightarrowtail \$101\underline{d1}d£ \rightarrowtail \$1011\underline{dd}£ \rightarrowtail \$1011£ \\
\rightarrowtail & \$1010£ \\
\rightarrowtail^* & \$1001£ \\
\rightarrowtail & \$1000£ \\
\rightarrowtail^* & \$0111£ \ldots \\
\rightarrowtail^* & \$0000£.
\end{array}
$$

Pour éviter cela, l'ordre strict $\ggg$ que nous évoquions précédemment, et les règles du système de réécriture, doivent satisfaire des propriétés supplémentaires. Par exemple, on peut exiger que les règles de réécriture fassent décroître strictement la longueur du mot qu'on réécrit à chaque étape. On dit dans ce cas que le SRM est *length-reducing*. Malheureusement, avec une contrainte aussi forte, les SRM ne permettent probablement pas (c'est un problème ouvert) de décrire l'intégralité de la classe des langages rationnels [NW01]. Donc des contraintes plus subtiles doivent être imposées, permettant de récupérer des dérivations de longueurs polynomiales sans (trop) perdre en expressivité.

Une définition du déterminisme pour les SRM

Il nous reste à aborder le problème du *déterminisme* : si toutes les dérivations de réécriture partant d'un même mot sont de longueur raisonnable, il est encore possible que chacune d'elles conduise à un mot irréductible différent. Par exemple, avec le SRM $\{00 \rightarrowtail 1\}$, on a $\underline{00}0 \rightarrowtail 10$ et $0\underline{00} \rightarrowtail 01$, donc le mot 000 admet deux formes irréductibles. Dans le pire cas, il faudrait donc, pour chaque mot, calculer toutes les dérivations possibles afin de déterminer si le mot initial appartient au langage ou pas, ce qui n'est évidemment pas raisonnable.

Pour passer outre ce problème, on impose généralement que les SRM soient *Church-Rosser* (ou *confluent*). Cette forme de déterminisme se définit de la façon suivante : si un mot w se réécrit de deux façons différentes en x_1 et x_2, *i.e.*, si $w \rightarrowtail^* x_1$ et $w \rightarrowtail^* x_2$, alors il existe un mot y tel que $x_1 \rightarrowtail^* y$ et $x_2 \rightarrowtail^* y$. Notons que si x_1 et x_2 sont irréductibles, alors on a nécessairement $x_1 = x_2$; donc si un SRM est Church-Rosser, alors pour tout $w \in \Sigma^*$, il existe *au plus un* mot irréductible w' tel que $w \rightarrowtail^* w'$.

Ainsi, couplée avec la notion de terminaison du paragraphe précédent, toute dérivation partant d'un mot w terminera nécessairement, et dans ce cas, le dernier mot w' qu'on obtiendra sera l'unique mot irréductible qu'on peut obtenir en réécrivant w. Il ne suffira donc plus que de tester si $w' \in I$ pour déterminer si w appartient au langage $\mathbb{L}(\langle R, I\rangle)$.

Malheureusement, le fait d'être Church-Rosser est un problème indécidable en général. On peut l'éviter en se concentrant sur des sous-classes de SRM avec des contraintes syntaxiques sur les règles, en supposant par exemple qu'elles sont *constructor-based* [O'D77] (dans le cas des système de réécriture de termes) ou *almost-non-overlapping* [Klo92].

3.3.3 Un jeu d'équilibriste

Comme nous l'avons vu, deux propriétés sont intéressantes pour qu'un système $\langle R, I\rangle$ reconnaissant un langage L soit intéressant en pratique : on souhaite (1) que toute dérivation de réécriture issue d'un mot w soit finie, de longueur polynomiale en $|w|$, et (2) que le SRM R soit Church-Rosser (*i.e.*, déterministe). On peut toujours obtenir ces propriétés en imposant des conditions plus ou moins complexes sur les règles de réécriture.

Malheureusement, il reste encore deux questions que nous n'avons pas encore abordées. Premièrement, plus que le fait de concevoir des SRM, nous souhaitons les *apprendre*, à l'aide d'un algorithme efficace, ce qui nécessite (intuitivement) de nouvelles contraintes sur les règles de réécriture. Deuxièmement, nous voudrions que la classe de langages reconnus soit suffisamment expressive pour contenir une bonne partie des langages hors-contextes. Or il est clair que chaque nouvelle contrainte imposée sur les règles de réécriture réduit leur expressivité, et ce de façon plus ou moins compréhensible ou prévisible.

C'est cette équation subtile que nous avons dû gérer dans [EdlHJ04], un jeu d'équilibriste où chaque contrainte que nous souhaitions imposer sur les règles pour obtenir telle propriété nous obligeait à renoncer à telle autre. Nous avons malgré tout fini par trouver un point d'équilibre (le lecteur intéressé est invité à consulter [EdlHJ04] pour les détails) :

- Nous avons défini les *hybrid almost-non-overlapping delimited string-rewriting systems* ;
- Nous avons montré que ces systèmes étaient confluents et qu'ils induisaient des dérivations de réécriture de longueur $\mathcal{O}(|w| \cdot |R|)$ à partir d'un mot w (où $|R|$ désigne le nombre de règles de réécriture).
- Nous avons montré qu'ils pouvaient reconnaître tous les langages rationnels, et de nombreux langages hors-contextes usuels (mais pas tous, en particulier pas les langages ambigus comme le langage des palindromes).
- Nous avons développé un algorithme d'identification, LARS, et déterminé une classe de langages qu'il identifiait à la limite à l'aide d'ensembles caractéristiques. Malheureusement, la taille de ces ensembles peut être exponentielle dans certains cas.

Le lecteur notera que comme la plupart des travaux sur l'apprentissage des GHC, nous avons fini par contourner les difficultés qu'elles posent en apportant des palliatifs aux problèmes de non-déterminisme, d'expansivité, de reconnaissance *etc.* Cependant, il reste du chemin à parcourir dans cette voie. En particulier, l'algorithme LARS évoqué précédemment ne peut pas apprendre tous les SRM pour lesquels il a été conçus. Nous connaissons même certains langages rationnels qui peuvent se décrire avec ces SRM mais que LARS ne peut pas identifier. Par ailleurs, nous avons bien un théorème d'identification pour une certaine classe de SRM dits *clos*, mais nous pensons que cette propriété de clôture est indécidable.

En fait, pour obtenir un résultat qui serait pleinement satisfaisant, il faudrait que nous disposions d'une caractérisation plus précise de la classe de langages reconnus par nos SRM. Pour l'obtenir, il serait sans doute préférable de restreindre encore les règles de réécriture, en choisissant par exemple des règles *length-reducing*, quitte à perdre en expressivité. En contrepartie, une meilleure compréhension de la classe de langages reconnus nous guiderait dans la recherche d'un algorithme et d'un théorème d'identification pour l'intégralité de la classe.

Enfin, au-delà de cette contribution sur l'apprentissage des systèmes de réécriture de mots, il est bon de noter que l'apprentissage des grammaires hors-contextes est un sujet désormais bien établi en Inférence Grammaticale. Ainsi, chaque nouvel opus d'ICGI est l'occasion de constater que le domaine avance vite [Cla10, CEH08, CE05].

Deuxième partie

Les interfaces possibles avec la Classification Supervisée

Préambule

Quand on cherche à situer l'Inférence Grammaticale dans le paysage de la Recherche, on la place volontiers au sein de l'Apprentissage Automatique, qu'on place lui-même volontiers dans le champ de l'Intelligence Artificielle. Ainsi, dans leur livre de référence [CM10], L. Miclet et A. Cornuéjols préfèrent-ils parler d'Apprentissage Artificiel plutôt que d'Apprentissage Automatique, et consacrent-ils un chapitre complet à l'Inférence Grammaticale.

C'est l'histoire du *Machine Learning* qui explique cette hiérarchie. Pourtant, en 2010, elle n'est pas toujours facile à justifier : combien de chercheurs dans le domaine du *Machine Learning* connaissent-ils le paradigme d'identification à la limite ? Et combien de chercheurs en Inférence Grammaticale maitrisent-ils la théorie de la régularisation utilisée en optimisation ? Il suffit de suivre des conférences comme ICGI ou ECML pour constater que les communautés sont différentes, tant sur le plan de leurs motivations que sur celui de leurs cultures scientifiques. En outre, lorsqu'on étudie l'histoire des deux domaines (voir [CM10] et [dlH10]), on observe des points de divergence depuis longtemps déjà.

D'un autre côté, plusieurs éléments consolident cette hiérarchie. En effet, tous les algorithmes d'identification à partir d'INFORMANTS fournissent *in fine* des grammaires qui acceptent les données positives et rejettent les données négatives. Donc les grammaires peuvent être vues comme des sortes de classifieurs, et un algorithme d'Inférence Grammaticale comme un apprenant visant à résoudre un problème de classification. De même, le but de l'Inférence Grammaticale Stochastique est d'identifier des distributions de probabilité sur Σ^*, et c'est une thématique qu'on retrouve également en *Machine Learning*.

L'objectif de cette seconde partie est d'étudier de façon plus détaillée les relations entre Inférence Grammaticale et Classification Supervisée. C'est une question sur laquelle j'ai souvent buté au cours des 10 dernières années : à mon arrivée à l'EURISE, je me suis reconverti dans ces domaines en travaillant avec Colin de la Higuera et Marc Sebban. Or dès le début, il était manifeste qu'ils avaient tous deux des points de vue souvent inconciliables quand ils parlaient d'Apprentissage. De plus, indépendamment des incitations à la schizophrénie, c'est une question qui n'est finalement pas souvent posée. Par exemple, dans [CM10], les auteurs présentent les fondements de l'Apprentissage Artificiel en se basant sur la théorie de Vapnik, mais omettent de l'évoquer quand ils présentent l'Inférence Grammaticale, en adoptant une présentation plus classique, proche de celle que nous avons utilisée dans la première partie de ce document. Prétendant à l'Habilitation, il me parait opportun de profiter d'un certain recul pour exposer un point de vue sur cette question, sans prétendre la trancher.

Plan de la seconde partie Je crois que l'Inférence Grammaticale et la Classification de séquences sont deux disciplines autonomes, qui partagent parfois mais pas toujours, des objectifs communs. Aussi, dans le premier chapitre, j'étudierai les divergences entre les deux disciplines, portant sur leurs fondements respectifs, et leurs points de vue complémentaires sur des thématiques comme l'apprentissage actif. Puis dans le second, j'exhiberai un exemple de travail où les disciplines collaborent naturellement, l'apprentissage à partir de mots bruités.

Chapitre 4

Des divergences qui persistent inexorablement

Quand on pense aux débats récents qui ont animé les diverses tendances de l'Apprentissage Artificiel, notamment lors de la conférence CAp, on peut sans doute dire qu'Inférence Grammaticale et Apprentissage Automatique (et plus précisément, Classification Supervisée) ne font pas toujours bon ménage. L'objectif de ce chapitre est de montrer qu'il existe des différences de fond entre ces deux disciplines, qui contribuent à en faire deux disciplines autonomes.

Suivant l'organisation de la première partie de ce manuscrit, on pourrait d'abord penser que ces différences se jouent sur la nature des objets qui sont utilisés : des mots, des langages et des grammaires pour l'une, des réels, des polytopes et des hyperplans pour l'autre. Pourtant ce n'est pas fidèle à la réalité. Ainsi, dans [KCM08], les auteurs ont utilisé des noyaux rationnels, définis à l'aide de transducteurs, pour apprendre les languages réguliers avec des Svm. De façon symétrique, les noyaux de chaînes ont servi en Inférence Grammaticale pour décrire de nouveaux langages dits planaires qu'on peut identifier à l'aide de données positives seulement [CFWS06, CFW06].

Ainsi, plus que les objets, c'est bien sur la notion d'apprentissage que les divergences sont les plus importantes. Rappelons qu'en Inférence Grammaticale, apprendre consiste à identifier une cible, de façon exacte ou approximative. C'est un problème qui ne manque pas d'application. En génie logiciel par exemple, les automates sont parfois utilisés pour modéliser le comportement d'un système ; si l'on dispose d'un algorithme d'apprentissage, alors on peut construire ce modèle de façon automatique en soumettant le logiciel à divers scénarii, puis en récupérant les résultats des tests [DDvL08]. De même, en architecture, des jeux de tests sur un circuit permettent de reconstruire sa spécification qui, comparée à la spécification initiale, conduit à détecter d'éventuelles erreurs d'implémentation [BGC01, HHNS02]. Dans un domaine différent, la théorie des jeux, ce sont les stratégies des joueurs qui peuvent être modélisée sous la forme d'automates ; si l'un d'eux réussit à identifier les stratégies de ses adversaires, il optimise ses chances de gains [CM99]. En robotique, un agent peut encore chercher à reconstruire la carte d'une zone géographique qu'il découvre [DBK+92, RS93].

Ce sont des exemples qui auraient pu relever de l'Apprentissage Automatique du temps de Valiant, mais qui ont cessé de l'être lorsque les idées de Vapnik et des statisticiens ont commencé à diffuser. A partir de ce moment-là, apprendre a consisté, dans cette communauté, à savoir construire un bon modèle à partir de données. Or c'est une conception de

l'apprentissage qui n'a pas du tout diffusé en Inférence Grammaticale. Nous verrons pourquoi dans la première section de ce chapitre.

Au-delà de cette divergence fondamentale sur ce qu'apprendre veut dire, on peut également remarquer qu'il existe des idées partagées entre les deux disciplines, mais sur lesquelles chacune a son propre point de vue, largement complémentaire de celui de l'autre. C'est typiquement le cas en apprentissage actif, où chaque discipline part de la même idée, apprendre en s'aidant d'un expert, mais la met en œuvre de façon différente et aboutit au final à des résultats largement incomparables. C'est sur ce point que portera la seconde section de ce chapitre.

Enfin, nous avons beaucoup exploité la notion de *correction* dans nos travaux (en particulier dans [BBdlHJT08, BBdlHJT07]). Cette même idée semble exister dans d'autres branches de l'Apprentissage Artificiel sous d'autres appellations, et nous lui consacrerons donc la dernière section de ce chapitre.

4.1 Les éclairages de l'Apprentissage Statistique ?

Si l'on veut étudier les divergences théoriques entre l'Apprentissage Automatique et l'Inférence Grammaticale, il parait incontournable de chausser les lunettes de la théorie dominante utilisée aujourd'hui en Classification, celle qu'a développée Vapnik [Vap95]. Même si elle a beaucoup évolué, cette théorie impacte toujours considérablement les travaux en Apprentissage, puisqu'elle a débouché sur les SVM par exemple, et qu'elle inspire tous les travaux visant à établir des bornes théoriques sur l'erreur (*loss*) d'une hypothèse apprise, quelle que soit la technique d'apprentissage utilisée.

4.1.1 L'analyse théorique de Vapnik

La PAC-apprenabilité introduite par Valiant [Val84] a servi de cadre de référence pendant au moins 10 ans, jusqu'à ce que les travaux de Vapnik diffusent en Apprentissage Automatique [Vap82] sous l'influence d'auteurs comme Haussler [BEHW89]. Plutôt que de se concentrer sur le problème de savoir si telle ou telle classe de fonctions était polynomialement PAC-apprenable, l'objectif de Vapnik était de déterminer sous quelles conditions il est possible, à partir d'un échantillon d'apprentissage, de trouver un modèle (hypothèse) qui généralise bien les données. C'est une étude purement statistique, menée dans un cadre très général.

Pour la présenter, nous nous plaçons dans le cas particulier d'un problème de classification de mots. On ne suppose pas disposer d'une cible particulière à identifier ou à approximer ; on suppose seulement qu'il existe une distribution μ inconnue sur $\Sigma^* \times \{-1, +1\}$. On se donne ensuite une classe de grammaires $\mathcal{G}$, c'est-à-dire une famille de fonctions (ou d'*hypothèses*) $h : \Sigma^* \to \{-1, +1\}$. Parmi celles-ci, il existe une hypothèse optimale h_{opt} prédisant au mieux la classe (dans $\{-1, +1\}$) d'un mot de Σ^*. Cette hypothèse n'est pas parfaite, puisqu'il n'y a pas de cible à identifier. C'est simplement l'hypothèse qui minimise l'*erreur réelle* (ou *erreur en généralisation*) suivante :

$$\varepsilon(h) = \mathbb{E}_{(x,y)\sim\mu}\left[h(x) \neq y\right] = \sum_{(x,y)\in\Sigma^*\times\{-1,+1\}} \mu(x,y)[\![h(x) \neq y]\!],$$

où $[\![\pi]\!]$ désigne la valeur de vérité d'un prédicat π dans $\{0, 1\}$.

Clairement, ne connaissant pas μ, on ne peut pas calculer l'erreur réelle. Par contre, pour

chaque échantillon[1] $E = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ tiré selon μ, et pour chaque hypothèse $h \in \mathcal{G}$, on peut calculer l'*erreur empirique* (ou *erreur en apprentissage*) commise par h sur E :

$$\hat{\varepsilon}(h, E) = \frac{1}{m} \sum_{i=1}^{n} [\![h(x_i) \neq y_i]\!].$$

La question centrale qu'attaque Vapnik est de déterminer dans quelle mesure le fait de minimiser l'erreur empirique permet d'obtenir des hypothèses dont l'erreur réelle est proche de l'erreur incompressible sur $\mathcal{G}$, c'est-à-dire celle commise par h_{opt}. C'est donc la validité du principe ERM (pour *Empirical Risk Minimization*) qui est étudiée, ce principe stipulant que pour apprendre une bonne hypothèse à partir de données, il convient de choisir celle qui minimise l'erreur empirique.

Vapnik montre qu'une condition suffisante pour que le principe ERM soit valide est que la *dimension de Vapnik-Chervonenkis* (VC-dim) de la famille d'hypothèses $\mathcal{G}$ soit *finie*. Nous définirons ce paramètre, noté $d_{\mathcal{G}}$, dans la section suivante ; il mesure la richesse de $\mathcal{G}$. Ainsi, lorsque $d_{\mathcal{G}} < \infty$, Vapnik établit deux bornes. En premier lieu, il montre que pour n'importe quelle hypothèse $h \in \mathcal{G}$ et pour tout échantillon E de $m \geq d_{\mathcal{G}}$ exemples tirés aléatoirement et indépendamment selon μ, on a, avec probabilité au moins $1 - \delta$:

$$|\varepsilon(h) - \hat{\varepsilon}(h, E)| \leq \sqrt{\frac{d_{\mathcal{G}}}{m} \left(1 + \ln \frac{2m}{d_{\mathcal{G}}}\right) + \frac{1}{m} \ln \frac{4}{\delta}}.$$

Autrement dit, pour n'importe quelle hypothèse, lorsque le nombre m d'exemples augmente, l'erreur empirique converge vers l'erreur réelle. C'est un résultat relativement intuitif, même s'il est parfois faux pour des échantillons issus de tirages défavorables (d'où l'intervention du paramètre δ), et qu'il est généralement faux lorsque $d_{\mathcal{G}}$ est infini[2]. Enfin, on notera que cette première borne s'applique à n'importe quelle hypothèse $h \in \mathcal{G}$, y compris celles dont l'erreur réelle $\varepsilon(h)$ est très grande : l'analyse de Vapnik est menée dans le pire cas.

En second lieu, Vapnik montre que si h_{emp} est une hypothèse de $\mathcal{G}$ qui minimise l'erreur empirique sur un échantillon E de $m \geq d_{\mathcal{G}}$ exemples tirés aléatoirement et indépendamment selon μ, alors avec probabilité au moins $1 - 2\delta$,

$$\hat{\varepsilon}(h_{emp}, E) - \varepsilon(h_{opt}) \leq \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}} + \sqrt{\frac{d_{\mathcal{G}}}{m} \left(1 + \ln \frac{2m}{d_{\mathcal{G}}}\right) + \frac{1}{m} \ln \frac{4}{\delta}}.$$

En conséquence, si pour chaque échantillon E, on considère l'hypothèse h_{emp} qui minimise l'erreur empirique, alors lorsque la taille de E augmente, on voit converger cette erreur (et l'erreur réelle de h_{emp}) vers l'erreur incompressible (commise par h_{opt}).

Ainsi, lorsque $d_{\mathcal{G}}$ est finie, chercher des hypothèses qui minimisent l'erreur empirique permet bien d'approximer l'hypothèse optimale, conformément au principe ERM. Mais bien entendu, il est tout à fait possible qu'aucun algorithme ne permette d'identifier les hypothèses h_{emp}, de façon exacte ou approximative, sur des échantillons E. C'est en ce sens que l'analyse de Vapnik est purement statistique et non informatique : aucun élément de Calculabilité, d'Algorithmique ou de Complexité n'est pris en compte dans cette analyse.

[1] Il y a ici un abus de notation : il est tout à fait possible qu'un même couple (x_i, y_i) soit tiré plusieurs fois selon μ ; il conviendrait donc d'utiliser des multi-ensembles plutôt que des ensembles. On gardera cependant les notations habituelles, en faisant comme si chaque donnée avait un numéro unique.

[2] Ce point a posé de nombreux problèmes lorsque l'algorithme ADABOOST a été inventé : la théorie de Vapnik semblait prédire que l'erreur réelle des hypothèses complexes retournées par ADABOOST devaient diverger, alors que les observations montraient l'inverse [SFBL98].

4.1.2 Sur la VC-dimension des boules et des AFD

Comme nous l'avons déjà dit, la dimension de Vapnik-Chervonenkis d'une famille d'hypothèses $\mathcal{G}$ mesure la richesse de cette famille. Au-delà de son intérêt en Apprentissage Statistique, c'est une mesure qu'on retrouve également en Théorie des Graphes, en particulier dans les travaux sur la coloration [KKR+95].

D'un point de vue formel, on dit qu'un ensemble de mots $E = \{x_1, x_2, \ldots, x_m\}$ est *pulvérisé* par $\mathcal{G}$ si pour tous $y_1, y_2, \ldots, y_m \in \{-1, +1\}$, il existe une hypothèse $h \in \mathcal{G}$ consistante avec l'ensemble $\{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$. La dimension de Vapnik-Chervonenkis de $\mathcal{G}$ est le cardinal maximum de toute famille E pulvérisable par $\mathcal{G}$.

On notera que même si une famille d'hypothèses a une grande VC-dimension, il est tout à fait possible que la plupart des ensembles de taille moindre ne soient pas pulvérisables par $\mathcal{G}$. De nouveau, la VC-dimension est un paramètre qui mesure le "pire" cas. D'autres dimensions, comme celle de Rademacher, ont été utilisées pour développer des bornes de l'erreur réelle ; elles sont relatives à l'échantillon d'apprentissage E, qui est fixé [BM02].

Concernant la VC-dimension de $\mathcal{AFD}(\Sigma)$, on peut facilement voir qu'elle est infinie. En effet, en considérant n'importe quel ensemble de mots étiquetés $\{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$, il suffit de construire le PTA des mots d'étiquette $+1$ pour obtenir un AFD consistant avec l'ensemble (voir Section 3.1.3). Aussi, pour tout $m \geq 0$, tout ensemble de m mots est pulvérisable par un AFD. Par contre, si on limite la classe des AFD en ne considérant que ceux dont la taille (nombre d'états) est inférieure à un certain entier $n \geq 1$, alors on obtient une classe, notée $\mathcal{AFD}_{\leq n}(\Sigma)$, de VC-dimension finie :

Théorème 13 ([IT97]) *La VC-dimension de $\mathcal{AFD}_{\leq n}(\Sigma)$ vaut*
- $(|\Sigma| - 1 + o(1))n \log_2 n$ *lorsque* $|\Sigma| \geq 2$,
- $n + \log_2 n - \mathcal{O}(\log_2 \log_2 n)$ *lorsque* $|\Sigma| = 1$.

Concernant les bonnes boules maintenant, on a le résultat suivant :

Théorème 14 *Si $|\Sigma| \geq 2$, alors la VC-dimension de $\mathcal{GOODBALL}(\Sigma)$ est infinie.*

Preuve: Soient $0, 1 \in \Sigma$ deux lettres distinctes. Soient $m \geq 1$ et $E_m = \{x_1, x_2, \ldots x_m\}$ un ensemble de m mots de longueur m tels que $x_i = 0^{i-1}10^{m-i}$ pour tout $1 \leq i \leq m$. On considère maintenant les étiquettes $y_1, y_2, \ldots y_m \in \{-1, +1\}$ et on cherche une boule consistante avec $\{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$. Supposons qu'il y ait $k > 0$ étiquettes $+1$ (et $m - k$ étiquettes -1). On définit alors la boule $B_r(o)$ en posant $r = k - 1$ et $o = c_1 c_2 \ldots c_m$ avec $c_i = 1$ si $y_i = +1$ et $c_i = 0$ si $y_i = -1$, pour tout $1 \leq i \leq m$. Comme $r = k - 1 < m = |o|$, cette boule est bonne. De plus, si $y_i = +1$, alors $d_E(o, x_i) = k - 1$ donc $x_i \in B_r(o)$. Enfin si $y_i = -1$, alors $d_E(o, x_i) \geq k$, donc $x_i \notin B_r(o)$. Par conséquent, $B_r(o)$ est consistante avec E. □

On peut aussi remarquer que la VC-dimension de $\mathcal{GOODBALL}(\Sigma)$ vaut 2 lorsque $|\Sigma| = 1$. De plus, on pourrait de nouveau poser des contraintes sur les tailles, en considérant la classe $\mathcal{GOODBALL}_{\leq n}(\Sigma)$ des bonnes boules dont le centre a une longueur plus petite que n. Compte tenu de la preuve précédente, sa VC-dimension est finie et plus petite que n. Toutefois, l'utilité de cette classe est restreinte, puisqu'en tant que classifieurs, ces boules rejettent tout mot de longueur supérieure à $2|o|$.

Pour finir, notons que plusieurs autres résultats sur la VC-dimension des classes de grammaires étudiées en Inférence Grammaticale existent. Citons par exemple [MSSS99] pour ce qui est des *pattern languages*.

4.1.3 Quelques conséquences en Inférence Grammaticale

Dans [Vap95], Vapnik justifie le fait que ses bornes soient relativement serrées. Pourtant, le cadre dans lequel elles s'appliquent est extrêmenent général. Et si on revient maintenant à l'Inférence Grammaticale, nous disposons effectivement d'hypothèses supplémentaires : on suppose (1) qu'il existe une hypothèse cible inconnue $f : \Sigma^* \to \{-1, +1\}$, (2) que la distribution μ porte sur Σ^* et non sur $\Sigma^* \times \{-1, +1\}$, et (3) que les exemples sont de la forme $(w, f(w))$ avec w tiré aléatoirement et indépendamment selon μ.

Dans ce cas, minimiser l'erreur empirique revient à chercher des hypothèses d'erreur empirique nulle dans $\mathcal{G}$, c'est-à-dire des hypothèses consistantes avec les données. Elles existent forcément puisque la cible f est l'une d'entre elles. De plus, on peut établir des bornes plus fines que les bornes précédentes : en se basant sur les travaux de Vapnik, les auteurs de [BEHW89] montrent que pour tout échantillon E de $m \geq d_{\mathcal{G}}$ exemples tirés selon μ, et pour toute hypothèse $h \in \mathcal{G}$ *consistante* avec E, on a, avec probabilité au moins $1 - \delta$:

$$\varepsilon(h) \leq \frac{2d_{\mathcal{G}}}{m} \ln \frac{2em}{d_{\mathcal{G}}} + \frac{2}{m} \ln \frac{2}{\delta}.$$

Cette borne est plus précise que celle de Vapnik (car la racine de la borne générale a disparue, et que $\sqrt{x} > x$ pour tout $0 < x < 1$). En outre, elle revient à dire [KV94] que pour tout $\epsilon > 0$, une hypothèse $h \in \mathcal{G}$ consistante est ϵ-bonne avec probabilité au moins $1 - \delta$ dès que

$$m = \mathcal{O}\left(\frac{d_{\mathcal{G}}}{\epsilon} \ln \frac{1}{\epsilon} + \frac{1}{\epsilon} \ln \frac{1}{\delta}\right).$$

Bref, d'un point de vue statistique, les algorithmes d'Inférence Grammaticale devraient permettre de trouver des classifieurs dont les performances en généralisation sont intéressantes. Pourtant les écueils restent nombreux. Concernant la classe $\mathcal{AFD}(\Sigma)$, sa VC-dimension est infinie et nous sommes donc hors des hypothèses précédentes. Ce n'est pas le cas de la classe $\mathcal{AFD}_{\leq n}(\Sigma)$, mais nous n'échappons par contre pas à l'algorithmique : la question de l'existence d'un AFD de moins de n états consistant avec un échantillon E est un problème $\mathcal{NP}$-complet [Gol78]. De même, concernant $\mathcal{GOODBALL}(\Sigma)$, outre sa VC-dimension infinie, le problème de trouver une boule consistante avec un échantillon E est $\mathcal{NP}$-difficile [dlHJT08].

Ainsi, l'approche de Vapnik ne s'applique pas facilement aux "classifieurs" qu'on considère en Inférence Grammaticale. Pourtant, les AFD et les grammaires sont des pierres angulaires de la science informatique : on les utilise partout, et c'est une des raisons pour lesquelles on cherche à les apprendre. Cette richesse n'est manifestement pas un atout dans le cadre de l'Apprentissage Statistique. Il est préférable d'utiliser des classifieurs très simples, comme les hyperplans séparateurs, ou des *stumps* (arbres de décision de hauteur 1) dont l'intérêt, en tant qu'objet informatique, est franchement limité, mais qui répondent aux exigences du cadre : des caractéristiques combinatoires basiques permettant de résoudre efficacement le problème de la consistance avec un échantillon, et une dimension de Vapnik-Chervonenkis faible.

Pour illustrer cet état de fait, citons Vapnik en personne, dans l'interview qu'il a donné à Ran Gilab-Bachrach et qui est publiée sur le site `www.learningtheory.org` : "*The uniform convergence theory was constructed to justify learning algorithms developed in the 1960s and, in particular, the optimal separating hyperplane. For me* PAC *theory that started in mid 1980's was a backward step to prior development presented in our joint with Alexey Chervonenkis book "Theory of pattern recognition" (1974) and in my book "Estimation of Dependencies Based on Empirical Data" (1979 Russian version and 1982 English translation).*" Tout informaticien

notera cependant que la théorie de la PAC-apprenabilité tient compte de la complexité des algorithmes d'apprentissage, donc de leur mise en œuvre, et que de ce point de vue, c'est plutôt l'approche de Vapnik qui marque un retour en arrière.

4.2 La dualité de l'apprentissage actif

Dans les paradigmes d'identification, exacte ou approximative, que nous avons présentés dans le chapitre 3, l'apprenant reçoit des données qui sont consistantes avec une certaine grammaire cible. Or même si la manière avec laquelle les données sont générées est formalisée, par l'entremise d'une présentation dans le premier cas, et d'un tirage aléatoire selon une distribution inconnue dans le second, l'apprenant ne fait jamais que *subir* ces données, et n'a aucun moyen de les *choisir*.

Pourtant, en tenant compte de notre expérience d'enseignant, il y a toutes les raisons de penser que la première situation est beaucoup plus pénalisante pour l'apprenant que la seconde. En effet, l'apprentissage à partir d'un flux ou d'un échantillon de données évoque la situation d'un étudiant ne suivant que des cours magistraux (CM) en amphithéatre. De retour dans sa chambre universitaire, l'étudiant n'a pas d'autres choix que d'éplucher ses notes pour trouver des réponses à ses éventuelles questions, en espérant qu'il les ait prises de façon correcte. C'est un apprentissage qu'on dit *passif*, ce qui est probablement injuste pour ce qui concerne l'étudiant.

En Inférence Grammaticale, les résultats présentés dans le chapitre 3 nous conduisent à des conclusions similaires sur la difficulté d'identifier un AFD. Si on cherche à l'identifier de façon exacte, alors il faut supposer que les données contiennent des informations précises qui le caractérisent parfaitement (ensemble caractéristique). En termes pédagogiques, cela revient à supposer que les notes de l'étudiant (et le cours de l'enseignant) sont suffisamment complets pour lui permettre d'acquérir parfaitement les concepts ; c'est évidemment rarement le cas. Et si on cherche à identifier un AFD de façon approximative, alors c'est tout bonnement impossible pour des raisons algorithmiques. Par exemple, il n'est pas possible de calculer en temps polynomial un AFD qui soit consistant avec des données et qui soit en même temps de petite taille (*i.e.*, d'une taille proche de celle de l'automate minimal) [PW93]. En poursuivant l'analogie avec nos étudiants, cela signifie qu'utiliser des notes pour apprendre un concept de façon approximativement correcte nécessite soit d'y consacrer un temps exponentiel, soit de monopoliser énormément de mémoire pour stocker un modèle exponentiellement plus grand que le concept cible.

A l'inverse, si l'étudiant suit des travaux dirigés (TD), qu'il a accès à un enseignant capable de répondre à ses questions, alors on constate en pratique que son apprentissage est plus efficace, et ses acquis beaucoup plus fiables. C'est un apprentissage qu'on dit *actif*. De même, nous verrons que les AFD sont efficacement identifiables dans ce contexte.

4.2.1 Deux conceptions complémentaires

D'un point de vue historique, c'est Dana Angluin qui formalisa la première cette idée d'apprentissage actif, en introduisant le paradigme d'*identification à partir de requêtes* [Ang81, Ang87b]. A ma connaissance, elle l'a fait pour attaquer des problèmes d'Inférence Grammaticale, et c'est dans ce cadre (ou ses variantes) que les inférentistes se placent encore aujourd'hui lorsqu'ils attaquent des problèmes d'apprentissage actif.

Angle d'attaque des inférentistes

Intuitivement, le paradigme d'Angluin mime un jeu de devinettes entre deux agents : un oracle et un apprenant. Comme d'habitude, on suppose fixée une classe $\mathcal{G}$ de grammaires, et l'oracle choisit l'une d'entre elles $G \in \mathcal{G}$. L'objectif de l'apprenant est de découvrir cette grammaire en posant des questions, qu'on appelle des *requêtes*. Le jeu s'apparente alors à un échange de questions-réponses et termine lorsque l'apprenant réussit à identifier le concept cible. Mais bien sûr, pour que ce paradigme ait du sens, il est nécessaire de poser quelques restrictions.

Concernant l'oracle, nous supposerons au moins dans un premier temps qu'il est honnête : il se doit de répondre aux requêtes sans mentir, et ne peut changer de concept cible en cours de jeu que si les réponses qu'il a déjà fournies à l'apprenant restent cohérentes par rapport à la nouvelle cible (donc de façon transparente pour l'apprenant). De plus, il est supposé omniscient, et ne s'abstient donc jamais. Par contre, il n'est pas tenu d'être bienveillant, et peut donc toujours chercher à répondre à l'apprenant de la façon la moins informative possible.

De son côté, comme dans tout jeu de devinettes, l'apprenant ne peut pas poser n'importe quelle question : le type de requêtes possibles est fixé à l'avance. De plus, le nombre de requêtes au cours d'une partie doit être raisonnable ; typiquement, on souhaite qu'il soit borné par un polynôme de la taille de la grammaire cible. C'est un point très important, car en pratique, l'oracle peut être un expert humain qui doit éventuellement conduire des expériences complexes pour répondre à l'apprenant. Aussi, plus que la durée du jeu, ce sont les accès à l'oracle qui sont coûteux, donc le nombre de requêtes qu'il faut minimiser.

Ce cadre d'apprentissage est devenu une pierre angulaire en Inférence Grammaticale pour au moins deux raisons. D'une part, sur le plan théorique, il permet d'établir de nombreux résultats d'apprenabilité, soit dans le paradigme lui-même, soit dans les autres par "effet de bord". D'autre part, à l'instar de Rpni ou Edsm en identification à la limite, l'algorithme L^*, inventé par Dana Angluin dans [Ang87a], permet d'identifier les Afd à partir de requêtes (d'appartenance et d'équivalence). Or cet algorithme et ses variantes ont été très utilisés dans des applications, typiquement celles que nous avons évoquées en introduction.

Angle d'attaque des apprentistes

De façon relativement indépendante, l'apprentissage actif s'est également beaucoup développé en Apprentissage Automatique pour attaquer des situations pratiques où les données sont trop rares, ou au contraire trop massives. Plus précisément, dans [Set10], un *survey* récent sur la question, l'auteur classe les différentes approches en deux catégories principales.

Dans certaines applications, obtenir des données étiquetées peut être extrêmement coûteux, en temps et en argent. C'est typiquement le cas de données biologiques ou de données résultant d'une expérience en Physique. Un algorithme d'apprentissage actif pourra alors synthétiser des données non étiquetées puis demander à un expert de mener des expérimentations ciblées pour déterminer leurs étiquettes. Dans ce cas, on parle d'apprentissage actif par *synthèse de requêtes* (*membership query synthesis*).

A l'inverse, dans d'autres applications, on dispose d'un volume considérable de données non étiquetées, comme en Traitement de la Langue ou en Recherche d'Information. Un algorithme d'apprentissage actif peut alors chercher à gagner en efficacité, en sélectionnant des

données qui lui paraissent intéressantes, puis en utilisant les compétences d'un expert pour obtenir leurs étiquettes. Il peut aussi chercher à optimiser ses performances, en obtenant plus d'informations sur des sous-ensembles de données sur lesquelles il a des difficultés pour apprendre. On parle alors d'apprentissage actif par *sélection des données* (*selective sampling*).

Au-delà de cette élégante présentation, il convient cependant de noter que parmi ces deux approches, la seconde (par sélection de données) est beaucoup plus étudiée que la première (synthèse de requêtes) par les apprentistes. Il y a des raisons théoriques qui expliquent ce développement très inégal. En effet, dans une tâche typique de classification, un algorithme cherche à produire une hypothèse dont l'erreur réelle est minimale. Cette erreur se définit et s'estime en fonction d'une distribution inconnue μ sur les données. Or si l'apprenant a bien accès à un échantillon tiré selon μ en sélection de données, il n'a en revanche aucun moyen d'accéder à la distribution dans un contexte où il doit synthétiser des requêtes, donc aucun moyen d'évaluer et de minimiser l'erreur des hypothèses qu'il apprend.

Complémentarité

Ainsi, nous sommes en présence d'une même idée, apprendre en posant des questions, que deux communautés de chercheurs mettent en œuvre de façon différente et qu'ils exploitent à des fins différentes. En empruntant la terminologie de [Set10] et en épluchant la littérature, on peut clairement stipuler que la très grande majorité des algorithmes développés en Inférence Grammaticale fonctionnent par synthèse de requêtes, alors que la très grande majorité des travaux en Apprentissage Automatique ne traitent que de sélection des données. C'est parfois une source de confusion quand les deux communautés se réunissent dans des sessions dites d'"Active Learning" de grandes conférences comme Ecml ...

4.2.2 Identification à partir de requêtes

Comme nous l'avons déjà dit, pour que le paradigme d'identification à partir de requêtes ait du sens, il convient de limiter les requêtes que l'apprenant peut poser à l'oracle. Comme c'est ce point qui permet l'identification ou non d'une classe de grammaires, il est clair que de nombreux types de requêtes ont été étudiés dans la littérature. Nous reprenons ici les deux plus classiques (voir [dlH10] pour une liste plus complète).

Requêtes d'appartenance et d'équivalence

La famille la plus courante de requêtes est celle des requêtes d'appartenance. Elle permet à l'apprenant de déterminer si un mot est reconnu ou non par la grammaire cible. En Apprentissage Automatique, c'est également ce type de requêtes qui est utilisé pour l'essentiel, par des algorithmes qui demandent à un expert (oracle) d'étiqueter des données qu'il a sélectionnées :

Définition 11 (Requêtes d'appartenance) *Soit $G \in \mathcal{G}$ la grammaire cible. Dans le cas d'une requête d'appartenance (*membership query*, MQ), l'apprenant soumet un mot $w \in \Sigma^*$ à l'oracle ; la réponse de ce dernier, notée* $\textsc{Mq}(w)$*, vaut soit* Oui *si $w \in \mathbb{L}(G)$, soit* Non *si $w \notin \mathbb{L}(G)$.*

Au contraire, le second type de requêtes, dites d'équivalence, est beaucoup moins pertinent en pratique. Ce sont des requêtes parfois qualifiées de "techniques" visant à déterminer si une grammaire hypothèse est bien la grammaire cible :

Définition 12 (Requêtes d'équivalence) *Soit $G \in \mathcal{G}$ la grammaire cible. Dans le cas d'une requête d'équivalence (*equivalence query*, EQ), l'apprenant soumet une grammaire hypothèse $H \in \mathcal{G}$ à l'oracle ; la réponse de ce dernier, notée* $\textsc{Eq}(H)$*, vaut :*

- *soit* Oui *si* $\mathbb{L}(H) = \mathbb{L}(G)$,
- *soit un contre-exemple de l'équivalence, c'est-à-dire un mot $w \in \Sigma^*$ appartenant à la différence symétrique des langages reconnus $\mathbb{L}(H) \oplus \mathbb{L}(G)$.*

De nombreuses variantes de cette dernière définition existent dans la littérature. Certaines sont plus restrictives : on parle de requêtes d'équivalence *faibles* quand l'oracle retourne Non plutôt qu'un contre-exemple dans le second cas. D'autres ne le sont pas assez : si on suppose que l'apprenant peut soumettre n'importe quelle représentation de langages plutôt que des grammaires de $\mathcal{G}$ à l'oracle (requêtes d'équivalence *impropres*), alors un algorithme de *halving* imaginé par Littlestone dans [Lit87] peut être adapté pour apprendre une classe extrêmement large de grammaires (voir [dlH10, p. 166]). La définition de requêtes d'équivalence que nous utilisons ici est la plus consensuelle aujourd'hui, dans la mesure où elle permet d'obtenir des résultats d'apprenabilité "raisonnables". Cependant, les difficultés qu'elle présente restent nombreuses.

D'une part, si on considère la classe $\mathcal{GHC}(\Sigma)$ des grammaires hors-contextes, il est clair que la fonction $\textsc{Eq}(H)$ n'est pas calculable puisque l'équivalence des Ghc n'est pas décidable. Utiliser de telles requêtes peut alors conduire à des résultats d'apprenabilité dont la portée est relativement faible, puisqu'on ne peut pas disposer d'un oracle[3]. *A contrario*, supposer que l'oracle a des capacités computationelles arbitraires est souvent intéressant lorsqu'on cherche à établir des résultats de non-apprenabilité ; ainsi, on admet souvent que l'oracle connait parfaitement l'apprenant, qu'il est capable d'anticiper ses déductions et ses questions, y compris si l'apprenant a recours à un générateur aléatoire pour les poser par exemple. C'est une façon de supposer que l'oracle peut toujours répondre à l'apprenant de la façon la moins informative possible, qu'il peut donc se comporter comme un *adversaire* plutôt que comme un professeur bienveillant. C'est donc un moyen d'éviter tout phénomène de *collusion* entre l'apprenant et l'oracle.

D'autre part, force est de constater qu'il n'est pas facile d'imaginer une application où l'on puisse disposer de requêtes d'équivalence en pratique, en particulier si l'oracle doit être simulé. En fait, il est possible de contourner cette difficulté en simulant les requêtes d'équivalence avec des tirages aléatoires et des requêtes d'appartenance (comme nous le verrons). Mais il faut alors renoncer à identifier la cible de façon exacte : seule une identification approximative est possible. En outre, il faut disposer d'une distribution de probabilité pertinente sur Σ^*, dont la définition, dans le cadre de l'application visée, n'est souvent pas facile à poser. Ce problème est identique à celui qui se pose en Apprentissage Automatique, quand on cherche à faire de l'apprentissage actif par synthèse de requêtes.

Enfin, de façon plus technique, les requêtes d'équivalence posent des problèmes de complexité dans la définition même de l'identification à partir de requêtes : il faut absolument éviter que l'apprenant puisse gagner du temps en forçant l'oracle à retourner des contre-exemples de taille exponentielle en la taille de la cible (voir [dlH10, Section 7.5.3]).

Ainsi, nous utiliserons la définition suivante :

Définition 13 (Identification à partir de requêtes) *Soient $\mathcal{G}$ une classe de grammaires et $\mathcal{Q}$ une classe de requêtes (typiquement $\mathcal{Q} = \{MQ, EQ\}$). On dit que $\mathcal{G}$ est* polynomialement

[3] Dans [Ang87a], l'auteur rapproche et compare ses résultats avec les travaux de Schapiro sur l'inférence de programmes logiques.

identifiable avec des requêtes de $\mathcal{Q}$ *s'il existe un apprenant* $\mathfrak{A}$ *tel que pour toute grammaire cible* $G \in \mathcal{G}$*, (1)* $\mathfrak{A}$ *questionne l'oracle avec des requêtes de* $\mathcal{Q}$ *et termine son exécution en retournant une représentation équivalente à* G*, et (2) lors d'une exécution visant à identifier* G*, et pour chaque appel à l'oracle par* $\mathfrak{A}$*, le nombre total de requêtes et le temps utilisé jusqu'à cet appel est polynomial en* $\|G\|$ *et en la taille de l'information retournée par l'oracle jusqu'à cet appel.*

Identification des boules et des AFD

La combinaison des requêtes d'appartenance et d'équivalence $\mathcal{Q} = \{\text{MQ}, \text{EQ}\}$ s'appelle un *minimum adequate teacher* (MAT). En effet, Angluin a démontré les résultats suivants, stipulant qu'il faut bien disposer des deux types de requêtes pour identifier les AFD, ce qui justifie de fait la terminologie qu'elle a proposée :

Théorème 15 ([Ang87a, Ang87b]) *La classe* $\mathcal{AFD}(\Sigma)$ *est polynomialement identifiable à partir de requêtes d'appartenance et d'équivalence. Par contre, elle n'est pas polynomialement identifiable à partir de requêtes d'appartenance uniquement, ni de requêtes d'équivalence uniquement.*

Concernant le résultat positif, Angluin a proposé un algorithme, L^*, qui est devenu un classique de l'Inférence Grammaticale [Ang87a]. C'est essentiellement lui et ses variantes qui sont utilisés lorsqu'une application nécessite d'apprendre un AFD à partir de requêtes. En conséquence, beaucoup de travaux ont porté sur l'optimisation de ses performances, en particulier la minimisation du nombre de requêtes nécessaires pour apprendre [RS93, KV94, BDGW97]. C'était également un des points sur lequel portait la compétition ZULU que nous décrirons dans la section suivante.

Maintenant, concernant les résultats négatifs, Angluin a développé une technique générique, dite des *approximate fingerprints*, qu'a reprise et approfondie Gavaldà dans [Gav93]. Elle consiste à exhiber, au sein de la classe $\mathcal{G}$ de grammaires, une sous-classe $\mathcal{G}'$ de taille finie N, telle qu'un apprenant ne puisse jamais éliminer plus de $\log N$ grammaires de $\mathcal{G}'$ en posant une requête. En conséquence, cet apprenant devra faire un nombre exponentiel de requêtes s'il veut trouver une grammaire cible dans $\mathcal{G}'$. Une technique plus simple (mais qui suit la même idée) permet de prouver le résultat suivant, établissant la non-apprenabilité des bonnes boules à partir des requêtes d'appartenance et d'équivalence :

Théorème 16 ([BBdlHJT08]) *Soit* $n \geq 0$ *et* $\mathcal{B}_{\leq n} = \{B_r(o) : o \in \Sigma^*, r \leq |o| \leq n\}$*. Tout algorithme qui identifie polynomialement les boules de* $\mathcal{B}_{\leq n}$ *à partir de* $\mathcal{Q} = \{MQ, EQ\}$ *ne peut pas utiliser moins de* $\Theta(|\Sigma|^n)$ *requêtes pour certaines d'entre elles.*

Preuve: Nous supposons que l'oracle est un adversaire. Il ne choisit pas une bonne boule à identifier, mais maintient un ensemble S de toutes les boules qui peuvent encore être des boules cibles après les réponses qu'il a déjà fournies à l'apprenant. Initialement, on a $S = \mathcal{B}_{\leq n} = \{B_r(o) : o \in \Sigma^*, r \leq |o| \leq n\}$. Puis à chaque nouvelle requête, il s'arrange pour répondre de sorte qu'un nombre minimum de boules de S soient éliminées. Plus précisément :

- pour une requête d'appartenance sur le mot o, il répond systématiquement $\text{MQ}(o) = \text{NON}$;
- pour une requête d'équivalence sur la boule $B_r(o)$, il retourne systématiquement le contre-exemple $\text{EQ}(o, r) = o$.

Par voie de conséquence, pour tout mot ou tout centre de boule o, l'apprenant déduit que o n'appartient pas à la boule cible. Cela conduit l'adversaire à éliminer de nombreuses boules de S, mais une seule de rayon nul. Or il existe $\Theta(|\Sigma|^n)$ boules de rayon nul dans S. Donc face à cet oracle, n'importe quel apprenant doit faire un nombre exponentiel de requêtes pour identifier l'une d'entre elles. □

Relations avec les autres paradigmes d'identification

Au-delà de son intérêt propre, se traduisant par des applications réelles, l'apprentissage à partir de requêtes a aussi un intérêt en relation avec les autres cadres d'apprentissage, en particulier ceux que nous avons présentés dans le chapitre 3.

Concernant l'identification à la limite, nous avons déjà évoqué le résultat suivant, que nous pouvons désormais prouver :

Théorème 17 ([Pit89]) *La classe $\mathcal{AFD}(\Sigma)$ n'est pas identifiable en temps IPE-polynomial à partir d'*Informants*.*

Preuve: On raisonne par l'absurde, en supposant qu'il existe un apprenant $\mathfrak{A}$ capable d'indentifier les Afd en temps IPE-polynomial. Dans ce cas, on peut utiliser $\mathfrak{A}$ pour construire un nouvel algorithme, $\mathfrak{B}$, qui identifie la classe des Afd avec des requêtes d'équivalence uniquement. Initialement, $\mathfrak{B}$ soumet n'importe quel Afd à l'oracle. Si cet Afd n'est pas la cible, l'oracle retourne un contre-exemple que $\mathcal{B}$ soumet à $\mathfrak{A}$ comme premier élément d'un Informant. Il récupère ainsi un second Afd qu'il soumet à l'oracle. De nouveau, si cet Afd n'est pas la cible, l'oracle retourne un contre-exemple que $\mathcal{B}$ soumet à $\mathfrak{A}$ comme second élément de l'informant, et ainsi de suite. Clairement, la séquence des contre-exemples qu'$\mathfrak{A}$ reçoit forme le début d'un Informant pour l'Afd cible. Comme $\mathfrak{A}$ converge vers cet Afd par hypothèse, $\mathcal{B}$ converge également. De plus, si $\mathfrak{A}$ reçoit un contre-exemple, c'est qu'il commettait une erreur implicite de prédiction. Or leur nombre total est limité par un polynôme sur la taille de la cible par hypothèse, donc $\mathcal{B}$ ne pose qu'un nombre polynomial de requêtes d'équivalence avant de converger. Autrement dit, $\mathcal{B}$ est capable d'identifier polynomialement $\mathcal{AFD}(\Sigma)$ avec des requêtes d'équivalence uniquement, ce qui contredit le Théorème 15. □

Une preuve très similaire permet de montrer le même résultat pour les bonnes boules, en utilisant le Théorème 16 :

Théorème 18 ([dlHJT08]) *La classe $\mathcal{GOODBALL}(\Sigma)$ n'est pas identifiable en temps IPE-polynomial à partir d'*Informants*.*

Concernant maintenant la Pac-apprenabilité, nous avons vu que les Afd n'étaient pas identifiables dans ce cadre (Théorème 7). Or le fait de pouvoir simuler des requêtes d'équivalence par des tirages aléatoires et des requêtes d'appartenance permet de montrer le résultat suivant :

Théorème 19 ([Ang87a]) *Il existe un algorithme L_a^* prenant en entrée les paramètres d'erreur $\epsilon > 0$ et de confiance $\delta > 0$, tel que pour tout* Afd *A et toute distribution μ sur Σ^*, ayant accès à l'oracle $\mathrm{Ex}(A, \mu)$ et à un oracle répondant à des requêtes d'appartenance, avec probabilité $1 - \delta$, L_a^* retourne un* Afd *B qui est ϵ-bon par rapport à A et μ. De plus, L_a^**

s'exécute en temps et en nombre total de requêtes polynomial en $1/\epsilon$, $1/\delta$, $\|A\|$ *et* m, *où* m *est la longueur du plus grand exemple obtenu par tirage aléatoire.*

Nous ne reprenons pas la preuve en détail ici. L'idée est d'utiliser l'algorithme L^* en remplaçant chaque requête d'équivalence par un tirage aléatoire de mots et des requêtes d'appartenance. Il est relativement simple de montrer qu'il suffit, pour simuler une requête d'équivalence sur un AFD B, de tirer $(1/\epsilon)\ln(1/\delta)$ mots selon μ puis de les soumettre comme requêtes d'appartenance, pour déterminer, avec probabilité $1-\delta$, si B est ϵ-bon par rapport à l'AFD cible. Cependant, cette simulation doit être faite pour chaque requête d'équivalence. Aussi, pour obtenir un nombre total raisonnable de requêtes, il est nécessaire d'affiner le nombre de tirages : celui-ci doit dépendre du nombre de requêtes d'équivalence déjà simulées. Ainsi, pour simuler la i-ème requête d'équivalence, on peut montrer qu'il suffit de tirer $(1/\epsilon)\left(\ln(1/\delta) + (i+1)\log 2\right)$ mots selon μ, et dans ce cas, on peut obtenir le résultat final [Ang87a]. Notons encore que ce nombre de requêtes a été considérablement réduits depuis la première preuve (voir en particulier [Gav09]).

4.2.3 Sur la compétition ZULU

Fort de ses succès sur les AFD, le paradigme d'identification à partir de requêtes a donné lieu a de nombreux travaux tout au long des 30 dernières années, portant notamment sur l'oracle (qui peut être probabiliste, peut s'abstenir, peut faire des erreurs, *etc*), ou sur le type de requêtes autorisées (requêtes d'appartenance, d'équivalence forte ou faible, de sous-ensemble, de correction, de traduction, *etc*) ou encore sur la classe de grammaires cibles (AFD et boules bien sûr, mais également automates non déterministes, GHC, *etc*).

Toutefois, en 2009, nous avons pensé qu'il existait encore plusieurs points à étudier dans le cadre de l'identification des AFD, en particulier lorsque les requêtes d'équivalence ne sont pas disponibles et que le nombre de requêtes d'appartenance est limité. Ce sont ces deux points qui ont induit l'organisation de la compétition ZULU[4] que nous allons présenter dans ce paragraphe (voir aussi [CdlHJ09]). Outre Colin de la Higuera, initiateur et pilote du projet, les participations de David Combe à Saint-Etienne [Com09] et Myrtille Ponge à Nantes ont été cruciales pour sa réalisation : tous deux ont développé la plateforme en vue de la compétition dans le cadre de leurs projets de M2, puis l'ont maintenue et complétée pendant toute la compétition (souvent sur leur temps libre).

ZULU et les autres compétitions

ZULU n'est pas la première compétition organisée en Inférence Grammaticale. La première fut ABBADINGO[5] en 1999. L'objectif était d'apprendre des automates à partir d'échantillon de données. C'est à cette occasion que fût inventé l'algorithme EDSM, un des algorithmes phares de l'Inférence Grammaticale aujourd'hui. Une version épurée de la plateforme est toujours en ligne[6], utile pour tester tout nouvel algorithme comme RPNI*. La seconde compétition fut OMPHALOS[7] en 2004, dont l'objectif était d'apprendre des grammaires hors-contextes à partir d'exemples et de contre-exemples, ou simplement d'exemples. A cette occasion, des propriétés théoriques de certaines grammaires comme la *substituabilité* [CE05] ont été mises en évidence

[4] http://labh-curien.univ-st-etienne.fr/zulu/
[5] http://abbadingo.cs.nuim.ie/
[6] http://www.irisa.fr/Gowachin/
[7] http://www.irisa.fr/Omphalos/

et utilisées avec succès. Ces mêmes propriétés continuent d'être exploitées aujourd'hui dans ce domaine.

Ainsi, les deux compétitions précédentes ont été des succès, dans la mesure où elles ont permis des avancées significatives. Mais ce n'est pas toujours le cas. En 2004, la communauté des Algorithmes Génétiques (GECCO) a organisé une compétition sur l'identification des automates à partir de données bruitées. A notre connaissance, il n'y a pas eu d'avancée notable sur ce problème qui reste très difficile à résoudre en Inférence Grammaticale. De même, la compétition TENJINNO[8] s'est déroulée en 2006. Elle portait sur l'apprentissage des transducteurs, une généralisation des automates de Mealy. De nouveau, le problème était extrêmement difficile, et la compétition n'a pas permis d'avancées réelles.

Clairement, pour qu'elle soit un succès, nous avons cherché à concevoir ZULU en tenant compte des compétitions passées, et nous avons en particulier retenu :

1. la nécessité de proposer des premiers problèmes qui soient accessibles avant de demander la résolution de problèmes plus complexes ;
2. la mise à disposition d'un premier algorithme de base (*baseline*) facilement modifiable par l'utilisateur et qui lui permette de se lancer dans la compétition : il n'y a rien de plus décourageant que de devoir développer une application *from scratch* avant de jouer ;
3. le choix des informations divulguées aux joueurs : il est particulièrement important que les joueurs ne se focalisent pas sur des problèmes annexes dont la solution biaise artificiellement la compétition sans véritable apport pour la communauté.

Descriptif de ZULU

ZULU est à la fois une plateforme en ligne qui simule un oracle permettant d'apprendre des AFD, et une compétition. En tant que plateforme, ZULU permet aux utilisateurs de générer des tâches, d'interagir avec un oracle pendant les sessions d'apprentissage et d'enregistrer les résultats et les performances des utilisateurs. En outre, ZULU permet de télécharger la *baseline*, écrite en JAVA, et renseigne les utilisateurs sur les manières de construire tout autre algorithme capable d'interagir avec le serveur. Cette *baseline* est une version de L^* remplaçant les requêtes d'équivalence par des tirages aléatoires de mots et des requêtes d'appartenance. Théoriquement, cette version de l'algorithme ne permet pas d'identifier tout automate de façon exacte (compte tenu du Théorème 15), mais permet dans certain cas de l'approximer correctement (grâce au Théorème 19). C'est un des points sur lequel nous souhaitions que les compétiteurs travaillent. Enfin, plusieurs pages donnent des informations techniques, par exemple sur la génération aléatoire des automates ou des mots (*test set*).

Ont accès au serveur tous les utilisateurs qui ont préalablement créé un compte. Le serveur joue ensuite le rôle d'un oracle répondant à des requêtes d'appartenance uniquement. Ainsi, un joueur peut se connecter et demander un nouvel automate cible. Le serveur calcule ensuite un nombre de requêtes N qui lui semble suffisant pour que le joueur apprenne une machine "raisonnable"[9]. Puis le serveur invite le joueur à identifier l'automate cible avec au plus N requêtes d'appartenance. A la fin de la phase d'apprentissage, le serveur donne à l'apprenant un ensemble de 1800 mots que ce dernier doit étiqueter à l'aide de l'automate appris : 1 si

[8] `http://web.science.mq.edu.au/tenjinno/`

[9] Ce nombre N correspond en fait au nombre de requêtes qui sont nécessaires au serveur pour apprendre lui-même approximativement un automate cible, en faisant 30% d'erreurs.

le mot est reconnu, 0 sinon. Il récupère enfin ces étiquettes et calcule le taux d'erreur de l'automate appris.

La plateforme a été mise en ligne en juillet 2009, soit environ un an avant le début de la compétition proprement dite. Nous avons utilisé ce temps pour en faire la promotion soit par le biais de mailings, soit lors de conférences. En particulier, nous avons visé la communauté Traitement de la Langue Naturelle lors de la conférence FSMNLP'09 qui s'est tenu à Pretoria, en Afrique du Sud. Et c'est dans ce cadre que le nom de la compétition a été trouvé. En effet, l'Inférence Grammaticale développe des techniques visant à apprendre des grammaires, ce qui peut logiquement servir à des linguistes. En outre, certaines langues essentiellement parlées sont en péril (comme celles pratiquées dans le Grand Nord, du fait du réchauffement climatique). Or les techniques d'identification à partir de requêtes sont particulièrement appropriées lorsque les ressources sont rares.

La compétition elle-même s'est déroulée au mois de juin 2010. Nous avons défini 9 catégories de 2 problèmes, fonction de la taille de l'alphabet et du nombre (maximum) d'états de l'automate cible. Nous avons ajouté 3 dernières catégories en réduisant le nombre de requêtes autorisées pour l'apprentissage, afin de départager les ex-æquo. Les automates proposés aux participants, les nombres de requêtes maximum pour l'apprentissage et les ensembles de test étaient les mêmes pour tous les participants. Nous avons évalué les algorithmes en mesurant leurs taux d'erreur sur chaque tâche. Disposer de plusieurs catégories de problèmes visait à mettre en évidence des algorithmes qui pouvaient être meilleurs dans certains contextes et moins bons dans d'autres.

Quant au classement final, il s'est avéré compliqué à construire dans la mesure où, en plus des diverses catégories de problèmes, nous avons cherché à limiter à 3 le nombre d'algorithmes avec lesquels chaque joueur pouvait concourir. En effet, si ce nombre n'est pas borné, il est tout à fait possible qu'un joueur termine premier dans chaque catégorie, voire sur chaque problème, en utilisant chaque fois un algorithme différent, ce qui ne permet plus d'apprécier la qualité réelle de tous ces algorithmes. Nous avons finalement établi un classement général en utilisant une méthode similaire à celle utilisée pour les compétitions de voile. Les résultats ont été proclamés lors d'un workshop, pendant la conférence ICGI'10 qui s'est tenue à Valencia en septembre[10].

Résultats de Zulu

En dehors des utilisateurs occasionnels, relativement nombreux avant le début de la compétition, 11 joueurs ont finalement participé avec 23 algorithmes différents. Les pays d'origine des compétiteurs sont variés : Espagne, Allemagne, USA, Argentine, Slovénie et Inde. Les trois joueurs (ou plutôt équipes) primés furent :

1. Falk Howar, Mark Merten & Bernhard Steffen, de la TU Dortmundt,
2. Borja Balle, de l'UPC Barcelona et
3. Sarah Eisenstat & Dana Angluin, du MIT et de l'université de Yale.

L'équipe de Falk Howar émarge dans la communauté Génie Logiciel. Elle est particulièrement spécialisée sur des problèmes de *model checking* : ils utilisent des variantes de L^* pour apprendre des automates de Mealy en faisant des tests sur logiciels. De son côté, Borja Balle est un doctorant de Ricard Gavaldà qui ne travaillait pas sur des problèmes

[10]Voir `http://users.dsic.upv.es/workshops/icgi2010/`. Les liens Final Program puis Zulu Workshop donnent accès aux résumés longs et aux *slides* des joueurs.

d'apprentissage actif jusqu'à présent. Leurs contributions sont relativement similaires dans l'esprit : ne pouvant disposer des contre-exemples habituellement fournis par les requêtes d'équivalence, ils les recherchent et les choisissent en utilisant des heuristiques qui visent à minimiser le nombre de requêtes d'appartenance qu'un tel choix va impliquer sur la suite de l'algorithme. Cependant, les algorithmes eux-mêmes sont deux variantes différentes de l'algorithme L^*, basées sur les *observation packs* étudiés dans [BDGW97] pour les premiers, et les *arbres binaires de classification* étudiés dans [KV94, Chap.8] pour le second.

Concernant maintenant la contribution de Sarah Eisenstat et Dana Angluin, c'était probablement la moins optimisée des trois mais la plus originale. Elles ont exploité les caractéristiques de la compétition, en particulier le fait que les automates cibles que nous proposions était tirés aléatoirement. Un vieux résultat [Kor67] repris dans [TB73] stipule que pour distinguer deux états dans la plupart des AFD à n états sur un alphabet de k lettres, ils suffit de considérer des suffixes de longueur $\mathcal{O}(\log_k \log_2 n)$. C'est une propriété fausse lorsqu'on considère n'importe quel automate, mais qui est généralement vraie pour des automates tirés au hasard, comme ceux de la compétition. Ainsi, pour un AFD de $n = 300$ états sur $k = 2$ lettres, des suffixes de longueur 3 ou 4 suffisent. Clairement, des mots si courts sont en petit nombre (*i.e.*, $k^{\mathcal{O}(\log_k \log_2 n)} = \mathcal{O}(\log_2 n)$). Ainsi, lorsque les compétitrices doivent chercher un contre-exemple pour remplacer une requête d'équivalence, elles utilisent des heuristiques fouillant dans des ensembles de suffixes courts. Cela leur permet manifestement d'identifier approximativement les AFD cibles avec une technique dont elles justifient la pertinence.

Au-delà de la compétition elle-même, l'idée de considérer des grammaires cibles "moyennes" pourrait sûrement être exploitée pour d'autres problèmes en Inférence Grammaticale car la plupart des résultats de non-apprenabilité reposent sur des problèmes $\mathcal{NP}$-complets. Or certains d'entre eux sont sans doute faciles à résoudre pour des instances aléatoires. Quelques travaux anciens ont déjà été faits dans cette voie [Lan92, FKR+97]. De plus, c'est un thème qui revient à la mode par le biais d'Henning Fernau (et ses travaux sur la complexité paramétrée), ou dans des contributions comme [AEKR10].

Enfin, concernant la suite de ZULU, la plateforme est toujours ouverte. De plus, ZULU devrait avoir une suite : c'est le thème d'une journée qui se tiendra en octobre lors de la conférence ISOLA'10 en Crête, organisée par les vainqueurs de ZULU. Au-delà, deux autres compétitions vont avoir lieu prochainement en Inférence Grammaticale. La première est STAMINA, organisée par Pierre Dupont, portant sur le même problème que la compétition ABBADINGO (voir ci-dessus). La seconde devrait être organisée en 2012 par Sicco Verwer, de la TU Delft, et portera sur l'apprentissage des automates stochastiques.

4.3 Quelques idées de correction

Dans la section précédente, nous avons vu que l'apprentissage actif était mis en œuvre de deux façons différentes en Inférence Grammaticale et en Apprentissage Automatique : dans le premier domaine, les algorithmes fonctionnent essentiellement par synthèse de requêtes, alors que dans le second, ils font de la sélection de données. Nous avons également travaillé sur une dernière notion qui semble partagée entre les deux disciplines sans pour autant qu'elles l'exploitent de façon identique, celle de *correction*.

4.3.1 Diverses notions de correction

On dit souvent qu'on apprend beaucoup de ses propres erreurs, mais ce n'est vrai que si, à la suite d'une erreur, on en obtient (ou on en déduit) un corrigé. Prenons par exemple le cas d'un enfant qui acquiert sa langue maternelle : si celui-ci emmagasine beaucoup d'informations en écoutant les autres parler, il lui arrive également, lorsqu'il parle, d'obtenir des suggestions de correction de la part de son interlocuteur ("On ne dit pas des chevals mais des chevaux"). Et cette interaction lui permet manifestement de progresser plus vite dans son apprentissage [BBY04].

C'est en partant de ce constat que nous avons commencé à étudier le potentiel de la notion de correction en Inférence Grammaticale. Notre objectif était encore d'étudier l'inférence de grammaires à partir de données bruitées. Or nous avions des difficultés pour travailler à partir d'échantillons bruités, dans un cadre d'apprentissage passif, et donc l'idée d'utiliser une technique d'apprentissage actif, avec un oracle capable de fournir des corrections, devait intuitivement nous simplifier la tâche. Nous avons en outre profité de la visite de Leonor Becerra-Bonache, de l'université de Tarragona, pour avancer : elle avait été l'auteur d'un travail pionnier sur une toute première définition des *requêtes de correction* [BBDT06]. Dans [BBdlHJT08, BBdlHJT07], nous avons étendu cette définition, puis établi les résultats que j'exposerai dans la section suivante.

Bien qu'il ne le sache pas forcément, le lecteur a déjà expérimenté de telles requêtes dans sa vie, lorsqu'il utilise un moteur de recherche. En effet, s'il tape un mot-clef, le moteur retourne généralement une liste de pages web qui sont sensées être pertinentes vis-à-vis du mot-clef. Mais lorsque le mot-clé est erroné ou rare, le moteur propose une correction plutôt qu'une liste d'Url (voir Figure 4.1). Ces corrections sont d'autant plus fiables que ce sont des milliards de pages web qui forment le dictionnaire de référence. Ainsi, le web et les moteurs de recherche se comportent comme des oracles puissants, capables d'identifier des mots-clés corrects ou de les corriger quand ils sont douteux[11].

Fig. 4.1 – La correction d'une requête erronnée.

En Apprentissage Automatique maintenant, il semble également que des notions de correction sont exploitées dans plusieurs branches du domaine. En particulier, c'est une notion qu'on retrouve plus ou moins en apprentissage par renforcement : les récompenses fournies par

[11] Nous omettons dans ce propos tous les inconvénients de ce type d'oracle, qui décrète comme douteux des mots-clés corrects mais rares : c'est le concept de correction qui nous intéresse ici.

l'environnement aident l'apprenant à converger vers une politique optimale [SB98]. En outre, en apprentissage actif par sélection de données, on peut facilement imaginer une application où, confronté à des données très bruitées, l'apprenant demanderait des corrections d'exemples qu'il considère comme suspect, de façon à minimiser l'erreur des hypothèses qu'il apprend. Nous avons nous-même utilisé une forme primitive de corrections dans la Section 5.2.2, dédiée à l'algorithme OraBoost : dans ce cadre, l'oracle fournit une indication de confiance sur la classe des exemples, soit proche de +1 si l'exemple semble sain, soit proche de -1 si l'oracle ne sait pas déterminer si la classe de l'exemple est correcte ou non. Dans ce cas, on faudrait sans doute parler de requête d'évaluation plutôt que de correction.

Ainsi, cette notion de correction semble exploitable aussi bien en Inférence Grammaticale qu'en Apprentissage Automatique, dans le cadre d'applications dont les objectifs sont malgré tout très différents. En tout cas, c'est une piste qu'il conviendrait de suivre.

4.3.2 Sur les requêtes de correction

Plusieurs notions de requêtes de correction ont été proposées dans la littérature (notamment dans [BBDT06]). Néanmoins, les contraintes posées sur celles-ci sont souvent très fortes : elles permettent d'obtenir des résultats d'apprenabilité mais tendent à éluder les difficultés d'exploitation d'une "vraie" notion de correction. Dans [BBdlHJT08], nous avons préféré choisir la pente raide en introduisant la définition suivante :

Définition 14 (Requête de correction) *Soient $G \in \mathcal{G}$ la grammaire cible et w un mot de Σ^* que l'apprenant soumet à l'oracle. Alors, la réponse de l'oracle, notée $\textsc{Cq}(w)$, est de deux types :*

- *si $w \in \mathbb{L}(G)$, alors $\textsc{Cq}(w) = \textsc{Oui}$;*
- *si $w \notin \mathbb{L}(G)$, alors l'oracle retourne une correction de w par rapport à G, c'est-à-dire un mot w' qui appartient à $\mathbb{L}(G)$ et qui est à distance minimum de w, au sens de la distance de Levenshtein :*

$$\textsc{Cq}(w) = \textit{un mot de } \left\{ w' \in \mathbb{L}(G) : d_E(w, w') \textit{ est minimum} \right\}.$$

La raideur de la pente que nous évoquions précédemment est liée à l'utilisation de la distance de Levenshtein, induisant des phénomènes combinatoires souvent complexes, par exemple le fait qu'un mot n'ait pas une correction unique. Malgré tout, si on repense à l'exemple du moteur de recherche, il est clair que la correction d'un mot-clé est bien proche du mot erroné, au sens d'une distance d'édition particulière.

Résultats sur les bonnes boules et les Afd

De façon intuitive, les requêtes de corrections sont susceptibles d'amener des résultats d'identification en Inférence Grammaticale. En effet, prenons l'exemple de l'identification d'un disque du plan pour la métrique euclidienne usuelle. Pour identifier un tel disque, il suffit de procéder de la façon suivante (voir Figure 4.2) :

1. On demande à l'oracle la correction d'un point.
2. Tant que l'oracle répond que le point appartient au disque, on demande la correction d'un autre point, en suivant toujours la même direction, de plus en plus loin.
3. Une fois que l'on a obtenu un point A hors de la boule et sa correction C, on recommence pour obtenir un second point B et sa correction D.

4. Le centre O est donné par l'intersection des droites (AC) et (BD), et le rayon est la longueur des segments $[OC]$ (ou $[OD]$).

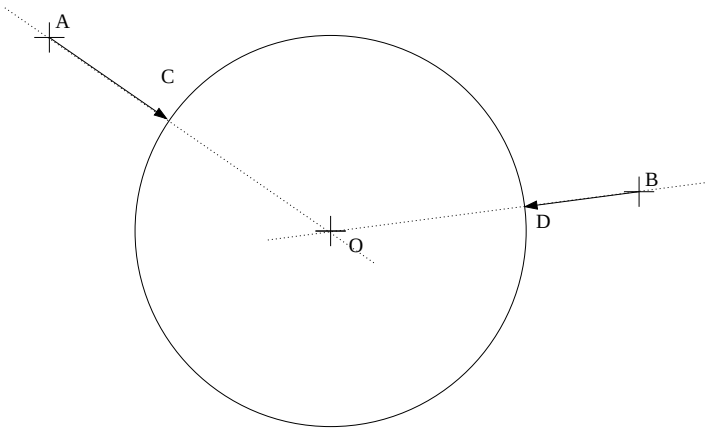


FIG. 4.2 – Trois étapes suffisent pour identifier un disque du plan avec des requêtes de correction : (1) trouver au hasard deux points A et B à l'extérieur du disque en utilisant quelques requêtes ; (2) demander à l'oracle des corrections de A et B, ce qui nous permet de récupérer les points C et D ; (3) utiliser la règle pour trouver le centre O, et le compas pour dessiner le cercle.

Ainsi, il est facile d'identifier des disques du plan, et si l'on reprend maintenant les requêtes de correction sur les mots, il n'y a pas de raison de penser qu'on ne puisse pas identifier les boules de mots de la même façon. De fait, nous avons montré le résultat suivant, dont la preuve nécessite plus de travail que nos esquisses précédentes :

Théorème 20 ([BBdlHJT08]) *La classe $\mathcal{GOODBALL}(\Sigma)$ est polynomialement identifiable à partir de requêtes de correction uniquement.*

En revanche, nous obtenons le résultat négatif suivant pour les AFD :

Théorème 21 ([BBdlHJT08]) *La classe $\mathcal{AFD}(\Sigma)$ n'est pas polynomialement identifiable à partir de requêtes de correction uniquement.*

Preuve: On utilise la technique classique avec un adversaire. Pour tout $n \geq 2$, on considère la classe $\mathcal{A}_{\leq n}$ des AFD avec moins de n états. Notons B_w l'automate minimal du langage $\Sigma^* \setminus \{w\}$ pour tout $w \in \Sigma^*$; le lecteur peut facilement vérifier que B_w a $|w| + 2$ états. Maintenant, concernant l'oracle, il répond OUI à chaque requête de correction sur un mot w. Cela élimine de nombreux AFD de $\mathcal{A}_{\leq n}$ mais au plus un AFD B_w (lorsque $|w| \leq n - 2$). Or comme il y a $\Theta(|\Sigma|^n)$ AFD de type B_w dans $\mathcal{A}_{\leq n}$, le nombre de requêtes nécessaires pour identifier l'un d'entre eux est exponentiel en face d'un tel oracle. □

Ce dernier résultat nous semble capital car il souligne l'originalité des requêtes de correction : les AFD sont identifiables à partir $\{\mathrm{MQ}, \mathrm{EQ}\}$ (Théorème 15) mais pas de $\{\mathrm{CQ}\}$, et à l'inverse, les bonnes boules ne sont pas identifiables à partir $\{\mathrm{MQ}, \mathrm{EQ}\}$ (Théorème 16) mais le sont à partir de $\{\mathrm{CQ}\}$. Ainsi, l'information apportée par des requêtes classiques et par des requêtes de correction est de nature différente ; il est manifestement impossible de simuler des requêtes de correction avec un MAT et réciproquement.

Algorithme d'identification des bonnes boules

Nous nous focalisons maintenant sur l'apprenabilité des bonnes boules. Pour l'établir, il est d'abord nécessaire de caractériser toutes les corrections possibles d'un mot :

Lemme 6 ([BBdlHJT08]) *Soient $B_r(o)$ une boule et $v \notin B_r(o)$. Alors l'ensemble des corrections possibles pour v est $\{u \in \Sigma^* : d(o,u) = r \text{ et } d(u,v) = d(o,v) - r\}$.*

Preuve: Soit $k = d(o,v)$, puis considérons une suite d'opérations d'édition de longueur minimum transformant o en v : $o \xrightarrow{k} v$. Comme $v \notin B_r(o)$, on a $k > r$, donc cette dérivation passe par un mot w_0 tel que $o \xrightarrow{r} w_0 \xrightarrow{k-r} v$. Définissons l'ensemble $W = \{w \in \Sigma^* : d(o,w) = r \text{ et } d(w,v) = k - r\}$. Clairement, $w_0 \in W$, donc $W \neq \emptyset$. De plus, $W \subseteq B_r(o)$. Maintenant notons U l'ensemble des corrections possibles pour v et montrons que $U = W$. Soient $u \in U$ et $w \in W$. Si $d(u,v) > d(w,v)$, alors w est un mot de $B_r(o)$ qui est strictement plus proche de v que u, donc u ne peut pas être une correction de v. Si au contraire $d(u,v) < d(w,v)$, comme $d(o,v) \leq d(o,u) + d(u,v)$, on en déduit que $d(o,u) \geq d(o,v) - d(u,v) > d(o,v) - d(w,v)$. Or $d(o,v) = k$ et $d(w,v) = k - r$, donc $d(o,u) > r$, ce qui est impossible puisque $u \in U \subseteq B_r(o)$. Ainsi, $d(u,v) = d(w,v) = k - r$, et donc les mots $w \in W$ et les corrections $u \in U$ sont tous à la même distance de v, *i.e.*, $W \subseteq U$. Enfin on a $d(o,v) \leq d(o,u) + d(u,v)$, donc $k \leq d(o,u) + k - r$, c'est-à-dire $d(o,u) \geq r$. Comme $u \in B_r(o)$, nous en déduisons $d(o,u) = r$, et comme on a aussi $d(u,v) = k - r$, on en conclut que $U \subseteq W$. □

Avec ce résultat, on peut facilement calculer les corrections possibles d'un mot par rapport à une boule. Par exemple, si on considère la boule $B_2(11)$ et le mot 0000, alors l'ensemble des corrections possibles est $\{00, 001, 010, 100, 0011, 0101, 0110, 1001, 1010, 1100\}$. On peut aussi interpréter le lemme précédent de façon "géométrique" : définissons le *segment* $[\![o,v]\!] = \{u \in \Sigma^* : d(o,u) + d(u,v) = d(o,v)\}$ et le *cercle* $C_r(o) = \{w \in \Sigma^* : d(o,w) = r\}$; une correction possible de v est un mot $u \in [\![o,v]\!] \cap C_r(o)$. Le fait que v ait plusieurs corrections possibles montre simplement que la géométrie (et plus précisément les segments de Σ^*) est très différente de celle du plan.

Maintenant, pour montrer le Théorème 20, nous avons développé un algorithme qui identifie toute bonne boule $B_r(o)$ en deux temps :

1. Il commence par chercher un mot appartenant à la *frontière supérieure* de la boule, c'est-à-dire un mot à distance r de o et qui soit de longueur maximum dans $B_r(o)$. La difficulté de cette étape vient du fait que la frontière d'une boule n'est pas régulière (comme nous l'avons déjà évoqué dans la Section 2.3.2) ; il faut donc mettre en place une stratégie qui garantisse cette propriété.
2. Un mot de la frontière extérieur de $B_r(o)$ est intéressant car il se construit à partir de o en faisant r insertions de lettres. Autrement dit, dans un tel mot, toutes les lettres de o sont présentes dans le bon ordre. Du coup, il suffit de déterminer les lettres qui ont été insérées pour retrouver o.

Commençons par chercher un mot de la frontière supérieure d'une boule $B_r(o)$ à identifier. Pour cela, supposons que l'alphabet soit $\Sigma = \{a_1, a_2, \ldots, a_n\}$ et considérons la famille de mots $\{m_k = (a_1 a_2 \ldots a_n)^k : k \geq 0\}$. Par exemple, si $\Sigma = \{0,1\}$, alors $m_3 = (01)^3 = 010101$. Si nous faisons augmenter k de 0 jusqu'à $|o|$, alors nous sommes sûrs que $o \preceq m_k$, *i.e.*, que o est un sous-mot de $m_{|o|}$. Or comme nous venons de le voir, tout mot w de la frontière

extérieur de $B_r(o)$ est obtenu en faisant r insertions dans o. Donc si nous continuons de faire augmenter k jusqu'à $|o| + r$ et au-delà, alors nous obtenons $o \preceq w \preceq m_k$ pour tout mot w de la frontière extérieure et tout $k \geq |o| + r$. Comme nous parlons de sous-mots, en termes de distance d'édition, nous avons alors $d_E(o, w) = r$ et $d_E(w, m_k) = d_E(o, m_k) - r$. Aussi, d'après le Lemme 6, tout mot w de la frontière supérieure sera une correction possible de m_k. Autrement dit, si nous arrivons à trouver un entier $k \geq |o| + r$, alors il suffira de demander la correction de m_k à l'oracle pour trouver un mot de la frontière supérieure.

Mais bien entendu, nous ne connaissons ni $|o|$, ni r. Le lemme suivant (dont nous omettons la preuve, assez technique) est d'un grand secours :

Lemme 7 ([BBdlHJT08]) *Soient $B_r(o)$ une boule, $a \in \Sigma$ une lettre et $j \geq 0$ un entier tel que $a^j \notin B_r(o)$. Soit $w = \textsc{Cq}(a^j)$. Si $|w| < j$, alors $|w|_a = |o|_a + r$.*

Autrement dit, lorsque j est suffisamment grand, la correction du mot a^j donne des informations sur le nombre d'occurrence de a dans o. Or comme la longueur de o est la somme des nombres d'occurrence pour chaque lettre (*i.e.*, $|o| = \sum_{a_i \in \Sigma} |o|_{a_i}$), il suffit d'obtenir des corrections $w_{a_1}, w_{a_2}, \dots w_{a_n}$ pour les mots $a_1^j, a_2^j, \dots a_n^j$ avec j suffisamment grand puis de sommer pour obtenir :

$$\sum_{a_i \in \Sigma} |w_{a_i}|_{a_i} = \sum_{a_i \in \Sigma} (|o|_{a_i} + r) = |o| + r|\Sigma|.$$

Cette dernière quantité est manifestement plus grande que $|o| + r$, et peut donc servir de valeur de k pour obtenir une correction w de m_k appartenant à la frontière extérieure. De plus, si $w = \textsc{Cq}(m_k)$, alors on a $|w| = |o|+r$ et comme nous connaissons la valeur de $|o|+r|\Sigma|$, nous pouvons à ce stade déterminer la valeur de r et de $|o|$.

Ces premières étapes sont reprises dans l'Algorithme 8 ci-dessous. Il nous reste à montrer comment on peut retrouver le centre de la boule à partir d'un mot de sa frontière extérieure.

Algorithm 8: IdfGoodBall(Σ)

Input: L'alphabet $\Sigma = \{a_1, \dots, a_n\}$
Output: La représentation (o, r) de la boule cible $B_r(o)$

```
1 j ← 1 ; k ← 0;
2 for i = 1 to n do
3     while Cq(a_i^j) = Oui or |Cq(a_i^j)| ≥ j do
4         j ← 2 · j;
5     k ← k + |Cq(a_i^j)|_{a_i};
6 w ← Cq((a_1 a_2 ... a_n)^k);
7 r ← (k − |w|)/(|Σ| − 1);
8 o ← ExtractCentre(w, r);
9 return (o, r);
```

Supposons maintenant qu'on connaisse un mot w de la frontière extérieure de $B_r(o)$. Comme nous l'avons dit, w a été obtenu à partir de o en faisant r insertions. Donc dans w, toutes les lettres de o apparaissent dans le bon ordre, et les autres sont dues à des insertions. Or si une lettre a été insérée à une position donnée dans w, on peut clairement la remplacer par n'importe quelle autre lettre insérée à n'importe quelle autre position et dans ce cas, le

mot w' qu'on obtient reste un sur-mot de o à distance r, *i.e.*, w' est toujours sur la frontière extérieure de la boule. C'est l'idée de l'Algorithme 9. On tente de remplacer chaque lettre de w par une lettre x placée en début de mot. Si le mot obtenu continue d'appartenir à la boule cible, alors la lettre qui a été remplacée était due à une insertion ; plus précisément, il existait une suite d'opérations d'édition transformant o en w où cette lettre était due à une insertion. Si au contraire, le mot n'appartient plus à la boule cible, alors la lettre remplacée vient nécessairement du centre o : dans aucune dérivation de o en w, cette lettre ne peut être due à une insertion. A la fin de ces itérations, on obtient le mot $x^r o$; et comme on connait r, on en déduit o.

Algorithm 9: ExtractCentre(w, r)

Input: Un mot w de la frontière extérieure, le rayon r
Output: Le centre o de la boule cible

```
1 x ← a_n ; // x est une lettre arbitraire
2 for i = 1 to n do
3 |   Assume w = a_1 ... a_n and let w' ← x a_1 ... a_{i-1} a_{i+1} ... a_n;
4 |   if Cq(w') = Oui then w ← w'
5 Assume w = a_1 ... a_n and return a_{r+1} ... a_n;
```

Concernant pour finir la complexité de notre algorithme, nous obtenons le résultat suivant :

Proposition 2 ([BBdlHJT08]) *L'algorithme* IdfGoodBall *identifie n'importe quelle bonne boule $B_r(o)$ après $\mathcal{O}(|\Sigma| + |o| + r)$ requêtes de correction.*

Preuve: La première étape de l'algorithme, visant à trouver un mot de la frontière extérieure, nécessite $\mathcal{O}(\log(|o| + r) + |\Sigma|)$ requêtes de correction. En effet, $\mathcal{O}(\log(|o| + r))$ sont nécessaires pour trouver des corrections w_{a_i} qui soient de longueurs plus petites que celles de a_i^j, et nous devons disposer de corrections pour chaque lettre de l'alphabet. Quant à la seconde étape de l'algorithme, elle consiste à visiter toutes les lettres du mot de la frontière extérieure, donc consomme $\mathcal{O}(|o| + r)$ requêtes. □

On notera ici que le fait que les boules soient bonnes est crucial pour que le Théorème 20 soit pertinent : le nombre de requêtes nécessaires est exponentiel en $\log r$, mais linéaire en $|o|$.

Sur les imperfections de l'algorithme

L'algorithme précédent permet bien de démontrer le Théorème 20 mais il présente malgré tout deux difficultés que nous voudrions évoquer par souci d'exhaustivité. Nous ne détaillerons cependant pas les solutions que nous avons développées pour y faire face, et invitons le lecteur intéressé à consulter [BBdlHJT08].

D'une part, sur le plan de la complexité, le nombre de requêtes de correction permettant d'identifier une bonne boule est linéaire. Or si l'on compare ce résultat avec celui nécessaire pour identifier un disque du plan, c'est un résultat finalement décevant : on aurait pu espérer un nombre de requêtes en $\mathcal{O}(\log |o|)$. En réalité, nous avons pu établir un tel résultat, mais en posant des hypothèses supplémentaires relativement contraignantes : nous avons dû abandonner la distance de Levenshtein au profit d'une distance d'édition où le coût des substitutions est un *irrationnel* ρ tel que $0 < \rho < 1$, et le coût des insertions et des suppressions vaut 1.

En outre, nous avons dû supposer que l'alphabet contenait au moins 3 lettres. Dans tous les autres cas, l'existence d'un algorithme d'identification avec un nombre logarithmique de requêtes de correction est un problème ouvert.

D'autre part, un des intérêts de l'apprentissage actif est la possibilité de remplacer l'oracle par un expert humain. Or dans le cadre des requêtes de correction, il est souvent extrêmement difficile pour un humain de répondre : si celui-ci peut proposer un semblant de correction, elle est souvent approximative et imparfaite. Elle peut par exemple dépendre d'un contexte auquel l'expert n'a pas accès. Il nous a donc paru opportun d'étudier de façon expérimentale le comportement de notre algorithme face à un oracle dont les réponses sont approximatives. Dans ce cas, nous avons constaté que les boules apprises étaient relativement proches des boules cibles, donc que l'algorithme résistait correctement à des réponses "bruitées". Cette idée de travailler à partir de réponses bruitées me semble prometteuse en apprentissage actif en général.

Chapitre 5

Des disciplines qui s'enrichissent mutuellement

Dans le chapitre précédent, nous avons tenté de démontrer qu'Inférence Grammaticale et Classification Supervisée étaient deux disciplines largement autonomes en Apprentissage Artificiel, portant des regards complémentaires ou divergents sur un certain nombre de notions. Pourtant ce sont des disciplines qui ont maintes raisons de rester mariées.

Des raisons politiques d'abord. De son côté, l'Inférence Grammaticale n'est pas une discipline qui draine suffisamment de chercheurs pour prétendre à l'autonomie totale. Il n'y a par exemple pas de revue qui soit spécialisée en Inférence Grammaticale, et les articles importants paraissent souvent dans des revues d'Apprentissage Automatique[1]. De son côté, l'Apprentissage Automatique s'allie régulièrement avec des partenaires en Informatique, contre-balançant ainsi son penchant pour les Mathématiques Appliquées. On pensera par exemple au rapprochement avec le Data Mining, consacré par l'organisation conjointe des conférences ECML et PKDD, ou bien celui avec les Sciences de la Découverte (*Discovery Sciences*), dans le cadre des conférences ALT et DS. L'Inférence Grammaticale est bien une branche de l'Informatique, dans la mesure où ce sont des problèmes d'algorithmique qui sont étudiés, et non des problèmes d'optimisation. Donc de ce point de vue, l'Apprentissage Automatique gagne également en s'en rapprochant.

Ensuite et surtout, Inférence Grammaticale et Apprentissage Automatique ont des raisons scientifiques de collaborer. En effet, considérons le domaine de l'Inférence Grammaticale *stochastique*, dont nous n'avons quasiment pas parlé. Son objectif est d'identifier, de façon exacte en théorie mais approximative en pratique, des distributions sur Σ^* ; la qualité des grammaires apprises se mesure alors à l'aide de la divergence de Kullback-Leibler. Or le même problème existe en Apprentissage Automatique, à savoir de trouver un modèle proche d'une distribution cible ; de plus, les apprentistes réalisent cet objectif en cherchant également des modèles qui minimisent la divergence de Kullback-Leibler. En somme, il n'y a pas de différence significative entre les deux domaines sur cette thématique, elle est commmune.

Enfin, même en Inférence Grammaticale classique, il existe des problèmes sur lesquels les deux communautés ont intérêt à travailler ensemble. C'est typiquement le cas de l'apprentis-

[1] comme *Machine Learning* ou *Journal of Machine Learning Research*. Beaucoup d'articles paraissent également dans les revues d'informatique fondamentale comme *Theoretical Computer Science* ou *Fundamenta Informaticæ*, ou de Reconnaissance des Formes comme *Pattern Analysis and Machine Intelligence*, ou de Traitement Automatique des Langues comme *Grammars*.

sage à partir de données textuelles bruitées. Ce que l'Inférence Grammaticale classique peut apporter dans ce cas, ce sont des algorithmes d'apprentissage à partir de mots, mais qu'il faut adapter pour qu'ils puissent traiter des données réelles. Sur ce point, qui constituera la première section du chapitre, j'ai particulièrement travaillé avec Marc Sebban, ainsi qu'Henri-Maxime Suchier pendant sa thèse [JS03, JTS04, JNSS05]. De son côté, l'Apprentissage Automatique a une longue expérience en matière de données bruitées, et de techniques d'optimisation des performances. Sur ce dernier point, nous avons avancé assez loin, notamment grâce à l'expertise de Richard Nock ; ce sont donc nos travaux communs [JNSS04, JNSS05, JSS09] qui serviront de fil conducteur à la seconde partie de ce chapitre.

5.1 Gestion du bruit dans les données textuelles

Bien qu'ils aient été développés pour satisfaire des critères de convergence théorique, comme on l'a vu dans le Chapitre 3, les algorithmes qui sont développés en Inférence Grammaticale restent également des heuristiques intéressantes à étudier en pratique, sur des données réelles, moyennant des adaptations.

Une des principales difficultés pour traiter des données réelles vient du bruit dans les données, intrinsèque au fait qu'elles résultent de mesures ou de saisies. C'est un problème sur lequel nous nous sommes particulièrement penché avec Marc Sebban, peu après son arrivée à Saint-Etienne, puis pendant la thèse d'Henri-Maxime Suchier [Suc06].

5.1.1 Typologie du bruit

Naturellement, les données bruitées sont au cœur de nombreux travaux en Apprentissage Automatique. Pour ce qui est des données de type mot, il existe plusieurs sortes de bruit qui peuvent les affecter. Après les avoir rappelées, nous verrons de quelle manière le bruit est habituellement pris en compte sur le plan théorique, et sur le plan pratique.

Notion de mot bruité

En tant que telle, la notion de bruit n'est pas facile à définir précisément. On peut dire que c'est un phénomène qui dégrade les données d'apprentissage, mais nous sommes loin d'une définition mathématique, et cela sous-entend que des données pures et parfaites pourraient exister. Or si cette hypothèse est concevable lorsqu'on cherche à identifier une grammaire cible en Inférence Grammaticale, elle est beaucoup plus discutable dans le cadre de données issues du monde réel, dont on cherche un bon modèle sans qu'un modèle parfait n'existe. Ainsi, plutôt qu'une *définition* du bruit, nous nous contenterons de décrire ses possibles effets sur des données.

En premier lieu, le bruit peut affecter la classe des exemples d'apprentissage uniquement, et transformer un mot reconnu par une grammaire cible en un mot qui ne l'est plus, et réciproquement. On parle alors de bruit de *classification* (ou d'*étiquette*) [AL88, KV94, JS03, JNSS04, JNSS05].

En second lieu, comme nous utilisons des mots comme données d'apprentissage, le bruit peut affecter la séquence même des lettres, par le biais d'opérations d'édition. Ce type de bruit est similaire au bruit, gaussien par exemple, qui peut affecter les attributs de données numériques (dans $\mathbb{R}^n$). Plusieurs variantes existent :

– Dans [GS91], les auteurs considèrent le *random attribute noise* : chaque lettre du mot peut être remplacée par une autre avec une certaine probabilité ; ainsi, seules des substitutions de lettres peuvent se produire, et la longueur du mot initial limite leur nombre.
– Dans le second modèle de bruit de [SS92] ($EDIT_2$ *noise*), cette contrainte est levée : chaque lettre est susceptible d'être substituée, effacée, ou bien une insertion peut se produire. Cependant, la longueur du mot initial continue de limiter leur nombre. Les auteurs étudient également un modèle plus simple ($EDIT_1$ *noise*) où une seule opération d'édition peut corrompre la totalité d'un mot.
– Enfin, dans [dlHJT06], nous nous sommes intéressés à des *bruits d'édition* plus généraux, où tout mot à distance d'édition k du mot original m est susceptible d'être le bruité de m.

En troisième lieu, le bruit peut être *malicieux* [Val85, KL93] : un oracle omniscient, susceptible de connaître l'algorithme qui utilisera les données en apprentissage, peut remplacer un exemple par un mot quelconque, classé de façon arbitraire. Dans ce cadre, aucune hypothèse n'est faite sur la déformation que subit l'exemple d'apprentissage : l'intérêt du bruit malicieux est d'étudier le pire cas.

Prise en compte du bruit dans les paradigmes d'apprentissage

Plusieurs travaux visant à prendre en compte le bruit dans les paradigmes d'apprentissage ont été menés [Ang92], essentiellement dans le cadre Pac : la plupart du temps, le bruit est vu comme un phénomène statistique.

Ainsi, dans [Val85], l'auteur s'intéresse au bruit malicieux dont il fixe le niveau $0 < \beta < 1$. Chaque fois qu'un exemple est tiré selon la distribution μ, un adversaire laisse filer l'exemple avec probabilité $1 - \beta$, et le remplace par autre chose sinon. En outre, l'apprenant doit être polynomial en $1/\beta$, en plus des paramètres habituels. Dans [KL93], les auteurs montrent de façon générale qu'il est impossible de Pac-apprendre un concept dès que $\beta > \epsilon/(1 + \epsilon)$. Or comme cette dernière expression vaut $\epsilon - \epsilon^2 + o(\epsilon^2)$ au voisinage de zéro, le niveau de bruit malicieux supportable par un algorithme de Pac-apprentissage doit être aussi faible que l'erreur ϵ visée. C'est donc un modèle extrêmement restrictif.

Concernant le bruit de classification, par contre, des résultats d'Angluin et Laird ont prouvé qu'il était relativement bénin pour l'apprentissage des k-Cnf dans le cadre Pac [AL88][2]. Repris par Kearns, ce travail a conduit au modèle d'apprentissage à partir de *requêtes statistiques*, permettant de montrer qu'en fait, la plupart des classes Pac-apprenables le sont encore en présence du bruit de classification [Kea98].

Maintenant, dans l'autre cadre d'identification que nous avons introduit, celui de l'identification à la limite, il existe très peu de travaux de modélisation du bruit. Dans [dlHJT06], nous avons étudié un bruit (d'édition) dit *systématique* : étant donné un langage cible L, nous supposons que tous les mots à distance d'édition au plus k d'un mot de L apparaissent dans un texte pour L. Dans ce cas, la classe $\mathcal{GOODBALL}(\Sigma)$ reste identifiable en temps Ipe-polynomiale à partir de Textes. Nous avons également introduit la notion de *débruitage à la limite* permettant d'obtenir, pour certaine classe de grammaires, des Informants non bruités à partir d'Informants qui le sont. Notons que des hypothèses alternatives sur la manière dont le bruit affecte les présentations ont été étudiées dans [Ste97].

[2]Angluin et Laird ont établi que les k-Cnf sont Pac-apprenables dès que $\beta < 1/2$ avec un algorithme polynomial en $(1/2 - \beta)^{-1}$.

Ingénierie du traitement des données bruitées

Indépendamment des modèles théoriques, et compte tenu du fait que le bruit est intrinsèque aux données du monde réel, les méthodes de traitement des données bruitées sont légions dans la littérature. On pensera par exemple à tous les travaux de la communauté "réduction de données" dans le cadre de l'*instance-based learning*. Dans [BL97], les auteurs utilisent une terminologie particulière pour toutes ces méthodes, que nous adoptons ici.

Ainsi, on peut commencer par distinguer les méthodes *filter* : elles visent à "nettoyer" les données avant tout processus d'apprentissage. Nous en présenterons quelques-unes, adaptées à l'apprentissage à partir de mots, dans la section suivante.

On distingue ensuite des méthodes *wrapper*, qui visent à tenir compte de l'existence du bruit pendant la phase d'apprentissage. On peut par exemple penser à l'introduction de variables d'écart dans le cadre des *support vector machines* [STC00] ou à l'introduction de marges douces dans les algorithmes de boosting [ROM99]. Dans [JS03, JTS04], nous avons développé de telles méthodes, en adaptant les algorithmes Rpni et Edsm (voir Section 5.1.3).

On peut encore imaginer des méthodes visant à traiter le bruit *après* l'apprentissage : d'une certaine manière, c'est ce que font les méthodes de lissage dans les modèle de langages [Goo01]. On rangera aussi dans cette catégorie les travaux visant à apprendre un modèle standard et un modèle du bruit, qui sont ensuite tous les deux utilisés par un algorithme de classification [DA00].

Enfin, il est intéressant de remarquer que de nombreux algorithmes d'apprentissage sont sensés fonctionner sans aucun traitement particulier des données bruitées. En Inférence Grammaticale, on lit parfois que l'apprentissage des automates non déterministes seraient plus à même de résister aux données bruitées. De même, les algorithmes d'Inférence Grammaticale Stochastique, visant à identifier des distributions de probabilité fidèles à une distribution cible μ, seraient plus résistants. Pourtant, au-delà des intuitions, ce sont des questions qui ne semblent pas avoir été abordées de façon théorique dans la littérature. Par exemple, certaines classes d'automates stochastiques déterministes sont Pac-apprenables [RST95, CT04] ; le sont-elles toujours en présence d'un bruit d'édition ?

5.1.2 Attrait expérimental des méthodes *filter*

Comme nous l'avons dit, les méthodes *filter* visent à détecter et traiter les exemples bruités avant toute utilisation en apprentissage. Pour ce faire, on peut utiliser un premier algorithme d'apprentissage, connu pour être résistant aux données bruitées, puis exploiter le classifieur qu'il retourne pour traiter les données mal classées. Parmi tous les algorithmes résistants connus, les graphes de k-plus proches voisins [CH67] sont très intéressants, en particulier parce que la littérature sur la détection de données bruitées est très abondante. Rappelons que la règle de classification des k-plus proches voisins attribue la classe d'un exemple w en calculant la classe majoritaire parmi les k exemples de E qui sont les plus proches de w. Cette méthode est très résistante au bruit car un exemple bruité isolé au milieu d'exemples de la classe opposée a un impact très limité sur la règle de classification. La propriété duale est qu'un exemple probablement bruité peut être facilement détecté en analysant son voisinage.

Naturellement, construire des graphes de plus proches voisins nécessitent de disposer d'une métrique sur les données. Concernant les données de type mot, on peut envisager d'utiliser les métriques standard (de Hamming, de Levenshtein, *etc.*), mais elles ne conduisent pas forcément à des résultats intéressants car elles sont généralistes, donc sans lien avec le problème

d'apprentissage initial. Au contraire, les *distances d'édition stochastiques* [RY98, OS06, BJS06] sont basées sur des modèles de langages qu'on peut entrainer sur les données. Elles sont plus performantes, en pratique, mais elles présentent aussi l'inconvénient de ne pas être de vraies métriques[3], ce qui pose ensuite des problèmes lors de la construction ou de l'utilisation des graphes de voisinage : toutes les méthodes efficaces sur ces graphes, décrites dans [CNBYM01] par exemple, exploitent intensivement l'inégalité triangulaire.

Une autre manière de définir des distances entre mots consiste à les transformer en vecteurs numériques, puis à utiliser une métrique classique dans les espaces vectoriels. Prenons l'exemple de l'application de Parikh $\Phi : \Sigma^* \to \mathbb{N}^{|\Sigma|}$ transformant tout mot w en un vecteur dont les composantes sont les nombres d'occurrences de chaque lettre dans w : si $\Sigma = \{a_1, a_2, \ldots, a_n\}$, alors $\Phi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ [Par66]. On pose ensuite $d(u,v) = \|\Phi(u) - \Phi(v)\|_2 = \sqrt{\sum_{x\in\Sigma}(|u|_x - |v|_x)^2}$. Si Φ était injective (ce qui n'est pas le cas), alors $d(\cdot,\cdot)$ hériterait des propriétés de la métrique euclidienne, et serait elle-même une métrique. Pour poursuivre dans cette voie, si $\kappa(\cdot,\cdot)$ est un noyau sur les mots (*string-kernel*) dont la projection associée est injective, comme c'est le cas pour le *gap-weighted subsequences kernel* [STC04], alors la distance $d(u,v) = \sqrt{\kappa(u,u) + \kappa(v,v) - 2\kappa(u,v)}$ est une métrique sur les mots.

Cependant, les distances associées à des noyaux souffrent de nouveau du fait qu'elles ne sont pas spécialisées sur le problème d'apprentissage qui nous intéresse. Dans [SJY02] et dans le cadre du travail de DEA d'Aziz Yahiaoui [Yah03], nous avons proposé une méthode alternative, qui a ensuite été revisitée dans [Dup06]. Elle consiste à entrainer des modèles de langages sur chaque classe d'exemples. On utilise ensuite les probabilités qu'attribue chacun de ces modèles à un mot pour le projeter dans un espace euclidien réel.

De façon plus précise, nous avons utilisé des bi-grammes lissés [Goo01], donc un modèle très simple. Dans un bi-gramme, la probabilité d'une lettre ne dépend que de la lettre qui précède dans un mot. Ainsi, étant donné un mot de n lettres $w = a_1a_2a_3\ldots a_n$, sa probabilité vaut :

$$P(w) = \prod_{i=0}^{n} p(a_{i+1}|a_i).$$

Les probabilités élémentaires $p(a_i|a_{i-1})$ sont estimées à partir des mots de l'échantillon d'apprentissage, par un simple calcul de fréquences, puis une méthode de lissage. Enfin, pour donner un sens aux probabilités initiales et finales $p(a_1|a_0)$ et $p(a_{n+1}|a_n)$, on utilise des marqueurs $a_0 = \$$ et $a_{n+1} = £$ de début et de fin de chaîne.

En supposant disposer de deux classes $+$ et $-$, nous pouvons désormais calculer $P_+(w)$ et $P_-(w)$, qui sont les probabilités de w relativement aux sous-classes positives et négatives de l'échantillon E. Dans le but de permettre la comparaison entre deux mots de longueurs différentes, nous normalisons chacune de ces probabilités en utilisant une moyenne géométrique : $P'_+(w) = \sqrt[n+1]{P_+(w)}$ et $P'_-(w) = \sqrt[n+1]{P_-(w)}$, où $n = |w|$. Nous pouvons maintenant projeter w en un point de coordonnées $\big(P'_+(w), P'_-(w)\big)$ dans l'espace $\mathbb{R}^2$. Remarquons que la projection considérée n'est pas forcément injective, mais qu'elle a toutes les chances de l'être si les données sont vraiment issues du monde réel.

La Figure 5.1 présente un exemple de cette projection pour une base de prénoms français dans laquelle les exemples positifs sont les prénoms féminins, et les exemples négatifs sont les prénoms masculins. On constate que la projection des exemples dans $\mathbb{R}^2$ permet de séparer

[3]Pour de telles distances, l'inégalité triangulaire, la symétrie, ou même le fait que $d(x,x) = 0$ ne sont généralement pas vérifiées.

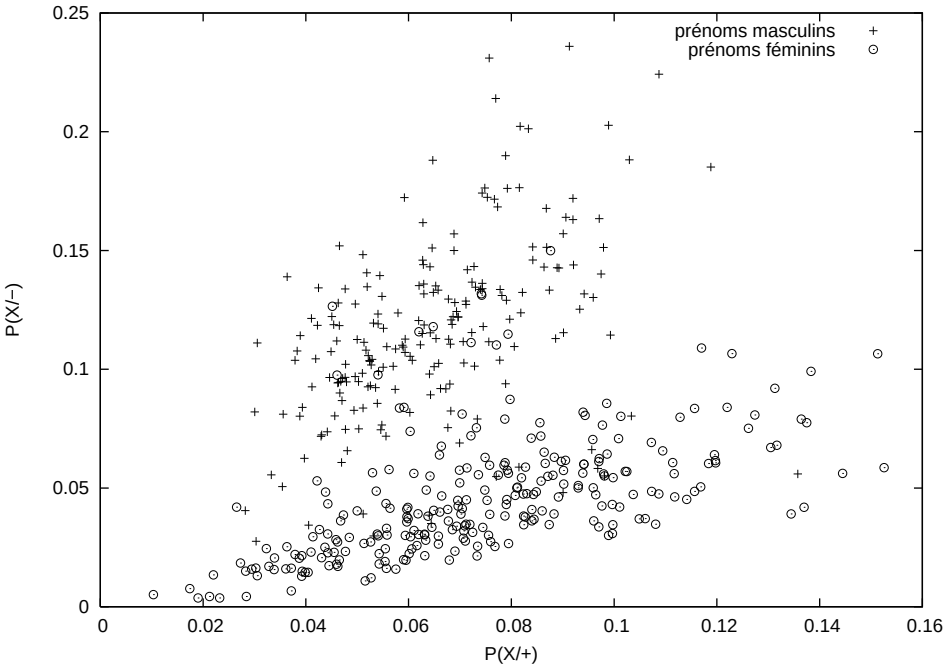


FIG. 5.1 – Projection d'une base de prénoms français dans un espace euclidien à deux dimensions à l'aide de bi-grammes. La classe + est la celle des prénoms féminins, et la classe -, celle des prénoms masculins.

relativement correctement les deux classes. Cela s'explique facilement par le fait que certaines séquences de lettres sont caractéristiques du genre pour les prénoms français. Ainsi, la terminaison "`ine`" est bien plus caractéristique d'un prénom féminin que masculin, ce qui se traduit directement dans les paramètres élémentaires des bi-grammes.

On peut maintenant considérer la métrique euclidienne dans $\mathbb{R}^2$ pour construire un graphe des k plus proches voisins sur l'échantillon d'apprentissage. Plus précisément, on utilise la distance suivante sur Σ^* :

$$d(u,v) = \sqrt{(P'_+(v) - P'_+(u))^2 + (P'_-(v) - P'_-(u))^2}.$$

La Figure 5.2 montre une partie du graphe obtenu à partir de la base de prénoms français précédente. Clairement, dans ce graphe, chaque prénom est souvent plus proche d'un prénom de même genre que d'un prénom de genre opposé. En outre, certains prénoms (grisés) comme Etienne ou Aimée sont mal classés (puisque la majorité des prénoms qui les entourent sont de la classe opposée), et d'autres, comme Claude sont nécessairement mal étiquetés (puisque les classes ont une intersection non vide).

Sur un graphe des k plus proches voisins, on peut également définir la *marge* m d'un exemple w de classe y en posant :

$$m(w) = \frac{2N_y(w) - k}{k},$$

où $N_y(w)$ désigne le nombre d'exemples qui sont de classe y parmi les k plus proches voisins de w. Ainsi, un exemple n'ayant que des voisins de même classe que lui, comme Maurice, reçoit

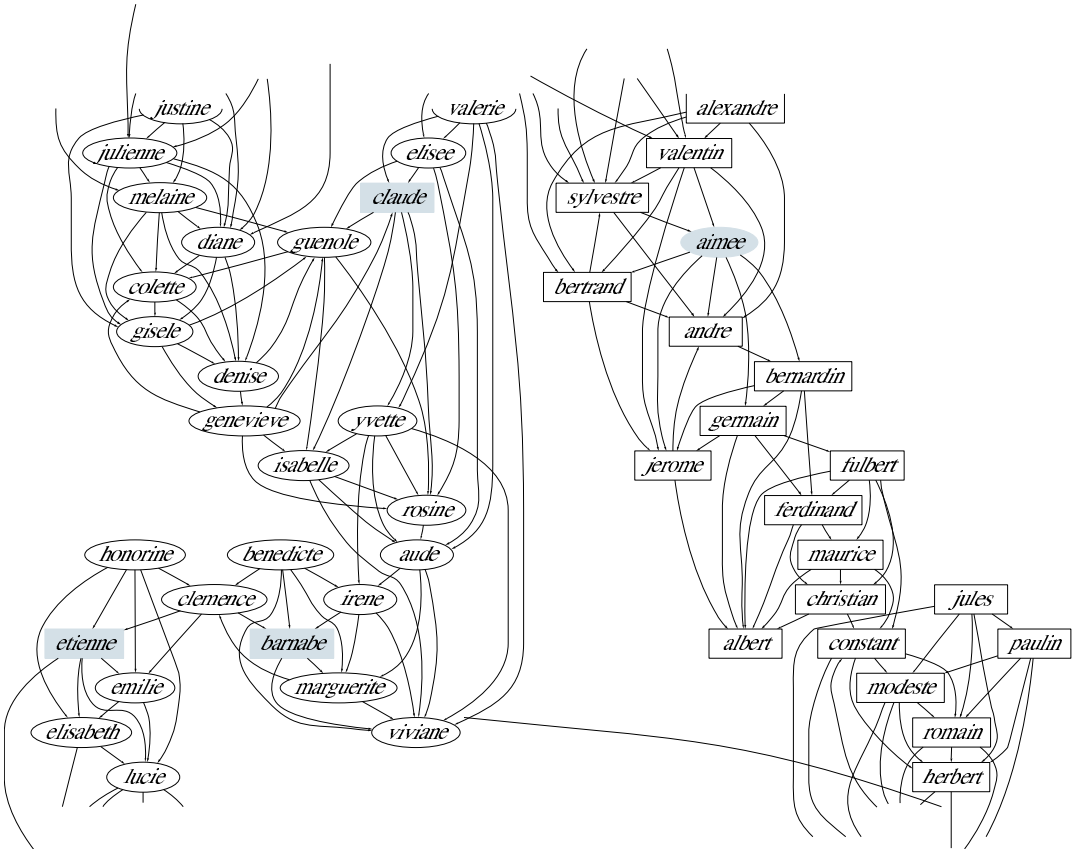


FIG. 5.2 – Une partie du graphe de voisinage construit à partir de la projection de la base de prénoms français dans $\mathbb{R}^2$.

une marge de +1, alors qu'un exemple n'ayant que des voisins de classe opposée à la sienne, comme Barnabé, reçoit une marge de −1. Quant aux exemples relevant des deux classes à la fois (comme Camille ou Dominique) ou les exemples rares, c'est-à-dire ceux contenant des lettres ou des sous-séquences inhabituelles (comme Gwenaël ou Zoïle), ils reçoivent des marges proches de 0.

Ainsi, non seulement le signe de $m(w)$, positif ou négatif, indique si l'exemple est bien classé au sens de la règle des k-plus proches voisins pour la distance choisie, mais sa valeur absolue $|m(w)|$ donne une indication de la confiance qu'on peut porter sur ce jugement. C'est donc une fonction intéressante pour nettoyer une base d'apprentissage, soit en changeant la classe d'un exemple, soit en l'éliminant. On notera qu'en Inférence Grammaticale, dans le cadre de l'apprentissage des AFD, la classe des exemples d'apprentissage détermine si un état est final ou pas, alors que les mots eux-mêmes permettent de déterminer la structure de l'AFD (états, transitions) ; l'élimination d'un exemple n'est donc pas toujours appropriée.

5.1.3 RPNI* : une méthode de type *wrapper*

Les méthodes *wrapper* visent à traiter le bruit pendant l'exécution de l'algorithme d'apprentissage. Dans [JS03], nous avons adapté l'algorithme RPNI [OG92] pour qu'il résiste au

bruit de classification. Nous avons fait un travail similaire sur l'algorithme EDSM [LPP98] dans le cadre du TER de Frédéric Tantini, ce qui nous a permis de participer à la compétition d'algorithmes *"Learning* DFA *from Noisy Data"* organisée par S. Lucas lors de GECCO'04 [JTS04].

Impact du bruit de classification sur RPNI

Comme nous l'avons vu dans la Section 3.1.3, RPNI reçoit un ensemble E_+ d'exemples (positifs) et un ensemble E_- de contre-exemples (ou d'exemples négatifs). Dans le contexte habituel d'utilisation de l'algorithme, on suppose que $E_+ \cap E_- = \emptyset$, mais en présence de bruit de classification, cette hypothèse ne tient plus. Malgré tout, nous continuerons temporairement de la supposer vraie pour mener cette analyse.

Reprenons maintenant l'exemple page 47 que nous avons utilisé pour illustrer le fonctionnement de RPNI. L'algorithme reçoit $E_+ = \{\lambda, 1, 01, 011, 101\}$ et $E_- = \{00, 10\}$, ce qui lui permet d'inférer l'AFD de gauche dans la Figure 5.3 ; cet AFD est bien la cible qui nous a permis de généré les données (*i.e.*, RPNI a convergé vers la cible grâce aux données). Nous ajoutons maintenant le mot 1111 à E_-. Ce mot devrait appartenir à E_+, en l'absence de bruit. Si maintenant nous utilisons RPNI, nous obtenons l'AFD de droite dans la Figure 5.3. Clairement, la présence d'une seule donnée bruitée se traduit, tout d'abord, par une augmentation du nombre d'états des AFD produits, et ensuite, par une chute souvent vertigineuse du taux de succès en généralisation des AFD.

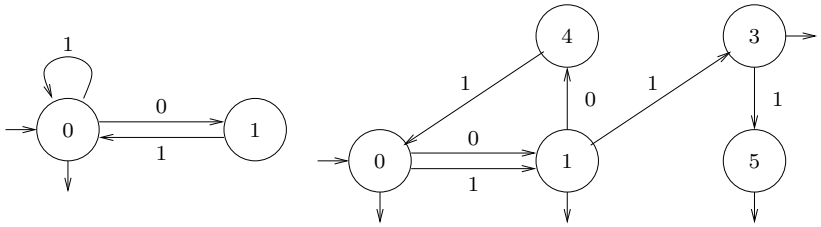


FIG. 5.3 – Deux AFD appris avec RPNI, en l'absence et en présence de données bruitées.

En fait, dans RPNI, deux états sont fusionnables si aucun exemple négatif n'est accepté par l'AFD résultant de leur fusion. De manière plus formelle, soit $B = A/\pi = \langle Q, \Sigma, i, F, \delta \rangle$ l'AFD résultant de la fusion. Considérons un état p de B et définissons les quantités n_1 et n_2 qui sont respectivement les nombres de mots de E_+ et de E_- contenus[4] par p, c'est-à-dire, $n_1 = |\{w \in E_+ : \delta(i, w) = p\}|$ et $n_2 = |\{w \in E_- : \delta(i, w) = p\}|$. Dans RPNI, p est final dès que $n_1 > 0$. De plus, pour qu'une fusion réussisse, il faut nécessairement que $(n_1 > 0 \Longrightarrow n_2 = 0)$.

Mais en présence de bruit, il se peut très bien que p contienne plusieurs mots de E_+ et de E_-, donc que $n_1 > 0$ et $n_2 > 0$, sans pour autant qu'une erreur d'apprentissage soit en train d'être commise. En fait, le bruit de classification a deux effets :

1. Lorsqu'un mot w appartient à E_+ du fait du bruit (il serait donc dans E_- en l'absence de bruit), l'état p qui le contient après une fusion acquiert le statut d'état final. Donc tout autre mot non bruité de E_- qui est contenu par p joue le rôle de contre-exemple à la fusion. Cette fusion va donc être rejetée par RPNI alors qu'elle serait acceptée en l'absence de bruit.

[4]Cette terminologie est pratique quoiqu'impropre ; on dira qu'un exemple d'apprentissage w est contenu par un état p si $\delta(i, w) = p$.

2. Lorsqu'un mot w appartient à E_- du fait du bruit (il serait donc dans E_+ en l'absence de bruit), l'état p qui le contient après fusion a malgré tout de fortes chances de contenir d'autres exemples de E_+ (non bruités, eux), donc d'être un état final. Le mot w joue alors le rôle de contre-exemple à la fusion. De nouveau, cette fusion va être rejetée par RPNI alors qu'elle aurait été acceptée en l'absence de bruit.

De l'analyse précédente, nous tirons deux conclusions. D'une part, nous devons relâcher la règle d'attribution du statut d'état final à un état, et nous utiliserons donc désormais la règle de majorité suivante : On dira qu'un état est *positif* (ou final) s'il contient strictement plus de mots positifs (de E_+) que de mots négatifs (de E_-) : $p \in F$ ssi $n_1 > n_2$. On dira qu'il est *négatif* sinon.

D'autre part, nous devons relâcher la contrainte d'acceptation d'une fusion en permettant à un certain nombre de mots de E_+ et de E_- d'être mal classés[5] par l'AFD résultant de la fusion : On dira qu'une fusion est *statistiquement acceptable* si la proportion p_2 d'exemples mal classés dans tout l'AFD après fusion n'est pas significativement plus grande que la proportion p_1 d'exemples mal classés avant fusion.

Comme nous allons le voir, cette manière de faire évite la construction d'AFD avec un grand nombre d'états et un faible pouvoir en généralisation. Nous acceptons de réduire, par fusion d'états, la taille des AFD appris si elle n'induit pas d'augmentation significative de l'erreur empirique. Comme les fluctuations d'échantillonnage peuvent avoir une influence lors d'une décision d'acceptation ou de rejet d'une fusion, une simple comparaison entre p_1 et p_2 n'est pas pertinente. nous allons donc mettre en œuvre un test statistique de comparaison de proportions avec des intervalles de confiance.

Mise en œuvre de l'heuristique

Commençons par revisiter les principes sous-tendant le bon fonctionnement de RPNI. Dans cet algorithme, les exemples positifs servent à construire la structure de l'AFD ; en outre, ce sont eux qui donne le statut d'état final à un état. Les exemples négatifs, eux, sont uniquement utilisés pour rejeter des fusions. Or en présence de bruit de classification, certains exemples positifs seraient négatifs en l'absence de bruit, et inversement. Donc il n'y a plus de raison de dissymétriser le rôle des exemples positifs et négatifs. En conséquence, le PTA initialement construit par RPNI ne doit plus l'être sur les exemples positifs uniquement, mais sur tous les exemples. En outre, nous avons choisi d'attribuer le statut d'état final à tout état contenant une *majorité* d'exemples positifs ; il est donc désormais tout à fait possible que $E_+ \cap E_- \neq \emptyset$. Enfin, en cours d'exécution, les exemples négatifs contenus par un état positif seront considérés comme des exemples bruités, et inversement. Mais cette qualification pourra changer au fur et à mesure des fusions, selon le nombre majoritaire d'exemples positifs ou négatifs contenus par cet état après une fusion.

Considérons maintenant la nouvelle règle d'acceptation des fusions : désormais, nous accepterons une fusion si la proportion p_2 d'exemples mal classés par l'AFD après fusion n'augmente pas significativement par rapport la proportion p_1 d'exemples mal classés avant fusion. Nous devons donc tester l'hypothèse nulle $\mathcal{H}_0 : p_1 = p_2$ contre l'hypothèse alternative $\mathcal{H}_a : p_2 > p_1$. Ce test est unilatéral puisque seule une valeur suffisamment grande de $p_2 - p_1$ doit conduire au rejet de l'hypothèse testée. A l'inverse, une petite valeur et plus encore une valeur négative, ne remet pas en cause la qualité d'une fusion. Les quantités p_1 et p_2 sont inconnues : elle cor-

[5] Un exemple négatif (resp. positif) sera dit *mal classé* s'il est contenu par un état positif (resp. négatif).

respondent respectivement aux erreurs réelles de l'AFD avant et après la fusion. On les estime par les erreurs empiriques $\hat{p}_1 = N_1/N$ et $\hat{p}_2 = N_2/N$ calculées sur l'ensemble d'apprentissage, où N_1 (resp. N_2) est le nombre d'exemples d'apprentissage mal classés avant (resp. après) la fusion, et N est le nombre total d'exemples (*i.e.*, $N = |E_+ \cup E_-|$).

Dans notre test, nous nous intéressons donc à la différence $\hat{p}_2 - \hat{p}_1$, qui, sous l'hypothèse nulle $\mathcal{H}_0$, a pour espérance $\mathbb{E}(\hat{p_2} - \hat{p_1}) = p_2 - p_1 = 0$, et pour variance, $\mathbb{V}\text{ar}(\hat{p_2} - \hat{p_1}) = \frac{p_2(1-p_2)}{N} + \frac{p_1(1-p_1)}{N} = \frac{2pq}{N}$, avec $p = p_1 = p_2$ et $q = 1 - p$. On estime habituellement p par la moyenne des deux proportions d'exemples mal classés avant et après fusion : $\hat{p} = (\hat{p_1} + \hat{p_2})/2$. Si les contraintes $Np > 5$ et $Nq > 5$ sont vérifiées, les conditions d'approximation par une distribution normale sont satisfaites, donc la variable $T = \hat{p_2} - \hat{p_1}$ suit une loi normale $\mathcal{N}(E(T), \text{Var}(T)) = \mathcal{N}\left(0, \sqrt{\frac{2pq}{N}}\right)$. Nous devons déterminer la *valeur critique* Z_α au risque α qui définit la borne du rejet de $\mathcal{H}_0$, et qui correspond au $(1-\alpha)$-percentile de la loi $\mathcal{N}\left(0, \sqrt{\frac{2pq}{N}}\right)$. Pour cela, on (centre et on) réduit T :

$$P(T > Z_\alpha) = P\left(\frac{T}{\sqrt{\frac{2pq}{N}}} > \frac{Z_\alpha}{\sqrt{\frac{2pq}{N}}}\right) = P\left(\mathcal{N}(0,1) > \frac{Z_\alpha}{\sqrt{\frac{2pq}{N}}}\right).$$

On a donc $P(T > Z_\alpha) = \alpha$ si $Z_\alpha = U_\alpha\sqrt{\frac{2\hat{p}\hat{q}}{N}}$, où U_α est le $(1-\alpha)$-percentile de la loi normale $\mathcal{N}(0,1)$; on rappelle que ces dernières valeurs sont tabulées, ce qui permet de calculer effectivement la valeur de Z_α. Bref, si $T > Z_\alpha$, on rejette l'hypothèse $\mathcal{H}_0$, donc la fusion, avec un risque α. Inversement, si $T \leq Z_\alpha$, la fusion est statistiquement validée, donc acceptée.

Nous sommes maintenant en mesure de présenter les modifications que nous apportons à RPNI de sorte qu'il tolère les données bruitées. Nous donnons le pseudo-code de notre nouvel algorithme, RPNI*, dans les Algorithmes 10 et 11.

Algorithm 10: RPNI*(E_+, E_-, α)

Input: Deux ensembles $E_+, E_- \subseteq \Sigma^*$, un paramètre $0 \leq \alpha \leq 0.5$
Output: Un AFD $A = \langle Q, \Sigma, i, F, \delta\rangle$

```
A ← CONSTRUIRE_PTA(E+, E−) ;
B ← A ; π ← {{q} : q ∈ Q};
for p = 1 to |Q| − 1 do
    for q = 0 to p − 1 do
        if p = min{π(p)} et q = min{π(q)} then
            π′ ← CALCUL_FUSION(π, p, q, δ);
            B′ ← ATTRIBUE_ÉTATS_FINAUX(A/π′, E+, E−);
            if STATISTIQUEMENT_COMPATIBLE(B, B′, E+, E−, α) then
                π ← π′ ; B ← B′ ;
return(B)
```

RPNI* fonctionne comme RPNI, à ceci près qu'il construit un PTA généralisé, qu'il attribue le statut d'état final aux états contenant une majorité d'exemples de E_+, et qu'il utilise le test statistique de comparaison de proportions précédent pour décider de la réussite ou de

Algorithm 11: STATISTIQUEMENT_COMPATIBLE$(B, B', E_+, E_-, \alpha)$

Input: Les ensembles E_+, E_-, les AFD B et B' avant et après fusion, le paramètre α
Output: Un booléen TRUE ou FALSE

$U_\alpha \leftarrow (1-\alpha)$-percentile de la loi $\mathcal{N}(0,1)$;
$N \leftarrow |E_+ \cup E_-|$;
$N_1 \leftarrow$ nombre d'exemples de $E_+ \cup E_-$ mal classés par B ;
$N_2 \leftarrow$ nombre d'exemples de $E_+ \cup E_-$ mal classés par B' ;
$\hat{p_1} \leftarrow N_1/N$; $\hat{p_2} \leftarrow N_2/N$;
$T \leftarrow \hat{p_2} - \hat{p_1}$;
$\hat{p} \leftarrow (\hat{p_1} + \hat{p_2})/2$; $\hat{q} \leftarrow (1-\hat{p})$;
$Z_\alpha \leftarrow U_\alpha \sqrt{2\hat{p}\hat{q}/N}$;
if $T > Z_\alpha$ **then return** FALSE **else return** TRUE;

l'échec d'une fusion. En pratique, le risque α est un paramètre de généralisation qu'on peut faire varier entre 0 et 0.5. Une bonne utilisation de ce paramètre consiste donc à tester RPNI* pour différentes valeurs de α, puis à garder le meilleur AFD produit, en termes de taux de succès en généralisation et de nombre d'états.

Notons que plus α est proche de 0, plus Z_α est grand, plus il faut un nombre important d'exemples mal classés pour rejeter une fusion, et donc plus souple est la règle de fusion. A la limite, quand $\alpha = 0$, RPNI* retourne l'automate universel. A l'inverse, plus α est proche de 0.5, plus Z_α est proche de 0, plus il faut un nombre faible d'exemples mal classés pour rejeter une fusion, et donc plus rigide est la règle de fusion. Quand $\alpha = 0.5$, une fusion est rejetée dès qu'un nouvel exemple est mal classé. Cela signifie qu'en l'absence de bruit, RPNI* se comporte exactement comme RPNI : aucun exemple n'est mal classé par le PTA, et une fusion échoue dès qu'un exemple est mal classé à la suite d'une fusion. Enfin, quand $\alpha > 0.5$, Z_α est négatif, ce qui signifie que même les fusions qui classent correctement tous les mots de l'ensemble d'apprentissage sont rejetées ; RPNI* retourne alors le PTA.

Quelques résultats expérimentaux sur RPNI*

Dans [JS03], nous avons mené plusieurs études expérimentales de RPNI* (dont certaines sur des données de l'UCI). Nous n'en reprenons qu'une ici, et laissons le lecteur intéressé consulter notre article pour plus d'information.

Dans celle qui nous intéresse, nous avons réalisé une étude comparative entre RPNI* lorsque le paramètre α était choisi de façon optimale, et RPNI* lorsque $\alpha = 0.5$; rappelons que dans ce cas, RPNI* n'accepte aucune fusion accroissant l'erreur empirique, et se comporte donc exactement comme RPNI, à ceci près que les états du PTA initial peuvent contenir des exemples positifs et négatifs.

Nous avons ensuite généré une base synthétique non bruitée à l'aide du simulateur Gowachin [LPC98]. L'utilisation de RPNI sur cette base permet d'inférer l'AFD cible, qui a 9 états et ne commet aucune erreur en généralisation. Enfin, nous avons fait varier le taux de bruit γ de 0 à 20% et comparons les deux algorithmes en termes de taux de succès en généralisation et de nombre d'états de l'AFD inféré (voir Figure 5.4). Les mesures sont faites à l'aide d'une procédure de 10-cross-validation ; les folders de tests sont les folders originaux non bruités.

Dans la Figure 5.4, on note que jusqu'à 6% de bruit, RPNI* est capable de "digérer"

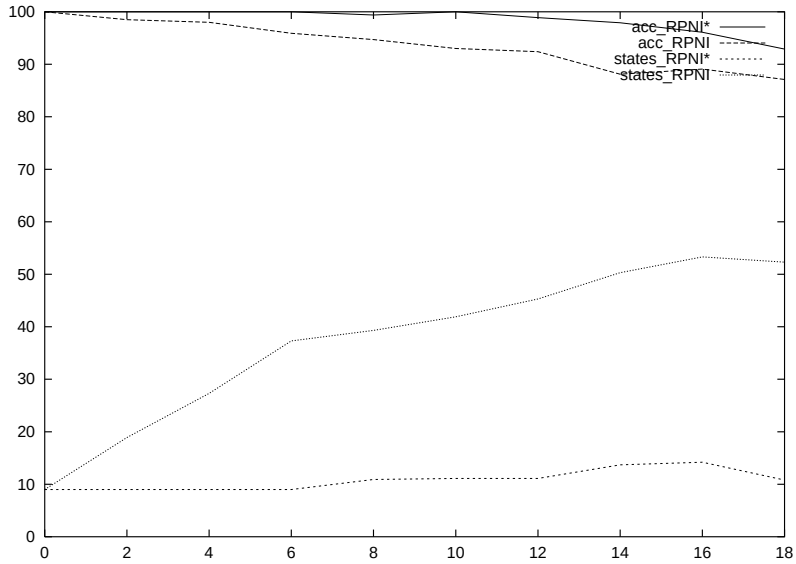


FIG. 5.4 – Résultats obtenus par 10-cross-validation sur la base artificielle : acc_RPNI* et acc_RPNI représentent respectivement le taux de succès en généralisation de RPNI* pour un α optimal, et pour $\alpha = 0.5$; states_RPNI* et states_RPNI représentent le nombre d'états moyen des AFD inférés.

complètement les données bruitées : sur les 10 folders de la procédure de cross-validation, RPNI* infère systématiquement la cible, garantissant ainsi un taux de succès de 100%. De 6% à 10% de bruit, RPNI* retourne des AFD qui ne font pas d'erreurs en généralisation malgré des nombres d'états légèrement supérieurs à 9. Au-delà de 10%, le bruit est trop important pour être totalement digéré, ce qui induit une légère divergence des performances en termes de taux de succès et de nombre d'états. A l'inverse, le bruit impacte RPNI dès le début de l'expérience : non seulement le taux de succès chute régulièrement, mais la taille des AFD inférés croît très vite.

Force est donc de constater que notre heuristique est capable de digérer une partie du bruit de classification. Il va cependant de soi qu'un tel constat ne serait pas possible si RPNI n'était pas capable, sans données bruitées, de retourner des AFD acceptables. C'est la raison pour laquelle nous avons aussi étudié l'extension de EDSM aux données bruitées dans [JTS04], que nous ne reprenons pas ici.

5.2 Optimisation des classifieurs de séquences

A la suite de nos travaux sur les méthodes *filter* et *wrapper* visant à apprendre à partir de séquences bruitées, nous nous sommes demandés dans quelle mesure nous pourrions optimiser nos résultats. Finalement, nous avons fait une sorte de synthèse de ces travaux en les combinant dans une technique de boosting adaptée aux données bruitées. C'est à l'occasion d'une visite de Richard Nock (Université des Antilles Guyanne) à l'EURISE, entre décembre 2003 et février 2004 que nous avons abordé ce problème [JNSS04, JNSS05]. C'est aussi sur ce thème du boosting qu'Henri-Maxime Suchier a fait sa thèse [Suc06].

5.2.1 Principes du boosting

La théorie du boosting a 20 ans. Dans les travaux initiaux de Schapire [Sch90], le problème était de savoir sous quelle condition une classe de grammaires $\mathcal{G}$ *faiblement* polynomialement PAC-apprenable, c'est-à-dire apprenable au sens de la Définition 8 mais pour des valeurs fixées de ϵ et δ, pouvait être polynomialement PAC-apprenable au sens strict, avec des valeurs arbitraires de ϵ et δ. Et la réponse fut qu'il n'y avait pas de condition (voir [KV94] pour une explication détaillée de la preuve).

Dans la conférence invitée qu'il a donné lors d'ICML'03, M. Kearns expliquait qu'à l'époque, le problème semblait résolu. Pourtant, c'était un résultat théorique, difficile à exploiter en pratique jusqu'à la naissance d'ADABOOST [FS97]. Ce dernier algorithme a fait l'objet d'un nombre considérable de travaux, certains visant une compréhension fine de son fonctionnement (par exemple, l'étude de ses comportements à la limite [RDS04] ou de son erreur en généralisation [SFBL98]), d'autres une mise en évidence de ses relations avec divers domaines de l'Apprentissage Automatique (par exemple, la régression [FHT00], ou les SVM), d'autres encore ses extensions possibles (aux données bruitées [Fre01], aux classes multiples [SS99], au tri de données (*ranking*) [SS00], *etc*).

Ainsi, en 2010, la présentation de la théorie du boosting, et les questions qui animent cette communauté ont profondément changé. Une visite du site `www.boosting.org` est très instructive (même s'il n'est pas souvent mis à jour). Loin des paradigmes d'apprentissage, le boosting est une méthode d'ensembles, permettant d'optimiser les performances d'un apprenant "faible" en combinant linéairement les classifieurs qu'il produit lorsqu'on change judicieusement la distribution sur les exemples d'apprentissage [MR02].

Notion d'apprenant faible

Plaçons nous dans un cadre où l'on dispose d'une distribution inconnue μ sur $\Sigma^* \times \{-1, +1\}$ et d'un échantillon d'apprentissage $E = \{(x_1, y_1), (x_2, y_2), \ldots (x_m, y_m)\}$ tiré aléatoirement et indépendamment selon μ. Nous nous donnons maintenant une classe de grammaires $\mathcal{G}$ dont les hypothèses, dites de *base*, sont des fonctions $h : \Sigma^* \to \{-1, +1\}$.

Nous supposons en outre disposer d'un algorithme $\mathfrak{F}$, dit de *base* lui aussi, qui prend en entrée (1) l'échantillon E et (2) *n'importe quelle distribution ν sur E*, et qui retourne des hypothèses de $\mathcal{G}$. La distribution ν est simplement une fonction de pondération des exemples telle que $\nu(x_i) \in [0, 1]$ pour tout $1 \leq i \leq m$ et $\sum_{i=1}^{m} \nu(x_i) = 1$. En pratique, l'algorithme $\mathfrak{F}(E, \nu)$ peut être capable d'utiliser directement des échantillons d'apprentissage pondérés comme c'est généralement le cas en apprentissage automatique, ou bien, comme dans le cadre PAC, simuler l'oracle $\text{EX}(E, \nu)$ pour "obtenir" des exemples et travailler[6].

On attend enfin de l'algorithme $\mathfrak{F}$ qu'il fournisse des hypothèses de façon non aléatoire, quelle que soit la distribution ν sur E. On attend plus précisément qu'il *apprenne* des hypothèses *faibles*. Dans les travaux pionniers sur la PAC-apprenabilité faible, cela signifiait que l'erreur en généralisation d'une telle hypothèse devait être significativement différente de $1/2$. Or dans le contexte d'un algorithme comme ADABOOST, cette définition n'est pas la bonne : c'est l'erreur *empirique* des hypothèses fournie par $\mathfrak{F}$ sur E et ν qui doit être $< 1/2$, car cela suffit pour établir les propriétés d'ADABOOST.

Ce point est important : il existe aujourd'hui autant de définitions d'apprenants faibles que d'algorithmes de boosting, et la seule définition qui les caractérise tous consiste à dire qu'un

[6] L'ensemble E est fini, la distribution ν est connue, il suffit donc d'avoir accès à un générateur aléatoire.

apprenant $\mathfrak{F}$ est faible *dans le contexte* d'un algorithme de boosting si ce dernier converge quand on l'utilise avec $\mathfrak{F}$ ... Cette relativité n'a d'ailleurs pas que des inconvénients : aujourd'hui, on travaille souvent sur la procédure de boosting elle-même, avant de décrire les propriétés de l'apprenant faible auquel elle pourra s'appliquer ; cela repousse, de fait, la question souvent difficile de savoir si un tel apprenant faible existe, en particulier si on peut montrer par des expérimentations que la procédure de boosting a d'excellentes performances en généralisation.

Dans cette partie, nous utiliserons donc une définition particulière, adaptée de [FS97], permettant à l'algorithme AdaBoost de fonctionner correctement. Cette condition sera suffisante, mais pas nécessaire [MR02].

Définition 15 (Hypothèse Γ_0-faible) *Soient $0 < \Gamma_0 < 1$ et $h = \mathfrak{F}(E, \nu)$. On dit que h est une* hypothèse Γ_0-faible *par rapport à E et ν au sens d'*AdaBoost *si*

$$\Gamma_0 \leq \sum_{i=1}^{m} \nu(x_i) y_i h(x_i) < 1.$$

La quantité $\gamma_h = \sum_{i=1}^{m} \nu(x_i) y_i h(x_i)$ s'appelle la crête[7] *de h par rapport à E et ν.*

D'un point de vue statistique, γ_h est l'espérance de la variable aléatoire $Yh(X)$ selon ν, *i.e.*, $\gamma_h = \mathbb{E}_{(x,y)\sim\nu}[yh(x)]$. Or cette variable vaut +1 si $h(x) = y$, et -1 si $h(x) \neq y$. Donc si $0 < \Gamma_0 \leq \gamma_h < 1$, alors la probabilité que h classe correctement un exemple d'apprentissage est significativement plus grande que celle qu'il fasse une erreur. A l'inverse, si $\gamma_h \simeq 0$, alors h classe les exemples de E de façon aléatoire ; dans ce cas, l'erreur empirique de h vaut 1/2 (comme nous le verrons ci-dessous). Enfin, si $\gamma_h \simeq 1$, alors h ne commet aucune erreur ; c'est une hypothèse "forte" dont les performances ne peuvent être améliorées par boosting.

En somme, si une hypothèse est Γ_0-faible, alors elle a un comportement significativement différent de l'aléatoire sur E, et un algorithme qui l'a produite a effectivement appris quelque chose sur les données. Notons que dans les articles initiaux sur AdaBoost, ce n'était pas la crête d'une hypothèse h qui était utilisée, mais son *erreur* : $\varepsilon_h = \sum_{i=1}^{m} \nu(x_i)[\![h(x_i) \neq y_i]\!]$. Or cette quantité n'est vraiment intéressante que lorsque les hypothèses sont à valeur dans $\{-1, +1\}$, au contraire de la notion de crête qui reste intéressante lorsque les hypothèses à valeur dans $[-1, +1]$, ou dans $\mathbb{R}$ [SS99][8]. Quoi qu'il en soit, erreurs et crêtes sont très directement liées dans le cas binaire :

Proposition 3 *L'hypothèse $h = \mathfrak{F}(E, \nu)$ à valeur dans $\{-1, +1\}$ est Γ_0-faible au sens d'*AdaBoost *si et seulement si $0 < \varepsilon_h \leq (1/2) - (\Gamma_0/2)$. En d'autres termes, h est une hypothèse faible si son erreur empirique est significativement plus petite que 1/2.*

Preuve: On a $\gamma_h = \sum_{i=1}^{m} \nu(x_i) y_i h(x_i)$. Divisons l'ensemble E en deux sous-ensembles $E(+)$ et $E(-)$ des exemples respectivement bien et mal classés par h, puis définissons le poids de ces ensembles :

$$E(+) = \{(x_i, y_i) \in E : y_i h(x_i) = +1\}, \qquad E(-) = \{(x_i, y_i) \in E : y_i h(x_i) = -1\},$$
$$W(+) = \sum_{(x_i,y_i)\in E(+)} \nu(x_i), \qquad W(-) = \sum_{(x_i,y_i)\in E(-)} \nu(x_i).$$

[7] *edge* en anglais.

[8] Dans le cas d'hypothèse à valeurs dans $\mathbb{R}$, c'est le signe de $h(x)$ qui donne la classe de x.

Clairement, on a :

$$\left\{\begin{array}{rcl}\gamma_h &=& W(+)-W(-)\\ 1 &=& W(+)+W(-)\end{array}\right. \iff \left\{\begin{array}{rcl}W(+) &=& (1+\gamma_h)/2\\ W(-) &=& (1-\gamma_h)/2\end{array}\right.$$

Par ailleurs, on a $\varepsilon_h = \sum_{i=1}^{m} \nu(x_i)[\![h(x_i) \neq y_i]\!]$, donc $\varepsilon_h = W(-) = (1-\gamma_h)/2$, c'est-à-dire $\gamma_h = 1 - 2\varepsilon_h$. Donc si $\Gamma_0 \leq \gamma_h < 1$, alors $0 < \varepsilon_h \leq (1/2) - (\Gamma_0/2)$. □

Nous pouvons maintenant poser la définition suivante d'*apprenant faible*, définition de nouveau non universelle :

Définition 16 (Algorithme d'apprentissage faible par rapport à un échantillon) *Soit $0 < \Gamma_0 < 1$. Etant donné un échantillon d'apprentissage E, on dit qu'un algorithme $\mathfrak{F}$ est un* apprenant Γ_0-faible pour E *au sens d'*AdaBoost *si pour toute distribution ν sur E, $\mathfrak{F}(E,\nu)$ retourne une hypothèse Γ_0-faible. On exigera, en outre, que la complexité de $\mathfrak{F}$ soit polynomiale en $\|E\|$ et $1/\Gamma_0$.*

Cette notion est relative à l'échantillon d'apprentissage E. Ainsi, dans le cas où l'échantillon E, provenant de la distribution μ sur $\Sigma^* \times \{-1,+1\}$, est bruité, au sens où il contient à la fois l'exemple $(w,+1)$ et l'exemple $(w,-1)$, alors aucune hypothèse ne peut être faible pour cet échantillon, puisque pour la pondération $\nu(w,+1) = \nu(w,-1) = 1/2$ et $\nu(x,\cdot) = 0$ pour tout $x \neq w$, on a $\gamma_h = 0$, c'est-à-dire $\varepsilon_h = 1/2$, quelle que soit $h \in \mathcal{G}$. Cependant, cela n'interdit pas l'existence d'hypothèses faibles pour d'autres échantillons E (non bruités) tirés selon μ. Nous reviendrons sur le boosting à partir de données bruitées dans la section suivante.

Quelques mots sur l'algorithme AdaBoost

Soit E un échantillon d'apprentissage fixé. Le principe de l'algorithme AdaBoost est de faire varier la distribution ν de façon astucieuse, donc de produire des distributions $\nu_0, \nu_1, \ldots \nu_T$ sur E, puis de récupérer les hypothèses $h_0, h_1, \ldots h_T$ produite par l'algorithme $\mathfrak{F}$ pour chaque distribution, et les combiner linéairement en une hypothèse $H_T : x \mapsto \text{signe}\left(\sum_{t=0}^{T} c_t h_t(x)\right)$, dite *forte*, avec des coefficients c_t judicieusement choisis (voir Algorithme 12).

De façon remarquable, cette technique dépend moins des performances de l'algorithme $\mathfrak{F}$ que des choix faits pour repondérer les exemples. Initialement, le poids de chaque exemple vaut $1/m$. Puis pour calculer ν_{t+1} à partir de ν_t, on choisit d'augmenter exponentiellement le poids des exemples mal classés par h_t (ceux pour lesquels $y_i h_t(x_i) = -1$), et de diminuer exponentiellement le poids des exemples bien classés par h_t (*i.e.*, $y_i h_t(x_i) = +1$) :

$$\nu_{t+1}(x_i) = \frac{\nu_t(x_i)\exp\left(-c_t y_i h_t(x_i)\right)}{Z_t}.$$

Le coefficient Z_t permet de normaliser ν_{t+1} de sorte que $\sum_{i=1}^{m} \nu_{t+1}(x_i) = 1$: on prend $Z_t = \sum_{i=1}^{m} \nu_t(x_i)\exp\left(-c_t y_i h_t(x_i)\right)$.

Il nous faut maintenant à commenter le choix du coefficient c_t : on fixe sa valeur à $(1/2)\ln\left((1+\gamma_t)/(1-\gamma_t)\right)$, où γ_t désigne la crête de h_t par rapport à E et ν_t. En fait, AdaBoost est dans la droite ligne du principe Erm : il cherche à diminuer l'erreur empirique de

Algorithm 12: AdaBoost(E, T)

Input: Un échantillon $E = \{(x_1, y_1), (x_2, y_2), \dots (x_m, y_m)\} \subseteq \Sigma^* \times \{-1, +1\}$, un apprenant faible $\mathfrak{F}$, un nombre maximum d'itération T

Output: Une hypothèse forte H_T

```
for i = 1 to m do ν_0(x_i) ← (1/m);
for t = 0 to T do
    h_t ← 𝔉(E, ν_t);
    γ_t ← Σ_{i=1}^m ν_t(x_1) y_i h_t(x_i);
    c_t ← (1/2) ln((1 + γ_t)/(1 − γ_t));
    Z_t ← Σ_{i=1}^m ν_t(x_i) exp(−c_t y_i h_t(x_i));
    for i = 1 to m do
        ν_{t+1}(x_i) ← ν_t(x_i) exp(−c_t y_i h_t(x_i)) /Z_t ;
return la fonction H_T = ( x ↦ signe( Σ_{t=1}^T c_t h_t(x) ) )
```

l'hypothèse H_T, définie par :

$$\varepsilon(E, H_T) = \frac{1}{m} \sum_{i=1}^{m} [\![H_T(x_i) \neq y_i]\!].$$

Rappelons que sous les conditions de validité du principe Erm, ceci doit permettre de limiter l'erreur réelle de H_T. Or on peut montrer que $\varepsilon(E, H_T) \leq \left(\prod_{t=0}^{T} Z_t \right)$, et la valeur choisie pour c_t est en fait celle qui minimise la quantité Z_t à chaque itération, donc $\varepsilon(E, H_T)$ (voir [FS97] pour un calcul détaillé).

Ce choix de la valeur de c_t permet en outre de réécrire l'algorithme AdaBoost de façon plus effective. Cette réécriture n'est jamais faite, car elle est évidente pour un spécialiste. Elle peut cependant ne pas l'être pour un simple utilisateur d'AdaBoost : il constatera que l'algorithme est finalement très simple, et c'est un élément essentiel pour expliquer sa popularité. Cette réécriture contredit en outre l'idée selon laquelle les poids des exemples augmentent ou diminuent de façon exponentielle à chaque itération : au contraire, le choix du coefficient c_t implique que les exemples soient repondérés de façon linéaire (voir l'Algorithme 13).

Ce choix des coefficients c_t permet enfin l'étude de la convergence d'AdaBoost. Par la convergence des procédures de boosting, on entend l'étude des erreurs commises par l'hypothèse forte H_T quand $T \to +\infty$. Concernant l'erreur empirique, on peut montrer que $\varepsilon(E, H_T) < \exp\left(-\sum_{t=0}^{T} \gamma_t^2 / 2 \right)$ (voir par exemple [MR02]). En conséquence :

Théorème 22 *Si $\mathfrak{F}$ est un apprenant Γ_0-faible sur E, alors $\varepsilon(E, H_T) < \exp\left(-(T+1)\Gamma_0^2/2\right)$. Donc l'erreur empirique de H_T est nulle dès que $T \geq \left\lceil 2(\ln m)/\Gamma_0^2 \right\rceil$.*

Preuve: Comme $\mathfrak{F}$ ne retourne que des hypothèses Γ_0-faible sur E, on a $\gamma_t \geq \Gamma_0$ pour tout $0 \leq t \leq T$, donc $-\sum_{t=0}^{T} \gamma_t^2 \leq -(T+1)\Gamma_0^2$, et donc $\varepsilon(E, H_T) < \exp\left(-(T+1)\Gamma_0^2/2\right)$. Or $\varepsilon(E, H_T) = (1/m) \sum_{i=1}^{m} [\![H_T(x_i) \neq y_i]\!]$, donc si $\exp\left(-(T+1)\Gamma_0^2/2\right) < 1/m$ alors H_T ne commet aucune erreur sur E, et c'est le cas dès que $T \geq \left\lceil 2(\ln m)/\Gamma_0^2 \right\rceil$. □

Enfin concernant l'erreur réelle de l'hypothèse H_T, nous n'en parlerons quasiment pas ici pour deux raisons. D'abord, les bornes intéressantes ne s'établissent pas dans le cadre usuel

Algorithm 13: AdaBoost(E, T), après réécriture.

Input: Un échantillon $E = \{(x_1, y_1), (x_2, y_2), \dots (x_m, y_m)\} \subseteq \Sigma^* \times \{-1, +1\}$, un apprenant faible $\mathfrak{F}$, un nombre maximum d'itération T

Output: Une hypothèse forte H_T

```
for i = 1 to m do ν(x_i) ← (1/m);
for t = 0 to T do
    h_t ← 𝔉(E, ν);
    γ ← Σ_{i=1}^{m} ν(x_i) y_i h_t(x_i);
    c_t ← (1/2) ln((1 + γ)/(1 − γ));
    for i = 1 to m do
        if h_t(x_i) = y_i then ν(x_i) ← ν(x_i)/(1 + γ) else ν(x_i) ← ν(x_i)/(1 − γ);
return la fonction H_T = ( x ↦ signe ( Σ_{t=1}^{T} c_t h_t(x) ) )
```

de Valiant, mais nécessitent de recourir à la théorie des marges [SFBL98]. Ensuite, nous avons peu travaillé sur ce sujet (sauf dans [JSS09]). Nous citons donc simplement le résultat suivant, combinant plusieurs résultats de la littérature :

Théorème 23 ([KP02, SFBL98]) *Soit $\mathfrak{F}$ un algorithme retournant des hypothèses prises dans une classe $\mathcal{G}$ et μ une distribution sur $\Sigma^* \times \{-1, +1\}$. Etant donné un échantillon d'apprentissage $E = \{(x_1, y_1), \dots, (x_m, y_m)\}$ tiré aléatoirement et indépendamment selon μ, pour tous $\delta > 0, \theta > 0, \Gamma_0 > 0$, si $\mathfrak{F}$ retourne des hypothèses Γ_0-faibles sur E, alors avec probabilité au moins $1 - \delta$, l'erreur en généralisation de l'hypothèse H_T produite par* AdaBoost *est plus petite que*

$$\left((1-\Gamma_0)^{1-\theta}(1+\Gamma_0)^{1+\theta}\right)^{\frac{T}{2}} + \mathcal{O}\left(\frac{1}{\theta}\sqrt{\frac{d_{\mathcal{G}}}{m}}\right) + \mathcal{O}\left(\sqrt{\frac{\log(1/\delta)}{m}}\right).$$

Si $\theta < \Gamma_0/2$, alors le premier terme décroit vers 0 lorsque $T \to +\infty$; les deux autres sont des termes de pénalisation dépendant, entre autres, de la VC-dimension $d_{\mathcal{G}}$ de la famille $\mathcal{G}$.

Conditions d'utilisation du boosting en Inférence Grammaticale

Nous avons vu qu'il existait deux grandes familles d'algorithmes en Inférence Grammaticale. Les premiers visent l'identification exacte d'une classe de grammaires $\mathcal{G}$. Dans ce cadre, *booster* un algorithme n'a pas vraiment de sens, dans la mesure où cet algorithme va tenter de construire une hypothèse consistante avec les données, donc une hypothèse ne commettant aucune erreur en apprentissage (*i.e.*, une hypothèse de crête +1). En d'autres termes, aucun apprenant faible n'est étudié en identification exacte, et l'amélioration (parfois souhaitée) des performances en généralisation d'un algorithme comme Rpni n'est pas possible par boosting.

Il faut donc se placer dans le cadre de l'identification approximative (Pac-apprenabilité). Mais la question de l'existence d'un apprenant faible dans ce cadre est loin d'être simple. Si nous étions dans le cadre de la Pac-apprenabilité faible, nous aurions le résultat suivant :

Théorème 24 ([Sch90, KV94]) *Soit $\mathcal{L}$ une classe de langages représentés par une classe $\mathcal{G}$ de grammaires. Si $\mathcal{L}$ est polynomialement faiblement* Pac*-apprenable en termes des grammaires de $\mathcal{G}$, alors $\mathcal{L}$ est polynomialement (fortement)* Pac*-apprenable en termes des "grammaires" de la classe $\mathcal{G}'$ des arbres ternaires de majorité avec feuilles dans $\mathcal{G}$.*

Comme les langages rationnels sont inhéremment non prédictibles (sous des hypothèses cryptographiques standard) [KV89], ils ne peuvent pas être polynomialement PAC-appris, quelle que soit la classe de grammaires choisie pour les représenter. Donc les AFD ne sont pas polynomialement faiblement PAC-apprenables car sinon, par boosting, on disposerait d'une classe de "grammaires" polynomialement équivalentes, reconnaissant les langages rationnels, et qui serait fortement PAC-apprenable ce qui est impossible. Le même raisonnement ne tient pas pour les bonnes boules car nous ne savons pas si ces dernières sont ou non polynomialement PAC-apprenables quand on les représente sous la forme d'arbres ternaires de majorité avec feuilles dans $\mathcal{GOODBALL}(\Sigma)$.

Au-delà de ces considérations, la définition d'apprenant faible qu'exploite ADABOOST est de toute façon complètement différente puisqu'elle est relative à un échantillon d'apprentissage. Or pour un échantillon donné E de mots, il se peut tout à fait qu'il existe un apprenant retournant des AFD (ou des bonnes boules) Γ_0-faible pour E. Autrement dit, même si les AFD (ou les bonnes boules) ne sont pas faiblement PAC-apprenables, il se peut qu'ADABOOST se comporte correctement sur certains, voire sur la plupart des échantillons d'apprentissage pour un algorithme qui serait sensible aux variations d'une distribution ν sur E et qui commettrait des erreurs en apprentissage sans avoir un comportement purement aléatoire.

L'algorithme RPNI* peut satisfaire ces contraintes : la règle de fusion qu'il utilise est statistique, tout comme la règle d'attribution du statut d'état final ; donc intrinsèquement, c'est un algorithme qui commet des erreurs en apprentissage (en acceptant à tort certaines fusions). En outre, on peut facilement modifier ces règles pour qu'elles tiennent compte des distributions ν. En revanche, RPNI* n'est pas fait pour travailler avec des données saines, mais des données bruitées. Et nous savons d'ores et déjà qu'ADABOOST ne résiste pas aux données bruitées (puisqu'avec des données bruitées, aucune hypothèse ne peut être faible au sens d'ADABOOST). Il est donc nécessaire de changer d'algorithme de boosting, ce que nous allons faire dans la section suivante.

5.2.2 RPNI* comme apprenant faible

Face à des données bruitées, les performances d'ADABOOST se dégradent fortement. On peut facilement le comprendre, car la règle de repondération des exemples force l'algorithme à classer correctement toutes les données, y compris les données bruitées. En pratique, ce phénomène de sur-apprentissage se traduit ensuite par une augmentation de l'erreur en généralisation au cours des itérations. Aussi, plusieurs travaux ont tenté de limiter l'impact des données bruitées en évaluant l'intérêt d'augmenter le poids d'un exemple mal classé. Certaines approches tentent de contrôler la mise à jour des poids en utilisant des fonctions de repondération plus douces que la fonction exponentielle originale [FHT00, DW00, Fre01]. Mais l'utilisation de ces nouvelles fonctions peut remettre en question les propriétés de convergence du boosting. On pourrait aussi penser que des exemples bruités, dont la classe est douteuse, devraient être supprimés de l'échantillon d'apprentissage. Or comme nous l'avons déjà évoqué, en Inférence Grammaticale, les mots permettent de déterminer la structure de l'AFD (états, transitions), indépendamment de leurs classes, qui servent, elles, à déterminer quels états sont finaux. Il parait donc essentiel d'intégrer les exemples bruités dans les processus d'apprentissage et donc aussi de boosting.

Quand Richard Nock nous a rendu visite, nous disposions, d'une part, d'algorithmes capable d'inférer des AFD en "digérant" le bruit (méthodes *wrapper*), et d'autre part, de techniques permettant d'assigner un niveau de confiance sur l'étiquette des exemples (méthodes

filter). Nous avons donc développé un nouvel algorithme de boosting utilisant ces informations pour combiner des AFD, dans un contexte bruité, tout en évitant le phénomène de sur-apprentissage. Plus précisément, nous avons choisi de conserver les exemples bruités et l'usage de la fonction exponentielle, mais nous supposons avoir accès à un *oracle de confiance*, qui fournit une estimation sur la confiance dans la classe d'un exemple.

Une valeur positive pour un exemple signifie que nous pouvons être certain de sa classe. En revanche, une valeur négative signifie que l'on doit émettre des doutes sur sa classe. Soulignons qu'une confiance négative ne signifie pas que l'exemple appartient à la classe opposée, mais plutôt que l'on ne connaît pas sa classe réelle. Il se pourrait tout à fait que l'oracle se trompe lorsqu'il affirme qu'un exemple est bruité, et dans ce cas l'impact de changer la classe de l'exemple serait bien plus nuisible que de traiter cet exemple en supposant qu'on ne connait pas sa classe réelle. De plus, il est nécessaire de supposer que l'oracle fournit des valeurs *réelles* dans $[-1, +1]$ plutôt que des valeurs *entières* dans $\{-1, +1\}$. Ce choix est lié au fait que l'oracle doit être simulé en pratique, donc qu'il est préférable de tenir compte des imperfections de cette simulation dès l'étude théorique.

L'utilisation de l'oracle de confiance pour traiter les données bruitées nécessite incontournablement une modification de la règle standard de mise à jour des poids : si AdaBoost augmente le poids de tous les exemples mal appris, nous voulons ici seulement augmenter les poids des exemples qui le "méritent", c'est-à-dire ceux qui ne sont probablement pas bruités. Si cette stratégie semble utile pour améliorer n'importe quel algorithme d'apprentissage, elle nous parait particulièrement adaptée pour repousser les limites d'un algorithme *wrapper* comme RPNI* puisqu'on pourra utiliser les outils des méthodes *filter*, en particulier le graphe de voisinage introduit dans la Section 5.1.2, pour simuler l'oracle.

L'algorithme OraBoost

Soit $E = \{(x_1, y_1, q_1), (x_2, y_2, q_2), \ldots, (x_m, y_m, q_m)\}$ un échantillon d'apprentissage fixé. Les quantités q_i sont des valeurs réelles non nulle dans $[-1, +1]\backslash\{0\}$ qui dénotent la confiance qu'accorde l'oracle dans la classe y_i attribuée à l'exemple x_i. Rappelons qu'une confiance positive pour un exemple (x_i, y_i) signifie que x_i appartient avec certitude à la classe y_i qui lui est assignée, donc que cet exemple n'est probablement pas bruité, tandis qu'une confiance négative signifie que l'on ne peut rien supposer sur la classe de x_i. Pour simplifier notre développement, nous supposerons que q_i n'est pas nul, ce qui signifie que l'oracle ne s'abstient pas, qu'il émet nécessairement un avis sur tous les exemples d'apprentissage ; en outre, les valeurs de q_i sont supposées constantes au cours du processus d'apprentissage. Toutes ces hypothèses sur les valeurs de confiance pourraient être levées moyennant des développements théoriques plus importants.

L'Algorithme 14, OraBoost, suit le même schéma qu'AdaBoost : on définit une première distribution ν_0 uniforme sur l'échantillon d'apprentissage E ; ensuite, à chaque itération t, on utilise $\mathfrak{F}$ sur E et ν_t pour obtenir une hypothèse h_t et on repondère les exemples d'apprentissage en construisant la distribution ν_{t+1} ; enfin, au bout de T itérations, on combine les T hypothèses produites en une hypothèse globale H_T "forte".

Cependant, le schéma de repondération des exemples change sensiblement, dans la mesure où il fait intervenir explicitement les réponses de l'oracle. Par cette mise à jour, nous désirons non seulement augmenter le poids de tout exemple mal classé pour lequel l'oracle se dit confiant, mais également baisser le poids de tout exemple "douteux", qu'il soit bien ou mal classé par l'hypothèse courante. La règle de mise à jour que nous proposons permet de réaliser

Algorithm 14: OraBoost(E, T)

Input: Un échantillon $E = \{(x_1, y_1, q_1), (x_2, y_2, q_2), \dots (x_m, y_m, q_m)\} \subseteq \Sigma^* \times \{-1, +1\} \times ([-1, 0[\cup]0, +1])$, un apprenant $\mathfrak{F}$, un nombre maximum d'itération T

Output: Une hypothèse forte H_T

```
for i = 1 to m do ν_0(x_i) ← (1/m);
for t = 0 to T do
    h_t ← 𝔉(E, ν_t);
    c_t ← calcul_coeff_boosting(E, ν_t, h_t);
    for i = 1 to m do
        a_t(x_i) ←(if q_i < 0 then −q_i else q_i y_i h_t(x_i)) ;
    Z_t ← Σ_{i=1}^{m} ν_t(x_i) exp(−c_t a_t(x_i));
    for i = 1 to m do
        ν_{t+1}(x_i) ← ν_t(x_i) exp(−c_t a_t(x_i)) /Z_t ;
return la fonction H_T = (x ↦ signe(Σ_{t=1}^{T} c_t h_t(x)))
```

cela. En effet, pour un exemple x_i à qui l'oracle attribue une confiance positive ($q_i > 0$) et qui n'est donc probablement pas bruité, on a :

$$\nu_{t+1}(x_i) = \frac{\nu_t(x_i) \exp(-c_t q_i y_i h_t(x_i))}{Z_t}.$$

Pour un exemple x_i de confiance négative ($q_i < 0$), donc un exemple suspect, on a :

$$\nu_{t+1}(x_i) = \frac{\nu_t(x_i) \exp(c_t q_i)}{Z_t}.$$

En montrant que $c_t > 0$ (ce qu'on établira dans le Lemme 8), nos objectifs seront bien atteints. En effet, dans le cas de confiances positives, les poids des exemples mal classés augmenteront, ceux des bien classés baisseront. Dans le cas de confiances négatives, que l'exemple soit correctement classé ou non, son poids baissera systématiquement, à chaque itération.

Une autre modification majeure intervient dans cet algorithme de boosting. Elle concerne le coefficient c_t, qui sera différent de celui d'AdaBoost mais doit être calculé de façon optimale pour préserver les propriétés théoriques du boosting. Si on peut obtenir une expression exacte pour ce coefficient dans le cas d'AdaBoost, ça ne sera pas le cas pour OraBoost car la règle de mise à jour des poids utilise les valeurs réelles q_i fournies par l'oracle de confiance. Néanmoins, nous pourrons obtenir une valeur approchée $\hat{c}_t$ de la valeur théorique de c_t avec une précision quelconque, par dichotomie. Et pour cela, nous aurons besoin de définir et d'exploiter une nouvelle notion d'*hypothèse faible*, adaptée à OraBoost et différente de celle utilisée pour AdaBoost. Cette notion d'hypothèse faible permettra, *in fine*, de prouver que l'erreur empirique de l'hypothèse globale H_T converge bien vers 0 avec le nombre T d'itérations.

Sur les conditions d'utilisation d'OraBoost

Pour mener l'étude des propriétés théoriques d'OraBoost, plusieurs difficultés se présentent. Tout d'abord, comme AdaBoost, nous souhaitons qu'OraBoost minimise l'erreur empirique de l'hypothèse H_T, en vertu du principe Erm. Or comme certains exemples de E sont bruités, il ne faut surtout pas considérer tous les exemples de E dans le calcul de l'erreur : cela impliquerait le phénomène de sur-apprentissage que nous cherchons à éviter. On définit donc l'erreur de H_T sur les exemples certifiés sains par l'oracle uniquement.

Pour cela, on divise l'ensemble E en deux sous-ensembles, l'ensemble E_N des exemples dont l'étiquette est douteuse selon l'oracle de confiance, et l'ensemble $E_{\overline{N}}$ des exemples déclarés sains par l'oracle : $E_N = \{(x_i, y_i, q_i) \in E : q_i < 0\}$ et $E_{\overline{N}} = \{(x_i, y_i, q_i) \in E : q_i > 0\}$. Puis on considère l'erreur empirique de H_T sur $E_{\overline{N}}$ uniquement :

$$\varepsilon(E_{\overline{N}}, H_T) = \frac{1}{|E_{\overline{N}}|} \sum_{(x_i, y_i, \cdot) \in E_{\overline{N}}} [\![H_T(x_i) \neq y_i]\!].$$

De même, la notion d'hypothèse faible ne doit logiquement pas tenir compte des exemples potentiellement bruités :

Définition 17 (Hypothèse Γ_0-faible) *Soient $0 < \Gamma_0 < 1$ et $h = \mathfrak{F}(E, \nu)$. On dit que h est une* hypothèse Γ_0-faible *par rapport à E et ν au sens d'*OraBoost *si*

$$\Gamma_0 \leq \frac{\sum_{(x_i, y_i, q_i) \in E_{\overline{N}}} \nu(x_i) q_i y_i h(x_i)}{\sum_{(x_i, \cdot, q_i) \in E_{\overline{N}}} \nu(x_i) q_i} < 1.$$

De plus, on dit que l'algorithme $\mathfrak{F}$ est Γ_0-faible sur E pour OraBoost *s'il fournit des hypothèses Γ_0-faible pour toute distribution ν sur E.*

Outre le fait qu'elle ne concerne qu'un sous-ensemble des exemples de E, on notera que les valeurs de confiance q_i interviennent directement dans cette définition. En fait, c'est une caractéristique qu'il est difficile d'expliquer de façon intuitive, sinon par le fait que cette définition d'hypothèse faible s'est imposée lors de l'étude des propriétés théoriques d'OraBoost. Autrement dit, c'est l'algorithme de boosting qui nous a conduit vers cette définition d'hypothèse faible, et non l'inverse.

Ceci posé, nous établissons maintenant nos résultats sans preuve : le lecteur peut les consulter dans [JNSS04, JNSS05]. Tout d'abord, concernant l'erreur empirique, nous obtenons la majoration suivante (dont la preuve est relativement standard) :

Proposition 4

$$\varepsilon(E_{\overline{N}}, H_T) \leq \frac{m}{|E_{\overline{N}}|} \left(\prod_{t=0}^{T} Z_t \right).$$

Comme le terme $(m/|E_{\overline{N}}|)$ est constant pour E et $E_{\overline{N}}$, cette proposition implique que pour minimiser l'erreur sur $E_{\overline{N}}$, il faut minimiser chaque $Z_t = \sum_{i=1}^{m} \nu_t(x_i) \exp(-c_t a_t(x_i))$, donc choisir les coefficients c_t de sorte que $\left(\frac{\partial Z_t}{\partial c_t}\right) = -\sum_{i=1}^{m} \nu_t(x_i) a_t(x_i) \exp(-c_t a_t(x_i)) = 0$.

Dans le cas d'AdaBoost, on sait résoudre cette équation et trouver une expression littérale du coefficient c_t. Malheureusement, dans le cas d'Oraboost, Z_t dépend des valeurs de confiance q_i, de valeurs réelles donc, qui sont certes constantes au cours des itérations, mais qui sont différentes pour chaque exemple. On peut malgré tout montrer le résultat suivant, dont la preuve utilise intensivement la définition d'hypothèse faible :

Lemme 8 *Si h_t est une hypothèse Γ_0-faible par rapport à E et ν_t au sens d'*OraBoost*, alors l'équation $\left(\frac{\partial Z_t}{\partial c_t}\right) = 0$ admet une solution unique, strictement positive, qui vérifie :*

$$\frac{1}{2\overline{q}} \ln\left(\frac{W_t(E) - W_t(E^-_{\overline{N},t})}{W_t(E^-_{\overline{N},t})}\right) \leq c_t \leq \frac{1}{2\underline{q}} \ln\left(\frac{W_t(E) - W_t(E^-_{\overline{N},t})}{W_t(E^-_{\overline{N},t})}\right),$$

où les différentes quantités utilisées sont définies par :

- $\underline{q} = \min_{(\cdot,\cdot,q_i)\in E} |q_i|$ *et* $\overline{q} = \max_{(\cdot,\cdot,q_i)\in E} |q_i|$,
- $E^-_{\overline{N},t} = \{(x_i, y_i, q_i) \in E : q_i > 0$ *et* $y_i h_t(x_i) = -1\}$ *et*
- $W_t(X) = \sum_{(x_i,\cdot,q_i)\in X} |q_i| \nu_t(x_i)$, *pour tout* $X \subseteq E$.

En d'autres termes, on a $c_t = r \ln\Big((W_t(E) - W_t(E^-_{\overline{N},t}))/W_t(E^-_{\overline{N},t})\Big)$ pour un certain $r \in [1/2\overline{q}, 1/2\underline{q}]$. Rappelons qu'on cherche c_t de sorte que $\left(\frac{\partial Z_t}{\partial c_t}\right) = 0$; du coup, avoir cet encadrement permet d'approximer c_t par dichotomie. Plus précisément, si on cherche à approximer c_t par $\hat{c}_t$ de sorte que $\frac{|\hat{c}_t - c_t|}{|c_t|} < \epsilon$ pour $0 < \epsilon < 1$, alors l'approximation dichotomique se fera en au plus $\mathcal{O}(\ln(\overline{q}/\underline{q}) + \ln(1/\epsilon))$ étapes. Cette faible logarithmique est d'autant plus appréciable qu'on doit réaliser l'approximation à chaque itération d'OraBoost.

La définition d'hypothèse faible est enfin essentielle pour démontrer les résultats sur la convergence de l'erreur en apprentissage qui suivent. De nouveau, les preuves ne sont pas triviales, dans la mesure où nous ne disposons pas d'une expression littérale pour c_t :

Théorème 25 *Soit $\mathfrak{F}$ est un apprenant Γ_0-faible sur E pour* OraBoost.

- *Lorsque les coefficients c_t sont optimaux, l'erreur empirique est majorée par :*

$$\varepsilon(E_{\overline{N}}, H_T) < \exp\left(-(T+1)\frac{\Gamma_0^2 \underline{q}^2}{2\overline{q}^2}\right),$$

  ce qui garantit (théoriquement) une convergence exponentielle de l'erreur empirique de H_T vers 0 lorsque $T \to +\infty$.
- *Lorsque les coefficient c_t sont approximés par des valeurs $\hat{c}_t$ de sorte que*

$$|\hat{c}_t - c_t| \leq \frac{(\ln 2)\Gamma_0^2 \underline{q}^2}{4\overline{q}^3},$$

  l'erreur empirique est majorée par :

$$\varepsilon(E_{\overline{N}}, H_T) < \exp\left(-(T+1)\frac{\Gamma_0^2 \underline{q}^2}{4\overline{q}^2}\right),$$

  ce qui garantit, en pratique, une convergence exponentielle de l'erreur empirique de H_T vers 0 lorsque $T \to +\infty$.

Quelques résultats expérimentaux avec OraBoost

Dans [JNSS04, JNSS05], nous avons mené plusieurs expérimentations sur OraBoost, quand on l'utilise pour améliorer les performances de Rpni*. Comme nous l'avons déjà évoqué, nous avons utilisé nos techniques *filter* pour simuler l'oracle de confiance. Plus précisément,

nous avons utilisé un graphe de k-plus proches voisins en suivant la méthodologie présentée en Section 5.1.2, puis utilisé la *marge* comme valeur de confiance sur l'étiquette des exemples. Nous ne présentons ici qu'une partie de l'étude.

Pour la mener, nous avons utilisé trois sortes de bases. Les premières sont des *benchmarks* connus du répertoire UCI[9] : AGARICUS et TICTACTOE. Les secondes sont des bases réelles. L'une, appelée USF, recense les mille prénoms les plus fréquemment donnés aux USA en 2002. L'autre, appelée WF, contient la plupart des prénoms qui, une fois francisés, commence par lettre "A". Pour ces deux bases, le concept à apprendre est le sexe correspondant au prénom. Enfin, les dernières bases sont artificielles et construites selon la présence ou l'absence de motifs (facteurs). Plus précisément, selon la base, nous avons fait varier la taille de l'alphabet entre 4 et 8 lettres, puis généré aléatoirement des mots dont la longueur est comprise entre deux valeurs fixées. Puis nous avons déclarés comme positifs ceux comportant des motifs fixés. La taille de ces motifs varie de 4 à 6 caractères d'une base à l'autre. Concernant les mots négatifs, nous avons suivi deux stratégies. Pour certaines bases, les exemples négatifs sont ceux qui ne contiennent pas les motifs utilisés pour les positifs. Pour d'autres bases, nous avons utilisé un second jeu de motifs, et procédé comme pour les positifs (ce qui induit un chevauchement de classes). Les 7 bases artificielles générées ont moins de 3000 mots, sauf la dernière, qui en a 10000.

Dans le Tableau 5.1, nous présentons les résultats que nous obtenons sur ces bases avec RPNI* et notre approche boostée (BOOST après 200 itérations). Cette étude est faite pour 5%, 10%, 15% et 20% de bruit. Nous avons ajouté une dernière colonne (PERF) montrant le comportement d'ORABOOST lorsque l'oracle est parfait : cet oracle connait les exemples non bruités auxquels il attribue la confiance +1, et ceux qui sont bruités auxquels il attribue la confiance -1. L'objectif de cette colonne est d'établir le taux d'erreur optimal (mais inaccessible en pratique) d'ORABOOST pour chacune des bases. Pour des raisons de temps de calculs importants, liés à la nature même du boosting, nous n'avons pas effectué de procédure de validation croisée. Pour chacune des onzes bases, nous avons utilisé 80% des exemples en apprentissage et les 20% restant comme échantillon de test. Finalement, nous avons artificiellement introduit du bruit dans l'ensemble des onze échantillons d'apprentissage, en retournant l'étiquette d'un certain pourcentage de mots. L'estimation des performances en généralisation se fait sur l'échantillon, non bruité, de test.

Nous pouvons faire les remarques suivantes. D'abord, la comparaison entre la colonne RPNI* et PERF valide notre approche : le schéma de repondération que nous avons choisi fait chuter le taux d'erreur de RPNI* de 32.7% à 16.6% en moyenne sur tous les niveaux de bruit. Si on observe maintenant la colonne BOOST, il est clair que le fait d'avoir à simuler l'oracle avec un graphe de voisinage ne permet plus de réduire l'erreur de façon aussi drastique, mais de façon significative malgré tout (de 32.7% à 29.3% en moyenne sur tous les niveaux de bruit). La qualité de la simulation de l'oracle est donc essentiel : si celui-ci ne filtre pas correctement les données bruitées, celles-ci continuent de pénaliser, voire même à dégrader les performances de RPNI*.

Pour une analyse plus fine, considérons la Figure 5.5, qui présente l'erreur en généralisation de l'algorithme sur quatre bases caractéristiques (pour 5% de bruit). Concernant les bases AGARICUS et WF, on observe une baisse rapide de l'erreur ; ce sont des courbes typiques du comportement d'ADABOOST dans un cadre non bruité. Pour ces bases, ORABOOST est manifestement efficace, ce qui signifie *a posteriori* que l'oracle était correctement simulé. A

[9]http ://www.ics.uci.edu/~mlearn/MLRepository.html

		5%			10%		
Base	Taille	rpni*	boost	perf	rpni*	boost	perf
Agaricus	5644	7.2%	4.3%	0.0%	10.0%	6.9%	0.0%
TicTacToe	809	34.5%	28.3%	4.9%	40.7%	34.5%	10.5%
USF	1871	31.2%	26.1%	26.4%	33.0%	31.2%	31.7%
WF	1887	25.3%	21.4%	20.6%	31.2%	22.7%	26.1%
Base1	2540	41.5%	41.1%	35.6%	44.8%	43.7%	36.2%
Base2	1505	21.9%	19.9%	14.3%	13.6%	30.5%	12.3%
Base3	1532	27.0%	26.7%	13.0%	39.0%	12.7%	12.7%
Base4	2969	12.0%	10.0%	0.0%	29.2%	23.4%	0.0%
Base5	2179	36.7%	22.2%	19.2%	18.6%	25.0%	16.2%
Base6	2004	7.0%	13.0%	2.0%	42.9%	25.6%	1.2%
Base7	10000	39.7%	37.2%	39.7%	42.7%	43.7%	39.4%
Moyenne	2994.5	25.8%	22.7%	16.0%	31.4 %	27.3%	16.9%
		15%			20%		
Agaricus	5644	10.0%	6.1%	0.0%	18.2%	11.9%	0.0%
TicTacToe	809	40.7%	40.7%	9.8%	40.1%	38.8%	11.7%
USF	1871	32.0%	27.7%	27.7%	32.2%	30.6%	28.8%
WF	1887	36.2%	24.6%	24.6%	32.0%	31.7%	22.7%
Base1	2540	42.9%	42.3%	38.5%	45.7%	45.2%	35.2%
Base2	1505	39.8%	37.5%	10.6%	46.8%	45.1%	12.6%
Base3	1532	39.4%	35.5%	14.9%	49.5%	41.6%	20.5%
Base4	2969	16.0%	26.9%	0.0%	35.0%	30.4%	0.0%
Base5	2179	43.8%	31.4%	15.1%	39.4%	34.6%	9.4%
Base6	2004	42.9%	33.1%	1.7%	41.9%	39.2%	1.0%
Base7	10000	40.7%	43.2%	40.7%	44.3%	41.8%	40.5%
Moyenne	2994.5	34.9%	31.7%	16.7%	38.6%	35.5%	16.6%

Tab. 5.1 – Taux d'erreurs obtenus sur 11 bases avec 5, 10, 15 et 20% de bruit.

l'inverse, OraBoost fait du sur-apprentissage sur les bases Usf et Base7, ce qui se traduit par une stagnation voire une hausse du taux d'erreur en généralisation. Concernant Base7 en particulier, on notera que la qualité de graphe de voisinage qui simule l'oracle repose sur une distance qui dépend de bi-grammes entrainés sur les données d'apprentissage (voir Section 5.1.2) ; or compte tenu des critères que nous avons utilisés pour construire cette base (alphabet petit, mots relativements courts, motifs aléatoires 2 à 3 fois plus longs que le bi-gramme), on peut penser que notre façon de simuler l'oracle avait peu de chances de donner de bons résultats. Et la sanction, en termes d'erreur en généralisation, est immédiate.

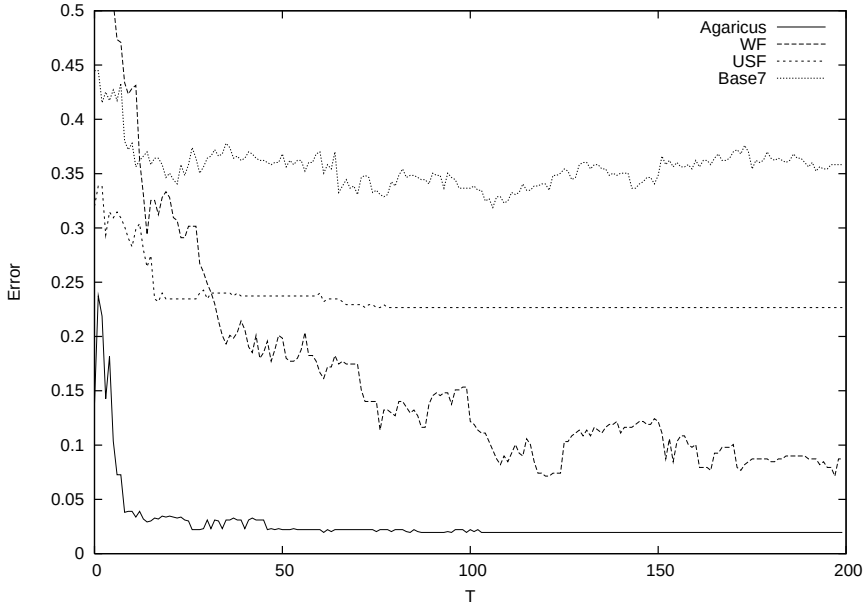


Fig. 5.5 – L'erreur en généralisation sur 4 bases caractéristiques, en fonction des itérations.

5.2.3 Rpni* comme co-apprenant faible

Indépendamment de la gestion des données bruitées, nous nous sommes posés une dernière question concernant l'utilisation pratique des algorithmes d'Inférence Grammaticale sur des problèmes d'Apprentissage Automatique. Le fait est qu'ils sont spécialisés sur des mots (ou plus généralement sur des structures discrètes), et si cette spécialisation peut être vue comme un gage d'efficacité, il n'en demeurre pas moins que les apprentistes préfèrent souvent utiliser des méthodes d'apprentissage génériques, en particulier des Svm.

Toute la théorie mathématique qui explique le bon fonctionnement des Svm se développe indépendamment du type de données qu'on peut chercher à classer, et cette abstraction est possible grâce à la notion de *noyau*, une fonction mathématique qui doit vérifier quelques propriétés mathématiques et algorithmiques. Ensuite, et de façon quasiment indépendante, toute une communauté de chercheurs s'est spécialisée sur le développement de noyaux aux divers types de données : des images, du sons, des données biologiques, et bien sûr des mots [KCM08]. Or si la généricité est un atout des Svm, elle élimine de fait tous les algorithmes spécialisés sur des types de données particuliers. Pourtant, les Svm n'ont pas que des qualités :

elles s'apparentent à des "boîtes noires", qu'un néophyte ne peut utiliser qu'en parcourant de façon hasardeuse une interminable liste de noyaux prédéfinis.

De façon alternative, nous avons voulu développer une procédure qui permettait de traiter des données complexes en combinant des algorithmes spécialisés. On peut typiquement penser à une base de données contenant des individus décrits pas leur prénom, leur taille et leur poids. L'objectif du problème d'apprentissage serait de déterminer le sexe des individus. Nous disons que ces données sont *hétérogènes*, au sens où chacune d'elle est décrite par des attributs de divers type : des mots pour les prénoms, des entiers pour les poids et les tailles.

Face à de telles données, on pourrait construire un arbre de décision sur les attributs numériques en omettant les prénoms, ou bien utiliser un algorithme d'Inférence Grammaticale pour apprendre les prénoms en omettant les mensurations. Mais cette stratégie d'omission décuple les erreurs de classification dues au chevauchement des classes. En effet, un individu de 185cm pesant 85kg est sans doute un homme, sauf s'il s'appelle Laure M. et qu'il a des prédispositions naturelles pour la natation. De même, en considérant l'exemple d'une certaine Dana A., chercheuse reconnue en Apprentissage Automatique et d'un certain Dana S., qui aurait posé les fondements de la Sémantique Dénotationelle, on ne peut pas déterminer leur sexe à partir de leur prénom, mais il en va tout autrement avec leurs mensurations.

Ce genre de données est fréquent dans la nature (mais plus rarement dans les bases du type Uci). Si on considère par exemple un site d'enchères en ligne comme `www.ebay.com`, chaque article est décrit par une image, une description textuelle, un prix de base, *etc.* Or si on cherche à construire un modèle d'utilisateur, permettant par exemple de lui présenter une gamme intéressante d'offres, tous ces attributs sont importants, bien qu'ils soient de type différents. De même, la base Biomet [GSBC$^+$03] contient des personnes décrites par leur photo, un échantillon de leur voix, leur empreinte digitale et leur signature. Si l'objectif du problème est de déterminer si une personne donnée est un faussaire ou non, alors tous ces attributs doivent être pris en compte, de nouveau.

On pourrait attaquer ces problèmes à l'aide d'un Svm en choisissant des noyaux spécialisés pour chaque type de données puis en les combinant[10]. Nous avons choisi une procédure différente : nous avons supposé disposer de k apprenants $\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_k$, chacun spécialisé sur un attribut, et développé une procédure de boosting qui les entraine tous, en "parallèle", à chaque itération. Puis nous utilisons l'ensemble des classifieurs produits pour repondérer les exemples, et construire l'hypothèse finale. La procédure qui en résulte s'appelle k-Boost [JSS09]. Nous l'étudions ci-après lorsque $k = 2$: c'est un cadre plus simple dans lequel nous pouvons faire des calculs exacts (ce qui n'est pas vrai pour un k quelconque).

L'algorithme 2-Boost

Soit $E = \{(x_1, y_1), \ldots (x_m, y_m)\}$ un échantillon d'apprentissage fixé. Chaque instance x_i appartient à un domaine $\mathcal{X}$ et sa classe est $y_i \in \{-1, +1\}$. Comme nous voulons modéliser la notion d'attributs hétérogènes, on suppose que $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$. Dans l'exemple précédent de la base d'individus, on aurait $\mathcal{X}_1 = \Sigma^*$ pour les prénoms, et $\mathcal{X}_2 = \mathbb{N} \times \mathbb{N}$ pour la taille et le poids. Puis on suppose disposer de deux algorithmes d'apprentissage $\mathfrak{F}_1$ et $\mathfrak{F}_2$ spécialisés sur $\mathcal{X}_1$ et $\mathcal{X}_2$. Toujours dans l'exemple précédent, $\mathfrak{F}_1$ peut faire voter deux bi-grammes (un par classe), alors que $\mathfrak{F}_2$ construit un arbre de décision avec l'algorithme C4.5. Maintenant,

[10] En effet, on peut par exemple montrer que si $\kappa_1(\cdot,\cdot)$ et $\kappa_2(\cdot,\cdot)$ sont des noyaux, alors $\alpha_1\kappa_1(\cdot,\cdot) + \alpha_2\kappa_2(\cdot,\cdot)$ est un noyau.

on fait collaborer ces deux algorithmes au travers de la procédure 2-Boost décrite dans l'Algorithme 15.

Algorithm 15: 2-Boost(E, T)

Input: Un échantillon $E = \{(x_1, y_1), (x_2, y_2), \dots (x_m, y_m)\} \subseteq \Sigma^* \times \{-1, +1\}$, deux apprenants faibles $\mathfrak{F}_1$ et $\mathfrak{F}_2$, un nombre maximum d'itération T

Output: Une hypothèse forte H_T

for $i = 1$ to m **do** $\nu_0(x_i) \leftarrow (1/m)$;
for $t = 0$ to T **do**
  $h_{1t} \leftarrow \mathfrak{F}_1(E, \nu_t)$;
  $h_{2t} \leftarrow \mathfrak{F}_2(E, \nu_t)$;
  $\gamma_{1t} \leftarrow \sum_{i=1}^{m} \nu_t(x_1) y_i h_{1t}(x_i)$;
  $\gamma_{2t} \leftarrow \sum_{i=1}^{m} \nu_t(x_1) y_i h_{2t}(x_i)$;
  $\delta_t \leftarrow (\sum_{i=1}^{m} \nu_t(x_1) h_{1t}(x_i) h_{2t}(x_i)) - \gamma_{1t}\gamma_{2t}$;
  $c_{1t} \leftarrow \frac{1}{4} \ln \frac{(\delta_t + (1+\gamma_{1t})(1+\gamma_{2t}))(-\delta_t + (1+\gamma_{1t})(1-\gamma_{2t}))}{(\delta_t + (1-\gamma_{1t})(1-\gamma_{2t}))(-\delta_t + (1-\gamma_{1t})(1+\gamma_{2t}))}$;
  $c_{2t} \leftarrow \frac{1}{4} \ln \frac{(\delta_t + (1+\gamma_{1t})(1+\gamma_{2t}))(-\delta_t + (1-\gamma_{1t})(1+\gamma_{2t}))}{(\delta_t + (1-\gamma_{1t})(1-\gamma_{2t}))(-\delta_t + (1+\gamma_{1t})(1-\gamma_{2t}))}$;
  $Z_t \leftarrow \sum_{i=1}^{m} \nu_t(x_i) \exp\left(-c_{1t} y_i h_{1t}(x_i) - c_{2t} y_i h_{2t}(x_i)\right)$;
  for $i = 1$ to m **do**
    $\nu_{t+1}(x_i) \leftarrow \nu_t(x_i) \exp\left(-c_{1t} y_i h_{1t}(x_i) - c_{2t} y_i h_{2t}(x_i)\right) / Z_t$;
return la fonction $H_T = \left(x \mapsto \text{signe}\left(\sum_{t=1}^{T} (c_{1t} h_{1t}(x) + c_{2t} h_{2t}(x))\right)\right)$

A chaque itération t de 2-Boost, une distribution ν_t pondère les exemples de E. Puis chaque apprenant $\mathfrak{F}_1$ et $\mathfrak{F}_2$ utilise sa propre vue des données (*i.e.*, les attributs sur lequels il est spécialisé) et la distribution ν_t pour produire des hypothèses h_{1t} et h_{2t}. Ces hypothèses sont ensuite combinées en une hypothèse pondérée $c_{1t}h_{1t} + c_{2t}h_{2t}$ dont la réponse *globale* est utilisée pour repondérer les exemples, en définissant ν_{t+1} à partir de ν_t. Toutes les hypothèses ainsi produites sont enfin combinées dans l'hypothèse finale H_T.

Sur les conditions d'utilisation de 2-Boost

L'étude des propriétés théoriques de 2-Boost suit un schéma très standard : on considère l'erreur empirique de H_T sur E, définie par $\varepsilon(H_T, E) = (1/m) \sum_{i=1}^{m} [\![H_T(x_i) \neq y_i]\!]$, puis on montre le résultat suivant :

Proposition 5 $\varepsilon(H_T, E) \leq \left(\prod_{t=1}^{T} Z_t\right)$,
avec $Z_t = \sum_{i=1}^{m} \nu_t(x_i) \exp\left(-c_{1t} y_i h_{1t}(x_i) - c_{2t} y_i h_{2t}(x_i)\right)$.

Minimiser l'erreur empirique consiste donc à calculer, à chaque itération, les coefficients c_{1t} et c_{2t} de sorte que Z_t soit minimal, donc les coefficients tels que $\left(\frac{\partial Z_t}{\partial c_{1t}}\right) = \left(\frac{\partial Z_t}{\partial c_{2t}}\right) = 0$.

Pour résoudre ces équations, nous sommes amenés à définir 3 paramètres. Les deux premiers, γ_{1t} et γ_{2t}, sont les crêtes des hypothèses h_{1t} et h_{2t} par rapport à E et ν_t. On retrouve donc les mêmes quantités que dans l'étude d'AdaBoost. Rappelons qu'elles dénotent l'espérance de la correction des réponses de h_{1t} et h_{2t} sur E : pour tout $j \in \{1, 2\}$,

$$\gamma_{jt} = \mathbb{E}_{(x,y)\sim\nu_t}\left[y h_{jt}(x)\right] = \sum_{i=1}^{m} \nu_t(x_i) y_i h_{jt}(x_i).$$

Le troisième paramètre, δ_t, dénote la covariance entre les deux variables aléatoires précédentes :

$$\delta_t = \mathbb{C}\mathrm{ov}_{(x,y)\sim\nu_t}\left[yh_{1t}(x), yh_{2t}(x)\right] = \left(\sum_{i=1}^{m} \nu_t(x_i)h_{1t}(x_i)h_{2t}(x_i)\right) - \gamma_{1t}\gamma_{2t}.$$

Intuitivement, ce paramètre est la masse de probabilité des exemples sur lesquels les hypothèses h_{1t} et h_{2t} ont le même avis. Son apparition dans la règle de repondération montre que les apprenants *collaborent* : la repondération d'un exemple tient bien sûr compte du fait que les hypothèses le classent correctement ou non, mais tient aussi compte de la qualité des hypothèses h_{1t} et h_{2t} (par le biais de γ_{1t} et γ_{2t}) et du degré d'accord entre ces hypothèses sur les données (par le biais de δ_t).

Une fois ces trois paramètres définis, on peut montrer que les coefficients c_{1t} et c_{2t} donnés dans l'Algorithme 15 sont optimaux : ce sont eux qui minimisent Z_t, donc l'erreur empirique $\varepsilon(H_T, E)$. Lorsque $\delta_t \simeq 0$, donc que les hypothèses sont totalement indépendantes, alors un rapide calcul prouve que $c_{1t} \simeq \frac{1}{2}\ln\frac{1+\gamma_{1t}}{1-\gamma_{1t}}$ et $c_{2t} \simeq \frac{1}{2}\ln\frac{1+\gamma_{2t}}{1-\gamma_{2t}}$. Dans ce cas, on retrouve donc les mêmes coefficients qu'AdaBoost ; tout se passe comme si on utilisait AdaBoost sur $\mathfrak{F}_1$ et $\mathfrak{F}_2$ en parallèle, chacun contribuant indépendamment de l'autre à la repondération des exemples. A l'inverse, s'il y a une forte corrélation (positive) entre h_{1t} et h_{2t}, alors $\gamma_{1t} \simeq \gamma_{2t} \simeq \frac{\gamma_{1t}+\gamma_{2t}}{2}$, et $\delta_t \simeq 1 - \left(\frac{\gamma_{1t}+\gamma_{2t}}{2}\right)^2$, et $c_{1t} \simeq c_{2t} \simeq \frac{1}{4}\ln\frac{1+\left(\frac{\gamma_{1t}+\gamma_{2t}}{2}\right)}{1-\left(\frac{\gamma_{1t}+\gamma_{2t}}{2}\right)}$; tout se passe donc comme si un unique apprenant, $\mathfrak{F}_1$ ou $\mathfrak{F}_2$, était entrainé avec AdaBoost. Enfin, dans un cas où la corrélation entre h_{1t} et h_{2t} est moyenne, nous n'avons affaire ni à deux classifieurs indépendants, ni à deux classifieurs identiques, mais à une sorte de classifieur résultant de la "fusion" de $\mathfrak{F}_1$ et $\mathfrak{F}_2$.

Une fois calculés les coefficients c_{1t} et c_{2t}, on peut montrer le résultat suivant :

Théorème 26 ([JSS09]) *Si $\mathfrak{F}_1$ est un apprenant Γ_1-faible et $\mathfrak{F}_2$ est un apprenant Γ_2-faible sur E, au sens d'*AdaBoost*, alors $\varepsilon(H_T, E) < \exp\left(-(T+1)\max(\Gamma_1^2, \Gamma_2^2)/2\right)$. Donc l'erreur empirique de* 2-Boost *converge exponentiellement vers 0 avec le nombre T d'itérations.*

De façon étonnante, la covariance δ_t des hypothèses produites par 2-Boost n'intervient pas dans ce théorème. En fait, ce n'est vrai que pour l'erreur empirique. En effet, concernant l'erreur en généralisation, on a :

Théorème 27 ([JSS09]) *Soient $\mathfrak{F}_1$ et $\mathfrak{F}_2$ deux algorithmes retournant des hypothèses prises dans des classes $\mathcal{G}_1$ et $\mathcal{G}_2$, et μ une distribution sur $\Sigma^* \times \{-1, +1\}$. Etant donné un échantillon d'apprentissage $E = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ tiré aléatoirement et indépendamment selon μ, pour tous $\delta > 0, \theta > 0, \Gamma_0 > 0$, si $\mathfrak{F}_1$ et $\mathfrak{F}_2$ retournent des hypothèses Γ_0-faibles sur E, alors avec probabilité au moins $1 - \delta$, l'erreur en généralisation de l'hypothèse H_T produite par* 2-Boost *est plus petite que*

$$\varepsilon^{\theta}(f_T, E) + \mathcal{O}\left(\frac{1}{\theta}\sqrt{\frac{\min\{d_{\mathcal{G}_1}, d_{\mathcal{G}_2}\}}{m}}\right) + \mathcal{O}\left(\sqrt{\frac{\log(1/\delta)}{m}}\right).$$

Les deux derniers termes sont des termes de pénalisation dépendant, entre autres, des VC-dimensions de $\mathcal{G}_1$ et $\mathcal{G}_2$. Le premier terme est une fonction dépendant de γ_{1t}, γ_{2t}, δ_t et θ qui, si $\theta < \Gamma_0/2$, et si δ_t n'est pas fortement négative, décroit exponentiellement vers 0 lorsque $T \to +\infty$.

Nous ne donnons pas plus de détails sur ce résultat théorique (voir [JSS09]).

Quelques résultats expérimentaux avec 2-Boost

Comme pour les algorithmes précédents, nous souhaitons ici reprendre une des expérimentations qui nous a permis d'étudier 2-Boost. Le lecteur est invité à consulter [JSS09] pour plus d'informations. Dans celle que nous présentons ici, nous étudions les performances en généralisation de 2-Boost sur une base artificielle dont les données sont hétérogènes. De cette manière, nous maitrisons l'ensemble des caractéristiques de la base.

Pour cela, nous sommes partis d'une première base contenant 1877 prénoms et le sexe associé. Compte tenu du chevauchement des classes (-1 pour les hommes, +1 pour les femmes), ces prénoms sont clairement insuffisants pour permettre de prédire correctement le sexe d'un individu. Puis nous avons ajouté un attribut "sport favori" qui pouvait être soit l'aqua-gym, soit le football, soit le tennis, et réparti aléatoirement ces valeurs sur l'ensemble de la base des prénoms, à l'aide d'une distribution telle que l'aqua-gym soit majoritairement pratiquée par les femmes, le football par les hommes, et le tennis par les deux. De nouveau, cet attribut seul ne permet pas discriminer les hommes et les femmes.

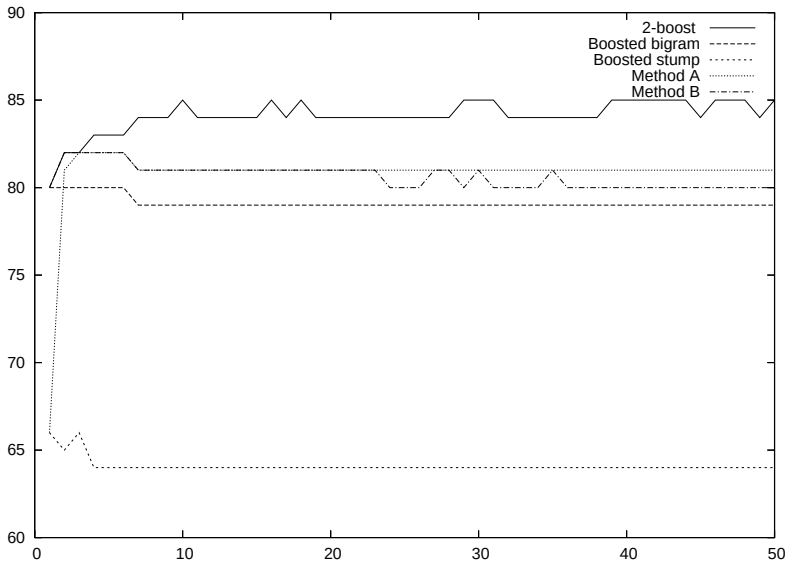
Nous avons ensuite cherché à construire un classifieur prédisant le sexe (-1 ou +1) d'un individu en fonction de son prénom et de son sport favori. Pour cela, on considère deux apprenants. Pour l'attribut "prénom", on fait voter deux bi-grammes lissés, un pour chaque classe, qu'on entraine à partir de mots pondérés (pour pouvoir ensuite l'utiliser dans le cadre du boosting, en tenant compte de la pondération ν sur les exemples). Et pour l'attribut "sport favori", on choisit un *stump*, c'est-à-dire un arbre de décision sur 1 niveau.

Puis pour étudier l'intérêt du schéma de repondération de 2-Boost, nous l'avons comparé aux deux techniques suivantes de combinaison des hypothèses :

Méthode A : Chaque apprenant est boosté individuellement avec AdaBoost. On appelle ensuite $f_T(x) = (\sum_{t=0}^{T} c_t h_t(x))/(\sum_{t=0}^{T} c_t)$ et $f'_T(x) = (\sum_{t=0}^{T} c'_t h'_t(x))/(\sum_{t=0}^{T} c'_t)$ les deux classifieurs résultants. La Méthode A consiste à renvoyer le signe de $f_T(x) + f'_T(x)$.

Méthode B : On suit le même schéma que la Méthode A, sauf au moment du vote final : on renvoie le signe de $(\sum_{t=0}^{T} c_t) f_T(x) + (\sum_{t=0}^{T} c'_t) f'_T(x))$.

Dans la Figure 5.6, la courbe présente le taux de succès en généralisation (estimé par 5-cross-validation) sur 50 itérations (1) de 2-Boost, (2) de chaque apprenant boosté individuellement sur son attribut de spécialité et (3) des classifieurs combinés avec la Méthode A et la Méthode B ; le tableau, lui, présente les moyennes des taux de succès en apprentissage et des taux de succès en généralisation sur les 50 itérations. On observe clairement que 2-Boost bat les Méthodes A et B, qui elles-mêmes battent AdaBoost lorsqu'on l'utilise indépendamment sur chaque attribut. C'est le cas aussi bien pour les taux de succès en apprentissage qu'en généralisation. Il est donc clair, sur cette petite expérience, que le schéma collaboratif de repondération qu'utilise 2-Boost se traduit *in fine* par de meilleures performances.



Algorithme	Succès en apprentissage	Succès en généralisation
Boosted stump	66.11	64.10
Boosted bigram	90.20	79.22
Méthode A	90.26	81.57
Méthode B	92.12	80.87
2-Boost	97.73	85.10

Fig. 5.6 – La courbe représente l'évolution sur 50 itérations du taux de succès en généralisation des divers techniques étudiées. La table montre les résultats moyens sur 50 itérations de ces méthodes en termes de taux de succès en apprentissage et en généralisation.

Chapitre 6

Conclusion et perspectives

Dans ce mémoire, nous avons tenté de défendre, à la lumière de nos travaux, la thèse selon laquelle l'Inférence Grammaticale est une discipline à part entière, émargeant au pays des Apprentissages Artificiels. Dans sa version stochastique, elle est indifférenciable d'une discipline comme la Régression, puisqu'elles s'attaquent au même problème, qu'elles ont le même objectif et des méthodes très similaires.

Pourtant, dans sa version classique, l'Inférence Grammaticale a également des objectifs propres, que d'autres disciplines proches comme la Classification Supervisée ne permettent pas d'atteindre. C'est alors la définition même de ce qu'apprendre veut dire qui est originale, et pertinente eu égard aux applications visées. Inférence Grammaticale et Classification Supervisée finissent même par exploiter de façon radicalement différente des idées-clés identiques, comme celle d'utiliser les compétences d'un expert en apprentissage actif.

Pourtant, je ne crois pas que l'Inférence Grammaticale gagnerait à se séparer de ses consœurs. Au contraire, sur un problème d'apprentissage à partir de séquences bruitées par exemple, il est intéressant d'exploiter conjointement certaines techniques développées en Inférence Grammaticale et d'autres en Classification Supervisée, comme nous l'avons vu dans le Chapitre 5.

Dit autrement, les interfaces de l'Inférence Grammaticale sont tout aussi intéressantes à explorer que le cœur de cette discipline.

Poursuivre en Inférence Grammaticale

L'Inférence Grammaticale, en temps que discipline autonome, voit régulièrement naître en son sein de nouvelles thématiques (comme l'inférence des automates temporisés [VdWW09]) et de nouvelles questions (listées dans [dlH06b] par exemple). Nous-mêmes en avons naturellement soulevé plusieurs dans ce manuscrit, qu'il sera intéressant de reprendre et d'approfondir.

De façon plus précise, j'ai d'abord envie de poursuivre mes travaux dans le cadre de l'identification à partir de requêtes. En effet, d'une part, les requêtes de correction (présentées dans le Chapitre 4) sont toujours porteuses. On pourrait par exemple les utiliser pour apprendre des grammaires hors-contextes "lipschitziennes", telles que deux mots proches au sens de la distance de Levenshtein aient des arbres de dérivation proches, au sens d'une distance d'édition d'arbres. En outre, il semble possible de combiner les techniques de synthèse de requêtes avec celles de sélection de données pour obtenir de nouveaux résultats en identification approximative. Enfin, je voudrais étudier de plus près les relations entre identification à

partir de requêtes et apprentissage par renforcement ; le projet ANR LAMPADA est un cadre formel idéal pour cela.

Par ailleurs, l'identification à partir de requêtes a des applications remarquables dans le domaine du *model checking*. Ce n'est pas un hasard si les vainqueurs de la compétition ZULU que nous avons organisée émargent dans ce domaine : sur le problème que nous avons soumis, il est manifeste qu'ils ont une expertise au moins équivalente à celle des spécialistes dans la communauté Inférence Grammaticale. Nous devrions donc gagner à les rencontrer et découvrir plus avant leurs problématiques. Un point de discussion possible concerne la terminaison de leurs algorithmes : les automates qu'ils cherchent à apprendre peuvent être extrêmement gros, et ils arrêtent donc "brutalement" le processus d'apprentissage quand ils estiment qu'ils ont suffisamment d'états. Or je pense que d'un point de vue formel, il y a probablement des critères plus objectifs qui pourraient être envisagés. De plus, ils infèrent des AFD pour ensuite prouver des propriétés dont ils connaissent *a priori* les caractéristiques (des propriétés de *sureté* par exemple). Pourquoi ne pas chercher à exploiter cette information pendant la phase d'apprentissage ?

J'ai également envie d'investir dans le domaine de l'Inférence Grammaticale stochastique. En effet, à courte échéance, l'identification des automates stochastiques va revenir sur le devant de la scène. C'est un domaine qui a foisonné ces dernières années. Sur un plan plutôt théorique, certains résultats de PAC-apprenabilité sur les automates stochastiques déterministes [CT04] ont récemment débouché sur de "vrais" algorithmes [CG08]. De plus, le problème de l'identification des automates à multiplicité a lui-même conduit à un algorithme d'apprentissage, DEES, qui paraît extrêmement puissant [DEH06]. Enfin, de très nombreux travaux ont été mené sur l'apprentissage des modèles de Markov, mais ils n'ont pas vraiment diffusé dans la communauté ICGI. Ainsi, dans ce domaine, il est temps de tirer les marrons du feu, de faire un bilan, une synthèse.

La prochaine compétition organisée par Sicco Verwer lors d'ICGI'12 portera justement sur ce problème. Je voudrais y participer en partant sur une technique basée sur des solveurs de contraintes. Celle-ci vient d'apparaître pour aborder le problème de l'identification des AFD [HV10]. Or non seulement elle semble en passe de détrôner tous les algorithmes classiques comme EDSM [LPP98], ce qui se vérifiera (peut-être) au moment de la compétition STAMINA en décembre 2010, mais c'est également une technique qui semble adaptable aux automates stochastiques. Une bonne idée consisterait donc à entamer une collaboration avec des spécialistes en contraintes (Christine Solnon, du LIRIS ?) et de jouer, lorsque la compétition sera effectivement lancée.

Partir sur un problème neuf

Comme dans beaucoup de disciplines, il existe des problèmes d'Inférence Grammaticale qui n'ont jamais été visités, souvent parce qu'ils sont considérés comme extrêmement difficiles. L'Inférence des grammaires de graphes est l'un d'entre eux.

Pour illustrer l'intérêt d'un tel problème, il faut préciser qu'à partir de 2007, l'EURISE a fusionné avec un laboratoire de Physique pour former une nouvelle UMR CNRS, le laboratoire Hubert Curien. Par la suite, nous avons commencé à travailler avec des spécialistes en Image sur des projets communs. En particulier, nous avons "décroché" ensemble le projet ANR SATTIC que je coordonne et qui finance la thèse d'Emilie Samuel. Or par effet de bord, nous avons récemment trouvé des résultats sur les problèmes d'isomorphisme de graphes plans, en collaborant avec Christine Solnon et Guillaume Damiand du LIRIS.

En effet, souhaitant travailler sur des données structurées plutôt que des matrices de pixels, nous avons utilisé et développé des techniques qui permettent d'extraire des graphes à partir d'images [SdlHJ10]. Nous en donnons un exemple dans la Figure 6.1, obtenu grâce à l'API qu'a développée Emilie[1]. Ces graphes sont loin d'être quelconques : non seulement ils sont planaires, mais ils sont également dessinés dans le plan, et ils ont une face infinie, ce qui leurs confèrent le statut de *plane graphs* [Fus07]. Or pour cette classe de graphes, nous avons montré dans [DdlHJ$^+$09b, DdlHJ$^+$09a] que les problèmes d'isomorphisme et de sous-isomorphisme étaient polynomiaux (à quelques conditions près). C'est remarquable car pour des graphes quelconques, ces problèmes sont considérés comme difficiles.

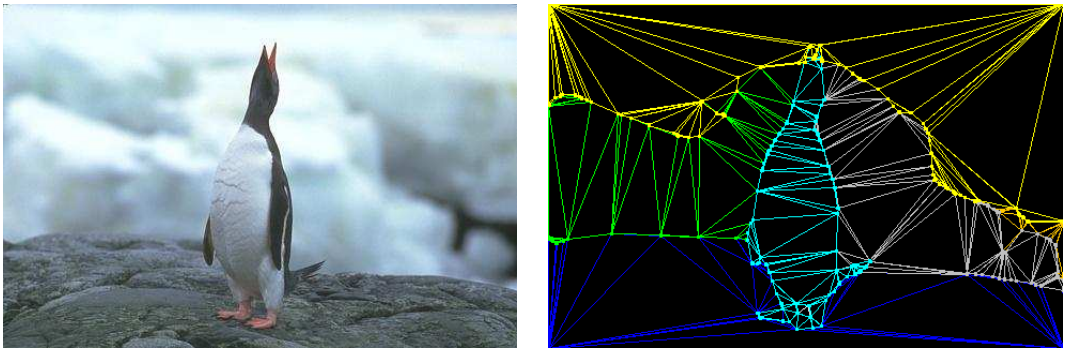


FIG. 6.1 – Un exemple de graphe plan extrait d'une image avec l'algorithme EPG [SdlHJ10].

Toutefois, en Traitement d'Images, ces résultats n'ont pas d'applications immédiate : grâce à nos algorithmes, nous sommes en mesure de retrouver une image dans une base à partir d'un motif qu'on aurait extrait de cette image cible. Cela peut éventuellement avoir un intérêt si cette base contient énormément d'images très ressemblantes, comme c'est sans doute le cas dans un contexte où des douaniers chercheraient à repérer des articles contrefaits. Cependant, si l'on voulait poursuivre dans cette voie pour attaquer des problèmes de classification d'images, alors il faudrait disposer de résultats beaucoup plus forts sur la polynomialité du calcul d'une distance d'édition de graphes plans par exemple, qui permettrait ensuite d'utiliser des techniques de classification par plus proches voisins. Or il y a tout à parier que le calcul d'une telle distance est $\mathcal{NP}$-difficile pour les graphes plans, comme il l'est pour les graphes quelconques.

D'un autre côté, disposer d'un algorithme polynomial de détection de motifs dans une image nous ouvre la porte d'une autre technique de classification prometteuse, basée sur les *grammaires (stochastiques) de graphes*. Disons deux mots de ces grammaires. D'abord, ce sont des objets qui sont étudiés depuis longtemps en Informatique "fondamentale" [ET96, RE97]. Elles ont des applications en Génie Logiciel par exemple, pour modéliser et faciliter les processus de développement. Elles ont également un intérêt en Bioinformatique, dès qu'on s'intéresse à la structure secondaire de molécules comme l'ARN. Elles ont encore servi pour modéliser des problèmes de concurrence en Programmation.

Sur le plan syntaxique, il existe principalement deux familles de grammaires (si l'on omet toutes les approches catégorielles) : les *node-replacement grammars* [ER90] et les *hyperedge-replacement grammars* [Hab92]. Or toutes deux posent des problèmes algorithmiques com-

[1] `http://labh-curien.univ-st-etienne.fr/EPG/`

plexes : non seulement, l'équivalence de grammaires est indécidable, mais le *parsing* des graphes est $\mathcal{NP}$-difficile[2]. C'est pourquoi peu de travaux ont porté sur leur identification en Inférence Grammaticale : il existe quelques articles pionniers [JK91], et les équipes de José Sempere à Valencià et Tim Oates à Baltimore commencent à les considérer en lien avec des problèmes de Bioinformatique.

Or travailler sur des graphes plans nous permet d'envisager un nouveau type de grammaires de graphes où l'on ne replacerait pas des sommets ni des (hyper-)arêtes, mais des faces. Il est clair qu'à ce stade, le mécanisme même de remplacement d'une face par un graphe, primordial pour utiliser de telles grammaires, reste à préciser. De plus, nous n'avons aucune idée des langages de graphes que ces grammaires pourraient engendrer. Néanmoins, il y a des raisons d'espérer qu'un algorithme de *parsing* polynomial puisse exister. De plus, la propriété de *substituabilité*, introduite par Alexander Clark dans ses travaux sur l'identification des grammaires hors-contextes [CE05], devrait nous aider à identifier de telles grammaires. Bref, d'un point de vue Inférence Grammaticale, il y a là des verrous qui doivent pouvoir sauter.

Sur le plan des applications, maintenant, on peut en évoquer deux. D'une part, l'extension d'un algorithme comme SEQUITUR [NMW97] aux grammaires de graphes plans permettrait sans doute de développer un nouvel algorithme de compression de graphes. D'autre part, en classification d'images, si l'on sait apprendre des grammaires stochastiques de graphes, une par classe, alors on peut attribuer à un nouveau graphe la classe du modèle qui maximise sa probabilité. Enfin, toutes nos expérimentations pourront utiliser des images comme données, dans la mesure où nous disposons déjà des outils pour extraire des graphes plans.

Explorer les frontières de l'Inférence Grammaticale

Comme nous l'avons vu dans le Chapitre 5, l'Inférence Grammaticale gagne en collaborant avec les autres disciplines en Apprentissage Artificiel. Il nous parait donc intéressant de terminer l'exposé de nos perspectives en abordant de nouveau ce point.

Beaucoup de nos travaux antérieurs ont porté sur les boules de mots : elles posent des problèmes combinatoires difficiles – que nous avons présentés dans le Chapitre 2, mais nous ont permis de comparer les divers paradigmes d'identification, et d'obtenir de nouveau résultats en identification à partir de requêtes et en identification à partir de données bruitées. Pourtant, nous avons échoué à exploiter les outils d'une véritable géométrie sur l'ensemble des mots : considérer des boules revient à supposer qu'on dispose d'un compas, mais pas d'une règle. Dit autrement, nous avons exploité les propriétés d'un espace métrique, alors qu'un espace affine aurait été plus intéressant. Au contraire, en Classification à partir de données numériques, on utilise aussi bien un compas (dans les techniques de classification par plus proches voisins, ou de classification non supervisée) qu'une règle (dès qu'on travaille avec des hyperplans séparateurs, des arbres de décision, un perceptron, ou un SVM).

Or en reprenant les idées des SVM, de nouvelles possibilités s'offrent à nous pour travailler avec des propriétés géométriques sur les mots. En effet, munissons Σ^* d'un noyau $\kappa(\cdot,\cdot)$. Alors on peut définir l'*hyperplan normal au mot* $n \in \Sigma^*$ comme étant l'ensemble $\{w : \kappa(n,w) = 0\}$. De plus, en posant $d_\kappa(u,v) = \sqrt{\kappa(u,u) + \kappa(v,v) - 2\kappa(u,v)}$, *i.e.*, en considérant la distance euclidienne sur l'espace des projections, alors on peut définir la *boule de centre* $o \in \Sigma^*$ *et de rayon* $r \geq 0$ comme étant l'ensemble $\{w : d_\kappa(o,w) \leq r\}$. En somme, nous avons maintenant une règle et un compas pour travailler avec des mots.

[2]En effet, reconnaître un motif dans un graphe quelconque est un problème $\mathcal{NP}$-complet [GJ79].

Et les questions qui s'ensuivent en Inférence Grammaticale ne manque pas : peut-on PAC-apprendre de telles boules ? est-il facile d'identifier ces boules avec des requêtes de correction ? peut-on exploiter à la fois la règle et le compas ? la notion de vecteur de support et celle de marge s'interprètent-elles dans l'espace des mots ? Autant de questions qui donnent envie de prendre une nouvelle feuille et un crayon pour commencer à griffonner ...

Annexe A

Quelques résultats non publiés

A.1 Minimalité de l'automate $A_r(o)$ pour les boules Lcs

Dans cette section, on montre le Théorème 4, stipulant la minimalité d'$A_r(o)$ lorsque nous travaillons avec la distance Lcs, un alphabet Σ contenant une lettre spéciale $\diamondsuit$, et une boule Lcs $B_r(o)$ telle que $o = c_1c_2 \dots c_n \in (\Sigma \setminus \{\diamondsuit\})^*$ et $r \geq 0$. Notons qu'ajouter une lettre joker à l'alphabet n'a généralement aucune conséquence dans la plupart des applications. Nous rappelons par ailleurs qu'un tuple $t_p = (x_0, x_1, \dots, x_n)$ est associé à chaque mot $p \in \Sigma^*$ avec $x_i = \min\left(d_\wedge(p, o[1..i]), r+1\right)$ pour tout $i \in 0, 1, \dots n$.

Pour démontrer ce théorème, il suffit de montrer que chaque état t_p est associé à un unique résiduel de $B_r(o)$, ce qu'établit le Théorème suivant :

Théorème 28 *Pour tout $p, q \in \Sigma^*$, $p^{-1}B_r(o) = q^{-1}B_r(o)$ si et seulement si $t_p = t_q$.*

Preuve du sens ($\Leftarrow$).

Ce sens est vrai quelque soit la distance d'édition choisi. Une première preuve est donnée par le Théorème 3 stipulant qu'$A_r(o)$ est déterministe. Le lemme suivant est une preuve directe, obtenue en dévoilant le lien entre les résiduels et les tuples t_p :

Lemme 9 *Considérons une $d_\mathcal{C}$-boule quelconque $B_r(o)$ et un tuple $t_p = (x_0, x_1, \dots, x_n)$ pour un certain $p \in \Sigma^*$. Alors*

$$p^{-1}B_r(o) = \bigcup_{\substack{0 \leq i \leq n \\ x_i \leq r}} B_{r-x_i}(o[i+1..n]).$$

En conséquence, pour tout $p, q \in \Sigma^*$, si $t_p = t_q$ alors $p^{-1}B_r(o) = q^{-1}B_r(o)$.

Pour le démontrer, on commence par établir la propriété suivante :

Proposition 6 *Soient $u, v \in \Sigma^*$ et supposons que $u = u_1u_2$ pour des chaînes $u_1, u_2 \in \Sigma^*$. Alors il existe $v_1, v_2 \in \Sigma^*$ tels que $v = v_1v_2$ et $d_\mathcal{C}(u, v) = d_\mathcal{C}(u_1, v_1) + d_\mathcal{C}(u_2, v_2)$.*

Preuve: Soit $S = (a_1, b_1)(a_2, b_2) \dots (a_s, b_s)$ n'importe quel alignement optimal de u et v, et soit $i_0 \in \{0, 1, 2, \dots s\}$ un indice tel que $u_1 = a_1a_2 \dots a_{i_0}$ (et donc $u_2 = a_{i_0+1} \dots a_s$). On définit $v_1 = b_1b_2 \dots b_{i_0}$ et $v_2 = b_{i_0+1} \dots b_s$ et $S_1 = (a_1, b_1)(a_2, b_2) \dots (a_{i_0}, b_{i_0})$ et

$S_2 = (a_{i_0+1}, b_{i_0+1}) \dots (a_s, b_s)$. Clairement, $v = v_1 v_2$ et $\mathcal{C}(S) = \mathcal{C}(S_1) + \mathcal{C}(S_2) = d_{\mathcal{C}}(u, v)$. De plus, S_1 est un alignement optimal de u_1 et v_1, puisque l'existence d'un alignement S_1' de coût plus faible nous permettrait de construire un nouvel alignement $S' = S_1' S_2$ de u et v tel que $\mathcal{C}(S') < d_{\mathcal{C}}(u, v)$. Par conséquent, $\mathcal{C}(S_1) = d_{\mathcal{C}}(u_1, v_1)$ et $\mathcal{C}(S_2) = d_{\mathcal{C}}(u_2, v_2)$ pour la même raison, donc $d_{\mathcal{C}}(u, v) = d_{\mathcal{C}}(u_1, v_1) + d_{\mathcal{C}}(u_2, v_2)$. □

Preuve: *(du Lemme 9)* ($\subseteq$) Soit $w \in p^{-1}B_r(o)$, donc $d_{\mathcal{C}}(pw, o) \leq r$. Par la Proposition 6, il existe un indice i tel que $d_{\mathcal{C}}(pw, o) = d_{\mathcal{C}}(p, o[1..i]) + d_{\mathcal{C}}(w, o[i+1..n])$. Comme $d_{\mathcal{C}}(pw, o) \leq r$, on en déduit que $d_{\mathcal{C}}(p, o[1..i]) \leq r$, donc $x_i = d_{\mathcal{C}}(p, o[1..i]) \leq r$. De plus, $d_{\mathcal{C}}(w, o[i+1..n]) \leq r - x_i$, c'est-à-dire, $w \in B_{r-x_i}(o[i+1..n])$. ($\supseteq$) Soit $w \in B_{r-x_i}(o[i+1..n])$ pour un certain indice $i \in 0, 1, \dots n$, tel que $x_i \leq r$. Alors $d_{\mathcal{C}}(w, o[i+1..n]) \leq r - d_{\mathcal{C}}(p, o[1..i])$. Aussi, par l'inégalité triangulaire, on a $d_{\mathcal{C}}(pw, o) \leq d_{\mathcal{C}}(pw, po[i+1..n]) + d_{\mathcal{C}}(po[i+1..n], o) = d_{\mathcal{C}}(w, o[i+1..n]) + d_{\mathcal{C}}(p, o[1..i]) \leq r$. Donc $w \in p^{-1}B_r(o)$. □

Une des raisons pour lesquelles l'automate $A_r(o)$ n'est pas minimal dans le cas général vient du fait que l'union de boules décrivant un résiduel n'est pas unique pour les boules quelconques. Ainsi, considérons la boule de Levenshtein $B_3(0110)$ sur l'alphabet $\Sigma = \{0, 1, 2\}$. Il est facile de vérifier que $t_{00} = (2, 1, 1, 2, 2)$ et $t_{02} = (2, 1, 1, 2, 3)$, donc $t_{00} \neq t_{02}$. Pourtant, le Lemme 9 amène $(00)^{-1}B_3(0110) = B_1(0110) \cup B_2(110) \cup B_2(10) \cup B_1(0) \cup B_1(\lambda)$ et $(02)^{-1}B_3(0110) = B_1(0110) \cup B_2(110) \cup B_2(10) \cup B_1(0) \cup B_0(\lambda)$, donc $(00)^{-1}B_3(0110) = (02)^{-1}B_3(0110)$, puisque $B_1(0) \cup B_1(\lambda) = B_1(0) \cup B_0(\lambda)$. En conséquence, les états t_{00} et t_{02} sont distincts dans $A_3(0110)$ alors qu'ils supportent le même résiduel.

Dans le cas de boules Lcs, ce contre-exemple est impossible, comme nous allons le voir maintenant.

Preuve du sens ($\Rightarrow$).

Il s'agit de montrer que si $p^{-1}B_r(o) = q^{-1}B_r(o)$, alors $t_p = t_q$. Nous procédons par induction sur les composantes des tuples. Pour le cas de base, nous commençons par établir une Proposition vraie dans le cas d'une distance d'édition quelconque, puis le Lemme 10.

Proposition 7 *Supposons que $p^{-1}B_r(o) = q^{-1}B_r(o)$ pour $p, q \in \Sigma^*$. Alors, pour tout $w \in \Sigma^*$, pour tout $0 \leq i \leq n$,*

$$d_{\mathcal{C}}(pw, o[1..i]) \leq r \text{ ssi } d_{\mathcal{C}}(qw, o[1..i]) \leq r. \tag{A.1}$$

En particulier, si $w = \lambda$, alors pour tout $0 \leq i \leq n$,

$$d_{\mathcal{C}}(p, o[1..i]) \leq r \text{ ssi } d_{\mathcal{C}}(q, o[1..i]) \leq r. \tag{A.2}$$

Preuve: Par définition, si $p^{-1}B_r(o) = q^{-1}B_r(o)$, alors pour tout $z \in \Sigma^*$, $d_{\mathcal{C}}(pz, o) \leq r$ ssi $d_{\mathcal{C}}(qz, o) \leq r$. En remplaçant z par $wo[i+1..n]$ dans cette relation, on obtient le résultat (puisque $\mathcal{C}(a, a) = 0$ pour tout $a \in \Sigma$). □

Lemme 10 *Soient $p, q \in \Sigma^*$ tels que $p^{-1}B_r(o) = q^{-1}B_r(o)$. Si $d_\wedge(p, \lambda) \leq r$, alors $d_\wedge(p, \lambda) = d_\wedge(q, \lambda)$. Autrement dit, si $|p| \leq r$, alors $|p| = |q|$.*

Preuve: Si $i = 0$, alors l'équivalence (A.1) dit que pour tout $w \in \Sigma^*$, $(d_\wedge(pw, \lambda) \leq r$ ssi $d_\wedge(qw, \lambda) \leq r)$, c'est-à-dire, $(|p| + |w| \leq r$ ssi $|q| + |w| \leq r)$. Donc on en déduit que $(|p| \leq k$ ssi $|q| \leq k)$ pour tout $0 \leq k \leq r$. Supposons que $|p| \leq r$. D'une part, on pose $k = |p|$ et on obtient $|q| \leq k = |p|$. D'autre part, l'équivalence (A.2) implique que $|q| \leq r$, donc on pose $k = |q|$ et on en déduit que $|p| \leq k = |q|$. Ainsi, $|p| = |q|$. □

De même, les deux cas d'induction établis par les Lemmes 11 et 12 nécessitent la Proposition suivante. Notons que la lettre joker $\diamondsuit$ ne joue un rôle que dans le dernier Lemme. La question de la nécessité de cette lettre est ouverte[1].

Proposition 8 *Soient $u, v \in \Sigma^*$ et $a \in \Sigma$. Alors soit $d_\wedge(ua, v) = d_\wedge(u, v) - 1$, soit $d_\wedge(ua, v) = d_\wedge(u, v) + 1$.*

Preuve: Soit $w \in \text{Lcs}(ua, v)$. Si $w \in \text{Lcs}(u, v)$, alors $\ell(ua, v) = \ell(u, v)$, donc $d_\wedge(ua, v) = d_\wedge(u, v) + 1$. Lorsque $w \notin \text{Lcs}(u, v)$, alors on a forcément $w = w'a$ avec $w' \in u \wedge v$. Si $|w'| = \ell(u, v)$, alors $\ell(ua, v) = \ell(u, v) + 1$, donc $d_\wedge(ua, v) = d_\wedge(u, v) - 1$. Sinon $|w'| = \ell(u, v) - 1$, donc $\ell(ua, v) = \ell(u, v)$, et donc $d_\wedge(ua, v) = d_\wedge(u, v) + 1$. □

Lemme 11 *Soient $p, q \in \Sigma^*$ tels que $p^{-1}B_r(o) = q^{-1}B_r(o)$, et supposons que $d_\wedge(p, o[1..i]) > r$ pour un certain $0 \leq i < n$. Si $d_\wedge(p, o[1..i+1]) \leq r$, alors $d_\wedge(p, o[1..i+1]) = d_\wedge(q, o[1..i+1])$.*

Preuve: Supposons que $d_\wedge(p, o[1..i]) > r$ et $d_\wedge(p, o[1..i+1]) \leq r$. Par l'équivalence (A.2), on en déduit que $d_\wedge(q, o[1..i]) > r$ et $d_\wedge(q, o[1..i+1]) \leq r$. Alors la Proposition 8 amène d'une part $d_\wedge(p, o[1..i]) = r + 1$ et $d_\wedge(p, o[1..i+1]) = r$, et d'autre part $d_\wedge(q, o[1..i]) = r + 1$ et $d_\wedge(q, o[1..i+1]) = r$. Donc $d_\wedge(p, o[1..i+1]) = d_\wedge(q, o[1..i+1]) = r$. □

Lemme 12 *Soient $p, q \in \Sigma^*$ tels que $p^{-1}B_r(o) = q^{-1}B_r(o)$, et supposons que $d_\wedge(p, o[1..i]) = d_\wedge(q, o[1..i])$ pour un certain $0 \leq i < n$. Si $d_\wedge(p, o[1..i+1]) \leq r$, alors $d_\wedge(p, o[1..i+1]) = d_\wedge(q, o[1..i+1])$.*

Preuve: Supposons que $d_\wedge(p, o[1..i+1]) \neq d_\wedge(q, o[1..i+1])$. Avec la Proposition 8, on suppose sans perte de généralité que $d_\wedge(p, o[1..i+1]) = d_\wedge(p, o[1..i]) - 1$ and $d_\wedge(q, o[1..i+1]) = d_\wedge(q, o[1..i]) + 1$. Soit maintenant $w \in \diamondsuit^*$ tel que $|w| = r - d_\wedge(p, o[1..i+1])$. Comme $d_\wedge(p, o[1..i+1]) \leq r$, le mot w est correctement défini. De plus, puisque $o \in (\Sigma \setminus \{\diamondsuit\})^*$, On obtient $\ell(pw, o[1..i+1]) = \ell(p, o[1..i+1])$ et $\ell(qw, o[1..i+1]) = \ell(q, o[1..i+1])$. Donc d'une part, $d_\wedge(pw, o[1..i+1]) = d_\wedge(p, o[1..i+1]) + |w| = r$. Et d'autre part, $d_\wedge(qw, o[1..i+1]) = d_\wedge(q, o[1..i+1]) + |w| = d_\wedge(q, o[1..i]) + 1 + r - d_\wedge(p, o[1..i]) + 1 = r + 2$ (puisque $d_\wedge(p, o[1..i]) = d_\wedge(q, o[1..i]$ par hypothèse). Ainsi il existerait un mot $w \in \Sigma^*$ tel que $d_\wedge(pw, o[1..i+1]) \leq r$ et $d_\wedge(qw, o[1..i+1]) > r$, ce qui contredit l'équivalence (A.1). Par conséquent, $d_\wedge(p, o[1..i+1]) = d_\wedge(q, o[1..i+1])$. □

Nous pouvons maintenant conclure :
Preuve: *(du Théorème 28)* Soient $p, q \in \Sigma^*$ tels que $p^{-1}B_r(o) = q^{-1}B_r(o)$. Notons $t_p = (x_0, x_1, \ldots, x_n)$ et $t_q = (y_0, y_1, \ldots, y_n)$ et montrons par induction que $x_i = y_i$ pour tout

[1] Compte tenu des phénomènes combinatoires dans les boules, nous pensons qu'elle est indispensable.

$0 \leq i \leq n$. **Cas de base** : D'une part, si $d_\wedge(p, \lambda) > r$ alors $d_\wedge(q, \lambda) > r$ par l'équivalence (A.2), donc $x_0 = y_0 = r + 1$. D'autre part, si $d_\wedge(p, \lambda) \leq r$ alors $d_\wedge(q, \lambda) = d_\wedge(p, \lambda)$ par le Lemme 10, donc $x_0 = y_0$. **Cas inductif** : Supposons que $x_i = y_i$ pour un certain $0 \leq i < n$. Si $d_\wedge(p, o[1..i + 1]) > r$, alors $d_\wedge(q, o[1..i + 1]) > r$ par l'équivalence (A.2), donc $x_{i+1} = y_{i+1}$. Sinon, $d_\wedge(p, o[1..i+1]) \leq r$, et il existe deux sous-cas. Tout d'abord, si $d_\wedge(p, o[1..i]) > r$, alors $d_\wedge(p, o[1..i+1]) = d_\wedge(q, o[1..i+1])$ par le Lemme 11, donc $x_{i+1} = y_{i+1}$. Dans le cas contraire, on a $d_\wedge(p, o[1..i]) \leq r$, donc par hypothèse d'induction, $d_\wedge(p, o[1..i]) = d_\wedge(q, o[1..i])$. Ainsi, $d_\wedge(p, o[1..i + 1]) = d_\wedge(q, o[1..i + 1])$, par le Lemme 12, donc $x_{i+1} = y_{i+1}$. □

A.2 Identification exacte des boules quelconques

Nous ne donnons ici que des idées de preuve. Le lecteur intéressé peut consulter [Tan09].

Pour montrer l'impossibilité d'identifier $\mathcal{BALL}(\Sigma)$ en temps MC (ou IPE)-polynomial à partir de Textes, on raisonne par l'absurde en supposant qu'il existe un algorithme $\mathfrak{A}$ réalisant l'identification sous ces contraintes ; on utilise ensuite la famille de boules emboitées $B_1(\lambda) \subseteq B_2(\lambda) \subseteq \ldots B_n(\lambda)$, et on construit une famille de Textes $f_1, f_2, \ldots f_n$, où chaque f_i est un texte pour $B_i(\lambda)$; ces Textes sont aux aussi "emboités" au sens où le début de f_{i+1} est la même séquence que f_i jusqu'au point de convergence de $\mathfrak{A}$ (sur f_i).

En conséquence, $\mathfrak{A}$ sur f_n fait nécessairement n changements d'hypothèses (ou erreurs implicites de prédiction), donc un nombre exponentiellement grand devant la taille $\mathcal{O}(\log n)$ de (la représentation de) $B_n(\lambda)$. L'impossibilité d'identifier en temps CS-polynomial est plus simple : les seuls mots distinguant $B_i(\lambda)$ et $B_{i+1}(\lambda)$ sont de tailles $i + 1$, donc exponentiellement grands par rapport à la taille (de la représentation) des boules ; or comme les ensembles caractéristiques doivent permettre de distinguer chaque paire de boules, aucun ensemble caractéristique polynomial n'existe pour les boules emboitées précédentes[2].

Concernant maintenant le résultat positif d'apprenabilité des boules en temps MC-polynomial à partir d'Informants, supposons qu'un Informant f contiennent (1) deux données positives $(x^k u, +)$ et $(y^k u, +)$ avec $u \in \Sigma^*$, $x, y \in \Sigma$ et $x \neq y$, et (2) tous les mots obtenus en ajoutant une lettre à n'importe quelle position dans $x^k u$ et $y^k u$ avec l'étiquette -. Alors on peut montrer que $B_k(u)$ est la seule boule admettant un tel Informant.

Donc un algorithme trivial qui se contente d'attendre de telles données sans changer d'avis identifie bien les boules quelconques en temps MC-polynomial. On notera pourtant que la longueur $|u| + k$ des mots utilisés est exponentielle en la taille $|u| + \log k$ de (la représentation de) la boule cible. Comme ce critère n'est pas pris en compte pour l'identification en temps MC-polynomial (il l'est pour l'identification en temps CS-polynomial), celà conduit à un résultat d'identification très/trop faible.

[2] En d'autres termes, la classe $\mathcal{BALL}(\Sigma)$ n'est pas polynomialement caractérisable, au sens de [dlH97].

Bibliographie

[ABB97] J.-M. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 1, Word Language Grammar, pages 111–174. Springer-Verlag, Berlin, 1997.

[AEKR10] D. Angluin, S. Eisenstat, L. Kontorovitch, and L. Reyzin. Lower bounds on learning random structures with statistical queries. In *Proc. 21st Internal Conference on Algorithmic Learning Theory (ALT'10)*, pages 194–208. LNCS 6331, 2010.

[AL88] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2 :343–370, 1988.

[Ang79] D. Angluin. Finding patterns common to a set of strings. In *Conference record of the eleventh annual ACM Symposium on Theory of Computing*, pages 130–141, New York, USA, 1979. ACM Press.

[Ang81] D. Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51 :76–87, 1981.

[Ang82] D. Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3) :741–765, 1982.

[Ang87a] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Control*, 39 :337–350, 1987.

[Ang87b] D. Angluin. Queries and concept learning. *Machine Learning Journal*, 2 :319–342, 1987.

[Ang88] D. Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, March 1988.

[Ang92] D. Angluin. Computational learning theory : Survey and selected bibliography. In *Proc. 24th ACM Symposium on Theory of Computing (STOC'91)*, pages 351–369, Victoria, B.C., Canada, 1992. ACM Press.

[AS83] D. Angluin and C. Smith. Inductive inference : theory and methods. *ACM computing surveys*, 15(3) :237–269, 1983.

[AS94] R. Alquezar and A. Sanfeliu. Incremental grammatical inference from positive and negative data using unbiased finite state automata. In *Proc. Int. Workshop on Structural and Syntactic Pattern Recognition (SSPR'94)*, pages 291–300, 1994.

[ASVB01] J.-C. Amengual, A. Sanchis, E. Vidal, and J.-M. Benedí. Language simplification through error-correcting and grammatical inference techniques. *Machine Learning Journal*, 44(1) :143–159, 2001.

[AvZ04] P. Adriaans and M. van Zaanen. Computational grammar induction for linguists. *Grammars*, 7 :57–68, 2004.

[BBdlHJT07] L. Becerra-Bonache, C. de la Higuera, J.-C. Janodet, and F. Tantini. Learning balls of strings with correction queries. In *Proc. 18th European Conference on Machine Learning (ECML'07)*, pages 18–29, Warsaw, Poland, 2007. LNAI 4701.

[BBdlHJT08] L. Becerra-Bonache, C. de la Higuera, J.-C. Janodet, and F. Tantini. Learning balls of strings from edit corrections. *Journal of Machine Learning Research*, 9 :1823–1852, 2008.

[BBDT06] L. Becerra-Bonache, A. H. Dediu, and C. Tirnauca. Learning DFA from correction and equivalence queries. In *Proc. of the 8th International Colloquium in Grammatical Inference (ICGI'06)*, pages 281–292. LNAI 4201, 2006.

[BBY04] L. Becerra-Bonache and T. Yokomori. Learning mild context-sensitiveness : Toward understanding children's language learning. In *Proc. 7th International Colloquium in Grammatical Inference (ICGI'04)*, pages 53–64. LNAI 3264, 2004.

[BDGW97] J. L. Balcazar, J. Diaz, R. Gavaldà, and O. Watanabe. Algorithms for learning finite automata from queries : A unified view. In *Advances in Algorithms, Languages and Complexity*, pages 53–72. Kluwer, 1997.

[BdlH01] M. Bernard and C. de la Higuera. Apprentissage de programmes logiques par inférence grammaticale. *Revue d'Intelligence Artificielle*, 14(3) :375–396, 2001.

[BDR09] R. Bailly, F. Denis, and L. Ralaivola. Grammatical inference as a principal component analysis problem. In *Proc. 26th International Conference on Machine Learning (ICML'090)*, pages 5–12. ACM International Conference Proceeding Series 382, 2009.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letter*, 24 :377–380, 1987.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4) :929–965, October 1989.

[BGC01] L. Bréhélin, O. Gascuel, and G. Caraux. Hidden Markov models with patterns to learn boolean vector sequences and application to the built-in self-test for integrated circuits. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9) :997–1008, 2001.

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BJS06] M. Bernard, J.-C. Janodet, and M. Sebban. A discriminative model of stochastic edit distance in the form of conditional transducer. In *Proc. 8th International Colloquium on Grammatical Inference (ICGI'06)*, pages 240–252, Chofu, Tokyo, Japan, 2006. LNAI 4201.

[BL97] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2) :245–271, 1997.

[BM00] E. Brill and R. Moore. An improved error model for noisy channel spelling correction. In *Proc. 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, pages 286–293, 2000.

[BM02] P. Bartlett and S. Mendelson. Rademacher and gaussian complexities : Risk bounds and structural results. *Journal of Machine Learning Research*, 3, 2002.

[BO93] R. Book and F. Otto. *String-Rewriting Systems*. Springer-Verlag, 1993.

[CdlHJ09] D. Combe, C. de la Higuera, and J.-C. Janodet. Zulu : an interactive learning competition. In *Proc. 8th International Workshop on Finite State Methods and Natural Language Processing (FSMNLP'09)*, pages 139–146, Pretoria, South Africa, 2009. LNAI 6062.

[CE05] A. Clark and R. Eyraud. Identification in the limit of substitutable context-free languages. In *Proc. 16th Int. Conference on Algorithmic Learning Theory (ALT'05)*, pages 283–296. LNCS 3734, 2005.

[CEH08] A. Clark, R. Eyraud, and A. Habrard. A polynomial algorithm for the inference of context free languages. In *Proc. 9th International Colloquium in Grammatical Inférence (ICGI'08)*, pages 29–42. LNAI 5278, 2008.

[CF03] F. Coste and D. Fredouille. Unambiguous automata inference by means of state-merging methods. In *Proc. 14th European Conf. on Machine Learning (ECML'03)*, pages 60–71. LNAI 2837, 2003.

[CFKdlH04] F. Coste, D. Fredouille, C. Kermorvant, and C. de la Higuera. Introducing domain and typing bias in automata inference. In *Proc. Grammatical Inference : Algorithms and Applications (ICGI'04)*, pages 115–126. LNAI 3264, 2004.

[CFW06] A. Clark, C. Costa Florencio, and C. Watkins. Languages as hyperplanes : Grammatical inference with string kernels. In *Proc. 17th European Conference on Machine Learning (ECML'06)*, pages 90–101. LNCS 4212, 2006.

[CFWS06] A. Clark, C. Costa Florencio, C. Watkins, and M. Serayet. Planar languages and learnability. In *Proc. International Colloquium in Grammatical Inference (ICGI'06)*, pages 148–160. LNAI 4201, 2006.

[CG08] J. Castro and R. Gavaldà. Towards feasible PAC-learning of probabilistic deterministic finite automata. In *Proc. 9th International Colloquium in Grammatical Inference (ICGI'08)*, pages 163–174. LNAI 5278, 2008.

[CGL$^+$03] J. Carme, R. Gilleron, A. Lemay, A. Terlutte, and M. Tommasi. Residual finite tree automata. In *Proc. 7th International Conference on Developments in Language Theory (DLT'03)*, pages 171–182. LNCS 2710, 2003.

[CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transaction on Information Theory*, 13 :21–27, 1967.

[Chi01] B. Chidlovskii. Schema extraction from XML : A grammatical inference approach. In M. Lenzerini, D. Nardi, W. Nutt, and D. Suciu, editors, *Proc. 8th Int. Workshop on Knowledge Representation meets Databases (KRDB 2001)*, volume 45 of *CEUR Workshop Proceedings*, 2001.

[CHL07] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[Cho57] N. Chomsky. *Syntactic Structures*. Mouton, 1957.

[Cla10] A. Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In *Proc. 10th International Colloquium in Grammatical Inference (ICGI'10)*, pages 24–37. LNAI 6339, 2010.

[CLR92] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1992.

[CM99] D. Carmel and S. Markovitch. Exploration strategies for model-based learning in multiagent systems. *Autonomous Agents and Multi-agent Systems*, 2(2) :141–172, 1999.

[CM08] A. Cornuéjols and L. Miclet. What is the place of machine learning between pattern recognition and optimization ? In *Proc.* Pascal *Workshop on Teaching Machine Learning*, page ? ?, Saint-Etienne, France, 2008.

[CM10] A. Cornuéjols and L. Miclet. *Apprentissage Artificiel : Concepts et algorithmes*. Eyrolles, 2010.

[CNBYM01] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Survey*, 33(3) :273–321, 2001.

[CO94] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proc. International Colloquium in Grammatical Inference (ICGI'94)*, pages 139–150. LNAI 862, 1994.

[Com09] D. Combe. *Apprentissage actif des automates déterministes (projet de M2R)*. PhD thesis, University of Saint-Etienne, 2009.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc 3rd Symposium on Theory of Computing (STOC'71)*, pages 151–158. ACM, 1971.

[CT04] A. Clark and F. Thollard. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5 :473–497, 2004.

[DA00] P. Dupont and J.-C. Amengual. Smoothing probabilistic automata : an error-correcting approach. In *Proc. Grammatical Inference : Algorithms and Applications (ICGI '00)*, pages 51–62. LNAI 1891, 2000.

[Dau89] M. Dauchet. Simulation of turing machines by a left-linear rewrite rule. In *Proc. of the 3rd International Conference on Rewriting Techniques and Applications (RTA'89)*, pages 109–120. LNCS 355, 1989.

[DBK+92] T. Dean, K. Basye, L. Kaelbling, E. Kokkevis, O. Maron, D. Angluin, and S. Engelson. Inferring finite automata with stochastic output functions and an application to map learning. In *Proc. 10th National Conference on Artificial Intelligence*, pages 208–214, San Jose, CA, 1992. MIT Press.

[DDE05] P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden Markov models : probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9) :1349–1371, 2005.

[DdG96] F. Denis, C. d'Halluin, and R. Gilleron. PAC learning with simple examples. In *13th Symposium on Theoretical Aspects of Computer Science, STACS'96*, LNCS, pages 231–242, 1996.

[DdlHJ+09a] G. Damiand, C. de la Higuera, J.-C. Janodet, E. Samuel, and C. Solnon. A polynomial algorithm for subisomorphism of open plane graphs. In *Proc. 7th International Workshop on Mining and Learning with Graphs (MLG'09), Poster Session*, Leuven, Belgium, 2009. Electronic Proceedings.

[DdlHJ+09b] G. Damiand, C. de la Higuera, J.-C. Janodet, E. Samuel, and C. Solnon. A polynomial algorithm for submap isomorphism : Application to searching patterns in images. In *Proc. 7th IAPR International Workshop on Graph-based*

Representation in Pattern Recognition (GbRPR'09), pages 102–112, Venice, Italy, 2009. LNCS 5534.

[DDvL08] P. Dupont, C. Damas, and A. van Lamsweerde. The QSM algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22(1) :77–115, 2008.

[DE08] F. Denis and Y. Esposito. On rational stochastic languages. *Fundamenta Informaticae*, 86(1-2) :41–77, 2008.

[DEH06] F. Denis, Y. Esposito, and A. Habrard. Learning rational stochastic languages. In *Proc. 19th International on Computational Learning Theory (COLT'06)*, pages 274–288. LNCS 4005, 2006.

[DEKM98] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 1998.

[Den01] F. Denis. Learning regular languages from simple positive examples. *Machine Learning Journal*, 44(1) :37–66, 2001.

[Die00] T. Dietterich. Ensemble methods in machine learning. In *Proc. 1st International Workshop on Multiple Classifier Systems (MSC'00)*, pages 1–15. LNCS 1857, 2000.

[DJ90] N. Dershowitz and J. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science : Formal Methods and Semantics*, volume B, chapter 6, pages 243–320. North Holland, Amsterdam, 1990.

[dlH97] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning Journal*, 27 :125–138, 1997.

[dlH05] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9) :1332–1348, 2005.

[dlH06a] C. de la Higuera. Data complexity issues in grammatical inference. In M. Basu and T. Kam Ho, editors, *Data Complexity in Pattern Recognition*, pages 153–172. Springer-Verlag, 2006.

[dlH06b] Colin de la Higuera. Ten open problems in grammatical inference. In *Proc. International Colloquium in Grammatical Inference (ICGI'06)*, pages 32–44. LNAI 4201, 2006.

[dlH10] C. de la Higuera. *Grammatical Inference : Learning Automata and Grammars*. Cambridge University Press, 2010.

[dlHC00] C. de la Higuera and F. Casacuberta. Topology of strings : Median string is NP-complete. *Theoretical Computer Science*, 230 :39–48, 2000.

[dlHJ01] C. de la Higuera and J.-C. Janodet. Inference of ω-languages from prefixes. In *Proc. 12th International Conference on Algorithmic Learning Theory (ALT'01)*, pages 364–377, Washington, D.C., USA, 2001. LNCS 2225.

[dlHJ04] C. de la Higuera and J-C. Janodet. Inference of ω-languages from prefixes. *Theoretical Computer Science*, 313(2) :295–312, 2004.

[dlHJT06] C. de la Higuera, J.-C. Janodet, and F. Tantini. Identification in the limit of systematic noisy languages. In *Proc. 8th International Colloquium on Grammatical Inference (ICGI'06)*, pages 19–31, Chofu, Tokyo, Japan, 2006. LNAI 4201.

[dlHJT08] C. de la Higuera, J.-C. Janodet, and F. Tantini. Learning languages from bounded resources : The case of the dfa and the balls of strings. In *Proc. 9th International Conference in Grammatical Inference (ICGI'08)*, pages 43–56, Saint-Malo, France, 2008. LNAI 5278.

[dlHO02a] C. de la Higuera and J. Oncina. Learning deterministic linear languages. In *Proc. COmputational Learning Theory (COLT'02)*, pages 185–200. LNAI 2375, 2002.

[dlHO02b] C. de la Higuera and J. Oncina. On sufficient conditions to identify in the limit classes of grammars from polynomial time and data. In *Proc. Grammatical Inference : Algorithms and Applications (ICGI'02)*, pages 134–148. LNAI 2484, 2002.

[dlHOV96] C. de la Higuera, J. Oncina, and E. Vidal. Identification of DFA : data-dependent versus data-independent algorithm. In *Proc Int. Colloquium on Grammatical Inference (ICGI'96)*, pages 313–325. LNAI 1147, 1996.

[dlHT00] C. de la Higuera and F. Thollard. Identication in the limit with probability one of stochastic deterministic finite automata. In *Proc. Grammatical Inference : Algorithms and Applications (ICGI'00)*, pages 15–24. LNAI 1891, 2000.

[DLT02] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51(4) :339–368, 2002.

[DLT04] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using RFSAs. *Theoretical Computer Science*, 313(2) :267–294, 2004.

[DM98] P. Dupont and L. Miclet. Inférence grammaticale régulière : fondements théoriques et principaux algorithmes. Technical report, IRISA, 1998.

[DM04] A. Delhay and L. Miclet. Analogical equations in sequences : Definition and resolution. In *Proc. 7th Int. Colloquium on Grammatical Inference (ICGI'04)*, pages 127–138. LNAI 3264, 2004.

[Dup06] P. Dupont. Noisy sequence classification with smoothed Markov chains. In *Proc. Conférence en Apprentissage (CAp'06)*, pages 187–202. Presses Universitaires de Grenoble, 2006.

[DW00] C. Domingo and O. Watanabe. Madaboost : a modification of adaboost. In *Proc. 3rd Annual Conference on Computational Learning Theory (COLT'00)*, pages 180–189. ACM Press, 2000.

[EdlHJ04] R. Eyraud, C. de la Higuera, and J.-C. Janodet. Representing languages by learnable rewriting systems. In *Proc. 7th International Colloquium on Grammatical Inference (ICGI'04)*, pages 139–150, Athens, Greece, 2004. LNAI 3264.

[EdlHJ07] R. Eyraud, C. de la Higuera, and J.-C. Janodet. LARS : A learning algorithm for rewriting systems. *Machine Learning*, 66(1) :7–31, 2007.

[ER90] J. Engelfriet and G. Rozenberg. Graph grammars based on node rewriting : an introduction to nlc graph grammars. In *Proc. 4th International Workshop on Graph Grammars and their Application to Computer Science*, pages 12–23. LNCS 532, 1990.

[ET96] H. Ehrig and G. Taentzer. Computing by graph transformation : a survey and annotated bibliography. *Bulletin of the EATCS*, 59 :182–226, June 1996.

[Eyr06] R. Eyraud. *Inférence grammaticale des langages hors-contextes*. PhD thesis, University of Saint-Etienne, 2006.

[Fer01] H. Fernau. Learning XML grammars. In *Machine Learning and Data Mining in Pattern Recognition (MLDM'01)*, pages 73–87. LNCS 2123, 2001.

[Fer05] H. Fernau. Algorithms for learning regular expressions. In *Proc. 16th Int. Conference on Algorithmic Learning Theory (ALT'05)*, pages 297–311. LNCS 3734, 2005.

[FHT00] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression : a statistical view of boosting. *The Annals of Statistics*, 38(2) :337–374, 2000.

[FJ86] M. Farber and R.E. Jamison. Convexity in graphs and hypergraphs. *SIAM Journal on Algebraic and Discrete Methods*, 7(3) :433–444, 1986.

[FKR+97] Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. Efficient learning of typical finite automata from random walks. *Information and Computation*, 138 :23–48, 1997.

[Fre01] Y. Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3) :293–318, 2001.

[FS97] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1) :119–139, 1997.

[Fus07] E. Fusy. *Combinatoire des graphes planaires et applications algorithmiques*. PhD thesis, Ecole Polytechnique, 2007. In English.

[Gav93] R. Gavaldà. On the power of equivalence queries. In *Proc. 1st European Conference on Computational Learning Theory (EUROCOLT'93)*, volume 53 of *The Institute of Mathematics and its Applications Conference Series, new series*, pages 193–203. Oxford University Press, 1993.

[Gav09] R. Gavalda. Note : More efficient conversion of equivalence-query algorithms to pac algorithms. Technical report, DLSI, UPC, 2009.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GM96] S. Goldman and H. Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2) :255–267, 1996.

[GO93] P. García and J. Oncina. Inference of recognizable tree sets. Technical Report DSIC-II/47/93, Departamento de Lenguajes y Sistemas Informáticos, Universidad Politécnica de Valencia, Spain, 1993.

[Gol67] E. Gold. Language identification in the limit. *Information and Control*, 10(5) :447–474, 1967.

[Gol78] E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37 :302–320, 1978.

[Goo01] J. Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 15(4) :403–434, 2001.

[Gre02] R. I. Greenberg. Fast and simple computation of all longest common subsequences. Technical report, Loyola University, 2002. http ://arXiv.org/abs/cs.DS/0211001.

[Gre03] R. I. Greenberg. Bounds on the number of longest common subsequences. Technical report, Loyola University, 2003. http ://arXiv.org/abs/cs/0301030v2.

[GS91] S. Goldman and R. Sloan. The difficulty of random attribute noise. Technical report, Departement of Computer Science, Washington University, 1991.

[GSBC+03] S. Garcia-Salicetti, C. Beumier, G. Chollet, B. Dorizzi, J. Leroux-Les Jardins, J. Lunter, Y. Ni, and D. Petrovska-Delacretaz. BIOMET : A multimodal person authentication database including face, voice, fingerprint, hand and signature modalities. In *Proc. 4th International Conference on Audio and Video-Based Biometric Person Authentication*, 2003.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology.* Cambridge University Press, 1997.

[Hab92] A. Habel, editor. *Hyperedge Replacement.* LNCS 643, 1992.

[Har78] M. H. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1978.

[HHNS02] A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model generation by moderated regular extrapolation. In *Proc. of the 5th Int. Conference on Fundamental Approaches to Software Engineering (FASE '02)*, pages 80–95. LNCS 2306, 2002.

[HKLW91] D. Haussler, M. Kearns, N. Littlestone, and M. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95(2) :129–161, 1991.

[HMU01] J.E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001.

[HV10] M. Heule and S. Verwer. Exact dfa identification using sat solvers. In *Proc. 10th International Colloquium on Grammatical Inférence (ICGI'10)*, pages 66–79. LNAI 6339, 2010.

[IT97] Y. Ishigami and S. Tani. VC-dimensions of finite automata and commutative finite automata with k letters and n states. *Discrete Applied Mathematics*, 74 :123–134, 1997.

[Jan00] J.-C. Janodet. *Sur les types de données dans les langages logico-fonctionnels : réécriture et surréduction des graphes adminissibles.* PhD thesis, University of Grenoble, 2000.

[JK91] E. Jeltsch and H.-K. Kreowski. Grammatical inference based on hyperedge replacement. In *Proc. Graph Grammars and their Application to Computer Science*, pages 461–474. LNCS 532, 1991.

[JNSS04] J.-C. Janodet, R. Nock, M. Sebban, and H.-M. Suchier. Boosting grammatical inference with confidence oracles. In *Proc. 21st International Conference on Machine Learning (ICML'04)*, pages 425–432, Banff, Alberta, Canada, 2004. AAAI Press.

[JNSS05] J.-C. Janodet, R. Nock, M. Sebban, and H.-M. Suchier. Adaptation du boosting à l'inférence grammaticale via l'utilisation d'un oracle de confiance. *Revue d'Intelligence Artificielle*, 19 :713–740, 2005.

[JORS99] S. Jain, D. Osherson, J. S. Royer, and A. Sharma. *Systems That Learn.* MIT press, 1999.

[JS03] J.-C. Janodet and M. Sebban. On state merging in grammatical inference : A statistical approach for dealing with noisy data. In *Proc. 20th International Conference on Machine Learning (ICML'03)*, pages 688–695, Washington, D.C., USA, 2003. AAAI Press.

[JSS09] J.-C. Janodet, M. Sebban, and H.-M. Suchier. Boosting classifiers built from different subsets of features. *Fundamenta Informaticae*, 94 :1–21, 2009.

[JTS04] J.-C. Janodet, F. Tantini, and M. Sebban. Blue* : A blue-fringe procedure to learn dfa from noisy data. In *Proc. Competition "Learning DFA from Noisy Data", a parallel event to the Genetic and Evolutionary Computation Conference (GECCO'04)*, Seattle, USA, 2004.

[Kar72] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations (Symposium Proceedings)*, pages 85–103. Plenum Press, 1972.

[KBdBB03] R. Kosala, M. Bruynooghe, J. Van den Bussche, and H. Blockeel. Information extraction from web documents based on local unranked tree automaton inference. In *Proc. 18th Int. Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 403–408. Morgan Kaufmann, 2003.

[KCM08] L. Kontorovich, C. Cortes, and M. Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3) :223–236, 2008.

[Kea98] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6) :983–1006, 1998.

[Kin08] E. Kinber. On learning regular expressions and patterns via membership and correction queries. In *Proc. 9th International Colloquium on Grammatical Inference (ICGI'08)*, pages 125–138. LNAI 5278, 2008.

[KKR+95] E. Kranakis, D. Krizanc, B. Ruf, J. Urrutia, and G. J. Woeginger. Vc-dimensions for graphs. In *Proc. 21st International Workshop on Graph-Theoretic Concepts in Computer Science (WG'95)*, pages 1–13. LNCS 1017, 1995.

[KL93] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal of Computing*, 22(4) :807–837, 1993.

[Klo92] J. W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science*, volume 2, pages 1–112. Oxford University Press, 1992.

[Kor67] A. Korshunov. The degree of distinguishability of automata. *Disket. Analiz.*, 10(36) :39–59, 1967.

[KP02] V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics*, 30(1) :1–50, 2002.

[Kru83] J. Kruskal. An overview of sequence comparison : Time marps, string edits and macromolecules. *SIAM Review*, 25(2) :201–237, 1983.

[KT05] J. Kleinberg and E Tardòs. *Algorithm Design*. Addison-Wesley, 2005.

[KV89] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proc. 21st ACM Symposium on Theory of Computing (STOC'89)*, pages 433–444, 1989.

[KV94] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.

[Lan92] K. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proc. International Conference on Computational Learning Theory (COLT'92)*, pages 45–52. ACM, 1992.

[Lev65] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4) :845–848, 1965.

[Lit87] N. Littlestone. Learning quickly when irrelevant attributes abound : a new linear threshold. *Machine Learning Journal*, 2 :285–318, 1987.

[LL09] M. Lange and H. Leiss. To CNF or not to CNF ? an efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8 :1–21, 2009.

[LPC98] K. Lang, B. A. Pearlmutter, and F. Coste. The Gowachin automata learning competition, 1998.

[LPP98] K. Lang, B. Pearlmutter, and R. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Proc. International Colloquium in Grammatical Inference (ICGI'98)*, pages 1–12. LNAI 1433, 1998.

[LS98] D. López and J. M. Sempere. Handwritten digit recognition through inferring graph grammars. In *Proc. Advances in Pattern Recognition, Joint IAPR International Workshops SSPR '98 and SPR '98*, pages 483–491. LNCS 1451, 1998.

[LU00] K. Lemström and E. Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *Proc. Symp. on Creative and Cultural Aspects and Applications of Artificial Intelligence and Cognitive Science (AISB'00)*, pages 53–60, 2000.

[LV91] M. Li and P. Vitanyi. Learning simple concepts under simple distributions. *Siam Journal of Computing*, 20 :911–935, 1991.

[Mai77] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25 :322–336, 1977.

[MCM83] R. Michalski, J. Carbonnell, and T. Mitchell. *Machine Learning : An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, California, 1983.

[Mic90] L. Miclet. *Syntactic and Structural Pattern Recognition, Theory and Applications*, chapter Grammatical Inference, pages 237–290. World Scientific, 1990.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[MNO88] R. McNaughton, P. Narendran, and F. Otto. Church-Rosser Thue systems and formal languages. *Journal of the Association for Computing Machinery*, 35(2) :324–344, 1988.

[MR02] R. Meir and G. Rätsch. An introduction to boosting and leveraging. In *Machine Learning Summer School*, pages 118–183. LNCS 2600, 2002.

[MSSS99] A. Mitchell, T. Scheffer, A. Sharma, and F. Stephan. The VC-dimension of subclasses of pattern languages. In *Proc. 10th International Conference on Algorithmic Learning Theory (ALT'99)*, pages 93–105. LNCS 1720, 1999.

[Nav01] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys*, 33(1) :31–88, 2001.

[Ner58] A. Nerode. Linear automaton transformations. *Proc. of the AMS*, 9 :541–544, 1958.

[NM05] K. Nakamura and M. Matsumoto. Incremental learning of context free grammars based on bottom-up parsing and search. *Pattern Recognition*, 38(9) :1384–1392, 2005.

[NMW97] C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences : a linear-time algorithm. *Journal of Artificial Intelligence Research*, 7 :67–82, 1997.

[NR05] F. Nicolas and E. Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms*, 3(2-4) :390–415, June 2005.

[NW01] G. Niemann and J. Waldmann. Some regular languages that are church-rosser congruential. In *Proc. 5th Int. Conference on Developments in Language Theory (DLT'01)*, pages 330–339. LNCS 2295, 2001.

[OABBA06] T. Oates, T. Armstrong, L. Becerra-Bonache, and M. Atamas. Inferring grammars for mildly context sensitive languages in polynomial-time. In *Proc. Grammatical Inference : Algorithms and Applications (ICGI'06)*, pages 137–147. LNAI 4201, 2006.

[O'D77] M. O'Donnell. *Computing in Systems Described by Equations*. LNCS 58, 1977.

[OG92] J. Oncina and P. García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[OGV93] J. Oncina, P. García, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5) :448–458, 1993.

[OS06] J. Oncina and M. Sebban. Learning stochastic edit distance : Application in handwritten character recognition. *Pattern Recognition*, 39(9) :1575–1587, 2006.

[OV96] J. Oncina and M. A. Varó. Using domain information during the learning of a subsequential transducer. In *Proc. Int. Colloquium on Grammatical Inference (ICGI'96)*, pages 301–312. LNAI 1147, 1996.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison Wesley, New York, 1994.

[Par66] R. Parikh. On context-free languages. *Journal of the ACM*, 13(4) :570–581, 1966.

[PH97] R. J. Parekh and V. Honavar. Learning DFA from simple examples. In *Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, ICML-97*, 1997.

[Pit89] L. Pitt. Inductive inference, DFA's, and computational complexity. In *Proc. Analogical and Inductive Inference (AII'89)*, pages 18–44. LNAI 397, 1989.

[PV88] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4) :965–984, 1988.

[PW93] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM*, 40(1) :95–142, 1993.

[RDS04] C. Rudin, I. Daubechies, and R. Schapire. The dynamics of adaboost : Cyclic behavior and convergence of margins. *Journal of Machine Learning Research*, 5 :1557–1595, 2004.

[RE97] G. Rozenberg and H. Ehrig. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1–3. World Scientific, 1997.

[Ric00] C. Rick. Simple and fast linear space computation of longest common subsequences. *Information Processing Letters*, 75(6) :275–281, 2000.

[ROM99] G. Rätsch, T. Onoda, and K.-R. Müller. Regularizing adaboost. In *Advances in Neural Information Processing Systems 11, [NIPS Conference, Denver, 1998]*, pages 564–570. The MIT Press, 1999.

[RS93] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103 :299–347, 1993.

[RST95] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proc. 8th Annual ACM Conference on Computational Learning Theory (COLT'95)*, pages 31–40. ACM Press, 1995.

[RY98] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(5) :522–532, 1998.

[Sak90] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76 :223–242, 1990.

[Sak97] Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185 :15–45, 1997.

[Sak03] J. Sakarovitch. *Eléments de théorie des automates*. Vuibert Informatique, 2003.

[SB98] R. Sutton and A. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[SBH+94] Y. Sakakibara, M. Brown, R. Hughley, I. Mian, K. Sjolander, R. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nuclear Acids Res.*, 22 :5112–5120, 1994.

[Sch90] R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2) :197–227, 1990.

[SCvZ05] B. Starkie, F. Coste, and M. van Zaanen. Progressing the state-of-the-art in grammatical inference by competition : The omphalos context-free language learning competition. *AI Communication*, 18(2) :93–115, 2005.

[SdlHJ10] E. Samuel, C. de la Higuera, and J.-C. Janodet. Extracting plane graphs from images. In *Proc. 13th International Workshop on Structural and Syntactic Pattern Recognition (SSPR'10)*, pages 233–243. LNCS 6218, 2010.

[Sén02] G. Sénizergues. L(A)=L(B) ? a simplified decidability proof. *Theoretical Computer Science*, 281(1-2) :555–608, 2002.

[Set10] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Winconsin–Madison, 2010.

[SFBL98] R. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin : A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26 :1651–1686, 1998.

[SJY02] M. Sebban, J.-C. Janodet, and A. Yahiaoui. Effet de la suppression des mots bruités en inférence grammaticale. In *Proc. 7ème Conférence Nationale de Classification*, pages 311–314, Toulouse, France, 2002. Service Central repro Université Paul Sabatier.

[SM02] K. U. Schulz and S. Mihov. Fast string correction with Levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1) :67–85, 2002.

[SS92] Y. Sakakibara and R. Siromoney. A noise model on learning sets of strings. In *Proc. International Conference on Computational Learning Theory (COLT'92)*, pages 295–302, 1992.

[SS99] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3) :297–336, 1999.

[SS00] R. Schapire and Y. Singer. BoosTexter : A bossting-based system for text categorization. *Machine Learning*, 39(2-3) :135–168, 2000.

[STC00] J. Shawe-Taylor and N. Cristianini. *Support-Vector Machines*. Cambridge University Press, 2000.

[STC04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[Ste97] F. Stephan. Noisy inference and oracles. *Theoretical Computer Science*, 185 :129–157, 1997.

[Suc06] H.-M. Suchier. *Etude du boosting en apprentissage automatique*. PhD thesis, University of Saint-Etienne, 2006.

[SY93] A. Saoudi and T. Yokomori. Learning local and recognizable ω-languages and monadic logic programs. In *Proceedings of EUROCOLT*, LNCS, pages 157–169. Springer-Verlag, 1993.

[Tan09] F. Tantini. *Inférence grammaticale en situations bruitées*. PhD thesis, University of Saint-Etienne, 2009.

[TB73] B. Trakhtenbrot and Y. Barzdin. *Finite Automata : Behavior and Synthesis*. North Holland Pub. Comp., Amsterdam, 1973.

[TDdlH00] F. Thollard, P. Dupont, and C. de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. 17th Int. Conference on Machine Learning (ICML'00)*, pages 975–982. Morgan Kaufmann, 2000.

[Tir08] C. Tirnauca. A note on the relationship between different types of correction queries. In *Proc. 9th International Colloquium on Grammatical Inference (ICGI'08)*, pages 213–223. LNAI 5278, 2008.

[TTT10] F. Torre, F. Tantini, and A. Terlutte. Sequences classification by least general generalisations. In *Proc. 10th International Colloquium on Grammatical Inference (ICGI'10)*, pages 189–202. LNAI (to appear), 2010.

[Ukk85] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6 :132–137, 1985.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11) :1134–1142, 1984.

[Val85] L. Valiant. Learning disjunctions of conjunctions. In *Proc. 9th International Joint Conference in Artificial Intelligence (IJCAI'85)*, pages 560–566, 1985.

[Vap82] V. Vapnik. *Estimation of Dependences based on Empirical Data*. Springer-Verlag, 1982.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[VdWW09] S. Verwer, M. de Weerdt, and C. Witteveen. One-clock deterministic timed automata are efficiently identifiable in the limit. In *Proc. 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, pages 740–751. LNCS 5457, 2009.

[VTdlH$^+$05] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite state automata. *Pattern Analysis and Machine Intelligence*, 27(7) :1013–1039, 2005.

[War89] M. Warmuth. Towards representation independence in PAC-learning. In *Proc. Analogical and Inductive Inference (AII'89)*, pages 78–103. LNAI 397, 1989.

[Wat03] B. Watson. A new algorithm for the construction of minimal acyclic DFAs. *Science of Computer Programming*, 48(2-3) :81–97, 2003.

[Wet80] C. S. Wetherell. Probabilistic languages : a review and some open questions. *Computing Surveys*, 12(4) :361–379, 1980.

[Yah03] A. Yahiaoui. *Amélioration des règles de fusion d'états en inférence exacte : approches statistiques et géométriques (projet de DEA)*. PhD thesis, University of Saint-Etienne, 2003.

[Yok89] T. Yokomori. Learning context-free languages efficiently : a report on recent results in Japan. In K. P. Jantke, editor, *Analogical and Inductive Inference : Proc. of the International Workshop AII'89*, pages 104–123. Springer-Verlag, Berlin, Heidelberg, 1989.

[Yok92] T. Yokomori. Inductive inference of 0l languages. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems*, Texts and Monographs in Computer Science, pages 115–132. Springer-Verlag, 1992.

[Yok95] T. Yokomori. On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning Journal*, 19 :153–179, 1995.

[Yok03] T. Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science*, 1(298) :179–206, 2003.