Inférence grammaticale en situations bruitées

Frédéric Tantini

N° d'ordre: 00000

École doctorale de Saint-Étienne

Inférence grammaticale en situations bruitées

Thèse préparée
pour obtenir le grade de :

Docteur de l'Université Jean Monnet de Saint-Étienne
Mention Informatique

par **Frédéric Tantini**

Laboratoire Hubert Curien
Faculté des Sciences et Techniques

À soutenir le 09 juin 2009 devant le jury composé de :

M. :	Président	du jury	Président
MM. :	Thierry	Lecroq	Rapporteurs
	Laurent	Miclet	
MM. :	Leonor	Becerra-Bonache	Examinateurs
	Marc	Tommasi	
	Marc	Sebban	
MM. :	Colin	de la Higuera	Directeur
	Jean-Christophe	Janodet	Co-directeur

Remerciements

À l'heure où j'écris ces remerciements, la rédaction des chapitres du manuscrits est terminée. C'est donc le cœur léger que je peux remercier tous ceux qui ont contribué de près ou de loin à l'écriture de cette thèse.

J'aimerais tout d'abord remercier Colin de la Higuera, mon directeur de thèse. Je le remercie d'avoir proposé ce sujet et de m'avoir accepté comme doctorant. Sans lui, ma thèse ne serait pas ce qu'elle est à présent. Il a su me diriger et m'encadrer quand j'en avais besoin, tout en me laissant libre dans mes décisions. J'ai apprécié travailler avec lui : les discussions que l'on a eues ainsi que ses conseils sont à la fois rigoureux, constructifs et enthousiastes. De plus, malgré ses moult fonctions, il a toujours su rester disponible.

Je tiens également à remercier grandement mon co-directeur de thèse Jean-Christophe Janodet. Merci à lui pour sa motivation sans faille, son entrain, son dynamisme ainsi que sa bonne humeur. Bien que pas toujours d'accord sur la forme plutôt que le fond de nos articles (j'ai tenu bon et ai réussi à ne pas mettre de blague :þ), son aide dans mon travail de recherche m'a été plus que d'une grande utilité.

Je les remercie tous les deux de m'avoir supporté, et ce depuis le stage de master, de m'avoir encouragé, et de m'avoir offert un cadre de travail agréable.

Je tiens ensuite à remercier les membres du jury, et en particulier les rapporteurs. Je remercie donc Thierry Lecroq ainsi que Laurent Miclet pour avoir bien voulu réaliser la lourde tâche de rapporter mon travail de thèse. Je tiens tout spécialement à les remercier pour la qualité des remarques faites sur mon mémoire, ainsi que sur mes travaux de recherche en général.

Je désire également remercier Marc Tommasi, président de ce jury, ainsi que Leonor Becerra-Bonache et Marc Sebban, pour leur travail d'examinateur, et les discussions que j'ai pu avoir avec eux.

J'aimerai ensuite remercier tous mes collègues de travail, ainsi que les personnes avec lesquelles j'ai, de près ou de loin, travaillé. Une mention spéciale pour Thierry, sans qui je n'aurai toujours pas de screen, de .bashrc digne de ce nom, de compte sur magoha-moth, et autres joyeusetés GNU. Merci donc à tous ceux avec qui j'ai discuté, travaillé, mangé, qui m'ont supporté (plus ou moins longtemps), que j'ai supporté, qui m'ont aidé, et inversement réciproquement cyclique : Alain, Amaury, Baptiste, Catherine, Cécile, Christine, Christophe, Cristina, Colin, Émilie, Élisa, Fabien, Fabrice, Franck, François, Hazael, Henri-Maxime, Jean, Jean-Christophe, Laurent, Leo, Marc B, Marc S, Mathias, Phillipe, Pierre, Rémi, Sabri, Sébastien, Toufik. Merci à tous ceux et celles qui ont partagé mon bureau, et merci à tous ceux que j'oublie.

Enfin, merci à tous ceux que je n'ai pas cité et qui ne comprendront que peu ou prou ce qui suivra ces premières pages mais qui m'ont soutenu au moins moralement ! Merci à la famille, les amis. Merci Amandine.

TXFkb 3Ugw6 AgQ2F vbS1v YXhtL CBtZy BRZ3V vdywg bWdqI G90bW docWU tZWFn ZHVlL CBtZ2 ogd2F teG1l IHFmI G1nai BidXp zYWd1 emUsI MOgIE 1hemV 1cWdk IFJnb nV3LC DDoCB 4J3V6 aHF6Z nFnZC BwZyB iZGFi YXJte i4K

GFjRe 2MrOP r3GyQ nXR5K gYCnJ gvXQ0 KkCpz VtUJm XXfKY 3hfPe rtpxN KGqlR FfEqM vd3e0 wLAbK eJ2ad HlfUg 0pZLQ ztoId pHA5W vpiwF yJ1ZH n5fIp WJbYy Xjn3D YlYuN zBoSh cF0N0 AbCto 2GLGX XODd0 9Uz==

0xec7 32238 93f47 7ada6 e7c1a 01497 e365d 0495a 045c0 54ce5 fdebe 9e7ee 29192 0f094 24861 56423 b8ceb ddb60 46c99 746da c901c 83b3a 9e264 e6378 71566 b577b ac005 08eb5 75f1c 44de4 79762 963b2 49d49 c8352 de53c d7fc8 c615a 4f2f3 c2c79 0424b 9bea5 749a8 547c1 fb3e2 0617a 1f083 1a2c5 c44b5 877b6 71694 92574 8194f 2b822 e6d12 9e079 334de 7c419 988f8 f8386 23e90 2bd64 4faf5 946d7 bd501 1dd75 2bc85 7396a 010

342414 413341 323124 521225 135544 415254 352114 225252 214154 213323 455245 313255 212125 211445 325434 152552 455523 225545 111315 532342 215225 522331 515544 124253 354415 235243 524232 511433 435112 211233 414113 334113 354211 133223 113323 111334 515324 331134 142541 524343 254132 331332 545524 554151 543553 533234 54

1f8b0 80031 52374 a0203 5d4f3 b52c3 400ce d7d8a d778d cac53 41380 5a420 17108 b1c6f b0578 e7637 136e4 3cb11 a87d3 1b4ce d030a 3467a 5fb5c fac3e 60fdc 24448 344d0 c4fc3 fa0d5 97fe0 3b4e1 997d2 4904e 12cc5 ee5c0 f8c73 77a59 09019 9916d 61dee 3e1ab cf8d1 382a6 f921d 4a148 793b8 a64d6 cebde b4018 b727c 77c86 1e62e a12ca e9a2a f7361 32750 c9ca6 9d7b4 4d7b3 77d95 f8592 92f21 6653c a5f58 ba143 20dc7 70731 88687 feb1d ff74f 0e336 9107c 905a6 235f4 a378a b93ac c5f16 038cd e1445 b2847 2fcad 69231 88bdf 32ffe 306ea c794b ab582 c9321 bfbe2 ba7df 34010 000

Bref, merci à tous, mais aussi aux autres.

Table des matières

Introduction

42

42 : voilà la réponse à « la grande question sur la vie, l'univers et le reste ». Tout du moins, voilà la réponse que donne Deep Though [Ada79], ordinateur super puissant créé par une race d'hyper-intelligences pan-dimensionnelles. Deep Though est un des nombreux ordinateurs de science-fiction qui possèdent une intelligence artificielle supérieure à celle des êtres humains. Bien que d'après de nombreux romans des ordinateurs aussi intelligents que Hal 9000 [CK68] auraient dû exister dès 2001, les progrès de la science informatique font que l'homme reste supérieur à la machine sur de nombreux domaines. Certes, Deep Blue a battu Garry Kasparov aux échecs alors que ce dernier était champion du monde. Cependant, ce n'est pas toujours le cas. Par exemple, les logiciels de jeu de go sont encore loin du niveau des professionnels, voire des meilleurs amateurs (le programme Crazy Stone [Cou06] peut battre des amateurs voire des professionnels, mais uniquement avec des pierres de handicap). Que dire alors de tromper un humain avec un test de Turing : une personne discute avec X et doit deviner au bout d'un certain temps, si X est un être humain ou une machine. Si ces dernières tâches sont loin d'être accomplies, l'*intelligence artificielle* s'emploie à développer techniques et théories pour que les ordinateurs puissent raisonner, planifier, percevoir, se mouvoir, déplacer des objets, apprendre, *etc.*

Parmi les nombreuses catégories de l'intelligence artificielle, l'apprentissage automatique vise à permettre à un ordinateur d'apprendre : apprendre à raisonner, apprendre à parler, *etc.* De façon générale, l'ordinateur a accès à des données dites *d'apprentissage* et en extrait une connaissance, des règles, afin de pouvoir être confronté à des données inconnues [CM08]. Cette extraction de règles permet alors de pouvoir reconnaître des données qu'il n'a pas encore vues. Cela lui permet de généraliser et ainsi d'éviter un apprentissage *par cœur*. Les applications sont alors multiples : de la reconnaissance de caractères manuscrits ou de la parole [Cas90, GSVG94], aux filtres de spams, en passant par les jeux (échecs, backgammon), la biologie [AM97, DEKM98], le traitement de la langue naturelle, le projet DARPA *etc.* Par exemple, on fournit des données à l'apprenant, comme des caractères manuscrits ou des courriers indésirables, et la généralisation lui permet de se confronter à de nouvelles données et donc de reconnaître des caractères écrits par une nouvelle personne, ou encore filtrer des messages qu'il n'avait jamais reçus.

Pour extraire ces règles, il existe plusieurs techniques, qui se révèlent plus ou moins

bonnes selon la tâche à accomplir. L'une d'entre elles s'appelle l'inférence grammaticale. Comme son nom l'indique, cette technique permet d'inférer une grammaire, c'est-à-dire un ensemble de règles basées sur des mots, chaque mot étant la concaténation de symboles issus d'un même alphabet. Nous pourrons par exemple avoir :

- $\{$*Chien, Le, Mange,* $\ldots\}$;
- $\{a, b, c, \ldots\}$;
- $\{0, 1\}$.

Ainsi, à partir d'un ensemble de *mots* comme :

- $\{$*LeChienMange*,$\ldots\}$;
- $\{abc, abcabc, \ldots\}$;
- $\{00001, 0111, \ldots\}$.

et de toute autre aide extérieure disponible, un algorithme essayera de généraliser en proposant des *grammaires* permettant de générer des mots qu'il n'a pas vu. Les règles de grammaires pouvant être :

- $\{$Phrase$\rightarrow$ *Sujet Verbe Complement*, *ArticleDefini* $\rightarrow$ *le* | *la* | *les*, $\ldots\}$;
- $\{S \rightarrow SS \mid abc\}$;
- $\{S \rightarrow ZU, Z \rightarrow ZZ|0, U \rightarrow UU|1\}$.

Ces grammaires permettront alors à l'algorithme de trouver le *langage* associé à ces règles

- la langue française,
- le langage formé des mots composés de concaténations du mot *abc*,
- le langage dont les mots débutent par des 0, et se terminent par des 1.

De toute évidence, l'inférence grammaticale est particulièrement adaptée à la recherche d'un langage cible. Mais elle l'est également lorsque les données sont des mots qui n'ont pas été générés par une grammaire. Nous parlons alors d'induction de grammaires. Par la suite, le terme inférence grammaticale regroupera l'inférence grammaticale à proprement parler ainsi que l'induction de grammaires [dlH05a].

Depuis 1957, une théorie est particulièrement utilisée en inférence grammaticale : celle de Noam Chomsky [Cho57]. Cette théorie permet de différencier les langages par leurs règles de grammaires, nous parlons alors de la *hiérarchie de Chomsky*. Les langages de base de cette hiérarchie sont appelés *langages rationnels*, ou encore *langages réguliers*. Ils sont sensés être plus faciles à apprendre que les langages de plus haut niveau, plus complexes.

La complexité de l'apprentissage

Revenons maintenant un instant à Deep Though et à la question qui lui a été posée. La réponse qu'il a donnée a nécessité de très longues années de calcul : plus de sept millions. Par conséquent, les ingénieurs en avaient même oublié la question initiale. Si nous pouvons comprendre que donner une réponse à « la grande question sur la vie, l'univers et le reste » prenne bien plus de temps qu'effectuer une addition, nous pouvons nous demander si sept millions d'années, ce n'est pas un peu trop long... Pour répondre à cela, il faut prendre en considération plusieurs points. Tout d'abord, la

difficulté de la question : faire une addition est plus simple que de calculer une racine carrée. Ensuite la taille des données : faire une addition de 12 avec 3 demande moins d'efforts qu'additionner deux nombres possédant chacun une centaine de chiffres, il est alors normal que la deuxième opération requière plus de temps que la première.

Lors de l'apprentissage de langages, la même question va se poser : mon programme n'a-t-il pas mis trop longtemps pour apprendre ? Encore une fois, pour y répondre, il faudra prendre en compte deux facteurs. Tout d'abord, la taille de la grammaire cible : un langage rationnel ne possédant que deux règles de grammaire doit être plus simple à apprendre qu'un langage en possédant une dizaine. De même, la quantité de données que le programme a à sa disposition va jouer un rôle important : s'il ne dispose que de peu de données, il ne devra mettre que peu temps pour formuler une hypothèse. L'algorithme devra alors identifier la cible en un temps *raisonnable* relativement à ces deux paramètres, plus formellement en un temps *polynomial*, pour pouvoir dire qu'il a bien appris.

Ce concept de polynomialité est alors pris en compte dans les différentes définitions d'apprentissage. En inférence grammaticale, il existe plusieurs façons distinctes d'apprendre : les données peuvent être subies ou choisies, l'inférence peut être exacte ou statistique. Trois paradigmes sont alors fréquemment utilisés :

1. L'identification à la limite [Gol67, Gol78] qui correspond à un apprentissage dit exact : un algorithme identifie à la limite une classe de langages si quelque soit le langage, lorsque nous lui donnons assez de données, il retourne exactement la cible. De nombreuses variantes sont apparues pour mieux prendre en compte les contraintes du monde réel : elles comptent le nombre de fois où l'apprenant change d'avis [AS83], le nombre d'erreurs qu'il fait [Pit89], ou bien bornent le temps autorisé en fonction de la taille de la cible [dlH97]. Des contraintes probabilistes ont également été ajoutées [CO94].
2. L'apprentissage actif [Ang87, Ang88b] qui modélise l'apprentissage de la langue par un enfant qui interagit avec sa mère : un apprenant peut poser un certain nombre de question à un *oracle* qui connaît le langage cible. Les variations de ce paradigme sont alors liées aux types de questions qu'il est possible de poser [Ang01, Tir08], ou à la façon de définir l'oracle [GM96, AKST97].
3. L'apprentissage PAC [Val84] pour Probablement Approximativement Correct : un algorithme PAC apprend si la probabilité qu'il a de faire le moins d'erreur possible est très grande. Ce paradigme sert essentiellement à prouver des résultats négatifs d'apprenabilité [PV88, KV89], mais certains travaux montrent qu'il est tout de même possible d'apprendre dans ce contexte [War89, TC04].

Une classe de langages apprenable dans un paradigme peut alors ne pas l'être dans un autre. Il convient donc de choisir convenablement le paradigme en fonction de la tache à effectuer.

Une des classes la plus intensivement étudiée est sans nul doute la classe des langages *rationnels*. Ces langages peuvent être représentés par des automates finis déterministes (AFD), modèles graphiques de leurs règles de grammaire. Depuis plusieurs années, de nombreux travaux expliquent à travers différentes méthodes comment les apprendre

dans ces différents paradigmes en *des temps raisonnables*, que ce soit en identification à la limite [Pit89, DMV94, dlH97, LPP98], en apprentissage actif [Ang90, BDGW94, BB06] ou encore en apprentissage PAC [LV91, Den01].

L'apprentissage à partir de données bruitées

Les données que vont utiliser les algorithmes d'apprentissage contiennent des erreurs. En effet, leur acquisition peut malheureusement être faussée pour plusieurs raisons : mauvais calibrage d'une sonde, défaillance de fabrication d'un capteur, erreur de recopie, oubli, *etc.* Nous parlons alors de données bruitées.

Supposons par exemple que nous voulions classer notre collection de CD de musique par genre. Nous allons utiliser un programme permettant de faire cela automatiquement. Au préalable, nous donnons à notre programme quelques CD en précisant à chaque fois leur genre. Le programme construira alors des règles lui permettant de généraliser aux CD qu'il n'a pas vu. Ainsi, il pourra classer automatiquement le reste des CD.

Plusieurs problèmes peuvent alors se poser. Tout d'abord, lorsque nous précisons les genres, nous pouvons nous tromper : la pochette nous indiquait de la musique classique, mais le CD n'était pas dans la bonne pochette ; c'était du jazz. Ensuite, lorsque nous donnons le genre au programme, nous pouvons nous tromper dans l'écriture du genre : nous pouvons taper « bluse » au lieu de « blues » par exemple. Enfin, dans l'échantillon que nous donnons au programme, nous pouvons ne pas avoir de CD de hip-hop alors qu'il en existe dans notre collection.

De même, si nous voulons extraire certaines règles à partir de plusieurs pages HTML, nous nous rendons très vite compte que la plupart des sites internet ne respectent pas la norme W3C. À chaque balise ouvrante, il faut une balise fermante (comme pour les parenthèses). Cependant beaucoup de développeurs omettent, volontairement ou non, de fermer les balises. De plus, certains utilisent des balises à des endroits où ils n'ont pas le droit selon leur DTD (la DTD ou « définition de type de document » permet de spécifier les règles d'utilisation des différentes balises dans une page).

Ces deux exemples permettent d'appréhender le concept de bruit : quelques données sont là par erreur, d'autres sont omises et enfin, certaines ont été modifiées. En pratique, quelques soient les moyens de récupérer des données, celles-ci sont généralement bruitées. Étudier l'apprentissage à partir de telles données semble alors essentiel.

Beaucoup de travaux essaient de résoudre ce problème difficile [SG86, Sak93, SN98]. La plupart utilisent des statistiques pour détecter si une donnée n'est pas erronée. Cependant, trop peu de travaux d'inférence grammaticale étudient l'apprentissage des AFD lorsque les données sont bruitées. De plus, ces travaux utilisent pour la plupart un modèle de bruit statistique, peu adapté à leur identification : les données d'apprentissage sont bruitées selon une certaine distribution puis fournies à l'apprenant.

Identifier les AFD se fait alors généralement de la façon suivante : des algorithmes « classiques » d'identification sont modifiés pour être résistants au bruit. Avec des tests statistiques, le nouvel algorithme décide si faire telle action est légitime compte tenu des informations parfois contradictoires qu'il possède (une donnée étiquetée positive et

négative par exemple). Il prend alors un certain risque à chaque étape de son processus d'inférence [HBS03, SJT04].

Malheureusement, aucun de ces travaux n'arrive à gérer un taux de bruit significatif pour des automates non triviaux : à chaque étape, la moindre erreur fausse l'automate dans son ensemble. De plus, l'introduction de bruit fait perdre l'apprentissage exacte des automates : les algorithmes vont trouver un AFD plus ou moins proche de l'AFD cible, mais il n'existe aucun moyen de savoir si l'algorithme a vraiment identifié la cible.

D'un autre côté, d'autres travaux ont défini des modèles de bruit non statistique. Cependant, les auteurs montrent que les automates ne sont pas identifiables dans ces cadres bruités, mais mettent en avant d'autres classes de langages, transversaux à la hiérarchie de Chomsky. Par exemple, Stephan puis Case, Jain et Sharma [Ste97, CJS01] montrent qu'au travers un bruit non statistique, certaines classes récursivement énumérables sont identifiables.

Une classe de langages apprenable en situations bruitées : les boules de mots

La situation est donc la suivante : d'un côté, il est devenu crucial de pouvoir apprendre à partir de données bruitées ; d'un autre, la plupart des travaux désirant identifier les langages rationnels, langages de base de la hiérarchie de Chomsky, échouent dans leur tâche.

Nous proposons donc dans ce travail ce que pourrait être la base d'une hiérarchie de classes de langages identifiables en présence de données bruitées : les langages topologiques.

Supposons que nous voulions classer des images. S'il est sans aucun doute difficile pour un non-initié de différencier une pipistrelle commune d'une pipistrelle de Kuhl, il est facile de différencier un rectangle d'un disque, et ce, même si les images sont floutées ou en mosaïque (voir Figure 1). En fait, toutes les images comprenant des formes géométriques simples peuvent aisément être identifiées avec n'importe quel type de modifications apportées à l'image.

Nous pensons qu'il en est de même pour les langages : les langages les plus simples, reposant sur des notions de distance et donc de topologie, doivent être reconnaissables, même en présence de bruit.

Nous nous intéresserons en particulier à la classe des boules de mots, une boule étant défini comme l'ensemble des mots dont la distance par rapport au centre est inférieure à un certain rayon. Contrairement à l'espace euclidien, les boules de mots ne sont pas des objets « sphériques » et parfaitement symétriques. Néanmoins, elles peuvent être apprises à partir de données bruitées.

De plus, les boules de mots sont en fait utilisées dans plusieurs autres travaux où elles ne sont que rarement nommées. Elles servent alors à effectuer des recherches approximatives de chaînes de caractères [SK83, Nav01], ce qui permet également de corriger des mots [SM02], mais aussi d'améliorer les performances des algorithmes de recherche de plus proches voisins [MOV94, BNC03].

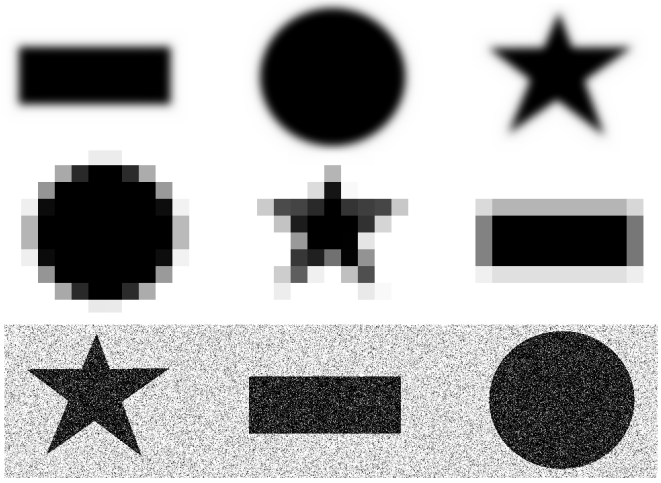

FIG. 1 – Figures géométriques bruitées.

Reste du manuscrit

Ce mémoire de thèse est composé de six chapitres. Le premier chapitre contient les définitions et notations élémentaires que nous utiliserons dans le reste du document. Nous y définissons entre autres la distance d'édition qui nous servira à calculer la distance entre deux chaînes de caractères, ainsi que la classe des langages rationnels qui sera notre point de comparaison pour l'apprentissage à partir de données bruitées.

Dans le chapitre 2, nous présentons les cadres d'apprentissage « standards » de l'inférence grammaticale. Nous y définissons en particulier la notion de requêtes de correction. Nous montrons alors comment apprendre les langages rationnels dans les différents paradigmes introduits.

Le chapitre suivant représente le cœur de la thèse : nous y expliquons dans un premier temps ce qu'apprendre à partir de données bruitées signifie. Puis nous montrons que les langages rationnels, bases de la hiérarchie de Chomsky, ne sont pas résistants au bruit. S'en suit alors une discussion sur la nécessité de considérer une nouvelle classe de langage comme point de départ pour une étude de l'apprentissage en situations bruitées.

Nous donnons dans le chapitre 4 la définition de plusieurs classes de langages basés sur la distance d'édition dont nous pensons que l'apprentissage à partir de données bruitées doit être possible. Nous étudions alors plus en détails les propriétés de l'une d'entre elles, les boules de mots.

Dans les deux derniers chapitres, nous montrons que la classe des boules de mots peut être apprise à partir de données bruitées. Tout d'abord nous exhibons deux techniques d'identification à la limite à partir de données bruitées systématiquement pour lesquelles cet apprentissage est possible. Puis nous terminons en proposant un modèle d'oracle pouvant répondre approximativement à des requêtes de correction. Nous montrons alors qu'un algorithme d'identification des boules de mots peut être facilement adapté pour faire face à ce type d'oracle.

Enfin, dans la conclusion, nous discutons des différents résultats présentés et donnons quelques perspectives possibles de recherche à ce travail.

Chapitre 1

Définitions

Le but de ce chapitre est d'introduire les différentes notations et définitions nécessaires à la compréhension du reste du manuscrit. Nous rappelons les définitions usuelles de la théorie des langages (par exemple les automates) puis nous nous intéressons aux distances entre chaînes de caractères, et en particulier à la distance d'édition.

1.1 Mot et ordre

Nous allons parler de langages formels. Dans la vie courante, un « langage » est un ensemble de phrases qui sont constituées de mots ordonnés suivant certaines règles (afin que la phrase appartienne au langage). Dans la théorie des langages, nous parlons de lettres et de mots pour parler respectivement de mots et de phrases.

Définition 1 (Alphabet et mot) *Un* alphabet Σ *est un ensemble fini non vide de symboles appelés* lettres. *Nous supposerons par la suite que* $|\Sigma| \geq 2$*, sauf précisé autrement. Un* mot u *(encore appelé chaîne ou séquence) est une suite finie* $u = a_1 a_2 \dots a_n$ *de lettres de* Σ*. Le mot vide (ne contenant aucune lettre) sera désigné par* λ*.*

Exemple 1 $ba, bbbbb, \lambda$ *et* a *sont quatre mots définis à partir d'un alphabet contenant au moins deux lettres, par exemple* $\Sigma = \{a, b\}$*.*

L'ensemble de tous les mots (y compris le mot vide) pouvant être construits à partir de Σ est noté Σ^*. Soit k un entier, Σ^k, $\Sigma^{\leq k}$ et $\Sigma^{>k}$ désignent respectivement l'ensemble des mots de longueur k, de longueur inférieure ou égale à k et de longueur strictement supérieure à k.

Nous notons $|u|$ la longueur du mot u, et $|u|_a$ désigne le nombre d'occurrences d'une lettre a dans u.

Exemple 2 *Les mots* $u = abba$ *et* $v = aaaa = a^4$ *sont de longueur* $|u| = |v| = 4$*, et appartiennent tous les deux à* Σ^4*,* $\Sigma^{\leq 10}$ *ou encore* $\Sigma^{>2}$*. Ils vérifient* $|u|_a = |u|_b = 2$*,* $|v|_a = 4$ *et* $|v|_b = 0$*.*

Définition 2 (Sous-mots et facteurs) *On dit qu'un mot u est un sous-mot de v, noté $u \preceq v$, si u est défini par $u = a_1 a_2 \dots a_n$ et s'il existe $n+1$ mots $u_0, u_1, \dots, u_n \in \Sigma^*$ tels que $v = u_0 a_1 u_1 \dots a_n u_n$.*

$\preceq$ est un ordre partiel sur Σ^*. De plus, s'il existe deux mots $u_1, u_2 \in \Sigma^*$ tels que $u_1 u u_2 = v$, alors u est un facteur de v.

Exemple 3 *Soient les mots $u = aba, v = aabaa$ et $w = ababbaa$. u est alors un sous-mot de v qui est lui-même un sous-mot de w : $u \preceq v \preceq w$. De plus, u est un facteur de v et de w, mais v n'est pas un facteur de w.*

Nous noterons $lcs(u, v)$ (*longest common subsequences*) l'ensemble des plus longs sous-mots communs à u et v.

Exemple 4 *Soient les mots $u = abba$ et $v = baab$, alors $lcs(u, v) = \{aa, ab, ba, bb\}$.*

Enfin, l'ordre que nous utiliserons pour trier les mots est l'ordre hiérarchique (ou encore *length-lex*) :

Définition 3 (Ordre hiérarchique) *L'ordre hiérarchique $<$ est défini par : $\forall u, v \in \Sigma^*, u < v \iff (|u| < |v|)$ ou $(|u| = |v|$ et $u <_{lex} v)$ où $<_{lex}$ est la relation d'ordre lexicographique, c'est-à-dire l'ordre utilisé dans le dictionnaire. Nous supposerons qu'il existe un ordre sur l'alphabet Σ : par exemple, $a < b < c < \dots$*

Exemple 5 *Soit l'alphabet $\Sigma = \{a, b\}$. Les mots de Σ^*, classés par ordre croissant selon l'ordre hiérarchique, sont λ, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, ...*

1.2 Langages et représentations standards

1.2.1 Définitions

Définition 4 (Langage) *On appelle langage toute partie $L \subseteq \Sigma^*$, c'est-à-dire tout sous-ensemble, fini ou infini, de Σ^*.*

Exemple 6 *Soit Σ un alphabet contenant au moins les lettres a et b. Les langages $\Sigma^{\leq 100}$ et $\{ab, ba, aab, baa\}$ sont des langages finis alors que $\Sigma^{>2}$ et $\{w \in \Sigma^* : |w|_b = 0\}$ (l'ensemble de tous les mots ne contenant pas de b) sont des langages infinis.*

Étant donné un ensemble fini de mots X, nous noterons $\|X\|$ la somme des longueurs des mots de X. La notation $|\cdot|$ est utilisée pour la cardinalité des ensembles.

Étant donnés L_1 et L_2 deux langages définis sur un alphabet Σ, nous pouvons alors définir :

- l'union des deux langages : $L_1 \cup L_2$ correspondant à l'ensemble des mots de Σ^* appartenant à L_1 ou L_2,
- l'intersection des deux langages : $L_1 \cap L_2$ correspondant à l'ensemble des mots de Σ^* appartenant à L_1 et L_2,

- la différence symétrique des deux langages : $L_1 \oplus L_2$ correspondant à l'ensemble des mots de Σ^* appartenant soit à L_1, soit à L_2 (mais pas aux deux en même temps), et
- la concaténation des deux langages : $L_1 \cdot L_2$ correspondant à l'ensemble des mots $w = uv$ de Σ^* dont u appartient à L_1 et v appartient à L_2.

1.2.2 Grammaires

Les langages sont représentables de plusieurs manières. L'une d'entre elles est sous forme de grammaire :

Définition 5 (Grammaire) *Une grammaire formelle est un quadruplet $\langle \Sigma, V, P, S \rangle$ où Σ est l'alphabet des terminaux (ou lettres), V l'alphabet des non-terminaux, $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ un ensemble de règles de production et $S \in V$ l'axiome.*

Au milieu des années 1950, Noam Chomsky a hiérarchisé quatre grandes classes de grammaire [Cho57]. Ces classes sont définies par des contraintes sur les règles de production :

Définition 6 (Hiérarchie de Chomsky)

Les grammaires de type 0 n'ont aucune contrainte sur les règles de production.
Les grammaires de type 1 (ou grammaires sensibles au contexte, en anglais context-sensitive*) ont des règles de production qui ne contiennent qu'un seul non-terminal en partie gauche et une partie droite différente de λ.*
Les grammaires de type 2 (ou grammaires hors-contextes, grammaires algébriques, ou encore grammaires context-free*) ont des règles de production dont les parties gauches sont formées d'un unique non terminal.*
Les grammaires de type 3 (ou grammaires régulières, grammaires rationnelles) ont des règles de production formées d'un non terminal en partie gauche et soit d'une unique lettre, soit d'une lettre puis d'un non terminal en partie droite.

À chaque type de grammaire correspond une classe de langages (à λ près). Clairement, un langage représentable par une grammaire de type 3 l'est par une grammaire de type 2, et donc 1 et bien sûr 0.

Un langage est dit régulier s'il est représentable par une grammaire régulière. De même, il est dit purement hors-contexte s'il peut être représenté par une grammaire hors-contexte, mais pas par une grammaire régulière.

1.2.3 Automates

Définition 7 (Automate fini) *Un automate fini est un quintuplet $A = \langle \Sigma, Q, I, F, \delta \rangle$ où :*

- *Σ est un alphabet fini,*
- *Q est un ensemble fini d'états,*
- *$I \subseteq Q$ est un ensemble d'états dits états initiaux,*

- $F \subseteq Q$ *est un ensemble d'états dits états finaux ou acceptants, et*
- δ *est une fonction de transition de* $Q \times \Sigma \to 2^Q$.

L'automate est dit fini car il possède un nombre fini d'états. Ce nombre d'états sert généralement comme taille de l'automate. Nous définissons donc la taille d'un automate A comme étant $|A| = |Q|$.

Nous étendons la fonction de transition à une fonction de transition sur les mots $\delta : Q \times \Sigma^* \to 2^Q$. On dit que l'automate A accepte le mot $w \in \Sigma^*$ si $\exists q \in I, \delta(q, w) \cap F \neq \emptyset$. Dans le cas contraire, on dit qu'il le rejette. Le langage reconnu par A est $L(A) = \{w \in \Sigma^* | \exists q \in I, \delta(q, w) \in F\}$

Exemple 7 *Soit l'automate* $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ *(voir Figure 1.1). A reconnaît le langage* a^+b^+. *La taille de l'automate est de* $|A| = |Q| = 3$.

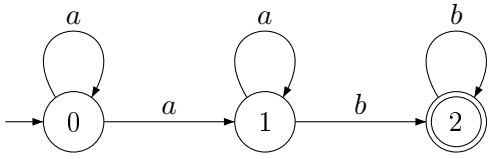

FIG. 1.1 – Automate reconnaissant le langage a^+b^+.

Les langages dit *réguliers* sont reconnus par des automates finis déterministes.

Définition 8 (Automate fini déterministe (AFD)) *Un automate fini déterministe est un automate fini tel que :*

- $I = \{q_0\}$, *c'est-à-dire qu'il n'y a qu'un état initial, et*
- $\forall q \in Q, \forall a \in \Sigma, |\{q' : \delta(q, a) = q'\}| \leq 1$, *c'est-à-dire que pour chaque état, il existe au plus une transition sortante par lettre de* Σ.

Autrement dit, un automate fini est déterministe, si quelque soit $w \in \Sigma^*$, il existe une unique façon d'accepter ou de rejeter w. L'automate donné Figure 1.1 n'est pas déterministe. En effet, le mot *aab* peut être accepté (ou *parsé*) de plusieurs façons :

- soit en lisant le premier a par la boucle sur q_0, puis le second par la transition de q_0 à q_1,
- soit en lisant le premier a par la transition de q_0 à q_1, puis le second par la boucle sur q_1.

Pour chaque automate fini non déterministe, il existe un automate fini déterministe reconnaissant exactement le même langage.

Exemple 8 *Si nous supprimons la boucle sur l'état* 0 *de l'automate de la Figure 1.1, nous obtenons un automate fini déterministe reconnaissant le même langage* a^+b^+.

L'étape pour passer d'un AFN (automate fini non déterministe) à un AFD est appelée *déterminisation*. De plus, pour chaque langage reconnu par un AFD, nous pouvons construire un AFD *minimal* en nombre d'états. Cet AFD minimal alors unique est appelé automate canonique. Minimiser un automate est un problème polynomial. Nous pouvons donc déterminer si deux AFD sont équivalents, c'est-à-dire s'ils reconnaissent le même langage, en les minimisant : ils le sont si les automates canoniques sont égaux.

1.3 Distances entre mots

Une notion importante que nous allons utiliser est celle de distance. Si lorsque nous parlons de distance dans la vie courante, il est établi que ce soit la distance euclidienne, qu'en est-il des distances entre mots ? Par exemple, le mot *carré* est-il plus proche du mot *rond* ou du mot *rectangle* ? Tout dépend en fait de la distance utilisée. Au préalable, rappelons la définition mathématique d'une distance :

Définition 9 *On appelle distance sur un ensemble E une application $d : E \times E \to \mathbb{R}_+$ vérifiant les propriétés suivantes :*

1. *Séparation : $d(x, x) = 0$*
2. *Symétrie : $d(x, y) = d(y, x)$*
3. *Non négativité : $d(x, y) \geq 0$*

Lorsque la distance vérifie en plus les deux conditions suivantes, elle définit une métrique[1] :

1. Identité des indiscernables : $d(x, y) = 0 \implies x = y$
2. Inégalité triangulaire : $d(x, y) + d(y, z) \geq d(x, z)$

Exemple 9 *Soient x et y deux vecteurs tels que $x = [x_1, \ldots, x_n]$ et $y = [y_1, \ldots, y_n]$, avec $\forall i \leq n$, $x_i \in \mathbb{N}$. Des exemples typiques de distance sont alors :*

- *la distance discrète, définie par* $d(x, y) = \begin{cases} 0 & \text{si } x = y \\ 1 & \text{sinon} \end{cases}$
- *la distance euclidienne, définie par* $\sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$
- *la distance de Minkowski, définie par* $\sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$ *avec $p \geq 1$ (ou p-norme). La distance euclidienne est alors la 2-norme.*

Ces distances sont largement utilisés sur des données numériques (vectorielles). Elles sont en revanche la plupart du temps inutilisables en l'état sur des données structurées comme les arbres, les graphes ou les chaînes.

Calculer la distance entre deux données structurées revient la plupart du temps à calculer la distance entre les structures. Bien souvent, on se donne un ensemble d'opérations réalisables et on compte le nombre minimum d'opérations à effectuer pour passer

[1] En réalité, une métrique au sens mathématique du terme doit vérifier (i) $d(x, y) \geq 0$,(ii) $d(x, y) = 0 \Leftrightarrow x = y$,(iii) $d(x, y) = d(y, x)$ et (iv) $d(x, z) \leq d(x, y) + d(y, z)$.Telle qu'on la définit, c'est-à-dire en remplaçant (ii) par $d(x, x) = 0$, l'application définit ce que l'on appelle un *écart* sur l'ensemble E induisant non plus un espace métrique mais un espace *pseudométrique* [Kel55].

d'une structure à une autre. Cependant, plus la structure est complexe, plus la distance est difficile à calculer : le calcul de distance entre deux graphes consiste simplement à supprimer et à ajouter des sommets et des arêtes pour passer d'un graphe à l'autre. Cependant, calculer cette distance peut alors être exponentiel dans le nombre de nœuds des graphes...

En revanche, calculer la distance entre deux chaînes se fait généralement en temps polynomial dans la longueur de ces chaînes.

1.3.1 Distance d'édition

Une distance fréquemment utilisée pour comparer deux chaînes est la distance d'édition, introduite en 1965 par Levenshtein [Lev65]. Cette distance utilise trois opérations, appelées opérations d'édition (ou opérations atomiques) :

- *l'opération d'effacement, ou de suppression* : elle transforme un mot $w = uav$ en $w' = uv$ avec $u, v \in \Sigma^*$ et $a \in \Sigma$;
- *l'opération d'insertion* : opération inverse de la suppression, elle transforme un mot $w = uv$ en $w' = uav$ avec $u, v \in \Sigma^*$ et $a \in \Sigma$;
- *l'opération de substitution* : pouvant s'apparenter à une suppression suivie d'une insertion, elle transforme un mot $w = uav$ en $w' = ubv$ avec $u, v \in \Sigma^*$, $a, b \in \Sigma$.

À chaque opération d'édition est attribuée un coût. La distance d'édition est alors définie comme la somme minimale des coûts des opérations permettant de passer d'un mot à un autre.

Il existe plusieurs variantes à la distance d'édition. Des variantes utilisant des transposition de deux lettres, voire des déplacements de sous-mots [CM07, SS07], et des variantes utilisant différents poids, comme par exemple la distance de Hamming [Ham50] qui n'utilise que les opérations de substitutions et qui est donc un cas particulier de la distance d'édition avec un coût infini pour les opérations d'effacement et d'insertion, et un coût unitaire pour l'opération de substitutions.

Dans certaines applications, comme en biologie par exemple, il est nécessaire d'avoir des poids d'édition différents, voire dépendants du contexte. Par exemple, une insertion en milieu de mot peut être plus coûteuse qu'en fin mot, le remplacement d'un a par un b être plus cher que celui d'un b par un c... Certains travaux visent donc à *apprendre* les poids de la distance d'édition avant de l'utiliser [RY96, BJS06, OS06]. D'autres enrichissent la distance d'édition en autorisant des opérations plus complexes comme substituer deux lettres par une seule, par exemple changer ph par f [BM00].

Sauf indications contraires, nous considérerons dans la suite du manuscrit que le coût des trois opérations d'édition est unitaire. Dans ce cas, nous pouvons alors redéfinir la distance d'édition comme suit :

Définition 10 (Distance d'édition) *La distance d'édition entre deux mots w et w', notée $d(w, w')$, est le nombre minimum d'opérations d'édition nécessaires pour réécrire w en w'.*

Exemple 10 *Soient $w = babab$ et $w' = abbaa$. $d(w, w') = 3$. En effet, pour passer du mot w à w' il suffit de supprimer le premier a de w, de l'insérer en tout début de mot*

et de remplacer le dernier b par un a. Comme il n'existe aucune opération permettant de passer directement de w à w′, la distance d'édition entre le mot w = babab et le mot w′ = abbaa est donc de trois.

Une propriété de la distance d'édition que nous utiliserons souvent stipule que la distance entre deux mots est au moins égale aux nombres d'insertions nécessaires pour égaliser les longueurs de ces deux mots :

Proposition 1 *Pour tous mots* $w, w' \in \Sigma^*$, $d(w,w') \geq \big||w| - |w'|\big|$. *De plus,* $d(w,w') = \big||w| - |w'|\big|$ *si et seulement si* ($w \preceq w'$ *ou* $w' \preceq w$).

Pour calculer la distance entre deux mots, il existe souvent plusieurs suites d'opérations possibles ayant un coût minimum. Dans l'exemple 10 ci-dessus, nous pouvons également utiliser trois substitutions pour passer de w à w' : substituer le premier b et le dernier b par des a puis le premier a par un b. La séquence d'opérations effectuées pour passer d'un mot à l'autre est alors appelée *script d'édition*.

Définition 11 (Script d'édition) *Le script d'édition est une suite de couples* (a_i, b_i) *tels que*

1. *chaque couple représente :*
   - *la suppression de la lettre* a_i *si* $b_j = \lambda$,
   - *l'insertion de la lettre* b_j *si* $a_i = \lambda$ *et*
   - *la substitution de la lettre* a_i *par* b_j.
2. *les concaténations* $a_0a_1a_2\cdots$ *et* $b_0b_1b_2\cdots$ *sont respectivement égales à w et w′, un script d'édition étant exécuté de gauche à droite : le script* $(a,b)(a,\lambda)$ *transforme donc le premier a de aa en b et supprime le second a.*

Exemple 11 *Pour passer de* $w = babab$ *à* $w' = abbaa$, *il existe quatre scripts d'édition permettant de calculer* $d(w,w')$:
- $(b,a),(a,b),(b,b),(a,a),(b,a)$
- $(b,\lambda),(a,a),(\lambda,b),(b,b),(a,a),(b,a)$
- $(\lambda,a),(b,b),(a,\lambda),(b,b),(a,a),(b,a)$
- $(b,\lambda),(a,a),(b,b),(\lambda,b),(a,a),(b,a)$

Une façon plus visuelle mais équivalente permettant de représenter la suite d'opération utilisée dans le calcul de la distance d'édition s'appelle l'*alignement*. On place des « espaces » à l'intérieur (ainsi qu'aux extrémités si nécessaire) de w et w' et on les place l'un au-dessus de l'autre afin qu'ils s'alignent :

Exemple 12 *Le script d'édition* $(b,\lambda),(a,a),(\lambda,b),(b,b),(a,a),(b,a)$ *est équivalent à l'alignement :*

$$\begin{array}{cccccc} b & a & \lambda & b & a & b \\ & | & & | & | & | \\ \lambda & a & b & b & a & a \end{array}$$

Une autre représentation, celle que nous utiliserons le plus souvent, s'inspire des dérivations de réécriture. Étant donnés deux mots $w, w' \in \Sigma^*$, on dit que w se réécrit en w' en un pas, noté $w \rightarrow w'$ si une des opérations d'édition a été appliquée. On note alors $w \xrightarrow{k} w'$ si w peut se réécrire en w' à l'aide de k opérations d'édition et $w \xrightarrow{*} w'$ la fermeture transitive et réflexive de la dérivation : $\xrightarrow{*} = \bigcup_{k \geq 0} \xrightarrow{k}$. Dans ce cas[2], la distance d'édition est le plus petit entier $k \in \mathbb{N}$ tel que $w \xrightarrow{k} w'$.

Exemple 13 *Le script d'édition* $(b,\lambda),(a,a),(\lambda,b),(b,b),(a,a),(b,a)$ *peut être représenté par la dérivation* $babab \rightarrow abab \rightarrow abbab \rightarrow abbaa$. *Il est à noter qu'une même dérivation peut représenter plusieurs scripts d'édition. Ainsi* $babab \rightarrow abab \rightarrow abbab \rightarrow abbaa$ *représente également* $(b,\lambda),(a,a),(b,b),(\lambda,b),(a,a),(b,a)$.

1.3.2 Programmation de la distance d'édition

Il existe plusieurs façons de programmer le calcul de la distance d'édition. La plupart visent à améliorer la complexité de l'algorithme général reposant sur la formule suivante :

$$d(ua,vb) = min \begin{cases} d(ua,v) & + & \text{cout(insertion)} \\ d(u,v) & + & \text{cout(substitution)} \\ d(u,vb) & + & \text{cout(suppression)} \end{cases} \tag{1.1}$$

où cout(insertion), cout(substitution), cout(suppression) représentent respectivement le coût d'une insertion, d'une substitution et d'une suppression dans le cas plus général où le coût de chaque opération peut être différent. Les algorithmes que nous allons donner seront généralement définis pour cout(insertion) = cout(substitution) = cout(suppression) = 1.

L'Algorithme 1 est l'algorithme de base du calcul de la distance d'édition entre deux mots u et v [WF74]. Son but est de remplir une matrice M de telle façon que $M\big[|u|+1\big]\big[|v|+1\big]$ contienne la distance d'édition $d(u,v)$, en calculant la distance d'édition entre chaque préfixe de u et v.

Par exemple, le calcul de la distance d'édition entre $u = babab$ et $w = abbaa$ par l'Algorithme 1 permet de remplir M de la manière suivante :

		u					
		$\lambda(0)$	$a(1)$	$b(2)$	$b(3)$	$a(4)$	$a(5)$
v	$\lambda(0)$	0	1	2	3	4	5
	$b(1)$	1	1	1	2	3	4
	$a(2)$	2	1	2	2	2	3
	$b(3)$	3	2	1	2	3	3
	$a(4)$	4	3	2	2	2	3
	$b(5)$	5	4	3	2	3	3

On comprend alors clairement pourquoi l'algorithme est correct : son invariant est qu'il permet de passer du mot $u_1 \cdots u_i$ au mot $v_1 \cdots v_j$ en utilisant un nombre minimal de $M[i][j]$ opérations d'édition (grâce à l'équation 1.1).

[2] En considérant que les poids des opérations valent 1.

Algorithme 1 : Algorithme général du calcul de la distance d'édition

Données : Deux mots u et v
Résultat : $d(u,v) = M\big[|u|+1\big]\big[|v|+1\big]$

$M[0][0] \leftarrow 0$;
`// Initialisation de la colonne`
pour $i \leftarrow 0$ **à** $|u|$ **faire**
  $M[i+1][0] \leftarrow M[i][0] + 1$;
finpour
`// Initialisation de la ligne`
pour $i \leftarrow 0$ **à** $|v|$ **faire**
  $M[0][i+1] \leftarrow M[0][i] + 1$;
finpour
pour $i \leftarrow 0$ **à** $|u|$ **faire**
  pour $j \leftarrow 0$ **à** $|v|$ **faire**

$$M[i+1][j+1] \leftarrow \min \begin{cases} M[i][j+1] + \text{cout(insertion)} \\ M[i][j] + \text{cout(substitution)} \\ M[i][j+1] + \text{cout(suppression)} \end{cases} ;$$

  finpour
finpour

1.3.3 Extensions de la distance d'édition

Bien qu'utilisant la programmation dynamique, l'Algorithme 1 a une complexité temporelle et spatiale de $(|u|+1) \times (|v|+1)$ (due au remplissage du tableau).

Cette complexité peut alors être trop importante si nous travaillons sur de très longues chaînes. Une première amélioration plutôt triviale concerne alors la complexité spatiale. En effet, plutôt que de garder en mémoire tout le tableau, seule la dernière ligne remplie est utile pour compléter la suivante. On peut alors adapter l'algorithme pour travailler avec un tableau à deux lignes et ainsi avoir une taille de $2 \times \big(1 + \min(|u|, |v|)\big)$.

Une autre amélioration concerne la complexité temporelle. Cette idée vient des auteurs Arlazarov, Dinic, Kronrod et Faradzev [ADKF70] et a été adaptée par Masek et Paterson [MP80, MP83]. La technique, dite des quatre Russes, consiste à partitionner le tableau M en blocs de taille $m \times m$, et à passer seulement un temps en $\mathcal{O}(m)$ plutôt qu'en $\mathcal{O}\big(m^2\big)$ sur chacun des blocs. Pour cela, elle part du constat que chaque case dans M ne peut différer qu'au maximum d'une valeur de 1 avec une case adjacente. L'amélioration consiste alors à coder les blocs avec des vecteurs de valeurs dans $\{-1, 0, 1\}$ dans les premières et dernières lignes et colonnes de chaque bloc. Un pré-calcul sur toutes les façons d'obtenir un bloc permet d'arriver au résultat. Au final, la technique des quatre Russes permet de calculer la distance d'édition entre deux mots de longueurs n en un temps $\mathcal{O}\left(\frac{n^2}{\log n}\right)$ grâce à un pré-calcul en $\mathcal{O}\big(n(\log n)^2\big)$.

Il peut également être intéressant d'utiliser d'autres algorithmes dans certains cas. Par exemple dans [Ukk83], Ukkonen présente un algorithme de complexité moyenne en

$\mathcal{O}\big(|u| + d(u,v)^2\big)$ et de pire complexité en $\mathcal{O}\big(|u| \times d(u,v)\big)$. Du coup, lorsque $d << n$, c'est-à-dire lorsque nous savons que les mots auront une faible distance par rapport à leur longueur (lorsque nous comparons une chaîne d'ADN et sa mutation par exemple), cet algorithme sera bien plus rapide que l'Algorithme 1. De même, dans [All92] Allison tire avantage de la programmation fonctionnelle[3] pour donner un algorithme en $\mathcal{O}\big(|u| \times (1+d(u,v))\big)$. Dans les deux cas, au lieu de remplir le tableau M ligne à ligne, les auteurs s'intéressent à la diagonale du tableau. La complexité spatiale est alors en $\mathcal{O}\big(d(u,v)^2\big)$.

Le lecteur désireux d'en savoir plus sur le calcul de la distance d'édition pourra consulter des ouvrages comme [Gus97, CHL01].

Il est également intéressant de noter qu'en gardant en mémoire un pointeur sur la case d'où provient le résultat de $M[i][j]$, on peut retrouver les scripts d'édition ainsi que les alignements en temps polynomial.

Outre dans les travaux cités ci-dessus, la distance d'édition entre chaînes est utilisée dans diverses applications. Elle peut servir en tant qu'aide à la reconnaissance de polygones lorsqu'elle est utilisée sur des mots cycliques [Mae91], d'esquisses faites à la main [LTZ96], d'empreintes digitales [JPHP00] ou plus simplement de caractères manuscrits [CSS99].

Des travaux de biologie l'emploient aussi pour calculer des distances entre structures ARN [JLMZ02] ou pour comparer des génomes [San92]. En linguistique, elle aide à la découverte de morphèmes [HMGS05], à la correction orthographique de langages agglutinatifs [OG94] et elle permet de définir des analogies entre séquences [MBD05]. On la trouve également en comparaison ou recherche de musiques [LU00], de vidéos [ALK99], d'ontologies [MS02] ou de fouille du web [GG08].

Enfin, elle peut aussi être étudiée en tant que distance. Dans [OR07] par exemple, les auteurs montrent que la distance d'édition peut être *ramenée* à une distance de Manhattan, lorsqu'elle est utilisée sur des mots binaires, avec une faible distortion. Des travaux plus classiques se servent de ses propriétés de distance pour de la recherche de plus proches voisins [MOV94, Ind04].

[3] Plus particulièrement de l'évaluation paresseuse (*lazy programming*).

Chapitre 2

Les cadres d'apprentissage

En inférence grammaticale, apprendre revient à présenter une hypothèse consistante avec les données qui sont portées à notre connaissance. Afin de savoir si nous avons « bien » appris, il convient de connaître ce que nous voulions apprendre, à partir de quoi, et combien de « temps » a été nécessaire pour formuler nos hypothèses.

La qualité du résultat d'apprentissage va fortement dépendre des exemples que l'algorithme aura. Par exemple, s'il ne voit qu'un ensemble fini de données positives, il va être difficile de généraliser : si un apprenant ne généralise pas suffisamment, il risque un apprentissage *par cœur*, c'est-à-dire reconnaître uniquement les données déjà vues, à l'inverse s'il généralise trop, il risque de retourner rapidement des hypothèses acceptant pratiquement tous les mots de Σ^*.

Dans le reste du manuscrit, nous allons nous intéresser à l'apprentissage de diverses classes de langages. Ce qui nous intéresse en particulier est l'apprentissage de langages à partir de données bruitées. En effet, en pratique, les données dont nous disposons sont bruitées, c'est-à-dire que parmi les données d'apprentissage, certaines ont subi des modifications. Afin de formaliser l'apprentissage à partir de données bruitées, nous allons au préalable nous intéresser à ce que signifie apprendre.

L'apprentissage se fait à partir de données et un apprenant peut les obtenir de deux moyens différents : soit une séquence d'informations existe et l'apprenant peut les prendre une par une, soit il peut demander de l'information à un oracle. Pour ces deux moyens, l'évaluation de l'apprentissage peut se faire d'un point de vue probabiliste (sommes-nous plus ou moins proche de la réponse ?), ou d'un point de vue asymptotique (avons-nous oui ou non la bonne réponse ?).

Dans ce chapitre nous nous intéresserons dans un premier temps aux objets à apprendre. Nous étudierons ensuite les trois paradigmes habituels d'apprentissage caractérisant quelles sont les « bonnes » façons d'apprendre :

- l'identification à la limite : les données sont subies et l'apprenant doit trouver la bonne réponse ;
- l'apprentissage actif ou apprentissage à partir de requêtes : l'apprenant choisit ses données via un oracle et l'apprenant doit trouver exactement la réponse ;
- l'apprentissage PAC : l'apprenant peut choisir ses données ou les subir, mais l'éva-

luation de l'apprentissage se fait de façon probabiliste.

Enfin, nous verrons comment apprendre les AFD dans ces différents cadres.

2.1 Classes de langages et représentations

Lorsque nous voulons identifier un langage, nous avons le choix entre plusieurs langages. Nous allons donc identifier la cible parmi une *classe de langages*. De plus, les hypothèses que nous émettons ne sont pas retournées sous formes de langages mais sous formes de représentations.

Soient $\mathcal{L}$ une classe de langages et $\mathcal{R}(\mathcal{L})$ une classe de représentations des langages de $\mathcal{L}$. Typiquement, les classes de langages que nous considérerons seront par exemple les langages réguliers ou algébriques, et les classes de représentations seront les grammaires algébriques ou les automates finis déterministes.

Les classes de langages $\mathcal{L}$ et de représentations $\mathcal{R}(\mathcal{L})$ sont reliées par une fonction dite de *nommage* $\mathbb{L} : \mathcal{R}(\mathcal{L}) \to \mathcal{L}$ qui pour n'importe quelle représentation retourne le langage correspondant. Cette fonction est une fonction qui est totale, c'est-à-dire $\forall G \in \mathcal{R}(\mathcal{L}), \mathbb{L}(G) \in \mathcal{L}$. De plus, elle est surjective : $\forall L \in \mathcal{L}, \exists G \in \mathcal{R}(\mathcal{L})$ *tel que* $\mathbb{L}(G) = L$. Il est à noter qu'elle n'est cependant pas nécessairement injective. En effet, un même langage peut être représenté de plusieurs façons différentes.

Pour n'importe quels mots $w \in \Sigma^*$ et langage L, nous écrirons $L \models w$ si $w \in L$. De même, comme les grammaires peuvent être vues comme un ensemble d'informations permettant à un analyseur, en anglais *parser*, de reconnaître des mots, pour n'importe quel mot $w \in \Sigma^*$ et grammaire G, nous écrirons $G \vdash w$ si l'analyseur reconnaît w. La syntaxe et la sémantique sont alors liées par la fonction de nommage : $G \vdash w \iff \mathbb{L}(G) \models w$.

Nous allons également nous intéresser par la suite à des problèmes de complexité : quels langages sont apprenables en peu de temps ? Quels sont ceux qui ne le sont pas même avec beaucoup de données ? Pour cela il nous faut pouvoir mesurer la *taille* des représentations et en particulier, les tailles des grammaires. Cette taille doit être raisonnable dans le sens où un langage possédant peu de données ou pouvant être décrit simplement, doit avoir une taille plus petite qu'un langage exponentiellement plus grand. Généralement, la taille d'une grammaire G est polynomialement liée au nombre de bits nécessaires pour encoder G. Nous utiliserons $\|G\|$ pour noter cette taille. En ce qui concerne les automates, leur taille sera leur nombre d'états.

2.2 Identification à la limite

Dans le paradigme standard d'identification à la limite de Gold, un apprenant reçoit une séquence infinie d'informations. Cette séquence doit l'aider à trouver exactement la représentation $G \in \mathcal{G}$ d'un langage cible inconnu $L \in \mathcal{L}$.

2.2.1 Notations préliminaires

Pour l'apprenant, tout se passe comme si une quantité infinie de données arrivent sans discontinuer. Cette suite d'information s'appelle une *présentation* :

Définition 12 (Présentation) *Soit $\mathcal{L}$ une classe de langages, une* présentation *de $L \in \mathcal{L}$ est une fonction $\mathbb{N} \to X$ où X est un ensemble.*

Étant donnée une présentation $\boldsymbol{f}$, nous noterons $\boldsymbol{f}_m$ l'ensemble $\{\boldsymbol{f}(j) : j < m\}$ des $m+1$ premiers éléments de $\boldsymbol{f}$, et $\boldsymbol{f}(n)$ son n-ème élément.

Nous notons $\boldsymbol{Pres}(\mathcal{L})$ un ensemble de présentations. Une présentation dénote un langage de $\mathcal{L}$. En d'autres termes, il existe une fonction yield : $\boldsymbol{Pres}(\mathcal{L}) \to \mathcal{L}$. Si $L = yield(\boldsymbol{f})$ alors nous dirons que $\boldsymbol{f}$ est une présentation de L, ou $\boldsymbol{f} \in \boldsymbol{Pres}(L)$.

Avec cette définition, les présentations sont à prendre au sens large : ce sont des séquences de n'importe quel type d'informations pouvant aider à l'apprentissage du langage.

Typiquement, les données qui arrivent sont des mots étiquetés du langage cible, mais ce n'est pas toujours le cas.

Exemple 14 *Considérons le langage $L = a^+b^+$.*

- *Si les mots de la présentation sont uniquement des mots appartenant au langage cible, X vaut Σ^*. Si en outre, $yield(\boldsymbol{f}) = \boldsymbol{f}(\mathbb{N})$, alors la présentation est appelée un texte. Dans ce cas, nous noterons alors $\boldsymbol{Pres} =$* Texte.
  Par exemple, $\boldsymbol{f} = \{aab, abbbb, ab, aab, \ldots\}$ n'est composée que de mots appartenant à L. Si tous les mots de L sont présentés alors $\boldsymbol{f}$ est aussi un texte.
- *Lorsque la présentation est composée de paires étiquetées (w, l) où $(w \in L \Rightarrow l = +)$ et $(w \notin L \Rightarrow l = -)$, c'est-à-dire à la fois d'exemples positifs et d'exemples négatifs, nous parlerons d'un informateur. Nous avons alors $X = \Sigma^* \times \{+, -\}$ et si $\boldsymbol{f}(\mathbb{N}) = \mathbb{L}(G) \times \{+\} \cup \overline{\mathbb{L}(G)} \times \{-\}$, nous noterons $\boldsymbol{Pres} =$* Informateur.
  Soit $\boldsymbol{f}' = \{(aab, +), (bab, -), (aa, -), (abb, +), (aaba, -), \ldots\}$. $\boldsymbol{f}'$ est une présentation de L qui est considérée comme un informateur si tous les mots de Σ^* apparaissent.
- Texte *et* Informateur *sont habituellement les deux types de présentations utilisées. Toutefois, X peut aussi être plus restreint : cela peut être une suite de préfixes des mots du langages, des indications sur la place des lettres dans les mots, leur nombre etc.*

Nous noterons **Pres** = Presentation lorsque nous voudrons parler indifféremment d'un Texte ou d'un Informateur.

Ensuite, pour apprendre, nous aurons besoin d'un *algorithme d'apprentissage*, c'est-à-dire d'un programme qui utilisera une présentation afin de retourner des hypothèses sur le langage cible à apprendre. Ces hypothèses seront faites sous forme de représentations :

Définition 13 (Algorithme d'apprentissage) *Un* algorithme d'apprentissage $\boldsymbol{Alg}$ *est un programme prenant les n premiers éléments d'une présentation et retournant une représentation :*

$$\boldsymbol{Alg} : \bigcup_{\boldsymbol{f} \in \boldsymbol{Pres}(\mathcal{L}), i \in \mathbb{N}} \{\boldsymbol{f}_i\} \rightarrow \mathcal{R}(\mathcal{L})$$

Notez qu'étant données deux présentations f et g, si $f(\mathbb{N}) = g(\mathbb{N})$ alors $yield(f) = yield(g)$. En effet, si tel n'était pas le cas, $\mathcal{L}$ ne serait pas apprenable à partir de $\mathbf{Pres}(\mathcal{L})$: deux langages partageant à l'infini une même présentation ne peuvent pas être distingués l'un de l'autre. Il en découle que si $f \in \mathbf{Pres}(\mathcal{L})$ et $g : \mathbb{N} \rightarrow X$ telles que $g(\mathbb{N}) = f(\mathbb{N})$ alors $g \in \mathbf{Pres}(\mathcal{L})$.

2.2.2 Identification à la limite sans contraintes de polynomialité

L'identification à la limite réussit si à un certain rang de la présentation, l'algorithme arrive à retourner toujours la même hypothèse, celle correspondant au langage cible. La définition suivante est une modification de celle de [Gol78] :

Définition 14 *On dit que $\boldsymbol{\mathcal{G}}$ est identifiable à la limite à partir de $\boldsymbol{Pres}$ s'il existe un algorithme $\boldsymbol{Alg}$ tel que pour tout $G \in \boldsymbol{\mathcal{G}}$ et pour toute présentation $\boldsymbol{f}$ de $\mathbb{L}(G)$, il existe un rang n tel que pour tout $m \geq n$, $\boldsymbol{Alg}(\boldsymbol{f}_m) = \boldsymbol{Alg}(\boldsymbol{f}_n)$ et $\mathbb{L}(\boldsymbol{Alg}(\boldsymbol{f}_m)) = \mathbb{L}(G)$.*

L'identification à la limite conduit parfois à des résultats négatifs. En particulier :

Théorème 1 ([Gol67]) *Soit $\mathcal{L}$ une classe de langages, si $\mathcal{L}$ contient tous les langages finis et au moins un langage infini (on dit alors que la classe est super finie), alors $\mathcal{L}$ n'est pas apprenable.*

Toutefois, l'absence de contraintes conduit souvent à des résultats d'apprentissage positifs, mais à des algorithmes inutilisables en pratique. C'est pourquoi plusieurs auteurs ont essayé de définir une identification à la limite *polynomiale*, en introduisant différents critères d'efficacité et en les combinant.

2.2.3 Contrainte de temps de mise à jour

Afin de pouvoir utiliser concrètement les algorithmes d'apprentissage, il est nécessaire que ceux-ci n'utilisent qu'une petite quantité de temps à chaque nouvelle donnée reçue, sous peine de ne pas pouvoir connaître le résultat de l'apprentissage avant un long moment. Il est donc raisonnable de penser que la polynomialité doit concerner la quantité de temps dont l'algorithme dispose pour apprendre :

Définition 15 (Temps de mise à jour polynomial) *On dit qu'un algorithme $\boldsymbol{Alg}$ a un* temps de mise à jour polynomial *s'il existe un polynôme $p()$ tel que, pour chaque présentation $\boldsymbol{f}$ et chaque entier n, construire $\boldsymbol{Alg}(\boldsymbol{f}_n)$ nécessite un temps en $\mathcal{O}(p(\|\boldsymbol{f}_n\|))$.*

Cette contrainte de temps de mise à jour polynomial n'est pas cependant pas satisfaisante. Dans [Pit89], Pitt montre en effet que même avec cette contrainte, un apprenant pourrait résoudre un problème $\mathcal{NP}$-difficile : il lui suffirait en effet d'attendre patiemment tout en recevant des exemples. Lorsqu'il a accumulé un nombre exponentiel d'exemples, il disposerait alors d'une quantité de temps suffisante pour résoudre un problème $\mathcal{NP}$-difficile...

Plusieurs autres critères sont alors envisageables en plus de la contrainte de temps de mise à jour : la quantité d'exemples reçus, le nombre de fois où l'apprenant fait une fausse hypothèse ou encore le nombre de fois où il change d'avis.

2.2.4 Contrainte de changement d'avis

Le second critère de polynomialité que nous présentons concerne donc le nombre de fois où l'apprenant change d'avis (*Mind Changes*, noté Mc) [AS83] :

Définition 16 (Changement d'avis (Mc)) *Étant donnés un algorithme* $\boldsymbol{Alg}$ *et une présentation* $\boldsymbol{f}$*, on dit que* $\boldsymbol{Alg}$ change d'avis au temps n *si* $\boldsymbol{Alg}(\boldsymbol{f}_n) \neq \boldsymbol{Alg}(\boldsymbol{f}_{n-1})$*. On dit que* $\boldsymbol{Alg}$ *est* conservatif *s'il ne change jamais d'avis lorsque son hypothèse courante est consistante avec le nouvel élément présenté.*

Dans ce cas, l'apprentissage sera alors « bon » si le nombre de changement d'hypothèses n'est pas trop grand :

Définition 17 (Identification Mc polynomiale) *Un algorithme* $\boldsymbol{Alg}$ *identifie une classe* $\boldsymbol{\mathcal{G}}$ *à la limite en temps* Mc *polynomial si*

1. $\boldsymbol{Alg}$ *identifie* $\boldsymbol{\mathcal{G}}$ *à la limite,*
2. $\boldsymbol{Alg}$ *a un temps de mise à jour polynomial et*
3. $\boldsymbol{Alg}$ *fait un nombre polynomial de changement d'avis : il existe un polynôme* $p()$ *tel que, pour chaque grammaire* G *et chaque présentation* $\boldsymbol{f}$ *de* $\mathbb{L}(G)$, $\#\text{Mc}(\boldsymbol{f}) \leq p(\|G\|)$ *où* $\#\text{Mc}(\boldsymbol{f})$ *est le nombre de* Mc *:* $\#\text{Mc}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \boldsymbol{Alg}(\boldsymbol{f}_k) \neq \boldsymbol{Alg}(\boldsymbol{f}_{k+1})\}|$.

Notons que la dernière condition n'implique pas les deux autres.

Exemple 15 *Soient le langage* $S_n = \{a^i : n \leq i \leq n+9\}$ *et la classe de langages* $\mathcal{S}$ *comprenant tous les langages* $S_n, \forall n \in \mathbb{N}^+$*. Chaque langage comprend donc* 10 *mots et* $\forall i, j \in \mathbb{N}, 0 < i < j, S_i \not\subseteq S_j$*, puisque* a^i *n'est pas dans* S_j *et* a^{j+10} *n'appartient pas à* S_i*.*

Soit l'algorithme suivant :

1. *Lire la donnée* $\boldsymbol{f}(n)$*, soit* u *la plus petite des données de* $\boldsymbol{f}$ *et* v *la plus grande.*
2. *Si* $|v| = |u| + 9$ *retourner* $S_{|u|}$ *sinon, retourner* S_1*.*
3. *Aller en 1.*

Cet algorithme identifie à la limite $\mathcal{S}$ *en faisant un seul changement d'hypothèse. En effet, l'algorithme n'a pas besoin d'être consistant. Il peut donc attendre d'avoir les* 11 *mots du langages (tout du moins les deux bornes) pour formuler son hypothèse. De plus, il a un temps de mise à jour polynomial.* $\mathcal{S}$ *est donc identifiable à la limite en temps* Mc *polynomial.*

Bien évidemment, il n'est pas toujours possible de faire un unique changement d'hypothèse, par exemple pour certains langages infinis. Cet exemple montre cependant qu'il peut être possible de ne faire que peu de changements d'hypothèses même si la taille de la grammaire cible est grande : si les nouveaux exemples de la présentation ne sont pas consistants avec l'hypothèse courante, l'algorithme peut attendre jusqu'à ce qu'il juge d'avoir assez de données pour formuler une autre hypothèse. Si un algorithme « passe » jusqu'à voir tous les mots de Σ^*, il pourrait donc n'effectuer qu'un seul changement d'hypothèse tout en vérifiant les deux autres points de la Définition 17. En revanche, si le fait de changer d'hypothèse est coûteux, il sera pertinent d'utiliser ce critère de polynomialité pour apprendre.

Enfin, une notion que nous retrouverons souvent avec les changements d'hypothèses est celle du *témoin de minimalité* :

Définition 18 (Témoin de minimalité) *Soit $G \in \boldsymbol{\mathcal{G}}$ une grammaire telle que $L = \mathbb{L}(G)$. Soit $\boldsymbol{f} \in \boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$. Un* témoin de minimalité *pour $G \in \boldsymbol{\mathcal{G}}$ est un ensemble $X \subseteq \boldsymbol{f}(\mathbb{N})$ consistant avec G tel que $\forall G' \in \boldsymbol{\mathcal{G}}$, si X est aussi consistant avec G', alors $\|G'\| > \|G\|$ ou bien $G' = G$. Il peut exister zéro, un ou plusieurs témoins pour G ; nous dirons alors que G* admet *un témoin s'il en existe au moins un.*

En d'autres termes, si nous pouvons exhiber un témoin de minimalité, nous pouvons garantir une borne inférieure quant à la taille de la représentation. De ce fait, nous verrons que s'il existe un témoin de minimalité pour toutes les grammaires d'une classe, ces dernières seront identifiables en temps Mc polynomial. Les témoins de minimalité sont différents des *tell tales* d'Angluin [Ang80] dans le sens où ces derniers garantissent la minimalité au sens de l'inclusion et non de la taille de la représentation. De plus, Les *tell tales* sont inclus dans les langages alors que les témoins de minimalité sont inclus dans les présentations et donc peuvent contenir des exemples négatifs.

Exemple 16 *Soient le langage $S_n = \{a^i : n \leq i \leq n+9\}$ et la classe de langages $\mathcal{S}$ comprenant tous les langages $S_n, \forall n \in \mathbb{N}^+$.*

Soit $\boldsymbol{f} \in \boldsymbol{Pres}(\mathcal{S})$ une présentation de type Informateur. *Supposons qu'il existe $u, v \in \boldsymbol{f}$ tels que v soit le plus petit exemple positif de $\boldsymbol{f}$ et u le plus grand exemple négatif de $\boldsymbol{f}$ tel que $|v - 10| < |u| < |v|$. L'ensemble $\{u, v\}$ est alors un témoin de minimalité pour le langage $S_{|u|+1}$. En effet, v garantit que la taille est comprise entre $|v| - 10$ et $|v| + 10$ et le contre-exemple u rend impossible toute grammaire de taille inférieure à $|u| + 1$.*

2.2.5 Contrainte de nombres d'erreurs implicites

Troisièmement, nous pouvons borner le nombre de fois où l'apprenant se trompe plutôt que le nombre de Mc. Plus formellement, il s'agit du nombre d'erreurs implicites [Pit89] que commet l'algorithme durant son apprentissage :

Définition 19 (Erreur implicite de prédiction (Ipe)) *Étant donnés un algorithme d'apprentissage $\boldsymbol{Alg}$ et une présentation $\boldsymbol{f}$, on dit que $\boldsymbol{Alg}$ fait* une erreur implicite de prédiction *(Implicit Prediction Error, noté* Ipe*)* au temps n *si $\boldsymbol{Alg}(\boldsymbol{f}_{n-1}) \not\vdash \boldsymbol{f}(n)$.*

***Alg** fait donc une* IPE *si le nouvel exemple positif n'appartient pas à l'hypothèse courante, ou si, dans le cas d'un* INFORMATEUR *par exemple, l'exemple négatif appartient à cette hypothèse.*

*Nous dirons que **Alg** est* consistant *s'il change d'avis chaque fois qu'une erreur de prédiction est détectée avec le nouvel élément.*

L'apprentissage sera alors « bon » si le nombre d'erreurs n'est pas trop grand par rapport à la taille de la grammaire :

Définition 20 (Identification IPE polynomiale) *Un algorithme **Alg** identifie une classe $\mathcal{G}$ à la limite en temps* IPE *polynomial si*

1. ***Alg** identifie $\mathcal{G}$ à la limite,*
2. ***Alg** a un temps de mise à jour polynomial et*
3. ***Alg** fait un nombre polynomial d'erreurs implicites de prédiction : il existe un polynôme $p()$ tel que, pour chaque grammaire G et chaque présentation $\boldsymbol{f}$ de $\mathbb{L}(G)$, $\#\text{IPE}(\boldsymbol{f}) \leq p(\|G\|)$ où $\#\text{IPE}(\boldsymbol{f})$ est le nombre d'*IPE *: $\#\text{IPE}(\boldsymbol{f}) = |\{k \in \mathbb{N} : \boldsymbol{Alg}(\boldsymbol{f}_k) \not\Vdash \boldsymbol{f}(k+1)\}|$.*

Exemple 17 *Reprenons le langage $S_n = \{a^i : n \leq i \leq n+9\}$ et la classe de langages $\mathcal{S}$ comprenant tous les langages $S_n, \forall n \in \mathbb{N}^+$.*

Soit l'algorithme suivant :

1. *Lire la donnée $\boldsymbol{f}(n)$, soit u la plus petite des données de $\boldsymbol{f}$.*
2. *Retourner $S_{|u|}$.*
3. *Aller en 1.*

Cet algorithme identifie à la limite $\mathcal{S}$ en faisant au plus 10 IPE. *Voici un exemple de l'exécution de l'algorithme sur une présentation du langage cible S_2 :*

i :	0	1	2	3	4	5	6	7	8	9	10	...
$f(i)$:	a^{10}	a^9	a^8	a^5	a^4	a^3	a^8	a^2	a^7	a^{11}	a^4	...
IPE :		oui	oui	oui	oui	oui	non	oui	non	non	non	...
Retour :	S_{10}	S_9	S_8	S_5	S_4	S_3	S_3	S_2	S_2	S_2	S_2	...

L'algorithme fera alors au maximum 9 IPE *si les données apparaissent dans l'ordre « inverse » (de a^{n+9} à a^n). En revanche, il est possible de faire moins d'*IPE *en changeant d'hypothèse même si les nouvelles données sont consistantes avec l'hypothèse courante. Soit l'algorithme suivant :*

1. *Lire la donnée u.*
2. *Retourner $S_{|u|-5}$ (ou S_1 si $|u| - 5 \leq 0$).*
3. *Lire la donnée, soient u la plus petite des données de $\boldsymbol{f}$ et v la plus grande.*
4. *Retourner $S_{\lceil \frac{|u|+|v|}{2} \rceil - 5}$ (ou S_1 si $\lceil \frac{|u|+|v|}{2} \rceil - 5 \leq 0$).*
5. *Aller en 3.*

*Cet algorithme fera un nombre plus petit d'*Ipe *car il essaie de regrouper le plus possible de données au « centre » du segment hypothèse. Voici un exemple de l'exécution de cet algorithme sur la même présentation que précédemment :*

i :	0	1	2	3	4	5	6	7	8	9	10	...
$f(i)$:	a^{10}	a^9	a^8	a^5	a^4	a^3	a^8	a^2	a^7	a^{11}	a^4	...
Ipe :		non	non	non	non	non	non	non	non	oui	non	...
Retour :	S_5	S_5	S_4	S_3	S_2	S_2	S_2	S_1	S_1	S_2	S_2	...

Changer d'hypothèse pour essayer de coller au mieux aux données permet donc ici à l'algorithme de ne faire qu'une erreur. Évidemment, ce n'est pas toujours le cas. Ici, l'algorithme fera un maximum de 3 Ipe *:*

i :	0	1	2	3	4	...
$f(i)$:	a^{10}	a^{15}	a^7	a^{16}	a^6	...
Ipe :		oui	oui	oui	non	...
Retour :	S_5	S_8	S_6	S_7	...	

Nous avons dit que lorsqu'un algorithme **Alg** est consistant, il change d'avis chaque fois qu'une erreur de prédiction est détectée. Il fera alors au moins autant de changements d'hypothèses que d'erreurs de prédiction : $\#\text{Ipe}(\boldsymbol{f}) \leq \#\text{Mc}(\boldsymbol{f})$. De la même façon, si **Alg** est conservatif alors il ne change jamais d'avis lorsque son hypothèse courante est consistante. Nous avons donc $\#\text{Mc}(\boldsymbol{f}) \leq \#\text{Ipe}(\boldsymbol{f})$. Aussi, nous pouvons en déduire le lemme suivant :

Lemme 1 *Si* $\boldsymbol{Alg}$ *identifie la classe* $\boldsymbol{\mathcal{G}}$ *à la limite en temps* Mc *polynomial et s'il est consistant, alors* $\boldsymbol{Alg}$ *identifie* $\boldsymbol{\mathcal{G}}$ *en temps* Ipe *polynomial. De même, si* $\boldsymbol{Alg}$ *identifie* $\boldsymbol{\mathcal{G}}$ *en temps* Ipe *polynomial et s'il est conservatif, alors* $\boldsymbol{Alg}$ *identifie* $\boldsymbol{\mathcal{G}}$ *en temps* Mc *polynomial.*

2.2.6 Contrainte d'ensemble caractéristique

Le quatrième critère concerne la quantité minimum de données qu'un algorithme doit recevoir pour apprendre. Il est en effet raisonnable de penser que l'apprenant a le droit d'apprendre plus « lentement » des grammaires qui ont des tailles plus importantes :

Définition 21 (Ensemble caractéristique polynomial) *Une classe de grammaires* $\boldsymbol{\mathcal{G}}$ *admet un* ensemble caractéristique polynomial *s'il existe un algorithme* $\boldsymbol{Alg}$ *et un polynôme* $p()$ *tels que pour toute grammaire* $G \in \boldsymbol{\mathcal{G}}$*, il existe* $\text{Cs} \subseteq X$ *avec*

1. $\|\text{Cs}\| \leq p(\|G\|)$,
2. $\mathbb{L}(\boldsymbol{Alg}(\text{Cs})) = \mathbb{L}(G)$ *et*
3. *pour tout* $\boldsymbol{f} \in \boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$*, pour tout* $n \geq 0$*, si* $\text{Cs} \subseteq \boldsymbol{f}_n$ *alors* $\boldsymbol{Alg}(\boldsymbol{f}_n) = \boldsymbol{Alg}(\text{Cs})$.

Un tel ensemble Cs *est appelé* ensemble caractéristique *de* G *pour* $\boldsymbol{Alg}$*, et si* $\boldsymbol{Alg}$ *existe, nous dirons que* $\boldsymbol{\mathcal{G}}$ est identifiable à la limite en temps Cs polynomial.

Exemple 18 *Soient le langage* $S_n = \{a^i : n \leq i \leq n+9\}$ *et la classe de langages* $\mathcal{S}$ *comprenant tous les langages* $S_n, \forall n \in \mathbb{N}^+$.

$\mathcal{S}$ *est alors identifiable à la limite en temps* Cs *polynomial. En effet, définissons l'ensemble caractéristique suivant pour chaque langage* S_n *:* $\text{Cs} = \{a^n, a^{n+9}\}$. *Nous pouvons alors facilement construire un algorithme qui vérifie l'existence d'un tel ensemble caractéristique dans la présentation et retourne le langage correspondant. Comme la taille des ensembles caractéristiques vaut* $\|\text{Cs}\| = n + n + 9 = 2n + 9$, *que* $\|S_n\| = 10n + 45$, *nous avons bien* $\|\text{Cs}\| \leq p(\|S_n\|)$. *De plus, l'algorithme retourne bien le bon langage à partir de l'ensemble caractéristique. Enfin, si l'ensemble caractéristique apparaît dans la présentation l'algorithme retournera toujours le même langage, alors* $\mathcal{S}$ *est alors identifiable à la limite en temps* Cs *polynomial.*

Notons que si une classe de grammaires admet seulement des ensembles caractéristiques dont la taille est exponentielle, alors aucun algorithme ne pourra converger sans recevoir une quantité déraisonnable de données ! L'existence d'un ensemble caractéristique polynomial est nécessaire mais pas suffisante. Des définitions plus contraignantes d'ensembles caractéristiques polynomiaux existent [dlH97].

2.3 Apprentissage actif

Le second paradigme d'apprentissage que nous utiliserons est dû à Angluin [Ang88b]. Dans ce cadre d'apprentissage, l'apprenant doit toujours trouver un langage cible mais, pour s'aider, il peut interroger un oracle qui connaît ce langage. Il peut alors poser des questions à l'oracle sous forme de différents types de requêtes et ce dernier doit y répondre sans mentir.

L'interaction oracle-apprenant rappelle alors l'interaction mère-enfant lorsque ce dernier apprend à parler. En effet, lorsque sa mère lui parle, nous pouvons dire que l'enfant reçoit des exemples positifs. De même, si l'enfant essaie de faire des phrases sa mère lui dira si sa phrase est correcte ou non.

L'apprentissage à partir de requêtes est alors intéressant d'un point de vue pratique. Par exemple, plutôt que de demander à un expert d'étiqueter tout un ensemble de données, un apprenant pourra lui demander d'étiqueter seulement quelques exemples qu'il aura choisi. Le fait que l'oracle puisse être un humain nous permet de comprendre que lors de l'apprentissage à partir de requêtes, ce qui va vraiment compter est le nombre de requêtes que l'apprenant fait.

2.3.1 Les requêtes usuelles

Depuis [Ang88b], de nombreux types de requêtes ont été définis (sous-ensemble, exhaustivité, ...). Toutefois, beaucoup de travaux amenant à des résultats d'apprenabilité (et d'inapprenabilité) n'en utilisent seulement que deux : les requêtes d'appartenance et celles d'équivalence.

Définition 22 (Requêtes d'appartenance) *Soit* $\mathcal{L}$ *une classe de langages et soit le langage cible* $L \in \mathcal{L}$ *connu par l'oracle et que l'apprenant essaie de trouver.*

Les requêtes d'appartenance, ou membership queries *(notées* MQ*), permettent à l'apprenant de soumettre un mot $w \in \Sigma^*$ à l'oracle. La réponse de celui-ci, notée $Mq(w)$, est alors :*

- OUI *si $w \in L$,*
- NON *si $w \notin L$.*

Définition 23 (Requêtes d'équivalence) *Soient $\mathcal{L}$ une classe de langages et $L \in \mathcal{L}$ le langage cible connu par l'oracle que l'apprenant essaie de trouver.*

Les requêtes d'équivalence, ou equivalence queries *(notées* EQ*), permettent à l'apprenant de soumettre une hypothèse H (en fait une représentation du langage H) à l'oracle. La réponse de ce dernier, notée $Eq(H)$, est*

- OUI *si H est équivalent à L,*
- *un contre-exemple sinon, c'est-à-dire un mot de la différence symétrique $L \oplus H$.*

En plus du temps de calcul global de l'apprenant, ce qui justifie qu'un algorithme est meilleur qu'un autre est le nombre total de requêtes effectuées :

Définition 24 *Soit* QUER *un ensemble de requêtes pouvant être utilisées. Une classe $\boldsymbol{\mathcal{G}}$ est* polynomialement identifiable à partir de QUER *s'il existe un algorithme **Alg** capable d'identifier chaque $G \in \boldsymbol{\mathcal{G}}$ et tel que, à chaque appel de requêtes, le nombre total de requêtes ainsi que le temps utilisé jusqu'à cet appel par **Alg** est polynomial en $\|G\|$ et en la taille de l'information présentée par l'oracle jusqu'à ce point.*

Exemple 19 *Prenons une sous-classe de $\mathcal{S}$: $\mathcal{S}' = \{S_n : n = 9k + 1, \forall k \in \mathbb{N}\}$. Les langages de $\mathcal{S}'$ sont donc disjoints deux à deux et pour tout $j \in \mathbb{N}^+$, a^j appartient à un unique langage de $\mathcal{S}'$: $S_{9*\lfloor j/9 \rfloor+1}$.*

Nous allons montrer que $\mathcal{S}'$ n'est pas apprenable avec un nombre polynomial de requêtes d'équivalence et d'appartenance. La preuve de la non apprenabilité repose sur une technique utilisée par Angluin dans [Ang90] : l'oracle est un adversaire jouant le rôle d'un oracle malveillant. Il peut avoir accès à l'algorithme de l'apprenant et le force à faire le plus grand nombre de requêtes possibles en maintenant à jour l'ensemble X des hypothèses consistantes avec les données.

Au début, X contient l'ensemble des langages de $\mathcal{S}'$. Pour chaque requête d'appartenance $Mq(w)$, l'oracle répond NON*. Si w contient la lettre b, aucun langage n'est enlevé de X. Si $w = a^j, j \in \mathbb{N}^+$, un seul langage est enlevé de X ($S_{9*\lfloor j/9 \rfloor+1}$). De même, pour chaque requête d'équivalence $Eq(S_n)$, la réponse de l'oracle est* NON *éliminant uniquement S_n de X.*

Pour chaque requête, un seul langage au maximum est enlevé de X, forçant ainsi l'apprenant à faire un nombre exponentiel de requêtes. $\mathcal{S}'$ n'est donc pas polynomialement apprenable avec des requêtes d'appartenance et d'équivalence.

Dans la preuve ci-dessus, les requêtes d'équivalence qui sont faites sont des requêtes dites *propres* : les langages hypothèses soumis à l'oracle sont des langages pouvant réellement être un langage cible. À l'inverse, si l'apprenant à le droit de soumettre un langage n'appartenant pas à la classe cible, les requêtes d'équivalence sont *impropres*.

Souvent, le fait d'avoir le droit de faire de telles requêtes permet une identification plus facile :

Exemple 20 *Reprenons le langage* $S_n = \{a^i : n \leq i \leq n+9\}$ *et la classe de langages* $\mathcal{S}$ *comprenant tous les langages* $S_n, \forall n \in \mathbb{N}^+$. *Soit l'algorithme suivant :*

1. *Faire la requête d'équivalence* $Eq\{\lambda\}$, *soient* u *le contre-exemple et* $j = |u|$ *sa longueur.*
2. *Faire la requête d'appartenance* $Mq(a^{j-1})$.
3. *Si l'oracle répond* OUI, $j \leftarrow j-1$ *puis aller en 2.*
4. *Sinon, retourner* S_j.

Cet algorithme identifie alors à la limite $\mathcal{S}$ *en faisant au plus une* EQ *et* 10 MQ.

Exemple 21 *Toujours pour la même classe de langages, soit l'algorithme suivant :*

1. *Faire la requête d'équivalence* $Eq(\emptyset)$, *soient* u *le contre-exemple et* $j = |u|$ *sa longueur.*
2. *Faire la requête d'équivalence* $Eq(S_{j-5})$ *(si* $j-5 \leq 0$, *faire la requête* $Eq(S_j)$*).*
3. *Si l'oracle répond* OUI, *l'inférence s'arrête. Sinon, soient* u *et* v *le plus petit et le plus grand contre-exemples retournés par l'oracle.*
4. *Poser* $j = \lceil \frac{|u|+|v|}{2} \rceil$ *et aller en 2.*

Cet algorithme identifie alors à la limite $\mathcal{S}$ *en faisant uniquement des* EQ.

Nous pouvons noter une similarité entre cet algorithme et celui identifiant $\mathcal{S}$ en temps IPE polynomial. Nous verrons par la suite que les deux concepts sont effectivement reliés dans certaines preuves d'identification.

De nombreux résultats et techniques d'apprentissage utilisant seulement ces deux types de requêtes ont été trouvés [Ang01, ASA01, Sak90, HPRW96]. Toutefois, les requêtes d'équivalence sont rarement utilisables en pratique. En effet, si nous reprenons l'exemple de la mère et de l'enfant, il est tout à fait concevable qu'un enfant demande à sa mère si telle ou telle phrase est correcte (cela correspondrait à une requête d'appartenance) ; en revanche les requêtes d'équivalence sont plus problématiques. En effet, une requête d'équivalence reviendrait à une question de l'enfant demandant à sa mère s'il connaît le français. Question à laquelle il serait bien difficile de répondre. De même, en inférence grammaticale, la question de l'équivalence entre deux langages n'est pas toujours possible. Un moyen de contourner ce problème est alors d'utiliser une méthode dite d'échantillonnage (ou *sampling*) [Ang87] où nous remplaçons les EQ par un tirage aléatoires de mots qui servent à poser des requêtes d'appartenance. Dans bien des cas cependant, l'échantillonnage n'est pas possible non plus car la distribution en question est soit inconnue, soit inaccessible [dlH06a].

2.3.2 Les requêtes de correction

Si la requête d'équivalence est souvent trop puissante et peu naturelle, un comportement se rapprochant plus de la réalité est le suivant : un enfant pose une question : « Est-ce que "J'ai vu deux chevals" est une phrase correcte ? », sa mère, plutôt que de lui répondre uniquement non (ce qui serait le cas d'une requête d'appartenance), va plutôt lui dire quelque chose comme : « Non, ce n'est pas français. Par contre, tu peux dire "J'ai vu deux chevaux". ». En effet, l'enfant n'apprend pas uniquement avec des exemples positifs ou des exemples négatifs, mais aussi à partir de corrections de ses phrases fausses. C'est à partir de ce constat que des travaux ont commencé sur un nouveau type de requête, les requêtes de correction [BBBD05] :

Définition 25 (Requêtes de correction) *Soient $\mathcal{L}$ une classe de langages et $L \in \mathcal{L}$ le langage cible connu par l'oracle que l'apprenant essaie de trouver.*

Les requêtes de correction permettent à l'apprenant de soumettre un mot $w \in \Sigma^*$ à l'oracle. La réponse de celui-ci, notée $Cq(w)$, est alors

- Oui *si $w \in L$ (ou Θ selon les travaux),*
- *une correction de w relativement à L sinon.*

Les requêtes de correction ont connu un rapide succès et plusieurs travaux ont proposés des modèles de correction différent :

- Dans [BBBD05], la correction correspond au plus petit mot v tel que $wv \in L$. La correction est donc basée sur le préfixe.
- Dans [TK07], la correction est la même mais elle borne la longueur de la correction.
- Dans [Kin08], Kinber propose une correction la plus proche possible de la requête en terme de distance d'édition, telle que la correction n'ait jamais été vue, qu'elle soit si possible de même longueur, sinon la plus petite selon l'ordre hiérarchique.
- Nous avons proposé dans [BBdlHJT07] des requêtes de correction basées sur la distance d'édition.

Dans les deux premiers cas, la correction est basée sur le préfixe. Ce modèle n'est alors pas très pertinent du point de vue de la langue naturelle. Dans le troisième cas, les corrections de l'oracle sont choisies de façon à aider l'apprenant. Les relations entre ces variantes ont été étudiées dans [Tir08].

Les requêtes de correction proposées dans [BBdlHJT07] et basées sur la distance d'édition semblent donc à la fois plus naturelles et plus pertinentes. C'est celles-ci que nous utiliserons par la suite :

Définition 26 (Requêtes de correction ($\mathbf{CQ_{Edit}}$)) *Soient $\mathcal{L}$ une classe de langages et $L \in \mathcal{L}$ le langage cible connu par l'oracle que l'apprenant essaie de trouver.*

Les requêtes de correction basées sur la distance d'édition, ou correction queries *($\mathrm{CQ_{Edit}}$), permettent à l'apprenant de soumettre un mot $w \in \Sigma^*$ à l'oracle. La réponse de celui-ci, notée $Cq(w)$, est alors*

- Oui *si $w \in L$,*

– *une correction proche de w relativement à L sinon, c'est-à-dire un mot w' appartenant au langage et à distance d'édition minimum de w :*

$$Cq(w) = \underset{w' \in L}{\operatorname{argmin}}\big(d(w, w')\big).$$

Lorsque plusieurs réponses sont envisageables, l'oracle en choisit une. De plus, celui-ci est consistant : même si plusieurs réponses sont possibles pour un mot w, $Cq(w)$ sera toujours la même correction.

Exemple 22 *Soit $L = \{a, b, aa, ab, ba, bb, aaaa, babbab\}$ un langage cible. La réponse à la requête de correction du mot ab est $Cq(ab) =$* Oui. *Le mot $baab$ étant à distance d'édition 2 des mots $aa, ab, ba, bb, aaaa$ et $babbab$, l'oracle choisira un de ces mots comme correction, par exemple aa, et toutes les réponses aux requêtes du mot $baab$ seront alors $Cq(baab) = aa$.*

Exemple 23 *Reprenons le langage $S_n = \{a^i : n \leq i \leq i + 10\}$ et la classe de langages $\mathcal{S}$ comprenant tous les langages $S_n, \forall n \in \mathbb{N}^+$.*

Soit l'algorithme suivant :

1. *Faire la requête $Cq(\lambda)$, soient u la correction et $j = |u|$ sa longueur.*
2. *Retourner S_j.*

Cet algorithme identifie alors à la limite $\mathcal{S}$ en faisant une unique requête de correction. En effet, la correction obtenue étant la plus proche possible de λ, cela ne peut qu'être le plus petit mot du langage cible S_j, c'est-à-dire a^j.

Si à la place de λ nous avions demandé la correction d'un mot w et que celui-ci appartenait au langage, il suffirait de demander la correction de wa^{10}. Supposons que $a^k = Cq(wa^{10})$ soit cette correction, le langage cible est alors S_{k-10}.

Dans notre définition, la correction donnée par l'oracle est potentiellement la plus mauvaise possible pour l'apprenant : si l'oracle est malveillant, il pourrait avoir accès à l'algorithme d'apprentissage, et proposer, lorsque plusieurs sont possibles, la correction la moins pertinente du point de vue apprentissage, c'est-à-dire celle apportant le moins d'information (par exemple un mot dont l'apprenant sait déjà qu'il est un exemple positif). Lors de l'apprentissage, il faudra donc toujours s'intéresser au nombre de requêtes effectuées dans le pire cas.

Si nous nous intéressons maintenant aux utilisations possibles des requêtes de correction, il est à noter que contrairement aux requêtes d'équivalence, elles sont bien plus faciles à simuler à partir du moment où nous connaissons le langage cible. De plus, elles existent déjà dans certaines applications. En effet, sur la plupart des moteurs de recherche sur internet, lorsque nous tapons des mots que le moteur ne connaît pas, ou lorsqu'ils ne sont pas fréquents, ce dernier propose une alternative aux mots-clés que nous avons tapés précédée d'un « Vouliez vous dire » ou « Essayez avec l'orthographe ». De même, le fonctionnement de programmes de correcteurs orthographique tel qu'*ispell*, est similaire à un apprenant qui ferait des requêtes de correction pour tous les mots d'un texte avec le dictionnaire comme langage cible.

2.4 PAC apprenabilité

Introduit par Valiant dans [Val84], le paradigme PAC (pour Probablement Approximativement Correct) a été largement utilisé en apprentissage automatique. Intuitivement, il vise à mesurer, avec une grande confiance, de bonnes approximations d'un concept cible inconnu. Plus formellement, un algorithme formule avec une probabilité proche de un, une hypothèse proche de la grammaire cible à ϵ près :

Définition 27 (Hypothèse ϵ-bonne) *Soient G la grammaire cible et H une grammaire hypothèse. Soient $\mathcal{D}$ une distribution sur $\Sigma^* \times \{+,-\}$ et $\epsilon > 0$. On dit que H est une* hypothèse ϵ-bonne *par rapport à G si $Pr_{\mathcal{D}}(x \in \mathbb{L}(G) \oplus \mathbb{L}(H)) < \epsilon$.*

L'apprenabilité PAC des grammaires à partir de mots (de longueur variable) a toujours été compliquée à définir [War89, KV89, KMR+94]. En effet, dans la définition standard de l'apprentissage PAC, l'algorithme peut demander à un oracle des exemples tirés aléatoirement selon la distribution $\mathcal{D}$. Or, dans le cas des mots, il existe toujours un risque de tirer un mot trop long pour être simplement lu en temps polynomial. Pour éviter ce problème, nous pouvons tirer des exemples à partir d'une distribution restreinte à des mots plus petits qu'une certaine valeur donnée par le lemme suivant :

Lemme 2 *Soit $\mathcal{D}$ une distribution sur $\Sigma^* \times \{+,-\}$. Alors $\forall \epsilon, \delta > 0$, avec probabilité supérieure à $1-\delta$, si nous tirons un échantillon X d'au moins $\frac{1}{\epsilon}\ln\frac{1}{\delta}$ mots suivant $\mathcal{D}$, la probabilité qu'un nouveau mot x soit plus long que n'importe quel autre mot de X est au plus ϵ. Formellement, si nous définissons $\mu_X = \max\{|y| : y \in X\}$, alors $Pr_{\mathcal{D}}(|x| > \mu_X) < \epsilon$.*

Démonstration :
Soit ℓ le plus petit entier tel que $Pr(\Sigma^{>\ell}) < \epsilon$. Une condition suffisante pour que $Pr_{\mathcal{D}}(|x| > \mu_X) < \epsilon$ est de prendre un échantillon X suffisamment grand pour être pratiquement sûr (c'est-à-dire avec probabilité supérieure à $1-\delta$) d'avoir au moins un mot plus long que ℓ. Clairement, la probabilité que n mots tirés selon $\mathcal{D}$ soient tous de longueur inférieure à ℓ est inférieure à $(1-\epsilon)^n$. Donc la probabilité qu'il y ait au moins un mot de longueur supérieure à ℓ parmi n mots est supérieure à $1-(1-\epsilon)^n$. Pour construire X, nous voulons donc $1-(1-\epsilon)^n > 1-\delta$, c'est-à-dire $(1-\epsilon)^n < \delta$. Comme $(1-\epsilon)^n \leq e^{-n\epsilon}$, il suffit que $n \geq \frac{1}{\epsilon}\ln\frac{1}{\delta}$ pour obtenir une valeur convenable de μ_X. □

Nous demandons maintenant à un algorithme d'apprendre une grammaire étant donné les paramètres d'*erreur* ϵ et de *confiance* δ. En outre, nous devons lui fournir une borne n sur la taille de la grammaire cible et une borne m sur la longueur des exemples d'apprentissage. Cette borne pourra ainsi être calculée grâce au Lemme 2.

L'algorithme peut alors demander des exemples à un oracle. Nous noterons EX() la requête d'un exemple ou d'un contre-exemple et POS-EX() la requête d'un exemple positif uniquement. Si en outre nous souhaitons restreindre la longueur des mots, nous utiliserons EX(m) (respectivement POS-EX(m)) la requête d'un exemple (respectivement d'un exemple positif seulement) de longueur inférieure à m. Formellement, l'oracle

retournera alors un mot tiré selon $\mathcal{D}$, $\mathcal{D}(\mathbb{L}(G))$, $\mathcal{D}(\Sigma^{\leq m})$ ou $\mathcal{D}(\mathbb{L}(G) \cap \Sigma^{\leq m})$ suivant la requête, où $\mathcal{D}(L)$ est la restriction de $\mathcal{D}$ aux mots de L : $Pr_{\mathcal{D}(L)}(x) = Pr_{\mathcal{D}}(x)/Pr_{\mathcal{D}}(L)$ si $x \in L$, 0 sinon. $Pr_{\mathcal{D}(L)}(x)$ n'est pas défini si $L = \emptyset$.

Nous pouvons alors définir ce qu'est bien apprendre dans le paradigme PAC :

Définition 28 (Polynomialement PAC apprenable) *Soit $\mathcal{G}$ une classe de grammaires. $\mathcal{G}$ est* PAC *apprenable s'il existe un algorithme **Alg** tel que $\forall \epsilon, \delta > 0$, pour toute distribution $\mathcal{D}$ sur $\Sigma^* \times \{+, -\}$, $\forall n \in \mathbb{N}$, $\forall G \in \mathcal{G}$ de taille $\leq n$, si **Alg** a accès à* EX*(), ϵ, δ, n et m, alors avec probabilité plus grande que $1 - \delta$, **Alg** retourne une hypothèse ϵ-bonne par rapport à G. Si **Alg** s'exécute en temps polynomial en $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|\Sigma|$, m et n, nous dirons que $\mathcal{G}$ est* polynomialement PAC apprenable.

Notons que pour pouvoir traiter la longueur non bornée des exemples nous pouvons utiliser $\epsilon' = \frac{\epsilon}{2}$ et une fraction de δ pour calculer m et accepter de faire une erreur d'au plus ϵ' sur tous les mots de longueur supérieure à m, et utiliser ainsi EX(m) au lieu de EX().

Enfin, par la suite, nous verrons que les techniques typiques prouvant la non apprenabilité PAC reposent souvent sur des hypothèses de complexité [PW88] : nous montrons que si nous pouvons apprendre une classe de langages $\mathcal{L}$ alors nous pourrions apprendre une classe $\mathcal{L}$' que nous savons ne pas être apprenable. En particulier, $\mathcal{RP}$ (pour *Randomised Polynomial Time*) est la classe de complexité des problèmes de décisions pour lesquels une machine de Turing probabiliste existe et

- elle fonctionne en temps polynomial en la taille des données d'entrée,
- sur une instance négative, elle retourne toujours NON et
- sur une instance positive, elle retourne OUI avec une probabilité supérieure à $\frac{1}{2}$ (sinon, elle retourne NON).

Un tel algorithme est dit *randomisé* : il pourrait alors avoir le droit de tirer une pièce à pile ou face pendant son exécution. L'algorithme ne fait ainsi des erreurs que dans le cas positif. Il est également important de noter que puisque cette erreur est inférieure à $\frac{1}{2}$, en répétant l'exécution de l'algorithme autant de fois que nécessaire, nous pouvons rendre aussi petite que nous voulons l'erreur d'apprentissage.

Nous utiliserons l'hypothèse communément admise que $\mathcal{RP} \neq \mathcal{NP}$. Pour une étude plus approfondie, voir par exemple [Pap94].

2.5 Étude du cas des AFD

Dans le prochain chapitre, nous allons nous intéresser à l'apprentissage à partir de données bruitées des langages rationnels (c'est-à-dire reconnus par des automates). En effet, ce sont sans aucun doute les langages formels les plus étudiés en apprentissage automatique. Ils correspondent aux langages de la base de la hiérarchie de Chomsky. S'ils n'étaient apprenables dans aucun paradigme, à quoi bon chercher à apprendre les langages *context-free* ? D'autre part, ils ont l'avantage d'avoir une représentation graphique sous forme d'automates. Ces derniers permettent alors souvent de visualiser les différents résultats d'apprentissage, ainsi que de comprendre le comportement des

algorithmes, identifier pourquoi ils fonctionnent correctement ou non, *etc.* Enfin, le champ d'utilisation des AFD est vaste : la biologie [Gus97], la musique [CV98, dlHPT04], la reconnaissance de la parole [PR96], ...

Avant d'essayer de les apprendre en situations bruitées, nous montrons ici comment les apprendre dans les paradigmes d'apprentissage que nous venons de voir à partir de données non bruitées. Comme nous l'avons dit, les automates sont étudiés depuis plusieurs années déjà. Les résultats que nous donnons ici sont donc pour la plupart tirés de la littérature.

Nous notons $\mathcal{AFD}(\Sigma)$ la classe des automates définis sur un alphabet Σ. De plus, une taille raisonnable de ces représentations est le nombre d'états de l'automate.

2.5.1 Identification à la limite

Nous commençons l'étude de l'apprentissage des AFD à partir de données non bruitées par leur identification à la limite.

Notons tout d'abord que la classe des réguliers étant une classe super finie, elle n'est pas identifiable à la limite à partir de TEXTE seulement (Théorème 1 [Gol67]). Nous ne nous intéresserons alors qu'à leur identification à partir d'INFORMATEUR.

Si nous considèrons dans un premier temps les erreurs implicites de prédiction comme critère de polynomialité, Pitt a montré qu'elles ne permettaient pas l'apprentissage des AFD :

Théorème 2 ([Pit89]) $\mathcal{AFD}(\Sigma)$ *n'est pas identifiable à la limite à partir d'*INFORMATEUR *en temps* IPE *polynomial.*

Démonstration :
Pour prouver ce théorème, Pitt s'appuie sur le théorème d'Angluin de 1990 et montre que si $\mathcal{AFD}(\Sigma)$ était identifiable en temps IPE polynomial, le théorème d'Angluin serait faux (Théorème 6 stipulant que les automates ne sont pas apprenables à partir de requêtes d'équivalence uniquement). L'idée de la preuve est de mettre en parallèle les contre-exemples des EQ avec les IPE.

Supposons qu'il existe un algorithme **Alg** qui identifie $\mathcal{AFD}(\Sigma)$ en temps IPE polynomial. Si nous ignorons toutes les hypothèses de **Alg** qui ne résultent pas en une IPE et que nous compressons l'exécution de l'algorithme pour ne retenir que les essais où l'AFD hypothèse est incorrect sur le prochain exemple, cela correspond alors à de l'identification avec des EQ. En effet, après chaque hypothèse, un contre-exemple sera fourni (ou si l'hypothèse est juste, l'inférence prend fin).

En d'autres termes, s'il existait un tel algorithme **Alg**, nous pourrions l'utiliser pour identifier $\mathcal{AFD}(\Sigma)$ en faisant un nombre polynomial d'EQ. $\mathcal{AFD}(\Sigma)$ n'étant pas identifiable polynomialement avec des EQ, $\mathcal{AFD}(\Sigma)$ n'est pas identifiable à la limite à partir d'INFORMATEUR en temps IPE polynomial. □

Toutefois, la plupart des autres résultats d'identification à la limite de $\mathcal{AFD}(\Sigma)$ sont essentiellement positifs. Beaucoup de ces résultats se basent sur des algorithmes d'inférence par fusion d'états.

Nous rappelons ici le fonctionnement d'un des plus connus de ces algorithmes : RPNI. Pour une étude plus approfondie sur les algorithmes par fusion d'états, le lecteur pourra se référer à [dlHOV96, Lan99, OS01]. Ces algorithmes diffèrent la plupart du temps seulement dans le choix des états à fusionner.

Algorithme 2 : Pseudo-code de l'algorithme RPNI

Données : Deux ensembles X_+ et X_-
Résultat : L'automate minimal reconnaissant tous les mots de X_+ et rejetant ceux de X_-

```
1  A ← PTA(X+, X−);
2  pour i ← 1 à |A| − 1 faire
       // Si l'état i a déjà été fusionné, passer à l'état suivant
3      j ← 0;
4      continuer ← Vrai;
5      tant que j < i et continuer faire
6          B ← fusion(A, i, j);
7          si Compatible(B, X+, X−) alors
8              A ← B;
9              continuer ← Faux;
10         finsi
11         j ← j + 1;
12     fintq
13 finpour
14 retourner A;
```

L'algorithme RPNI (pour Regular Positive Negative Inference) a été présenté par Oncina et García dans [OG92] (voir Algorithme 2). À partir d'un ensemble de mots positifs X_+ et d'un ensemble de mots négatifs X_-, il retourne l'AFD minimal reconnaissant X_+ et rejetant X_-, lorsque (X_+, X_-) contient un ensemble caractéristique. Étant un algorithme par fusion d'états, RPNI part de l'arbre accepteur de préfixes (ou PTA pour *prefix tree acceptor*) reconnaissant uniquement les mots de X_+ et rejetant ceux de X_-. RPNI choisit ensuite de façon itérative de fusionner des états deux par deux (la fusion est récursive afin de garder le déterminisme). Si la fusion n'aboutit pas sur un automate compatible avec X_+ et X_-, la fusion est rejetée, sinon elle est conservée. L'algorithme continue ensuite avec le nouvel automate ainsi créé jusqu'à ce qu'il ne puisse plus choisir d'états. La Figure 2.1 présente un exemple du résultat de l'exécution de RPNI sur les données, $X_+ = \{aa, bb, aaaa, abab, baab, baba, bbbbbb, baaabbbbab, baabbaaaabbbbaba\}$ et $X_- = \{a, ab, ba, aba, abb, bbb, aaab, baaa, aaaaaaaaa, bbbaababaa, ababbbbbbbba\}$. Le langage reconnu par l'automate retourné est le langage $\{w \in \Sigma^* : |w|_a \mod 2 = |w|_b \mod 2 = 0\}$.

La représentation sous forme d'automate permet de comprendre plus facilement le langage : l'état 0 accepte tous les mots ayant un nombre pair de a et un nombre pair de b, l'état 1 ceux ayant un nombre pair de b mais un nombre impair de a, l'état 2 ceux

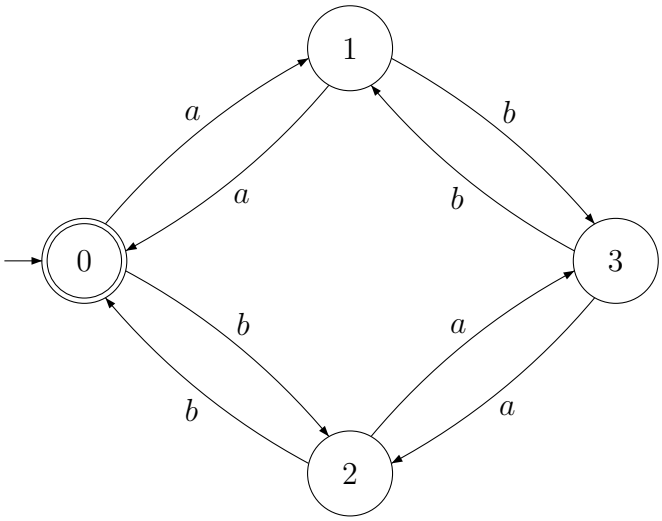

FIG. 2.1 – Automate résultant de l'exécution de RPNI sur les ensembles $X_+ = \{aa, bb, aaaa, abab, baab, baba, bbbbbb, baaabbbbab, baabbaaaabbbbaba\}$ et $X_- = \{a, ab, ba, aba, abb, bbb, aaab, baaa, aaaaaaaaa, bbbaababaa, ababbbbbbbba\}$.

ayant un nombre pair de a mais impair de b, et l'état 3 ceux ayant un nombre impair de a ainsi qu'un nombre impair de b. Comme seul l'état 0 est final, le langage reconnu par l'automate est donc bien celui contenant les mots ayant un nombre pair de a ainsi qu'un nombre pair de b.

Concernant l'apprentissage de RPNI, nous pouvons montrer sous certaines contraintes que RPNI identifie les AFD à la limite si un ensemble caractéristique est présent dans les données :

Théorème 3 ([Gol78, OG92]) $\mathcal{AFD}(\Sigma)$ *est identifiable à la limite à partir d'*INFORMATEUR *en temps* CS *polynomial.*

Démonstration :
(Idée de la preuve) Concernant la polynomialité, l'algorithme effectue une boucle sur tous les états du PTA. La taille de celui-ci est polynomiale en $\|(X_+, X_-)\|$. De plus, à chaque tour de boucle, RPNI vérifie avec (au maximum) tous les autres états du PTA s'il peut effectuer une fusion. Enfin, la vérification de la compatibilité est possible en temps polynomial en $\|(X_+, X_-)\|$, ce qui justifie donc l'exécution en temps polynomial de RPNI.

Afin de prouver l'existence d'un ensemble caractéristique, nous pouvons le construire à partir de l'automate canonique reconnaissant le langage cible, en utilisant les mots permettant d'accéder à tous les états et permettant de distinguer les états qui ne doivent pas être fusionnés entre eux. □

En fait, RPNI identifie à partir d'un ensemble de données fixées. Il faut alors exécuter RPNI à chaque nouvelle de données de la présentation. À un certain moment, l'ensemble caractéristique de la cible sera présent dans la présentation et RPNI retournera alors toujours le bon automate. Des versions incrémentales de RPNI ont donc été proposées pour palier à ce problèmes [Dup96].

Nous terminons cette étude par l'identification des AFD en utilisant le critère de changement d'avis. Dans [Pit89], Pitt affirme que les AFD sont identifiables en temps MC polynomial en disant qu'il peut construire un algorithme tel qu'au plus un AFD de taille i sera émis comme hypothèse, pour chaque i de 1 à n. Pour cela, son algorithme émet l'hypothèse nulle comme point de départ. Si ensuite l'hypothèse courante est un automate A_i de taille i, alors A_i est constamment retourné jusqu'à ce que, pour un certain $j > i$, assez d'exemples aient été reçus de sorte que, en temps polynomial en la taille des nouveaux exemples, il peut être vérifié qu'il n'existe qu'un seul automate A_j de taille j tel que A_j soit le plus petit automate canonique consistant avec les données. A_j est alors retourné. Cet algorithme identifie bien à la limite en temps MC polynomial, mais Pitt conclut en remarquant que les AFD qui sont produits ne sont pas particulièrement utiles jusqu'à ce qu'un nombre exponentiel en n d'exemples aient été vus.

Cependant, Pitt ne donne pas de techniques pour vérifier l'unicité du plus petit AFD consistant avec un ensemble en un temps polynomial.

Les témoins de minimalité définis précédemment permettent cela :

Proposition 2 *L'*AFD $A = \langle \Sigma, Q, q_0, F, \delta \rangle$ *admet un témoin* $X = \langle X_+, X_- \rangle$ *si*

1. *tous les états et toutes les transitions sont exercés par au moins une chaîne de* X*,*
2. *pour chaque état final, il existe un mot de* X_+ *accepté par cet état et pour chaque état non final, il existe un mot de* X_- *terminant dans cet état,*
3. *pour chaque lettre* a*, pour chaque transition* $\delta(q, a) = q' \neq q''$*, il existe des mots* w, w', f *tels que* w *(respectivement* w'*) soit le plus petit mot (de* Σ^**, au sens de l'ordre hiérarchique) tel que* $\delta(q_0, w) = q$ *(respectivement* $\delta(q_0, w') = q'$*) et* $waf \in X_+, w'af \in X_-$ *ou* $waf \in X_-, w'af \in X_+$*.*

Démonstration :
Soit X un ensemble tel que décrit la Proposition 2. Il n'existe pas d'autres AFD de taille inférieure ou égale à $|A|$ compatible avec cet ensemble. En effet, toute transition absente est rendue impossible par le témoin et chaque état est rendu nécessaire par les couples de mots $(waf, w'af)$ incompatibles deux à deux. De plus, X est un ensemble caractéristique de A pour l'algorithme RPNI (voir par exemple [DM98]). Aussi, A est bien *l'unique* AFD de taille minimale compatible avec X. □

Nous pouvons maintenant compléter l'algorithme de Pitt en proposant un algorithme identifiant $\boldsymbol{\mathcal{AFD}}(\Sigma)$ en temps MC polynomial. Soit **Alg** l'algorithme qui comme première hypothèse émet l'hypothèse vide. **Alg** change ensuite d'avis uniquement si $\boldsymbol{f}_i$ est un témoin de minimalité pour l'AFD A_i retourné par RPNI sur $\boldsymbol{f}_i$. Clairement, cette

vérification se fait en temps polynomial en $\|\boldsymbol{f}_i\|$. Si $\boldsymbol{f}_i$ n'est pas un témoin pour A_i, **Alg** retourne l'hypothèse précédente (qui n'est pas forcément consistante).

Chaque fois que **Alg** change d'avis, le nombre d'états de l'AFD retourné augmente strictement (grâce au témoin de minimalité). Comme il est borné par le nombre d'états de l'AFD cible, le nombre de changement d'avis est polynomial en la taille de l'AFD cible. Enfin, il arrivera forcément un moment où le témoin de minimalité de l'AFD cible apparaîtra, et alors **Alg** convergera. **Alg** identifie donc bien $\boldsymbol{\mathcal{AFD}}(\Sigma)$ à la limite en temps MC polynomial.

Nous en déduisons donc le théorème :

Théorème 4 ([dlHJT08]) $\boldsymbol{\mathcal{AFD}}(\Sigma)$ *est identifiable à la limite à partir d'*INFORMATEUR *en temps* MC *polynomial.*

2.5.2 Apprentissage actif

Si nous nous intéressons maintenant à l'apprentissage actif, Angluin a montré que les AFD ne sont pas apprenables avec un seul type de requête à la fois, mais le sont si l'oracle utilisé peut répondre à la fois à des requêtes d'appartenance et à des requêtes d'équivalence. L'algorithme utilisé pour apprendre les automates à partir de MQ et EQ se nomme LSTAR. Son principe est le suivant :

1. Construire une table, dite d'observation, consistante avec les données reçues jusqu'à ce point.
2. Soumettre un AFD (construit grâce à la table d'observation) en tant que requête d'équivalence.
3. Utiliser le contre-exemple donné par l'EQ pour mettre à jour la table (ou terminer si la réponse est OUI).
4. Compléter la table grâce à des MQ.
5. Retourner au point 1.

Il est alors facile de montrer que LSTAR se termine, qu'il identifie l'AFD cible, et qu'il fait un nombre polynomial de requêtes d'équivalence et d'appartenance. Nous en déduisons alors :

Théorème 5 ([AL87]) $\boldsymbol{\mathcal{AFD}}(\Sigma)$ *est polynomialement identifiable à partir de* MQ *et* EQ.

Toutefois, il est nécessaire de disposer à la fois des MQ et des EQ pour identifier en un temps polynomial :

Théorème 6 ([Ang88b, Ang90]) $\boldsymbol{\mathcal{AFD}}(\Sigma)$ *n'est pas polynomialement identifiable à partir de* MQ *ou de* EQ *seules.*

En outre, si nous connaissons les corrections des requêtes, cela n'aide pas à l'apprentissage :

Théorème 7 ([dlHJT08]) $\boldsymbol{\mathcal{AFD}}(\Sigma)$ *n'est pas polynomialement identifiable à partir de* CQ_{EDIT}.

Démonstration :
La preuve de la non apprenabilité des automates repose sur la technique utilisée par Angluin dans [Ang90] pour montrer que les automates ne sont pas apprenables avec des EQ seulement : l'oracle est un adversaire qui maintient à jour un ensemble d'automates hypothèses consistants avec les données. À chaque réponse de l'oracle, seul un petit nombre d'automates seront éliminés de l'ensemble d'AFD hypothèses, nous forçant ainsi à faire un nombre exponentiel de requêtes. L'adversaire change de cible durant le processus d'identification pour pénaliser l'apprenant, mais toutes ses réponses sont consistantes avec l'AFD final.

Soient A_w l'AFD reconnaissant $\Sigma^* \setminus \{w\}$, $n \in \mathbb{N}$ et $\boldsymbol{\mathcal{AFD}}_{\leq n} = \{A_w : w \in \Sigma^{\leq n}\}$. Comme dans [Ang90], nous décrivons un adversaire qui maintient un ensemble X des AFD possibles. Au début, $X = \boldsymbol{\mathcal{AFD}}_{\leq n}$. À chaque requête de correction w, l'adversaire répond OUI, éliminant de ce fait un unique AFD de X : A_w. En effet, $\forall A \in X$ tel que $A \neq A_w$, $w \in L(A)$, la réponse à la requête de correction n'apporte donc aucune information sur les autres AFD de X. Comme il y a $\Omega(|\Sigma|^n)$ AFD dans $\boldsymbol{\mathcal{AFD}}_{\leq n}$, les identifier nécessite alors $\Omega(|\Sigma|^n)$ requêtes. □

2.5.3 Apprentissage PAC

Nous terminons cette étude par l'apprentissage PAC. Le but de cet apprentissage est de construire une *bonne* hypothèse, c'est-à-dire, ici, un AFD proche de l'automate cible, avec grande probabilité en un temps polynomial. En 1994, Kearns et Vazirani ont montré dans [KV94] que si les AFD étaient polynomialement PAC-apprenables, alors nous pourrions résoudre efficacement plusieurs problèmes connus comme étant $\mathcal{NP}$-difficiles, comme par exemple le cassage de clefs RSA ou la factorisation des entiers de Blum. S'en suit donc le théorème :

Théorème 8 ([KV94]) $\boldsymbol{\mathcal{AFD}}(\Sigma)$ *n'est pas polynomialement* PAC *apprenable.*

Toutefois, il est à noter que ce résultat négatif est en partie dû au fait que les distributions utilisées pour tirer les exemples ne sont pas restreintes.

En effet, Li et Vitányi ont proposé un modèle pour l'apprentissage PAC avec des exemples dits simples [LV91], c'est-à-dire tirés de la distribution universelle de Solomonoff-Levin, repris par Denis, d'Halluin et Gilleron [DdG96] puis Parekh et Honavar [PH97], qui lui, permet l'apprentissage PAC de $\boldsymbol{\mathcal{AFD}}(\Sigma)$. Un autre modèle d'apprentissage PAC est celui où les exemples sont seulement des exemples positifs, modèle introduit par Denis [Den01].

Si certaines sous classes des AFD sont bien PAC apprenables à partir d'exemples positifs seulement, ce n'est pas le cas en général. Il est en effet facile de montrer que si dans une classe il y a deux langages L_1 et L_2 tels que $L_1 \cap L_2$ n'est pas vide, alors la classe n'est pas polynomialement PAC apprenable à partir d'exemples positifs seulement :

Lemme 3 *Si* $\boldsymbol{\mathcal{L}}$ *contient deux langages* L_1 *et* L_2 *tels que* $L_1 \cap L_2 \neq \emptyset$, $L_1 \not\subset L_2$ *et* $L_2 \not\subset L_1$, *alors* $\boldsymbol{\mathcal{L}}$ *n'est pas polynomialement* PAC *apprenable à partir d'exemples positifs seulement.*

Démonstration :
(Voir Figure 2.2) Soient $w_1 \in L_1 - L_2$, $w_2 \in L_2 - L_1$ et $w_3 \in L_1 \cap L_2$. Ces trois mots existent par définition. Considérons maintenant la distribution $\mathcal{D}_1$ où $\Pr_{\mathcal{D}_1}(w_1) = \Pr_{\mathcal{D}_1}(w_3) = \frac{1}{2}$ et la distribution $\mathcal{D}_2$ où $\Pr_{\mathcal{D}_2}(w_2) = \Pr_{\mathcal{D}_2}(w_3) = \frac{1}{2}$. Nous pouvons aisément constater que lors de l'apprentissage de L_1 à partir de $\mathcal{D}_2$ ou de L_2 à partir de $\mathcal{D}_1$, les exemples tirés ne seront que des répétitions du mot w_3. Par conséquent, lors du test, l'erreur sera forcément supérieur à $\frac{1}{2}$. □

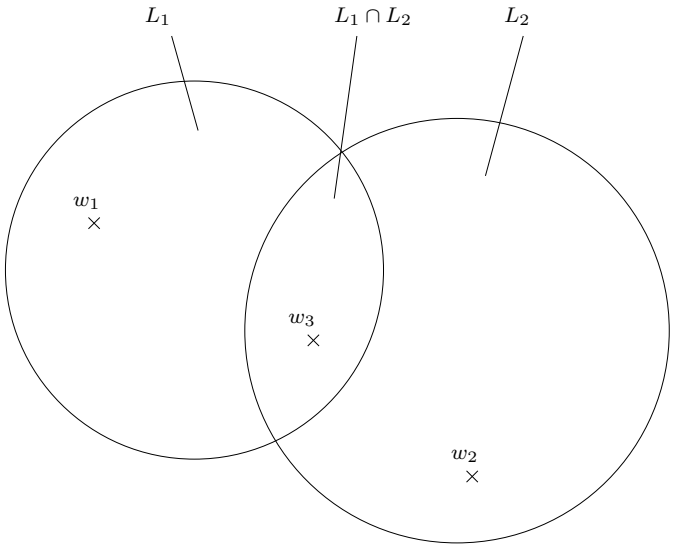

FIG. 2.2 – Exemple illustrant le Lemme 3.

Nous pouvons ainsi montrer qu'en général, les AFD ne sont pas PAC apprenables à partir de données positives seulement :

Théorème 9 $\mathcal{AFD}(\Sigma)$ *n'est pas polynomialement* PAC *apprenable à partir d'exemples positifs seulement.*

Démonstration :
Soient $L_1 = \{\lambda, a, b, aa, ab, ba\}$ et $L_2 = \{\lambda, a, b, ab, ba, bb\}$. Posons $w_1 = aa$, $w_2 = bb$ et $w_3 = ab$. Considérons ensuite les AFD reconnaissant L_1 et L_2. Le résultat est alors immédiatement obtenu grâce au Lemme 3. □
Si nous nous intéressons d'un peu plus près au modèle PAC, nous pouvons voir que les distributions sont faites sur l'ensemble des données possibles. L'apprentissage PAC correspond alors en quelque sorte à de l'apprentissage à partir de données bruitées : en effet, bruiter des données issues d'une distribution $\mathcal{D}$ revient à tirer les données d'une distribution différente $\mathcal{D}'$. De ce fait, si l'apprenant n'arrive pas à apprendre à partir de $\mathcal{D}'$, il ne peut pas PAC apprendre au sens général, puisque des données non bruitées aurait très bien pu être tirées selon $\mathcal{D}'$. Nous n'étudierons pas par la suite l'apprentissage PAC à partir de données bruitées.

Conclusion

L'apprentissage des AFD est ainsi possible dans la plupart des paradigmes. Toutefois, l'étude que nous venons de faire suppose que les données d'apprentissage sont *pures*. Or, en pratique, les données subissent souvent des modifications lors de leurs acquisition : erreur de recopie, sonde défaillante, *etc.* Nous pouvons donc nous demander si l'apprentissage des automates reste aussi aisée lorsque nous essayons de les identifier à partir de telles données. De plus, quelles sont les modifications possibles que les données peuvent subir ? Comment les prendre en compte dans les paradigmes d'apprentissage ? C'est ce que nous allons voir dans le chapitre suivant.

Chapitre 3

Le bruit en inférence grammaticale

Maintenant que nous avons vu les cadres d'apprentissage dans lesquels nous allons travailler, ainsi que ce que signifiait identifier les AFD dans ces paradigmes, nous allons nous intéresser à l'apprentissage en situations bruitées. En effet, en pratique, l'acquisition de données peut être perturbée de différentes manières entraînant la présence de données erronées. Les cadres théoriques du chapitre précédent permettent-ils toujours d'obtenir des résultats d'identification lorsque les données d'apprentissage sont bruitées ?

Nous allons voir dans ce chapitre ce que nous entendons par bruit : d'où vient-il, comment est-il modélisé, comment est-il traité ou encore, comment adapter les paradigmes d'identification pour prendre en compte les données bruitées ? Nous nous intéresserons ensuite à l'identification de la classe $\mathcal{AFD}(\Sigma)$ à partir de données bruitées selon les différents types de bruit et dans différents paradigmes. Enfin, nous discuterons des différents résultats d'apprentissage obtenus.

3.1 Typologie du bruit

Si l'inférence grammaticale, y compris de langages réguliers, continue à se développer, un problème clef continue de se poser : celui de la résistance au bruit des algorithmes proposés. Cette question est légitime puisque la plupart des données non artificielles, à partir desquelles nous voulons construire des automates par exemple, sont souvent bruitées.

Quelque soit le médium utilisé pour acquérir des données, entre le moment où les données sont réellement émises et celui où l'apprenant va les recevoir, les données peuvent subir toutes sortes de modifications : qu'il s'agisse d'une défaillance d'un capteur, d'une erreur de recopie humaine, d'une donnée incomplète, les sources d'erreurs sont multiples et fréquentes. Les données résultantes sont alors des données erronées. L'apprenant devra apprendre à partir de *données bruitées* dans le sens où il reçoit à la fois des vraies données ainsi que des données erronées.

Exemple 24 *Supposons qu'un apprenant doive classer des images en deux catégories : des images de mammifères, et des images de poissons. Apprendre à partir de données*

bruitées signifiera par exemple que certaines images d'apprentissage sont floutées, qu'on lui a présenté des périophthalmes (poissons pouvant vivre à l'air libre) comme étant des mammifères, qu'on ne lui a jamais montré de dauphins, etc.

Plus formellement, lorsqu'un apprenant essaie d'identifier un langage L à partir d'une présentation $\boldsymbol{f} \in \mathbf{Pres}(L)$, il doit alors s'attendre, quand les données sont bruitées, à ce que $\boldsymbol{f}(\mathbb{N}) \neq L$ (le bruit n'est bien entendu pas limité aux présentations de type Texte) :

- Une donnée peut voir son étiquette changer : l'exemple $(aba, +)$ peut se transformer en $(aba, -)$, et réciproquement, un exemple négatif devient un exemple positif. Nous parlons alors de *bruit d'étiquette*. Dans le cas d'un apprentissage multiclasse, une donnée d'une classe i appartiendra alors à une classe $j \neq i$. Ce type de bruit n'est donc pas très adapté à l'apprentissage à partir de données positives seulement.
- Une donnée peut voir sa valeur changer : elle peut subir un certain nombre d'opérations d'édition (suppression, insertion, modification). Ainsi, l'exemple $(aba, +)$ pourra devenir $(abba, +)$. Le bruit est dans ce cas appelé *bruit d'édition*. Notons que si le mot *abba* appartient en fait au langage, aucune erreur n'est réellement introduite. En revanche, le cas où le mot *abba* est en fait un exemple négatif, est identique à celui où le mot *abba* (bien que le mot qui ait été originellement bruité soit *aba*) subit l'influence du bruit d'étiquette.
- Une donnée peut ne jamais être présente : nous parlons alors de bruit par omission. C'est par exemple le cas lorsque des données sont non-étiquetées ou bien incomplètes. Ne sachant pas comment traiter ces données, elles sont souvent ignorées. De même, des exemples peuvent ne pas être présents dans les données d'apprentissage alors que ce sont des exemples *clés* du langage cible.

Exemple 25

- *Lorsque l'apprenant reçoit une image floutée de poisson, il connaît l'étiquette de l'image, mais l'image a été modifiée : c'est du bruit d'édition.*
- *Si l'apprenant reçoit l'image d'un périophthalme comme étant un mammifère, il s'agit du bruit d'étiquette : l'image n'a pas été modifiée, mais sa classe si.*
- *Enfin, si dans ses données d'apprentissage il n'y a aucun dauphin, cela représente un bruit par omission. Cela est bien entendu valable pour n'importe quel type de présentation. Dans le cas d'un* Informateur, *l'omission a lieu lorsque le dauphin n'est jamais montré, avec une étiquette positive comme avec une étiquette négative.*

Que ce soit les données d'une présentation, celles d'un ensemble fini, ou encore les réponses d'un oracle, identifier à partir de données bruitées signifie donc apprendre lorsqu'une partie, ou toutes les données d'apprentissage ont subi l'influence du bruit.

3.2 Traitement du bruit en apprentissage automatique

Plusieurs stratégies peuvent alors être envisagées pour apprendre à partir de données bruitées. Elles correspondent à une approche permettant de réduire le nombre

d'exemples d'apprentissage [BL97]. La première consiste à nettoyer ou filtrer les données d'apprentissage. Nous parlons alors d'algorithme de type *filter* : par exemple, si un mot n'a pas d'étiquette, s'il est étiqueté à la fois positif et négatif, ou encore s'il a une lettre qui n'appartient pas à l'alphabet, il peut être judicieux de l'enlever de l'ensemble d'apprentissage ou de le corriger. Une fois les données *purifiées*, nous pouvons alors utiliser un algorithme d'apprentissage classique. Cette méthode a en revanche un inconvénient majeur : si nous ne disposons que de peu de d'exemples ou si les données étaient très bruitées, le nombre d'exemples restant peut devenir trop faible pour réussir à identifier la cible.

La seconde approche vise à apprendre tout en sachant que les données sont bruitées, et donc à construire des algorithmes résistants au bruit *pendant* l'inférence. C'est la stratégie utilisée par les algorithmes de type *wrapper*. Si de toute évidence, un algorithme qui apprend à partir de données *pures* ne peut pas servir en l'état à l'apprentissage à partir de données bruitées, il est parfois possible de le modifier sans faire trop de changements afin de le rendre résistant au bruit.

Quelque soit la stratégie employée, le but de l'apprentissage consiste alors à essayer d'apprendre tout en sachant que les données sont bruitées et donc à essayer de distinguer les vraies données des données bruitées, que ce soit avant ou pendant l'inférence.

Supposons maintenant que nous ayons construit un algorithme qui soit censé identifier à partir de données bruitées. Comment tester cet algorithme ? En effet, les paradigmes que nous avons présentés au Chapitre 2 sont *a priori* inadaptés à l'apprentissage en situations bruitées. Nous allons voir maintenant comment gérer le bruit lors de l'identification à la limite et en apprentissage actif.

3.2.1 Le bruit dans l'identification à la limite

Afin de modéliser des données bruitées, la plupart des travaux d'identification à la limite se basent sur une distribution de bruit statistique. Ainsi, le but est d'apprendre à partir d'un ensemble de données, ou d'une présentation, contenant un certain nombre de mots bruités suivant cette distribution. De ce fait, certains travaux utilisent des automates stochastiques pour gérer les données bruitées, c'est-à-dire, des automates où nous associons à la fonction de transition une certaine probabilité [Ang88a]. De même les états seront tous finaux selon une certaine probabilité. L'automate appris est alors bien souvent seulement une approximation de l'automate cible, dans le sens où la différence symétrique entre les mots pouvant être générés par l'automate appris et ceux reconnus par l'automate cible est faible, mais rarement nulle. L'identification n'est alors plus exacte.

Mesurer si un algorithme est résistant au bruit se fait alors en calculant le pourcentage de mots correctement identifiés par l'hypothèse apprise. Nous parlons alors de *taux de succès en généralisation*. Le taux de succès en généralisation mesure ainsi la capacité de l'automate à classifier correctement les données appartenant au langage mais qu'il n'a pas vues, c'est-à-dire sa capacité à généraliser.

Plusieurs problèmes se posent alors. Un moindre problème, lié au protocole expérimental, est qu'il faut générer des données et pour cela construire aléatoirement un auto-

mate stochastique, ce qui est un problème loin d'être trivial. D'autre part, si l'automate a un taux de succès de 95%, pouvons-nous dire que nous avons appris correctement ? Lorsque le bruit repose sur une distribution statistique, l'apprentissage ne peut plus être considéré réellement comme de l'identification à la limite si nous disons que la cible est identifiée lorsque le taux d'apprentissage n'est pas de 100%. L'inférence n'étant plus exacte, le cadre mis en place relève alors plus du modèle PAC.

Néanmoins, il est à noter qu'un certain nombre de travaux ont proposé d'autres modèles de bruit plus adaptés à l'identification à la limite à partir de présentation car ne reposant pas sur des statistiques. En effet, il n'existe généralement pas de distribution sur les données. L'introduction de fonctions de bruit reposant sur des probabilités est alors inadéquat.

Parmi ces travaux, ceux de Stephan [Ste97] utilisent le fait qu'une présentation est infinie pour définir le bruit :

- Un TEXTE bruité pour un langage L est tel que chaque mot $w \in L$ apparaît infiniment souvent alors que seul un nombre fini de mots de la présentation n'appartiennent pas au langage.
- Un TEXTE très bruité pour un langage L est tel que chaque mot $w \in L$ apparaît infiniment souvent dans la présentation.

Contrairement au TEXTE bruité, un nombre infini d'erreurs peut donc apparaître dans un TEXTE très bruité, mais chaque erreur n'apparaîtra qu'un nombre fini de fois. Les critères d'identification sont alors notés *NoisyTxt* et *VeryNoisyTxt*. Sur le même principe Stephan définit les INFORMATEUR bruités et très bruités. Ce type de bruit est repris dans des travaux tels que [CJS01] où une condition supplémentaire apparaît : les présentations bruitées doivent être calculables.

Un autre exemple de bruit non statistique permettant une identification à la limite exacte est celui introduit dans [TdlHJ06], le bruit dit *systématique*[1]. Avec ce type de bruit, la présentation comporte tous les mots du langage ainsi que tous les mots proches du langage, c'est-à-dire tous les mots de Σ^* à distance au plus k de chaque mot du langage :

Définition 29 (Bruité d'un langage) *Soit L un langage de Σ^*. Le* k-bruité *de L est* $N_k(L) = \{w \in \Sigma^* : \exists v \in L, d(v,w) \leq k\}$.

Exemple 26 *Soient les langages $L_1 = \Sigma^{\leq n}$, $L_2 = \bigcup_{k \leq n/2} \Sigma^{2k}$, $L_3 = a(a|b)^*$ et $L_4 = \{w \in \Sigma^* : |w|_a \neq |w|_b\}$, les 1-bruités des langages correspondants sont $N_1(L_1) = \Sigma^{\leq n+1}$, $N_1(L_2) = \Sigma^{\leq n+1}$, $N_1(L_3) = \Sigma^*$ et $N_1(L_4) = \Sigma^*$.*

Le 1-bruité de $L = \{abbab\}$ est $N_1(L) = \{abab, abba, abbb, bbab, aabab, abaab, abbaa, abbab, abbbb, bbbab, aabbab, ababab, abbaab, abbaba, abbabb, abbbab, babbab\}$.

Une présentation bruitée d'un langage est alors définie comme étant la présentation du langage bruité :

[1] Dans [SG86], un bruit systématique est aussi proposé. Le bruit est systématique dans le sens où le bruit se fait toujours de la même façon : un capteur de température indiquera par exemple toujours 5°C de plus que la vraie température, seuls des exemples négatifs seront bruités en positifs mais pas l'inverse, *etc.* Toutefois, ce modèle n'a pas été repris dans le cadre de l'inférence grammaticale.

Définition 30 (Présentation k-bruitée) *Soit L un langage de Σ^*. Une présentation k-bruitée de L est une présentation de $N_k(L)$.*

Il est à noter que notre bruit systématique diffère de celui de Stephan dans le sens où seuls les éléments de $N_k(L) \setminus L$ *bruitent* la présentation, et que même ces éléments peuvent apparaître un nombre infini de fois.

3.2.2 Le bruit en apprentissage actif

Intéressons-nous maintenant au cadre de l'apprentissage actif. La perfection de l'oracle est difficile à obtenir ; par conséquent, le bruit est introduit par le biais de l'oracle. Si généralement le problème d'appartenance d'un mot à un langage est facile à résoudre, ce n'est pas toujours le cas. En effet, en pratique l'oracle est typiquement remplacé par un humain ; c'est par exemple le cas du système SQUIRREL [CGLN07]. Il peut donc lui arriver de ne pas arriver à décider si tel chiffre manuscrit est un 0 ou un 6, ou si telle lettre manuscrite est un a ou un u. Il peut ainsi se tromper, voire dire qu'il ne sait pas si un exemple appartient au langage ou non. De même, la réponse à une requête de correction n'est pas toujours évidente : quelle est la correction du mot $(ab(bba)^{11}b)^7$ relativement au langage $L = \Sigma^{\leq 113} \setminus \{w : (ab)^{31} \preceq w\}$? Si dans ce cas la réponse à une requête d'appartenance MQ est (presque) instantanée, donner une correction, c'est-à-dire répondre à une CQ_{EDIT}, est plus ardu.

Plusieurs modèles d'oracles peuvent alors être considérés afin de simuler un oracle humain :

- les MQ peuvent être limitées afin que l'oracle puisse répondre « je ne sais pas » à certains mots (appelées *omissions* ou LMQ pour *limited membership queries* [GM92, AS94, FGMP94]), un nombre fini ℓ d'omissions étant fixé à l'avance,
- la réponse de l'oracle à $MQ(w)$ peut être fausse pour certains mots (les requêtes sont alors appelées MMQ pour *malicious membership queries* [AK94]), un nombre fini ℓ d'erreurs étant fixé à l'avance[2],
- la réponse à $CQ(w)$ peut être OUI si w appartient au langage cible, et sinon, une correction appartenant au langage, plus ou moins proche de w suivant une certaine probabilité [BBdlHJT08].

Dans les travaux ci-dessus, les réponses aux requêtes sont dites *persistantes* : différentes requêtes avec le même mot w recevront toujours la même réponse, qu'elle soit juste ou fausse. Dans [Sak91], Sakakibara présente un modèle dans lequel chaque réponse à une requête d'appartenance peut être fausse suivante une certaine probabilité. Toutefois, dans son modèle, chaque requête constitue un événement indépendant. En d'autres termes, l'oracle peut donner deux réponses différentes à une même requête suivant le moment où la question est posée. Il suffit donc de répéter la requête un nombre de fois *suffisant* pour savoir si l'oracle a menti ou non. Sakakibara montre alors que toute classe qui est apprenable sans bruit est apprenable (avec grande probabilité) dans son modèle, ce qui n'est pas vraiment pertinant. Nous n'étudierons ainsi que les modèles de bruit où les réponses aux requêtes sont persistantes.

[2]Les ℓ omissions ou les ℓ erreurs peuvent être choisies par un oracle malveillant connaissant la stratégie d'apprentissage.

Maintenant que les cadres théoriques sont formellement posés, nous allons nous intéresser à l'apprentissage des langages les plus étudiés en inférence grammaticale : les langages réguliers. En effet, leur apprentissage en situations non bruitées est largement étudié. Ces études permettent de mieux comprendre et aborder l'apprentissage de langages de plus haut niveau dans la hiérarchie de Chomsky.

Ainsi, pour mieux appréhender l'étude de l'apprenabilité des langages de la hiérarchie de Chomsky à partir de données bruitées, nous nous attaquons en premier aux langages rationnels.

3.3 Apprentissage des AFD en situations bruitées

Nous allons maintenant voir les différents résultats des travaux sur l'apprentissage des AFD à partir de données bruitées. Pour cela nous considérerons les différents modèles de bruit étudiés précédemment.

3.3.1 Dans le cadre de l'identification à la limite

Nous allons nous intéresser dans un premier temps à l'apprentissage des AFD dans le cadre de l'identification à la limite. Afin de mieux cerner les enjeux du problèmes, nous allons reprendre l'exemple de l'apprentissage par RPNI en ajoutant du bruit.

Soient la cible $L = \{w \in \Sigma^* : |w|_a \mod 2 = |w|_b \mod 2 = 0\}$, et les échantillons $X_+ = \{bb, aaaa, abab, baab, baba, bbbbbb, baaabbbbab, baabbaaaabbbbaba\}$ et $X_- = \{a, aa, ab, ba, aba, abb, bbb, aaab, baaa, aaaaaaaaa, bbbaababaa, ababbbbbbbba\}$. Par rapport à l'exemple de la Figure 2.1, nous avons donc introduit le mot aa en tant que mot négatif alors qu'il était classé auparavant comme étant positif (sa vraie étiquette). L'automate résultant de l'apprentissage de RPNI sur ces nouveaux ensembles X_+ et X_- est donné Figure 3.1.

Comme nous pouvons le constater, les effets sur la complexité de l'automate sont importants : d'un automate à 4 états, nous passons à un automate à 7 états. Il a donc suffit de bruiter un seul et unique mot pour faire perdre au modèle sa capacité de généralisation. Si nous voulons inférer un automate à partir de données bruitées, il est alors nécessaire de traiter le bruit d'une façon ou d'une autre pour pouvoir utiliser les algorithmes à fusion d'états tels que RPNI. Autrement, le même problème se posera et l'automate appris risque de ne correspondre en rien à l'automate cible.

Comme nous l'avons dit précédemment, il existe deux manières de traiter le bruit : avant l'inférence (algorithme de type *filter*), ou pendant l'inférence (algorithme de type *wrapper*). En apprentissage automatique, ces deux types d'algorithme sont généralement utilisés pour réduire le nombre de données de l'ensemble d'apprentissage[3]. Toutefois, la plupart des travaux s'intéressent aux données numériques uniquement. Nous allons voir

[3]Par exemple, en enlevant les outliers (bruit) et les centres de clusters (peu intéressant pour étiqueter les données).

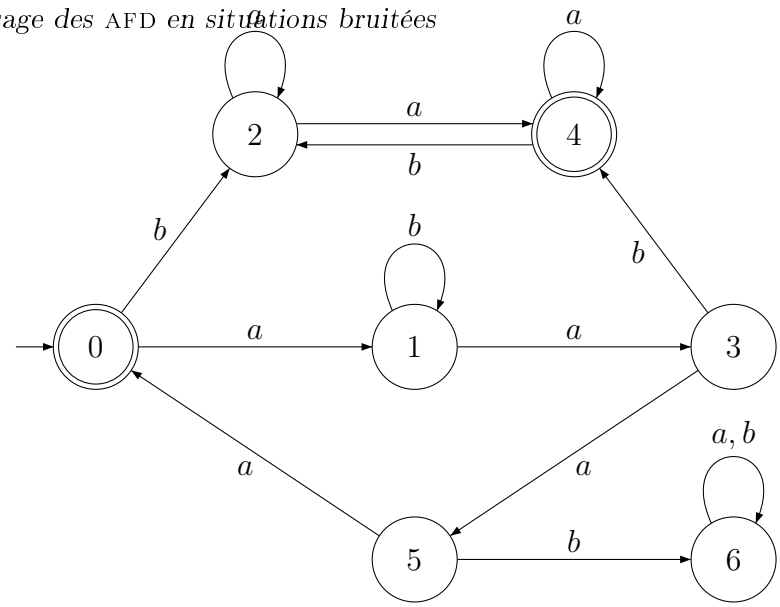

FIG. 3.1 – Automate résultant de l'exécution de RPNI sur un ensemble bruité.

dans un premier temps comment nettoyer les données d'apprentissage, puis dans un second temps les travaux qui visent à détecter le bruit pendant la phase d'apprentissage.

Filtrer les données en situations bruitées consiste à détecter quels exemples sont mal étiquetés. Dans [SJY02] cela est effectué via un test statistique. La probabilité conditionnelle mise en place permet alors de détecter les *outliers*, c'est-à-dire les vrais positifs (ou vrais négatifs) ayant des caractéristiques peu communes voire aberrantes[4] : si un mot w est étiqueté positif, mais que le test statistique indique que la probabilité pour que w soit positif est faible et que la probabilité pour que w soit négatif est forte, il y a de grandes chances pour que w ait subi l'influence du bruit. Une fois les données filtrées, RPNI est exécuté. Sur des automates cibles de taille 10, avec 8% de bruit, le taux de succès en généralisation passe alors d'environ 70% à 90%.

Le problème de cette approche concerne la définition des tests statistiques. En effet, ceux-ci ne prennent pas du tout en compte le fait que les langages soient rationnels. De plus, les tests reposent sur le voisinage des mots, et sur certains ensembles d'apprentissage ce test peut ne pas être fiable (typiquement, lorsqu'il y a peu de données). Pour éviter cela, les techniques dites *wrapper* détectent et traitent les données pendant l'apprentissage lui-même.

La plupart des travaux de ce type se basent sur l'adaptation des algorithmes de type RPNI. Ils modifient les algorithmes de telle sorte que la fusion de deux états est acceptée non plus si l'automate résultant est compatible avec les données X_+ et X_-, mais si la distribution engendrée par le nouvel automate n'est pas *trop* différente de celle engendrée par l'automate avant fusion.

[4]Un outlier est distant numériquement des autres données. Par exemple, dans une pièce la température des objets sera d'une dizaine de degré, exceptée celle du four allumé qui sera supérieure à 100°C : le four est alors un outlier.

Plus formellement, au lieu de tester uniquement si l'automate obtenu après fusion est compatible avec les données (ligne 7 de l'Algorithme 2), nous testons si les nombres d'erreurs avant fusion et après fusion ne sont pas significativement différents grâce à une loi normale et un paramètre de risque α. L'Algorithme 3 présente le principe de RPNI* étudié dans [SJ03].

Algorithme 3 : Pseudo-code de l'algorithme RPNI*

Données : Deux ensembles X_+ et X_-, un réel $\alpha \in [0; 0,5]$
Résultat : L'automate minimal reconnaissant tous les mots de X_+ et rejetant ceux de X_-

```
1  A ← PTA(X+, X−);
2  pour i ← 1 à |A| − 1 faire
3  |  j ← 0;
4  |  continuer ← Vrai;
5  |  tant que j < i et continuer faire
6  |  |  B ← fusion(A, i, j);
7  |  |  si Statistiquement_Compatible(B, X+, X−, α) alors
8  |  |  |  A ← B;
9  |  |  |  continuer ← Faux;
10 |  |  finsi
11 |  |  j ← j + 1;
12 |  fintq
13 finpour
14 retourner A;
```

Le paramètre α est lié au risque que nous prenons d'affirmer que les différences avant-après fusion ne sont pas significatives. Par conséquent, si $\alpha = 0$, l'algorithme acceptera toutes les fusions et retournera l'automate universel. À l'inverse, si $\alpha = 0,5$ une fusion sera rejetée dès qu'un exemple sera mal classé : s'il n'y a pas de bruit, le comportement sera alors le même que celui de RPNI.

Des améliorations de RPNI peuvent également être modifiées en suivant le même principe. RPNI teste pour chaque état s'il peut le fusionner avec un des états qu'il a déjà traité. S'il peut effectuer la fusion, il le fait immédiatement. Le problème est que cette fusion va peut-être avoir pour conséquence d'empêcher une autre fusion qui aurait été meilleure. Les algorithmes de type *blue-fringe* (*à frontière bleue*) résolvent ce problème en maintenant à jour un ensemble d'états dits rouge (qui ne peuvent plus être fusionnés entre eux) ainsi qu'un ensemble d'états dits bleus (les candidats pour une fusion avec un état rouge). Ils peuvent ainsi choisir le meilleur couple (état rouge, état bleu) à fusionner suivant un certain score (comme par exemple, le nombre de mots bien classé avant et après la fusion, le nombre d'états de l'automate, une combinaison des deux *etc.*). Nous avons proposé une telle modification avec l'algorithme BLUE* dans [SJT04]. Tout comme RPNI*, BLUE* effectue des tests statistiques afin de mieux prendre en compte le fait que certains exemples puissent être bruités. Tout comme les algorithmes *blue-fringe* ont de

meilleurs résultats que RPNI, les résultats de BLUE* sont meilleurs que ceux de RPNI*.

Concernant les résultats d'apprentissage, RPNI* peut, dans certains cas, correctement identifier des automates à dix états en ayant vu 1000 exemples dont environ 10% sont bruités. Ce n'est malheureusement pas toujours le cas. Nous discuterons d'autres résultats d'apprentissage de RPNI* et BLUE* dans la section suivante.

Il est à noter qu'il existe également des algorithmes permettant d'inférer des automates stochastiques (comme ALERGIA [CO94] ou MDI [TDdlH00]). Étant de nature probabiliste, certains de ces algorithmes peuvent identifier exactement des automates avec un faible niveau de bruit.

Nous terminons cette étude sur l'identification à la limite des automates en milieu bruité en nous intéressant au bruit non statistique.

Tout d'abord, il est à noter que si deux langages d'une classe ne sont pas distinguables une fois du bruit ajouté, alors la classe n'est pas identifiable à partir de données bruitées. Par exemple, soient les langages $L_1 = \{w : \exists k \in \mathbb{N}, |w| = 2k\}$, contenant les mots de longueurs paires, et $L_2 = \{w : \exists k \in \mathbb{N}, |w| = 2k + 1\}$, contenant les mots de longueurs impaires. Ces deux langages étant représentables par des automates (voir Figure 3.2), ils appartiennent donc à $\mathcal{AFD}(\Sigma)$.

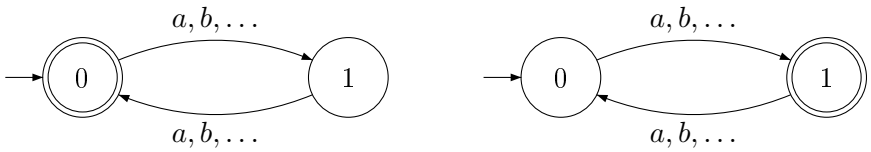

FIG. 3.2 – Automates reconnaissant le langage L_1 contenant tous les mots de longueurs paires (à gauche), et le langage L_2 contenant tous les mots de longueurs impaires (à droite).

Prenons maintenant le bruit systématique de degré 1, nous avons alors $N_1(L_1) = N_1(L_2) = \Sigma^*$. En effet, pour tout $w \in \Sigma^*$, il existe un mot de longueur paire (donc de L_1), ainsi qu'un mot de longueur impaire (donc de L_2), à distance au plus 1 de w (cela peut être w lui-même). En d'autres termes, nous ne pouvons distinguer L_1 de L_2 à partir d'une présentation bruitée. Nous en déduisons donc :

Théorème 10 $\mathcal{AFD}(\Sigma)$ *n'est pas identifiable à la limite à partir de présentation bruitée systématiquement.*

Enfin, dans [Ste97] puis [CJS01], les auteurs définissent certaines conditions permettant l'identification à partir de ce type de *NoisyTxt*. Par exemple, Case et Jain

introduisent la notion de *locking sequence*, similaire à celle de l'ensemble caractéristique, et montrent que toute classe de langages apprenable à partir *NoisyTxt* possède une *locking sequence* pour chacun des langages de la classe. Ils montrent alors, entre autre, que dans ce contexte toutes les familles indexées de langages récursifs peuvent être apprises. En revanche, dans [Ste97] il est montré que la classe de langages contenant tous les singletons peut être appris à partir de *NoisyTxt* mais pas de *VeryNoisyTxt*. De plus, si deux langages L, L' sont tels que $L \subset L'$, alors $\{L, L'\}$ ne peut être appris à partir de *NoisyTxt*. Nous avons alors :

Théorème 11 ([Ste97]) $\mathcal{AFD}(\Sigma)$ *n'est pas identifiable à la limite à partir de* VeryNoisyTxt *ni à partir de* NoisyTxt.

En revanche, Stephan puis Case, Jain et Sharma montrent qu'il existe plusieurs classes de langages apprenables à partir de *VeryNoisyTxt* ou de *NoisyTxt*.

3.3.2 Dans le cadre de l'apprentissage actif

Nous allons nous intéresser maintenant à l'apprentissage actif lorsque les oracles peuvent ne pas répondre à des requêtes (LMQ), ou répondre faux (MMQ). L'étude de l'apprentissage à partir d'oracle pouvant être *limité* ou *malicieux* est largement étudié dans [AKST97]. Comme les AFD ne peuvent être appris à partir de MQ seules, l'étude porte sur des oracles pouvant répondre à des EQ *et* des LMQ ou des EQ *et* des MMQ. Dans cet article, il est montré que l'identification polynomiale à partir de MMQ et EQ implique l'identification polynomiale à partir de LMQ et EQ. En effet, soit **Alg** un algorithme identifiant une classe $\mathcal{L}$ à partir de MMQ et EQ. Pour construire un algorithme identifiant $\mathcal{L}$ à partir de LMQ et EQ, il suffit d'utiliser **Alg** en remplaçant chaque réponse « je ne sais pas » par OUI ou NON. Ainsi, il est au moins aussi dur d'apprendre à partir de MMQ que de LMQ.

De plus, il est montré que $\mathcal{AFD}(\Sigma)$ est apprenable à partir de MMQ et de EQ. Pour ce faire, il faut modifier l'algorithme LSTAR vu précédemment[5] de façon à ce que le plus d'exemples (et de contre-exemples) possibles proviennent des requêtes d'équivalence qui elles sont sûres. L'algorithme tient alors à jour deux tables : la table CE des contre-exemples donnés par les EQ et la table *MA* des réponses aux MMQ. Il exécute alors LSTAR normalement, sauf qu'au lieu de faire une requête d'appartenance pour un mot w, il regarde d'abord si w n'est pas dans la table CE. Si ce n'est pas le cas, il effectue une MMQ et la réponse est ajoutée à la table MA. Après avoir reçu un contre-exemple d'une EQ, l'algorithme stocke la réponse dans CE et vérifie si elle est déjà contenue dans MA. Si elle y est avec la mauvaise étiquette, alors l'algorithme recommence de zéro en gardant uniquement la table CE, sinon, l'algorithme continue normalement.

Si l est le nombre de mots sur lesquels l'oracle peut mentir, alors nous pouvons montrer que cette technique permet d'apprendre en effectuant au plus $l + 1$ remise à zéro. En outre, le temps passé entre chaque mise à zéro est polynomial en la taille de l'automate cible. Nous en déduisons alors :

[5] Ou n'importe quel algorithme qui identifie à partir d'EQ et de MQ.

Théorème 12 ([AKST97])
- $\mathcal{AFD}(\Sigma)$ *est polynomialement identifiable à partir de* MMQ *et* EQ
- $\mathcal{AFD}(\Sigma)$ *est polynomialement identifiable à partir de* LMQ *et* EQ

Enfin, $\mathcal{AFD}(\Sigma)$ n'étant pas apprenable à partir de requêtes de correction, nous n'étudierons pas leur apprentissage lorsque l'oracle peut donner une correction erronée.

Le fait que peu de paradigmes permettent l'apprentissage des AFD à partir de données bruitées, mais que d'autres classes de langages le sont nous amène alors à une discussion du résultat de l'apprentissage des AFD en situation bruitées.

3.4 Discussion

Si l'apprentissage en milieu non bruité est *relativement* facile (dans le sens où un nombre considérable d'algorithmes apprenant correctement existent), ce n'est pas le cas lorsque les données subissent une quelconque forme de bruit. En effet, nous venons de voir que l'apprentissage exact n'est possible que lorsque nous apprenons avec un oracle malicieux, ou bien un oracle qui peut répondre « je ne sais pas »[6]. Dans la plupart des autres cas, nous pouvons montrer qu'il existe des cas où nous ne pouvons pas apprendre, et ce, sans même parler de polynomialité.

Le seul cadre dans lequel l'étude de l'apprentissage à partir de données bruitées semble être encourageant est celui de l'identification à la limite. Toutefois, si nous pouvons en effet montrer que plusieurs algorithmes montrent une certaine résistance au bruit, il faut en fait prendre ces résultats avec précaution.

Nous avons dit précédemment que RPNI*, comme BLUE*, pouvait apprendre avec environ 10% de bruit dans les données. Cependant, si les résultats d'apprentissage sont bons dans certains cas, ils peuvent ne pas l'être dans d'autres. En effet, s'il existe des AFD pour lesquels l'algorithme obtient un taux de succès en classification à plus de 98%, voire 100%, ce n'est pas toujours le cas. Pour certains automates, de même taille que ceux donnant un bon taux de classification, en fixant les mêmes paramètres à l'algorithme, le taux de succès peut descendre à 80%. De plus, bien que sensé être une amélioration de RPNI*, les résultats de BLUE* ne sont pas à la hauteur. Ils ne sont meilleurs que de quelques pourcents.

Deuxièmement, il est vrai que nous obtenons quelques bons résultats, mais ils sont obtenus pour des automates de très petites tailles (automates cible à 10 états) par rapport aux automates réellement utilisés en pratique (dont la taille dépasse facilement le millier d'états pour des applications comme le traitement de la langue naturelle par exemple). Si nous regardons ne serait-ce que l'apprentissage d'AFD à 60 états, le taux de succès varie entre 55 et 60%, ce qui est loin d'être acceptable.

L'apprentissage d'AFD à partir de données bruitées est donc un problème dur. Pour s'en rendre compte, il suffit de regarder le site de GOWACHIN [LPC98] permettant entre

[6] L'apprentissage repose alors le plus possible sur les requêtes d'équivalence qui elles ne sont pas bruitées.

autre d'obtenir des problèmes d'apprentissage d'automates en milieu bruité. Très peu d'auteurs se sont essayés à relever le défi. De plus, les taux de succès obtenus étant ceux cités précédemment, nous constatons clairement que l'apprentissage d'automates de taille supérieur à une dizaine d'états, ou avec un peu plus de bruit, est un problème qui est loin d'être résolu.

Nous pouvons également constater la difficulté de cet apprentissage en s'intéressant au concours proposé sur l'apprentissage d'AFD à partir de données bruitées par Lucas [Luc04]. Il prévoyait l'apprentissage d'automates à $10, 20, 30, 40$ et 50 états. Cependant, aucun des algorithmes présentés n'a réussi à dépasser un taux de réussite de 90% pour des automates à plus de 30 états. Les meilleurs du concours n'utilisent non pas les algorithmes « classiques » d'apprentissage d'automates (de type RPNI par exemple) mais les algorithmes génétiques[7].

Ces algorithmes à fusion d'états échouent dans l'identification d'automates à partir de données bruitées car le moindre mot ayant subit une modification aura des conséquences sur l'ensemble de l'automate : une mauvaise fusion acceptée, ou une fusion rejetée à tort, modifie complètement la structure de l'automate final. Le langage reconnu change alors radicalement. Les fusions faites après la (ou les) mauvaise fusion, ne feront qu'aggraver la situation car la généralisation (grâce aux cycles et aux boucles) sera faite aux mauvais endroits. Le fait que RPNI n'arrive pas à apprendre en présence de bruit est compréhensible car il n'est pas prévu pour cela. Toutefois son comportement en face de données bruitées permet de comprendre qu'une mauvaise fusion due au bruit à des conséquences dramatique sur le reste de son exécution. Dès lors, même si RPNI* ou BLUE* essaient de gérer ces données bruitées, les tests statistiques ne permettent pas de détecter tout le bruit. Ainsi, à la moindre fusion acceptée à tort, le comportement sera identique à celui de RPNI et la généralisation mauvaise.

Enfin, il est à noter que calculer le plus petit automate consistant avec des données X_+ et X_- est $\mathcal{NP}$-difficile [Gol78, Ang78]. De plus, Angluin a montré que dans le pire des cas, le problème de l'apprentissage d'un AFD à n états était $\mathcal{NP}$-difficile lorsque $2n + 1$ exemples sont absents [Ang78].

Nous pouvons ainsi clairement constater les limites de l'apprentissage d'AFD à partir de données bruitées. Il y a en fait plusieurs raisons qui font que c'est un problème dur.

Tout d'abord, le bruit habituellement considéré n'est pas du tout adapté à l'identification à la limite. En effet, le bruit statistique ne permet pas d'apprendre convenablement les langages : tout au plus, nous pouvons espérer ne pas nous tromper en affirmant que, parce qu'il n'apparaît que peu souvent, tel mot n'appartient pas au langage, ou que tel autre a une mauvaise étiquette, mais nous ne pourrons jamais en être certain. De plus, le problème change avec les statistiques : l'apprenant n'essaie en fait plus d'identifier le langage cible mais essaie plutôt d'identifier une distribution.

Ensuite, les problèmes de combinatoires soulevés lors de l'apprentissage d'AFD à partir de données bruitées et la complexité des langages font qu'il est quasiment impos-

[7] Ces travaux utilisent toutefois la connaissance du nombre d'états de l'automate cible. Ils font alors évoluer la matrice des transitions entre les états par mutations.

sible étant donné un ensemble X et un mot w de dire si w appartient bien à l'automate sensé reconnaître X. Que nous prenons le langage contenant les mots commençant par la lettre a, celui contenant les mots ayant un nombre pair de a, celui contenant les mots dont la parité du nombre de a est la même que celle du nombre de b, n'importe quel mot du langage sera à distance 1 d'un mot qui n'appartient pas au langage et réciproquement. Bien évidemment, cela n'est pas toujours le cas, mais ces exemples typiques montrent bien la difficulté de l'apprentissage à partir de données bruitées. De même, si nous reprenons les automates utilisés pour la preuve du Théorème 5.4, c'est-à-dire les automates reconnaissant Σ^* sauf un mot w, et que nous bruitons les données d'apprentissage, comment savoir parmi les mots (bruités) qui nous sont présentés comme négatifs, lequel est vraiment un mot négatif (si w n'a pas été lui même été bruité) ? Ainsi, il sera parfois impossible d'apprendre des automates à partir de données bruitées. Dans les autres cas, l'amélioration des algorithmes de type RPNI pour les rendre résistants au bruit doit passer par un changement des tests statistiques effectués pour mieux détecter les éléments bruités.

Conclusion

Nous venons de voir différents modèles de bruit envisageables. Toutefois, certains nous semblent plus raisonnables que d'autres. En particulier, le bruit doit se baser sur la distance d'édition ou sur des variantes (ce qui est souvent le cas en biologie par exemple). Cependant, dans ce contexte, l'apprentissage des AFD à partir de données bruitées est difficile. Pour autant, cela ne signifie pas que l'apprentissage à partir de données bruitées en général l'est obligatoirement.

Nous pourrions alors envisager deux solutions pour apprendre en situations bruitées : revoir le protocole de l'identification des AFD ou changer de classes de langages. Toutefois, nous venons de voir que les différents types de bruit envisagés précédemment permettent de conclure que quelque soit le type de bruit utilisé, les AFD ne seront pratiquement jamais identifiables. Ainsi, puisque les langages de plus bas niveau de la hiérarchie de Chomsky ne sont pas apprenables en situations bruitées, il en sera de même avec les langages de plus haut niveau (les langages rationnels y étant inclus).

D'un autre côté, le fait est que si la hiérarchie proposée par Chomsky a des fondements du point de vue de la linguistique, elle n'en a que peu du point de vue de l'apprentissage. De ce fait, l'apprentissage à partir de données bruitées ne peut se faire en considérant comme point de départ les AFD[8]. Nous pouvons alors choisir de s'intéresser à des classes de langages plus simples d'un point de vue topologique, qui elles seraient plus résistantes au bruit. Intuitivement, nous arrivons plus facilement à retrouver un disque bruité dans le plan qu'une figure plus complexe. Comme la définition du bruit repose sur la distance d'édition, les candidats raisonnables semblent alors être les langages dont la définition se base aussi sur la distance d'édition, comme par exemple les boules de mots. Cependant, nous pouvons nous demander si les boules de mots ne

[8]Il est à noter que les langages rationnels ne sont pas les seuls à souffrir du bruit. Tous les langages comme $\{a^n b^m : n, m \in \mathbb{N}\}$ où il faut compter les lettres, ne seront que peu ou prou résistants au bruit.

sont pas simplement des langages « jouets » ayant certainement de belles propriétés théoriques, mais n'ayant aucun intérêt en pratique. De plus, sont-elles résistantes au bruit ?

Chapitre 4

Les langages à base de distance

Dans le chapitre précédent, nous avons vu que les langages de base de la hiérarchie de Chomsky que sont les langages réguliers ne sont pas adaptés à un apprentissage à partir de situations bruitées. En effet, en présence de bruit, ils ne sont que rarement apprenables. Nous proposons ici ce que pourraient être les premières briques d'une hiérarchie alternative à la hiérarchie de Chomsky qui serait résistante au bruit : les langages topologiques.

Dans ce chapitre nous allons donc voir les éléments essentiels à la compréhension des langages topologiques. Dans un premier temps nous donnerons une définition formelle de plusieurs langages topologiques. Nous nous intéresserons ensuite plus particulièrement aux boules de mots (un langage topologique particulier). Nous étudierons alors quelques propriétés de ces langages afin de montrer que derrière la simplicité de leur définition se cache entre autre une réelle complexité combinatoire mais aussi une bonne résistance au bruit. Nous montrerons également que les boules de mots ne sont pas simplement des *exemples-jouets* théoriques mais sont déjà utilisés en pratique, par exemple pour faire de la correction orthographique, le centre de la boule représentant alors le mot à corriger et les mots de la boule les corrections possibles. Enfin, puisque nous voulons étudier l'apprentissage des boules en situations bruitées, nous conclurons ce chapitre en montrant que sous certaines contraintes d'apprentissage, les boules de mots ne sont pas apprenables.

4.1 Langages topologiques et boules de mots

Nous avons vu au Chapitre 2 que la distance d'édition jouait un rôle prépondérant dans la notion de bruit d'édition. Nous allons donc nous servir de cette distance pour définir de nouveaux langages : les langages dits *topologiques*. En effet, si nous prenons l'exemple de l'espace euclidien, intuitivement, les figures qui résisteraient le mieux au bruit seraient les figures géométriques les plus simples : les disques, les parallélogrammes, *etc.* En d'autres termes, les figures pour lesquelles la distance joue un rôle important dans leurs définitions.

4.1.1 Définitions de langages topologiques

Dans Σ^*, le langage le plus « simple » d'un point de vue topologique que nous puissions définir est la *boule de mots* :

Définition 31 (Boules de mots) *Étant donné $d(\cdot,\cdot)$ une distance sur Σ^*, la boule de centre $o \in \Sigma^*$ et de rayon $r \in \mathbb{N}$ est définie par $B_r(o) = \{w \in \Sigma^* : d(w,o) \leq r\}$.*

Nous noterons $\mathcal{B}_\Sigma$ l'ensemble de toutes les boules : $\mathcal{B}_\Sigma = \{B_r(o) : r \in \mathbb{N}, o \in \Sigma^*\}$.

Exemple 27 *Soit $\Sigma = \{a,b\}$. On a $B_1(ba) = \{a,b,aa,ba,bb,aba,baa,bab,bba\}$ et $B_r(\lambda) = \Sigma^{\leq r}$ pour tout rayon $r \in \mathbb{N}$.*

Dans le même esprit, nous pouvons définir une *ellipsoïde* à partir de deux foyers :

Définition 32 (Ellipsoïdes) *Soient $u,v \in \Sigma^*$, l'ellipsoïde de foyers u et v est définie par $\bigcirc_k(u,v) = \{w \in \Sigma^* : d(u,w) + d(w,v) \leq 2 \times k\}$ où $k \in \mathbb{N}/2$ est la distance du grand axe.*

Exemple 28 *Soit $\Sigma = \{a,b\}$. L'ellipsoïde de foyers aba et bab ayant une distance de grand axe une distance de $1,5$ est donc $\bigcirc_{1,5}(aba,bab)$=$\{w \in \Sigma^* : d(aba,w) + d(w,bab) \leq 3\} = \{aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaba, abaa, abab, baba, babb, bbab\}$.*

Une classe de langages montrant que Σ^* ne se comporte pas comme l'espace euclidien est celle des segments :

Définition 33 (Segments) *Soient $u,v \in \Sigma^*$. Le segment d'extrémités u et v est défini par $[u,v] = \{w \in \Sigma^* : d(u,w) + d(w,v) = d(u,v)\}$.*

Exemple 29 *Soit $\Sigma = \{a,b\}$. Le segment d'extrémités aba et bab est alors défini par $[aba,bab] = \{ab, ba, aba, bab, abab, baba\}$.*

Nous pourrions nous attendre à ce qu'un segment entre deux mots de longueurs 3 ne contiennent que des mots de longueurs 3, mais comme nous venons de le voir, ce n'est pas vrai. Plus surprenant encore, soit $[X,Y]$ un segment de $\mathbb{R}^n$ et Z un point de ce segment. Nous avons alors, $\forall Z' \in [X,Y], d(X,Z) = d(X,Z') \implies Z = Z'$, c'est-à-dire une unicité du point Z comme étant *le* point à distance $d(X,Z)$ de X. Dans Σ^* avec la distance d'édition, ce n'est plus le cas : $d(aba,ab) = d(aba,baba) = 1$ et $d(bab,ab) = d(bab,baba) = 1$, les mots ab et $baba$ pourraient donc tous les deux être des « milieux » du segment $[aba,bab]$.

En fait, un segment est un cas particulier d'une ellipse : la condition $d(u,w) + d(w,v) = d(u,v)$ peut se décomposer en $d(u,w)+d(w,v) \geq d(u,v)$ (condition d'inégalité triangulaire qui est toujours vraie) et $d(u,w) + d(w,v) \leq d(u,v)$ que nous pouvons réécrire en $d(u,w) + d(w,v) \leq 2k$. Nous avons alors bien $[u,v] = \bigcirc_{\frac{d(u,v)}{2}}(u,v)$.

Notons enfin, qu'un simple mot suffit à définir un langage topologique infini :

Définition 34 (Cônes et co-cônes) *Soit un mot $u \in \Sigma^*$, le cône de sommet u, noté $\hat{u}$ est défini par l'ensemble des sur-mots de u : $\hat{u} = \{w \in \Sigma^* : u \preceq w\}$. De même, l'ensemble des sous-mots de u, noté $\check{u}$ défini le co-cône de sommet u : $\check{u} = \{w \in \Sigma^* : w \preceq u\}$*

Pour chaque figure géométrique de l'espace euclidien, nous pourrions donner une définition d'un langage topologique dans Σ^*. Toutefois, notre intention n'est pas ici de faire une liste exhaustive des langages topologiques, mais plutôt de montrer que de tels langages existent et qu'ils sont intéressants du point de vue de la théorie des langages. De plus, nous voulons montrer que les langages topologiques sont résistants au bruit. Pour se faire, nous nous intéresserons par la suite uniquement à une classe de ces langages : la classe des boules de mots.

Contrairement aux AFD où il est plus que délicat de prédire ce que sera le bruité d'un langage reconnu par un automate, les boules se comportent bien en présence de bruit. En particulier, le bruité (systématique) d'une boule est une boule :

Théorème 13

$$N_k(B_{k'}(u)) = B_{k+k'}(u)$$

Démonstration :

- ($\subseteq$) Si x appartient au bruité de la boule, $x \in N_k(B_{k'}(u))$. Il existe alors un mot $y \in B_{k'}(u)$ tel que $d(y,x) \leq k$, c'est-à-dire un mot y tel que $d(u,y) \leq k'$ et $d(y,x) \leq k$. Nous avons donc $d(u,x) \leq k' + k$.
- ($\supseteq$) Soit $x \in B_{k+k'}(u)$ alors $d(u,x) \leq k + k'$. Supposons que $d(u,x) > k'$ (sinon c'est trivial). Le fait que $k' < d(u,x) \leq k + k'$ signifie que u peut être transformé en x par au plus $k + k'$ opérations d'édition. Soit y la chaîne obtenue après les k' premières opérations. Alors $d(u,y) = k'$ et $d(y,x) \leq k$; il s'en suit que $y \in B_{k'}(u)$ et $x \in N_k(B_{k'}(u))$.

□

4.1.2 Représentations des boules de mots

Même si la définition des boules de mots paraît simplissime, il ne faut pas se fier aux apparences. Par exemple, le nombre de mots qu'elles contiennent est exponentiel en fonction du rayon dès que $|\Sigma| \geq 2$. C'est le cas par exemple de la boule $B_r(\lambda) = \Sigma^{\leq r}$: pour construire un mot de longueur r, il y a $|\Sigma|$ façons de choisir la première lettre, $|\Sigma|$ façons de choisir la seconde, ... soit $|\Sigma|^r$ mots de longueurs r. Nous en déduisons que $|B_r(\lambda)| = 1+2+2^2+\cdots+2^{r-1}+2^r$, ce qui correspond à une suite géométrique. La boule de centre λ et de rayon r contient donc $1+2+2^2+\cdots+2^{r-1}+2^r = (|\Sigma|^{r+1}-1)/(|\Sigma|-1)$ mots. Il est cependant difficile d'estimer ce nombre dans le cas d'une boule quelconque. Toutefois, nous pouvons nous faire une idée expérimentalement (voir Table 4.1.2). Nous constatons clairement que pour une longueur de centre fixée avec $|\Sigma| = 2$, le nombre moyen de mots fait plus que doubler chaque fois que le rayon augmente de un. Compte

Longueur du centre	Rayon					
	1	2	3	4	5	6
0	3,0	7,0	15,0	31,0	63,0	127,0
1	6,0	14,0	30,0	62,0	126,0	254,0
2	8,6	25,6	56,5	119,7	246,8	501,6
3	10,8	41,4	101,8	222,8	468,6	973,0
4	13,1	61,4	173,8	402,9	870,9	1850,8
5	16,3	91,0	285,1	698,5	1584,4	3440,9
6	17,9	125,8	441,2	1177,5	2771,3	6252,9
7	21,2	166,9	678,0	1908,8	4835,8	11233,5
8	24,3	200,2	1034,2	3209,9	8358,1	19653,6
9	26,0	265,4	1390,9	5039,6	13677,8	34013,1

TAB. 4.1 – Nombre moyen de mots dans une boule pour un alphabet à 2 lettres. Les centres sont tirés au hasard, chaque calcul se fait en moyennant sur 20 centres.

tenu de ces chiffres, il est alors inenvisageable de stocker l'ensemble de tous les mots d'une boule en machine pour travailler.

La quantité de mots contenus dans une boule grandissant de façon exponentielle, cela nous amène alors directement à la question de la représentation d'une boule. En effet, puisque les boules sont des langages finis et donc réguliers, nous serions tentés de prime abord d'utiliser des automates pour les représenter. Il est par exemple facile de montrer que la boule $B_r(o)$ peut être reconnue par un automate avec λ-transitions ayant $\mathcal{O}(|o| \cdot r)$ états (voir Figure 4.1). Un automate avec λ-transitions correspondant à un automate fini non déterministe dont les transitions peuvent se faire sur λ.

Cependant, les automates non-déterministes ne sont pas de bons candidats pour représenter les boules de mots. En effet, il a été montré qu'ils ne sont pas apprenables dans la plupart des paradigmes d'identification [AK91, dlH97]. Les automates finis déterministes correspondants, eux, n'ont pas ce fâcheux désavantage. Cependant, les expérimentations montrent qu'ils ont souvent un nombre d'états exponentiel.

Il existe plusieurs algorithmes de construction d'AFD reconnaissants $B_r(o)$. En particulier, Schulz et Mihov introduisent dans [SM02] les *automates de Levenshtein* en proposant une construction directe dont le nombre d'états, ainsi que le temps et l'espace de construction, sont linéaires en $|o|$ mais exponentiels en r (voir Table 4.2). La taille de l'alphabet elle-même ne joue pas sur le nombre d'états, mais seulement sur le nombre de transitions.

Rayon r	1	2	3	4
Nombre d'états	$5 \cdot \|o\|$	$30 \cdot \|o\|$	$180 \cdot \|o\|$	$1353 \cdot \|o\|$

TAB. 4.2 – Nombre d'états de l'automate de Levenshtein [SM02] reconnaissant $B_r(o)$ lorsque $|\Sigma| \geq 2$.

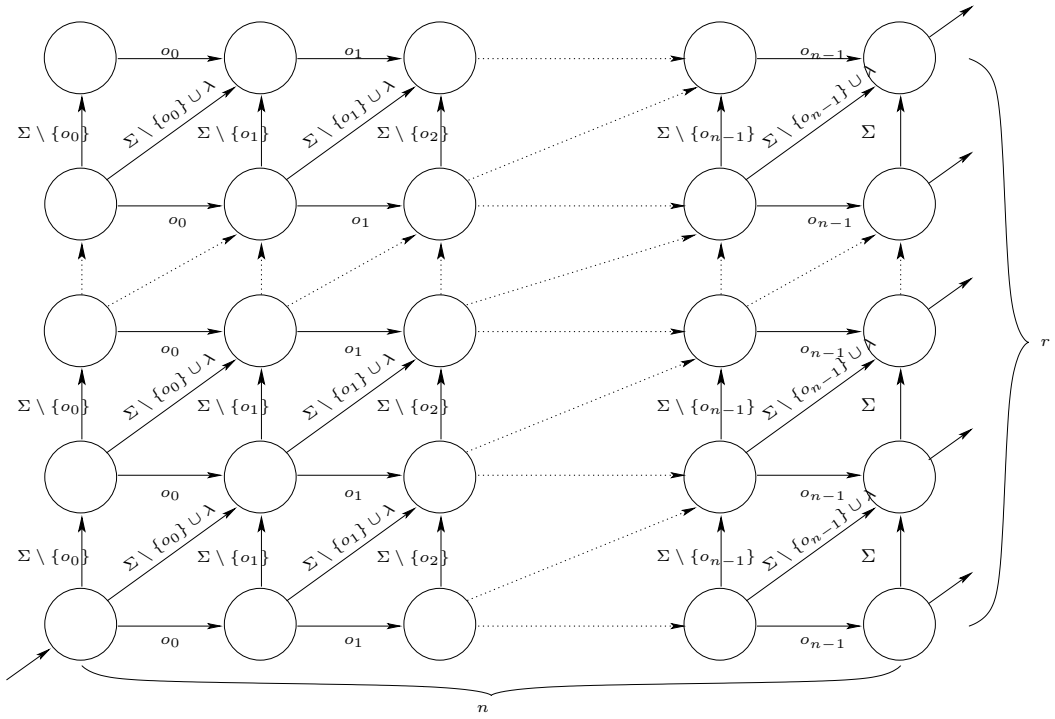

FIG. 4.1 – Automate avec λ-transitions reconnaissant la boule $B_r(o)$.

Malheureusement, les automates de Levenshtein retournés par leur algorithme ne sont pas minimaux, et prouver que les AFD minimaux reconnaissants $B_r(o)$ ont aussi un nombre d'états exponentiel est un vrai challenge combinatoire. Nous poserons donc ici la conjecture :

Conjecture 1 *Dans le pire cas, l'*AFD *minimal de* $B_r(o)$ *contient* $\Omega(2^r \cdot |o|)$ *états.*

Il est assez frustrant de ne pas avoir plus qu'une conjecture. Malheureusement, les seuls travaux s'intéressant à la taille de l'AFD minimal reconnaissant une boule de mots de notre connaissance utilisent la distance de Hamming [EM97].

Quoiqu'il en soit, il est clair au vu des expérimentations que choisir un AFD pour représenter une boule n'est pas très pertinent. De plus, si passer d'un ensemble de mots correspondant à une boule à un automate est chose facile, il n'en est pas de même de la réciproque, c'est-à-dire de la question : étant donné un automate A fini, déterministe et minimal, quelle est, si elle existe, la boule reconnue par l'automate ? C'est une question que nous laissons ouverte et sur laquelle nous reviendrons au chapitre suivant.

D'un autre côté, pourquoi ne pas représenter la boule $B_r(o)$ par la paire (o, r) ? La taille de cette représentation correspond au nombre de bits nécessaires pour encoder cette paire, soit une taille de $|o|+\log r$. La taille de la paire n'est en effet pas $|o|+r$: cela correspondrait à un encodage en base 1 du rayon r, ce qui n'est pas raisonnable (voir par exemple [GJ79]). De plus, décider si un mot w appartient à la boule $B_r(o)$ est quasiment immédiat. Il suffit en effet de calculer dans un premier temps la distance entre w et le centre de la boule, le calcul de $d(o, w)$ étant en $\mathcal{O}(|o| \times |w|)$. Puis vérifier si cette distance est plus petite que le rayon, c'est-à-dire si $d(o, w) \leq r$. Le problème de l'appartenance du mot w à la boule $B_r(o)$ peut alors être résolu en temps $\mathcal{O}(|o| \times |w| + \log r)$.

Nous prendrons donc le couple (o, r), de taille $|o| + \log r$, comme représentation de la boule $B_r(o)$. En effet, quand l'alphabet a au moins deux lettres, (o, r) est un représentant unique, donc *canonique*, de la boule $B_r(o)$:

Théorème 14 *Si* $|\Sigma| \geq 2$ *et* $B_{r_1}(o_1) = B_{r_2}(o_2)$, *alors* $o_1 = o_2$ *et* $r_1 = r_2$.

Afin de montrer l'unicité de la représentation des boules, nous allons nous servir de deux lemmes. Le premier lemme montre que si deux boules sont égales, les longueurs de leurs plus grands mots sont égales :

Lemme 4 $B_{r_1}(o_1) = B_{r_2}(o_2) \Rightarrow |o_1| + r_1 = |o_2| + r_2$.

Démonstration :
Soit $w \in \Sigma^{r_1}$, alors $d(o_1, o_1 w) = |w| = r_1$ par la Proposition 1. Le mot $o_1 w$ appartient donc à la boule $B_{r_1}(o_1)$, et donc à $B_{r_2}(o_2)$. Nous avons donc $d(o_1 w, o_2) \leq r_2$, et $d(o_1 w, o_2) \geq |o_1 w| - |o_2|$, toujours d'après la Proposition 1. Nous en déduisons alors que $r_2 \geq |o_1 w| - |o_2| = |o_1| + r_1 - |o_2|$, soit $r_2 + |o_2| \geq |o_1| + r_1$. De façon symétrique, nous pouvons montrer que $r_2 + |o_2| \leq |o_1| + r_1$. Nous en déduisons alors que $|o_1| + r_1 = |o_2| + r_2$. □

D'après le second lemme, s'il existe deux boules telles que le centre de la première n'est pas un sur-mot du centre de la seconde, nous pouvons construire un mot w de

longueur maximale de la première boule telle que le centre de la seconde boule ne soit pas un sous-mot de w :

Lemme 5 *Si* $|\Sigma| \geq 2$ *et* $o_2 \not\preceq o_1$*, alors, il existe un mot* $w \in \Sigma^*$ *et un entier* $r \in \mathbb{N}$ *tels que :*

1. $|w| = r + |o_1|$,
2. $o_1 \preceq w$ *et*
3. $o_2 \not\preceq w$

Démonstration :
Supposons, sans perte de généralité, que o_2 commence par la lettre a, et soit b une lettre de Σ^* différente de a : $b \in \Sigma \setminus \{a\}$. Nous construisons alors le mot w comme étant la concaténation de r fois la lettre b et du centre o_1 : $w = b^r o_1$. La longueur de w est alors bien de $|w| = r + |o_1|$ (1). De plus, par construction, $o_1 \preceq w$ (2). Enfin, comme $o_2 \not\preceq o_1$, que o_2 commence par un a et w par un b, $o_2 \not\preceq w$ (3). □

Les Lemmes 4 et 5 permettent alors de déduire le Théorème 14 :
Démonstration :

($\Leftarrow$) Trivial.

($\Rightarrow$) Supposons dans un premier temps que les deux centres soient différents : $o_1 \neq o_2$. Deux cas sont alors possible : soit $o_1 \not\preceq o_2$, soit $o_2 \not\preceq o_1$. Supposons, sans perte de généralité, que $o_2 \not\preceq o_1$. Par le Lemme 5, il existe alors une chaîne w telle que

1. $|w| = r_1 + |o_1|$,
2. $o_1 \preceq w$ et
3. $o_2 \not\preceq w$.

Le point 2 nous dit que $o_1 \preceq w$, nous en déduisons d'après la Proposition 1 que $d(o_1, w) = |w| - |o_1|$. Et donc, d'après le point 1 que $d(o_1, w) = |w| - |o_1| = r_1$. En d'autres termes, cela signifie que w appartient à la boule $B_{r_1}(o_1)$.

Maintenant, le point 3 précise que $o_2 \not\preceq w$. Toujours d'après la Proposition 1, nous en déduisons que $d(o_2, w) > \big||w| - |o_2|\big| = \big|r_1 + |o_1| - |o_2|\big| = r_2$ (d'après le point 1 et le Lemme 4). D'où $w \notin B_{r_2}(o_2)$.

w appartiendrait donc à $B_{r_1}(o_1)$ mais pas à $B_{r_2}(o_2)$, ce qui est impossible. L'hypothèse que les centres sont différents est donc fausse. En conséquence, nous avons nécessairement $o_1 = o_2$ et, d'après le Lemme 4, $r_1 = r_2$.

□

Il est à noter que le théorème n'est vrai que si l'alphabet est de taille au moins deux. En effet, si l'alphabet n'a qu'une lettre, une même boule peut admettre plusieurs représentations.

Exemple 30 *Soit $\Sigma = \{a\}$, la boule contenant les mots $\{\lambda, a, aa, aaa, aaaa\}$ peut être représenté par les couples $(\lambda, 4), (a, 3), (aa, 2)$. En d'autres termes, si $\Sigma = \{a\}$ alors $B_4(\lambda) = B_3(a) = B_2(aa)$.*

De même, elles ne pourront être identifiées à partir de bruit systématique :

Exemple 31 *Soit $\Sigma = \{a\}$. Nous pouvons avoir deux boules différentes : $B_2(a) \neq B_1(aa)$, mais $N_1(B_2(a)) = N_1(B_1(aa))$. En effet, $N_1(B_2(a)) = B_3(a) = B_2(aa) = N_1(B_1(aa))$.*

Enfin, certaines boules peuvent poser des problèmes de complexité. Par exemple, aucun mot en dehors de la boule $B_r(\lambda) = \Sigma^{\leq r}$ n'est de taille polynomiale en $0 + \log r$, c'est-à-dire en la taille de la boule. C'est le cas également de toutes les boules telles que $r > 2^{|o|}$. Nous distinguerons donc par la suite ce que nous appellerons les *bonnes* boules des *mauvaises* boules :

Définition 35 (Bonnes boules) *La boule $B_r(o)$ est une* bonne boule *si et seulement si $r \leq |o|$. Nous dirons que c'est une mauvaise boule sinon.*

Cette définition n'est pas farfelue : si nous prenons l'exemple des travaux sur la correction orthographique, cela signifie que nous cherchons des corrections de petite longueur si le mot est petit, ou encore, que nous concevons que nous faisons plus de fautes sur un mot plus long. Toutefois, afin d'être moins strict, nous pouvons également introduire la notion de boule $q()$-bonne :

Définition 36 ($q()$-bonnes boules) *Soit $q()$ une fonction polynôme*[1]*. La boule $B_r(o)$ est une* $q()$-bonne boule *si et seulement si $r \leq q(|o|)$. Une bonne boule est donc une $q()$-bonne boule pour la fonction polynôme $q(x) = x$.*

Nous noterons alors $\mathcal{BB}(\Sigma)$ la classe de toutes les *bonnes* boules et $\mathcal{BOULE}(\Sigma)$ la classe de toutes les boules de mots.

4.2 Avantages et inconvénients des boules de mots

Nous allons maintenant nous concentrer sur certaines propriétés des boules de mots. En particulier, sur des propriétés souvent surprenantes et qui peuvent parfois poser problème lors de l'étude des boules : les intuitions qui sont vraies pour les disques de $\mathbb{R}^2$ ou les boules de $\mathbb{R}^3$ sont fausses pour celles de Σ^*. Dans un second temps nous verrons que les boules de mots sont déjà utilisées dans divers travaux, bien que rarement nommées.

[1] En fait, $q()$ ne peut être une fonction polynôme quelconque. En effet, pour la fonction polynôme $q(x) = -x$, une $q()$-bonne boule serait telle que $r \leq -|o|$... Nous choisirons donc de préférence des fonctions polynômes monotone croissante telles que $q(0) \geq 0$, ou encore, des polynômes à coefficients dans $\mathbb{N}$.

4.2.1 Propriétés et contre-intuitions

Tout d'abord, avec le problème de la représentation, nous avons pu constater que les boules de mots sont différentes de celles de l'espace euclidien et ne sont pas aussi simples que leurs définitions le laissent croire. Le théorème suivant montre que cette difficulté est bien réelle. En effet, trouver la boule de rayon minimum contenant un ensemble de points est $\mathcal{NP}$-difficile :

Théorème 15 ([dlHC00]) *Étant donnés un ensemble fini de mots $W = \{w_1, \ldots, w_n\}$ et une constante K, décider si un mot $z \in \Sigma^*$ existe tel que $\sum_{w \in W} d(z,w) < K$ (respectivement $\max_{w \in W} d(z,w) < K$) est $\mathcal{NP}$-complet.*

Malgré ce problème, connu sous le nom de *median string* [dlHC00, JABC03], et la grande taille des boules de mots, ces dernières présentent une propriété intéressante qui nous sera souvent utile pour identifier les boules :

Proposition 3 *Soit $B_r(o)$ une boule. Parmi tous les couples de la forme $(a^i u, b^i u)$ tels que $|u| + i = |o| + r$, c'est-à-dire tels que les mots sont parmi les plus longs de la boule, $(a^r o, b^r o)$ est l'unique couple avec le plus grand i.*

Démonstration :
Raisonnons par l'absurde. Soit $(a^i u, b^i u)$ un couple tel que $|u| + i = |o| + r$ avec $i > r$, et donc $|o| > |u|$. Nous avons alors $|a^i u| = |o| + r$ donc $d(o, a^i u) \geq r$ par la Proproposition 1. Comme $a^i u$ appartient à la boule $d(o, a^i u) \leq r$ donc $d(o, a^i u) = r$. Nous en déduisons alors que $o \preceq a^i u$ toujours par la même proposition. Par le même raisonnement, nous avons $o \preceq b^i u$. Nous en déduisons donc que $o \preceq u$ ce qui contredit $|o| > |u|$. □

Les mots $a^r o$ et $b^r o$ correspondent aux mots construits à partir du centre, en faisant une insertion de r fois la lettre a (respectivement la lettre b) en début de mot. Ils sont donc à distance r du centre, et appartiennent aux mots de longueurs maximales de la boule. Il est à noter que d'autres couples partagent cette propriété, par exemple le couple (oa^r, ob^r), mais nous nous concentrerons sur les mots $a^r o$ et $b^r o$ par la suite.

Ensuite, il est faux de penser que les boules de mots sont des objets symétriques et homogènes. Nous avons en effet pu le constater avec le Tableau 4.1.2 : la plupart du temps, dans l'ensemble de mots représenté par une boule, la moitié de ces mots sont de longueurs maximales. En d'autres termes, dans la boule $B_r(o)$ pratiquement la moitié des mots sont de longueur $|o| + r$ (par exemple, $|B_r(\lambda)| = |\Sigma^{\leq r}| = \frac{|\Sigma^{r+1}|-1}{|\Sigma^r|-1} = 2^{r+1} - 1$ si $|\Sigma| = 2$ et $|B_r^{max}(\lambda)| = |\Sigma^r| = 2^r$).

De même, si nous comparons deux boules de même rayon et dont les centres sont de même longueur, nous sommes en mesure de s'attendre à ce qu'elles contiennent le même nombre de mots, tout comme dans $\mathbb{R}^2$ deux disques qui ont un même rayon ont une même surface. Dans Σ^*, la cardinalité de deux boules d'apparence identique peut varier pratiquement du simple au double :

Exemple 32 *Soient les boules $B_2(aaaabbbb)$ et $B_2(abababab)$:*
- $|B_2(aaaabbbb)| = 172$

- $|B_2(abababab)| = 254$

Une autre propriété qui est vraie dans l'espace euclidien est qu'un disque de rayon r ne peut contenir que des disques dont le rayon est inférieur à r. Encore une fois, cette propriété est fausse dans Σ^* :

Exemple 33 *Une boule de rayon* 5 *peut être incluse dans une boule de rayon* 4 *:* $B_5(ab) \subset B_4(abab)$.

La cardinalité d'une boule n'est donc pas une fonction croissante en fonction du rayon.

Enfin, la propriété qui est certainement parmi les plus contre-intuitives est que les boules de mots de sont pas « convexes ». Toujours à titre de comparaison, si nous prenons deux points dans un disque de $\mathbb{R}^2$ et que nous traçons le segment reliant ces deux points, le segment est intégralement inclus dans le disque, quels que soient les deux points pris. Dans Σ^*, un mot peut ne pas appartenir à une boule alors qu'il le devient en effectuant n'importe quelle opération d'édition :

Exemple 34 *Soient la boule* $B_4(aabb)$ *et le mot bbbaaa. Comme* $d(aabb, bbbaaa) = 5$ *le mot n'appartient pas à la boule. Cependant :*

- *En effectuant une insertion, le mot peut appartenir à la boule. Par exemple, en insérant un a en début de chaîne, ou un b en fin de chaîne :* $\underline{a}bbbaaa, bbbaa\underline{b}a \in B_4(aabb)$.
- *Le mot obtenu en substituant n'importe quelle lettre de bbbaaa, appartient lui aussi à la boule :* $b\underline{a}baaa, bbb\underline{b}aa \in B_4(aabb)$.
- *Enfin, tous les mots obtenus en supprimant une lettre à bbbaaa sont à distance* 4 *du centre de la boule :* $\underline{\cdot}bbaaa, bbb\underline{\cdot}aa \in B_4(aabb)$.

Le mot *bbbaaa* n'est pas dans $B_4(aabb)$, mais des mots de longueur $5, 6$ et 7 à distance 1 de *bbbaaa* le sont. Cela montre donc que la frontière des boules n'est pas du tout régulière, comme l'est celle des disques de $\mathbb{R}^2$, et qu'une fois de plus, les intuitions que nous pouvons avoir grâce à $\mathbb{R}^2$ ne sont pas forcément vraies dans Σ^*.

4.2.2 Utilité et applications

Bien qu'ayant des propriétés pouvant être déroutantes, les boules de mots sont des objets réellement utiles. En effet, elles sont utilisées dans certains travaux, bien que leurs noms n'apparaissent pas explicitement.

La première catégorie de travaux utilisant les boules de mots est celle de la recherche approchée d'un mot dans un texte (*approximate string matching* ou *fuzzy string matching*). Pour une étude plus approfondie, se référer à [SK83, Nav01]. Le but général est de trouver des correspondances d'un *mot* dans un *texte* où soit le mot, soit le texte (voire les deux) ont subi une forme quelconque de corruption : retrouver des signaux d'origine après leurs transmissions à travers un milieu bruité, trouver des séquences ADN après des mutations, ou simplement rechercher un mot dans un texte en présence de fautes d'orthographe.

Le problème, sous sa forme générale, est donc de trouver les endroits dans un texte T où un mot M apparaît, en autorisant un nombre limité k d'erreurs dans la correspondance mot/texte. En d'autres termes, trouver les mots de la boule de centre M et de rayon k dans T. Les applications peuvent se servir de différents modèles d'erreur, mais la plupart emploient, ou se ramènent à, la distance d'édition. Les domaines d'utilisation vont de la biologie [Gus97], à la recherche de musique [Lem00].

Si le terme de *boules de mots* n'est pas employé, ces dernières sont souvent utilisées. Par exemple, dans [Ukk85, Mel95, BYN99, BYN02], les auteurs construisent un automate reconnaissant les mots de $\Sigma^* \cdot B_k(M)$ afin de se retrouver dans un état final lorsqu'ils ont trouvé une occurence approchée de M dans T.

Un problème similaire à l'*approximate string matching* est celui de la correction. Il s'agit plus particulièrement de trouver de bonnes corrections à un mot qui a été altéré. Ce problème est donc important dans bien des applications : correction orthographique, reconnaissance de la parole, reconnaissance manuscrite, *etc.* Beaucoup possèdent un dictionnaire (correspondant au texte T du problème précédent). Une façon typique de procéder est alors la suivante : étant donné un mot M, chercher si M est dans le dictionnaire. Si ce n'est pas le cas, les mots du dictionnaire les plus similaires à M sont proposés comme corrections possibles. La similarité à M avec les corrections étant encore une fois basée sur la distance d'édition.

Dans [Ofl96] et [SM02] par exemple, le dictionnaire est vu comme un automate. Le premier parcours le dictionnaire d'une façon exhaustive, tandis que le second construit un automate de Levenshtein, c'est-à-dire un AFD reconnaissant une boule de mots. Ainsi, l'intersection des deux automates permet de retrouver les corrections à une certaine distance de M présents dans le dictionnaire.

Hormis ces deux champs d'applications, qui sont ceux où nous utilisons le plus souvent les boules de mots, nous pouvons les retrouver dans divers autres travaux.

En recherche de plus proches voisins par exemple, lorsque les données sont des mots et que les calculs des distances se font à l'aide de la distance d'édition, le temps de calcul peut vite devenir trop important pour les gros ensembles de données. Ainsi, pour accélérer le temps de recherche du plus proche voisin, certains travaux utilisent les boules de mots et l'inégalité triangulaire pour rechercher des *pivots* [MOV94, BNC03, RJM03, MSMO03].

Dans [MS01, SMT02], les auteurs définissent la k-voisins fermeture (k-*neighbor closure* dans le texte) d'un mot w, par rapport à une distance d, comme étant $\bar{w}^{(d,k)} = \{v \in \Sigma^* : d(v,w) \leq k\}$. En d'autres termes, $\bar{w}^{(d,k)} = B_k(w)$ pour d étant la distance d'édition. La k-voisins fermeture est ensuite étendue aux langages en posant $\bar{L}^{(d,k)} = \{v \in \Sigma^* : \exists w \in \Sigma^*, d(v,w) \leq k\}$. Ils autorisent alors à un algorithme d'apprentissage d'inférer la fermeture d'un langage plutôt que le langage cible comme approximation admissible.

4.3 Quand les boules de mots ne sont-elles pas apprenables ?

Nous allons dans les chapitres suivants montrer que les boules sont apprenables à partir de données bruitées. Toutefois, il existe des paradigmes pour lesquels les boules ne sont pas apprenables, et ce, même lorsque les données ne sont pas bruitées. Ce sont les cas que nous allons traiter ici.

4.3.1 Identification à partir de requêtes

Contrairement aux automates, les requêtes d'appartenance avec les requêtes d'équivalence ne permettent pas l'apprentissage des boules[2] :

Théorème 16 *Soient $n, m \in \mathbb{N}$ et $\mathcal{B}_{\leq n,m} = \{B_r(o) \mid o \in \Sigma^*, |o| \leq n, r \leq m\}$. N'importe quel algorithme qui identifie exactement chaque boule hypothèse de $\mathcal{B}_{\leq n,m}$ en utilisant EQ et MQ fait, dans le pire cas, au moins $2^n - 1$ requêtes.*

Démonstration :
Comme dans [Ang90] et le Théorème 7, nous décrivons un adversaire qui force n'importe quelle méthode d'identification exacte utilisant MQ et EQ à faire $\Omega(|\Sigma|^n)$ requêtes dans le pire des cas.

L'adversaire maintient un ensemble X de toutes les boules possibles. Au début, $X = \mathcal{B}_{\leq m,n}$. Tant que X contient au moins deux boules, l'adversaire procède comme suit : à chaque requête d'équivalence $B_r(o)$, l'adversaire retourne le contre-exemple o. À chaque requête d'appartenance o, l'adversaire répond NON. En d'autres termes, après chaque requêtes, o ne peut appartenir à la boule cible. Toutes les boules de X contenant o sont donc enlevées de l'ensemble X. Beaucoup de boules peuvent ainsi disparaître de X, mais, à chaque étape, une seule boule de rayon 0 est enlevée : $B_0(o)$. Comme il existe $\Omega(|\Sigma|^n)$ boules de rayon 0 dans $\mathcal{B}_{\leq m,n}$, n'importe quel apprenant sera forcé de faire au minimum $\mathcal{B}_{\leq m,n}$ requêtes pour identifier une telle boule. □

Ce résultat peut paraître surprenant. En effet, les boules peuvent être représentées par des automates et nous avons vu au Chapitre 2 que $\mathcal{AFD}(\Sigma)$ est apprenable à partir de MQ et EQ (voir le Théorème 5 de ce même chapitre). Soit **Alg** l'algorithme qui identifie les AFD à partir de MQ et EQ. Ne pourrions-nous pas utiliser **Alg** pour identifier les boules ? En effet, il suffirait de construire un automate hypothèse A en donnant les réponses aux requêtes de l'oracle à **Alg**. Ensuite, pour chaque hypothèse de **Alg**, nous extrayons le rayon r et le centre o de A et nous effectuons une nouvelle requête d'équivalence $B_r(o)$. Voir Figure 4.2.

Plusieurs problèmes se posent si nous faisons cela. Tout d'abord, **Alg** construit des AFD ne reconnaissant pas forcément une boule. Que signifie alors extraire le centre et le rayon d'un tel automate ? Deuxièmement, est-il possible d'extraire le rayon et le centre d'un automate dont nous savons qu'il reconnaît une boule en un temps polynomial ?

[2] Par conséquent, les MQ seules ainsi que les EQ seules ne le permettent pas non plus.

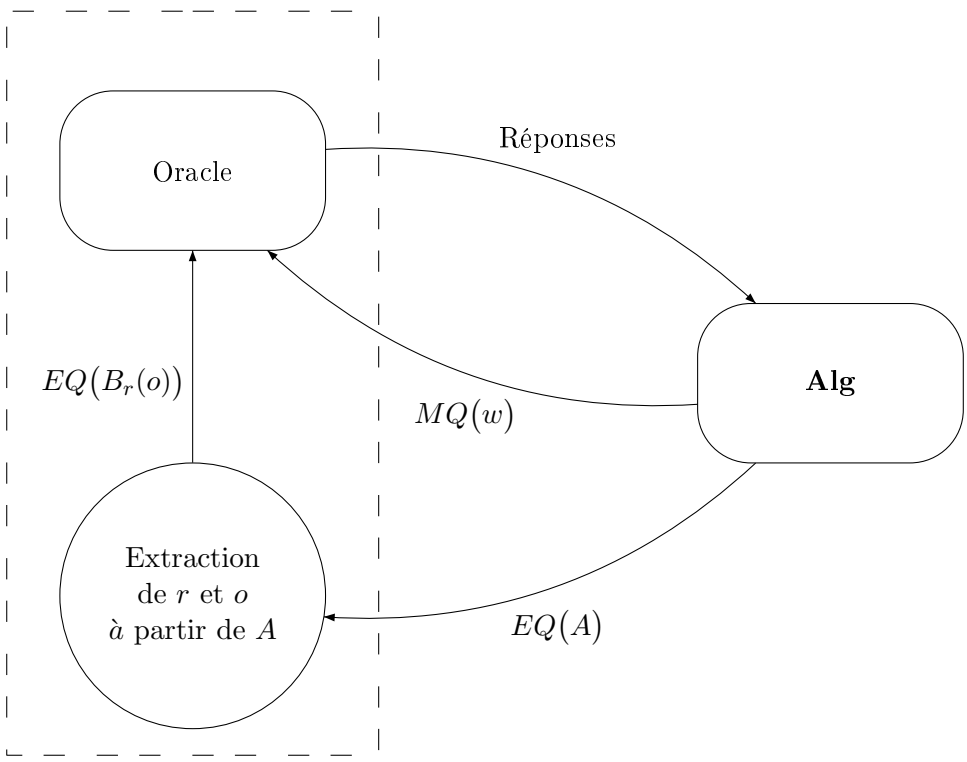

FIG. 4.2 – Diagramme montrant comment apprendre les boules à travers les automates.

Enfin, si comme nous le supposons la taille de l'automate minimal reconnaissant $B_r(o)$ est exponentiel en r, le problème reste inchangé : **Alg** aura bien appris un automate A reconnaissant $B_r(o)$ en temps polynomial en $|A|$, mais $|A|$ est exponentiel en r, et donc en la taille de la boule cible ($|o|$ pour les bonnes boules, et $|o| + \log r$ dans le cas général).

Finalement, nous pouvons aussi noter que la preuve du Théorème 16 ne tiendrait plus si nous pouvions faire des requêtes d'équivalence dites *impropres* : dans le cadre que nous avons présenté, l'apprenant à le droit de faire des EQ uniquement de langages appartenant à la classe des langages cibles[3]. Dans le cas présent, il ne peut faire de requêtes uniquement qu'avec des boules de mots. Si tel n'était pas le cas, alors l'apprenant pourrait faire la requête d'équivalence sur l'ensemble vide. L'oracle serait alors forcé de donner un contre-exemple, c'est-à-dire un mot de la boule cible.

4.3.2 Apprentissage PAC

Dans cette section, nous allons montrer que les bonnes boules ne sont pas polynomialement PAC apprenables, c'est-à-dire qu'il existe au moins une distribution $\mathcal{D}$ sur

[3]Par exemple, au Chapitre 2, un apprenant ne pouvait soumettre que des langages reconnus par des AFD.

Σ^* telle que n'importe quel algorithme d'apprentissage ne peut retourner une hypothèse ϵ-bonne avec une probabilité plus grande que $1-\delta$, en un temps polynomial. En d'autres termes, il existe une distribution pour laquelle aucun algorithme ne retournera en temps polynomial une hypothèse suffisamment proche de la cible avec une confiance élevée.

La preuve de cette non-apprenabilité suit les lignes classiques de ce genre de résultats : nous montrons d'abord que le problème de consistance associé est $\mathcal{NP}$-difficile, par réduction à un problème $\mathcal{NP}$-complet (*plus long sous-mot commun*). S'en suit que s'il existait un algorithme polynomial PAC-apprenant les boules, cet algorithme nous fournirait une preuve que ce problème $\mathcal{NP}$-complet est dans $\mathcal{RP}$.

Le problème que nous allons utiliser pour notre réduction est lui-même obtenu par réduction à un problème $\mathcal{NP}$-complet :

Lemme 6 *Les problèmes de décidabilité suivants sont $\mathcal{NP}$-complets :*

1. ***Problème du plus long sous-mot commun*** *(*PLSC*) : étant donnés n mots $x_1 \dots x_n$ et un entier k, existe-t-il un mot w de longueur k et sous-mot de chaque x_i ?*
2. ***Problème du plus long sous-mot communs à des mots d'une longueur donnée*** *(*PLSCMLD*) : étant donnés n mots $x_1 \dots x_n$ de longueurs $2k$, existe-t-il un mot w de longueur k sous-mot de chaque x_i ?*

Démonstration :

1. Voir [Mai77, GJ79].
2. Voir [dlHC00], problème LCS0.

□

Afin de montrer que $\boldsymbol{\mathcal{BB}}(\Sigma)$ n'est pas PAC-apprenable, nous allons montrer qu'il est difficile de trouver une boule consistante avec des données positives et négatives :

Lemme 7 *Le problème suivant est $\mathcal{NP}$-complet :*
Problème de la boule consistante *(*BC*) : étant donné deux ensembles X_+ et X_- de mots sur un alphabet Σ, existe-t-il une bonne boule contenant X_+ et aucun mot de X_- ?*

Démonstration :
Nous utilisons une réduction du problème PLSCMLD (Lemme 6). Dans X_+, nous mettons le mot vide λ et les mots de Σ^{2k}, c'est-à-dire tous les mots de longueurs $2k$. Nous construisons X_- en prenant chaque mot de Σ^{2k+1}

Une boule qui contient X_+ et aucun mot de X_- a alors nécessairement un centre de longueur k et un rayon de k (puisque nous nous concentrons sur les bonnes boules). Le centre est alors un sous-mot commun à tous les mots de longueurs $2k$.

Inversement, si une boule est construite avec un sous-mot de longueur k comme centre, cette boule est de rayon k, contient aussi λ et, à cause du rayon, ne contient pas d'éléments de X_-.

Finalement, le problème est dans $\mathcal{NP}$, puisqu'étant donné un centre u, il est facile de vérifier si $\max_{x \in X_+} d(u,x) < \min_{x \in X_-} d(u,x)$, c'est-à-dire de vérifier si la boule centrée sur u contient bien tous les mots de X_+ mais aucun de X_-. □

Nous allons maintenant montrer par l'absurde que $\boldsymbol{\mathcal{BB}}(\Sigma)$ ne peut être PAC-appris par réduction au problème précédent :

Théorème 17 $\boldsymbol{\mathcal{BB}}(\Sigma)$ *n'est pas polynomialement* PAC *apprenable.*

Démonstration :
Supposons que $\boldsymbol{\mathcal{BB}}(\Sigma)$ soit polynomialement PAC apprenable avec **Alg** et prenons l'instance $\langle X_+, X_- \rangle$ du problème BC. Posons $h = |X_+| + |X_-|$ et soit la distribution $Pr(x) = \frac{1}{h}$ si $x \in X_+ \cup X_-$, 0 sinon définit sur Σ^*.

Soient $\epsilon = \frac{1}{h+1}$, $\delta < \frac{1}{2}$ et $m = n = \max\{|w| : w \in X_+\}$. Soit maintenant $B_r(o)$ la boule retournée par une exécution de $\mathbf{Alg}(\epsilon, \delta, m, n)$. Testons si $X_+ \subseteq B_r(o)$ et si $X_- \cap B_r(o) = \emptyset$. Deux cas se présentent alors. Soit il n'existe pas de boule consistante, alors $B_r(o)$ est nécessairement inconsistante avec les données, donc le test ci-dessus est faux. Soit il existe une boule consistante, alors $B_r(o)$ est une hypothèse ϵ-bonne, avec $\epsilon < \frac{1}{h}$. Donc, avec probabilité au moins $1 - \delta > \frac{1}{2}$, il n'y a pas d'erreur du tout et le test sera vrai.

Clairement, cette procédure s'exécute en temps polynomial en $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|\Sigma|$, m et n. Ainsi, si les bonnes boules étaient PAC apprenables, il existerait un algorithme randomisé pour le problème BC, qui est $\mathcal{NP}$-complet par le Lemme 7. □

Enfin, de même que $\boldsymbol{\mathcal{AFD}}(\Sigma)$, $\boldsymbol{\mathcal{BB}}(\Sigma)$ ne peut être PAC-appris à partir d'exemples positifs seuls :

Théorème 18 $\boldsymbol{\mathcal{BB}}(\Sigma)$ *n'est pas polynomialement* PAC *apprenable à partir d'exemples positifs seulement.*

Démonstration :
Soient $L_1 = B_1(a)$ et $L_2 = B_1(b)$. Posons $w_1 = aa$, $w_2 = bb$ et $w_3 = ab$. Le résultat découle alors du Lemme 3. □

Les bonnes boules (et donc les boules) ne sont donc pas PAC apprenables polynomialement, que ce soit à partir de TEXTE ou d'INFORMATEUR. Comme pour les AFD, nous n'étudierons pas la PAC-apprenabilité de $\boldsymbol{\mathcal{BB}}(\Sigma)$ dans un contexte bruité au chapitre suivant.

4.3.3 Cas des situations bruitées

Dans le Chapitre 3, nous avons vu plusieurs types de bruit et les boules ne sont pas apprenables à partir de n'importe quelle situation bruitée.

Par exemple, comme les boules ne sont pas apprenables à partir de MQ et de EQ, elles ne peuvent pas l'être à partir de MMQ ni de LMQ, c'est-à-dire lorsque des erreurs ou des omissions peuvent se glisser dans les réponses de l'oracle.

De même, les boules ne sont pas apprenables à partir de *NoisyTxt*, car Stephan a montré dans [Ste97] que si deux langages L, L' sont tels que $L \subset L'$, alors $\{L, L'\}$ ne peut être appris. Or pour n'importe quel mot $o \in \Sigma^*$, nous avons par exemple $B_r(o) \subset B_{r+1}(o)$.

Conclusion

Les langages dont la définition est basée sur la distance d'édition semblent de prime abord n'être que des exemples pris pour montrer que des classes de langages peuvent être résistantes aux bruit. Cependant, bien qu'ayant une définition simple, les boules de mots n'en sont pas pour autant inintéressantes : les applications dans lesquelles elles sont utilisées sont nombreuses et variées, et leurs propriétés montrent une réelle complexité combinatoire.

Il reste maintenant à prouver que, contrairement aux AFD, la classe des boules de mots peut être apprise en situations bruitées. Nous nous intéresserons donc dans un premier temps à leur identification à la limite à partir de données bruitées.

Chapitre 5

Identification à la limite des boules à partir de données bruitées

L'apprentissage des AFD n'étant pas résistant au bruit, nous concentrons notre attention sur une nouvelle classe de langage, celle des boules de mots. Après avoir vu quelques propriétés des boules de mots, nous allons ici montrer que les boules de mots sont apprenables à partir de données bruitées systématiquement, c'est-à-dire lorsque tout le *voisinage* de chaque mot de la boule apparaît dans la présentation.

Pour cela, nous supposerons dans un premier temps que les boules de mots sont identifiables à la limite, puis à travers deux techniques, la réduction et le débruitage à la limite, nous les identifierons en situations bruitées. Nous prouverons ensuite qu'effectivement les boules de mots sont identifiables à la limite et que les techniques proposées en première partie de ce chapitre sont donc valables.

5.1 Techniques d'apprentissage

Afin d'identifier à la limite les boules de mots lorsque les données sont bruitées systématiquement, nous supposerons au préalable que $\mathcal{BOULE}(\Sigma)$ est identifiable à la limite (nous verrons en Section 5.2 que ceci est bien le cas). Dans un premier temps, Section 5.1.1, nous utiliserons une technique de réduction : nous apprenons une boule correspondant aux données de la présentation bruitée, puis nous retrouvons la boule cible grâce à la boule apprise. Dans un second temps, Section 5.1.2, nous utiliserons une technique de débruitage à la limite : nous tenterons de trouver quels sont les exemples bruités et non bruités de la présentation.

Ainsi, nous pouvons voir ces deux solutions de la façon suivante : à partir d'une présentation bruitée ($\mathbf{Pres}(\mathcal{L})$), soit nous apprenons le langage correspondant à cette présentation ($\mathcal{L}$'), soit nous débruitons la présentation ($\overline{\mathbf{Pres}}(\mathcal{L})$), puis nous identifions

le langage cible ($\boldsymbol{\mathcal{L}}$) :

$$\begin{array}{ccc} \mathbf{Pres}(\boldsymbol{\mathcal{L}}) & \longrightarrow & \mathcal{L}\text{'} \\ \downarrow & & \downarrow \\ \overline{\mathbf{Pres}}(\boldsymbol{\mathcal{L}}) & \longrightarrow & \boldsymbol{\mathcal{L}} \end{array}$$

5.1.1 Technique de réduction

La première façon d'identifier à la limite en situation bruitée que nous présenterons est donc la réduction. Cette technique est mise en œuvre dans de nombreux domaines de l'informatique. Les réductions permettent d'obtenir un grand nombre de résultats négatifs en montrant que tel problème est au moins aussi difficile que tel autre, connu comme étant trop dur : nous pouvons par exemple montrer que le problème du sac à dos[1] est $\mathcal{NP}$-complet en effectuant une réduction à partir de celui de la couverture exacte[2], que nous savons être un problème $\mathcal{NP}$-complet. Des résultats positifs existent cependant. Dans [PW88] par exemple, Pitt et Warmuth montrent que l'apprentissage des AFD implique l'apprentissage des formules booléennes. Dans [dlH05b], de la Higuera montre qu'avec la réduction, des preuves simples de résultats d'apprentissage déjà connus peuvent être exhibées.

Ce que nous étudions ici est l'apprentissage de *classes de langages*, selon certaines *représentations* et à partir de *présentations*. Nous allons voir comment relier ces trois concepts pour utiliser la réduction.

Soient $\boldsymbol{\mathcal{L}}$ et $\mathcal{L}$' deux classes de langages représentées respectivement par $R(\boldsymbol{\mathcal{L}})$ et $R(\mathcal{L}\text{'})$. Notons $\mathbb{L}_{\boldsymbol{\mathcal{L}}}$ l'application surjective $R(\boldsymbol{\mathcal{L}}) \to \boldsymbol{\mathcal{L}}$ et $\mathbb{L}_{\mathcal{L}'}$ l'application surjective $R(\mathcal{L}\text{'}) \to \mathcal{L}$'. Étant alors donnée une application surjective $\phi : \boldsymbol{\mathcal{L}} \to \mathcal{L}$', notons ψ une application $R(\boldsymbol{\mathcal{L}}) \to R(\mathcal{L}\text{'})$ pour laquelle le diagramme suivant commute :

$$\begin{array}{ccc} R(\boldsymbol{\mathcal{L}}) & \xrightarrow{\psi} & R(\mathcal{L}\text{'}) \\ {\scriptstyle \mathbb{L}_{\boldsymbol{\mathcal{L}}}}\downarrow & & \downarrow{\scriptstyle \mathbb{L}_{\mathcal{L}'}} \\ \boldsymbol{\mathcal{L}} & \xrightarrow{\phi} & \mathcal{L}\text{'} \end{array}$$

ψ est alors l'unique application telle que l'égalité suivante soit vraie :

$$\phi \circ \mathbb{L}_{\boldsymbol{\mathcal{L}}} = \mathbb{L}_{\mathcal{L}'} \circ \psi$$

Plaçons nous maintenant du point de vue des présentations. Étant donnée une application surjective $\phi : \boldsymbol{\mathcal{L}} \to \mathcal{L}$', notons ξ une application $\mathbf{Pres}(\boldsymbol{\mathcal{L}}) \to \mathbf{Pres}(\mathcal{L}\text{'})$ pour

[1] Le problème du sac à dos consiste à maximiser la valeur totale des objets mis dans le sac à dos sans en dépasser le poids total autorisé.

[2] Étant donnés un ensemble X et un ensemble S de sous-ensemble de X, une couverture exacte est une sous-collection de S telle que ses éléments sont disjoints deux à deux et leur union forme X. Le problème de la couverture exacte consiste alors à trouver une couverture exacte à partir d'un ensemble X et d'une collection S.

laquelle le diagramme suivant commute :

$$\begin{array}{ccc} \boldsymbol{\mathcal{L}} & \xrightarrow{\phi} & \mathcal{L}\text{'} \\ {\scriptstyle yield_{\boldsymbol{\mathcal{L}}}}\uparrow & & \uparrow{\scriptstyle yield_{\mathcal{L}'}} \\ \mathbf{Pres}(\boldsymbol{\mathcal{L}}) & \xrightarrow{\xi} & \mathbf{Pres}(\mathcal{L}\text{'}) \end{array}$$

Il en résulte alors :

$$\phi \circ yield_{\boldsymbol{\mathcal{L}}} = yield_{\mathcal{L}'} \circ \xi$$

La fonction ξ transforme donc les présentations du premier langage en présentation du second langage. Cela ne veut pas dire que c'est une transformation point par point. En effet, la fonction ξ peut très bien transformer un élément de la première présentation en plusieurs éléments de la seconde.

Comme une présentation peut ne pas être une fonction calculable, l'application calculable associée à ξ se fait alors par morceaux :

Définition 37 *Soient $\boldsymbol{\mathcal{L}}$ une classe de langages représentés dans $R(\boldsymbol{\mathcal{L}})$ avec des présentations dans $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}}) : \mathbb{N} \to X$ et $\mathcal{L}$' une classe de langages représentés dans $R(\mathcal{L}')$ avec des présentations dans $\boldsymbol{Pres}(\mathcal{L}') : \mathbb{N} \to Y$.*

Nous dirons alors qu'une réduction de présentations $\xi : \boldsymbol{Pres}(\boldsymbol{\mathcal{L}}) \to \boldsymbol{Pres}(\mathcal{L}')$ telle que $\xi(\boldsymbol{f}) = \boldsymbol{g}$ est calculable *si et seulement si il existe une fonction calculable $\overline{\xi} : X \to 2^Y$ avec $\bigcup_{i \in \mathbb{N}} \overline{\xi}(\boldsymbol{f}(i)) = \boldsymbol{g}(\mathbb{N})$.*

La réduction ξ est polynomiale *si $\overline{\xi}$ est une fonction polynomiale.*

$\overline{\xi}$ est la description en chaque point de la fonction ξ. Nous supposerons en outre que $\forall i \in \mathbb{N}$, $\overline{\xi}(\boldsymbol{f}(i))$ est un ensemble fini.

Ainsi, en combinant les deux diagrammes précédents nous obtenons :

$$\begin{array}{ccc} R(\boldsymbol{\mathcal{L}}) & \xrightarrow{\psi} & R(\mathcal{L}\text{'}) \\ {\scriptstyle \mathbb{L}_{\boldsymbol{\mathcal{L}}}}\downarrow & & \downarrow{\scriptstyle \mathbb{L}_{\mathcal{L}'}} \\ \boldsymbol{\mathcal{L}} & \xrightarrow{\phi} & \mathcal{L}\text{'} \\ {\scriptstyle yield_{\boldsymbol{\mathcal{L}}}}\uparrow & & \uparrow{\scriptstyle yield_{\mathcal{L}'}} \\ \mathbf{Pres}(\boldsymbol{\mathcal{L}}) & \xrightarrow{\xi,\overline{\xi}} & \mathbf{Pres}(\mathcal{L}\text{'}) \end{array}$$

Enfin, grâce à ce diagramme, nous pouvons formuler le théorème d'apprentissage utilisant les réductions :

Théorème 19 *Si*

1. *$\mathcal{L}$' est apprenable à partir de $\boldsymbol{Pres}(\mathcal{L}')$,*
2. *il existe une fonction calculable $\chi : R(\mathcal{L}') \to R(\boldsymbol{\mathcal{L}})$ et une fonction calculable $\psi : R(\boldsymbol{\mathcal{L}}) \to R(\mathcal{L}')$ telle que $\psi \circ \chi = Id$ et*
3. *ξ est une réduction calculable,*

alors $\boldsymbol{\mathcal{L}}$ *est identifiable par* $R(\boldsymbol{\mathcal{L}})$ *à partir de* $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$.

Le diagramme découlant du théorème est donc le suivant :

$$\begin{array}{ccc} R(\boldsymbol{\mathcal{L}}) & \xleftarrow{\chi} & R(\mathcal{L}') \\ {\scriptstyle \mathbb{L}_{\boldsymbol{\mathcal{L}}}}\downarrow & & \downarrow{\scriptstyle \mathbb{L}_{\mathcal{L}'}} \\ \boldsymbol{\mathcal{L}} & \xrightarrow{\phi} & \mathcal{L}' \\ {\scriptstyle yield_{\boldsymbol{\mathcal{L}}}}\uparrow & & \uparrow{\scriptstyle yield_{\mathcal{L}'}} \\ \mathbf{Pres}(\boldsymbol{\mathcal{L}}) & \xrightarrow{\xi,\overline{\xi}} & \mathbf{Pres}(\mathcal{L}') \end{array}$$

La preuve du Théorème 19 est une preuve constructive :
Démonstration :
Soit **Alg2** un algorithme d'apprentissage qui identifie $\mathcal{L}'$. Considérons l'Algorithme 4 ci-dessous, qui prend les n premiers éléments $(\boldsymbol{f}_n)$ d'une présentation $\boldsymbol{f}$ et retourne une grammaire.

Algorithme 4 : Algorithme de réduction

Données : Les n premiers éléments d'une présentation $\boldsymbol{f}_n$
Résultat : Une grammaire
1 $\boldsymbol{g}_m \longleftarrow \overline{\xi}(\boldsymbol{f}_n)$;
2 $G_{\mathcal{L}'} \longleftarrow$ **Alg2**$(\boldsymbol{g}_m)$ `// ` $G_{\mathcal{L}'}$ `est donc une grammaire de` $R(\mathcal{L}')$
3 $G_{\boldsymbol{\mathcal{L}}} \longleftarrow \chi(G_{\mathcal{L}'})$ `// ` $G_{\boldsymbol{\mathcal{L}}}$ `est donc une grammaire de` $R(\boldsymbol{\mathcal{L}})$
4 **retourner** $G_{\boldsymbol{\mathcal{L}}}$

Comme ξ est calculable, alors $\boldsymbol{g}_m$ peut effectivement être construit. De plus, si ξ et χ sont des fonctions polynomiales alors l'Algorithme 4 est polynomial si **Alg2** l'est. □

Le Théorème 19 permet ainsi de retrouver des résultats connus comme par exemple l'apprentissage des grammaires linéaires équilibrées [Tak88] par réduction à partir des automates finis déterministes.

$$\begin{array}{ccc} \text{Grammaires linéaires équilibrées} & \xleftarrow{\chi} & \textsc{Dfa} \\ {\scriptstyle \mathbb{L}_{\boldsymbol{\mathcal{L}}}}\downarrow & & \downarrow{\scriptstyle \mathbb{L}_{\mathcal{L}'}} \\ \text{Langages linéaires équilibrés} & \xrightarrow{\phi} & \boldsymbol{\mathcal{AFD}}(\Sigma \cup \{\#\}) \\ {\scriptstyle yield_{\boldsymbol{\mathcal{L}}}}\uparrow & & \uparrow{\scriptstyle yield_{\mathcal{L}'}} \\ \textsc{Informateur} & \xrightarrow{\xi,\overline{\xi}} & \textsc{Informateur} \end{array}$$

Le diagramme est alors correct si la fonction $\overline{\xi}$ prend un mot $w_1w_2\cdots w_{2n}$ et retourne $(a_1, a_{2n})\cdots(a_{n-1}, a_n)$ $\big($ou un mot $w_1w_2\cdots w_{2n+1}$ et retourne $(a_1, a_{2n+1})\cdots(a_n, \#)\big)$, χ

transformant un AFD sur l'aphabet $\Sigma \cup \{\#\}$ en une grammaire linéaire équilibrée sur Σ.

Concernant les boules, il permet de montrer qu'elles sont identifiables à partir de données bruitées systématiquement :

Théorème 20 $\mathcal{BOULE}(\Sigma)$ *est identifiable à la limite à partir de texte k-bruité.*

Démonstration :
D'après le Théorème 13 page 63, le bruité d'une boule est une boule. De plus, par hypothèse, $\mathcal{BOULE}(\Sigma)$ est identifiable à la limite à partir de texte. Ainsi, en prenant comme fonction χ=*si le rayon de la boule est au moins k, déduire k du rayon, sinon identité*, nous pouvons construire le diagramme suivant :

$$\begin{array}{ccc}
\mathcal{B}_\Sigma & \xleftarrow{\chi} & \mathcal{B}_\Sigma \\
{\scriptstyle \mathbb{L}_{\mathcal{L}}}\downarrow & & \downarrow{\scriptstyle \mathbb{L}_{\mathcal{L}'}} \\
\mathcal{B}_\Sigma & \xrightarrow{Id} & \mathcal{B}_\Sigma \\
{\scriptstyle yield_{\mathcal{L}}}\uparrow & & \uparrow{\scriptstyle yield_{\mathcal{L}'}} \\
\text{Texte } k\text{-bruité} & \xrightarrow{Id,\overline{Id}} & \text{Texte}
\end{array}$$

Nous pouvons alors conclure à l'aide du Théorème 19. □

La technique de réduction, combinée avec le fait que le bruité d'une boule est une boule, permet donc de mettre en place un mécanisme simple d'apprentissage des boules à partir de texte bruité systématiquement. Le seul inconvénient à cette méthode est le fait de devoir connaître le niveau de bruit k. Cependant, cet inconvénient n'est pas inhérent à la méthode mais au type de bruit utilisé. En effet, comment distinguer $N_1(B_2(o))$ de $N_2(B_1(o))$ uniquement par le niveau de bruit ? Connaître k (ou le rayon de la boule cible) est donc nécessaire lors de l'apprentissage à partir de bruit systématique.

Nous allons nous intéresser maintenant à une seconde technique permettant d'apprendre à partir de données bruitées systématiquement, le débruitage à la limite.

5.1.2 Technique de débruitage à la limite

Nous l'avons vu au Chapitre 2, une façon d'apprendre à partir de données bruitées est de débruiter les données, puis d'apprendre le langage à partir de ces données non bruitées. Les algorithmes utilisant cette technique sont les algorithmes de type *wrapper*. Ils disposent généralement de l'ensemble des données d'apprentissage pour pouvoir détecter celles qui semblent douteuses. Une fois les données nettoyées, ces dernières permettent alors généralement un meilleur apprentissage. Cette technique est par exemple utilisée dans [SHY+07] où les auteurs débruitent des données de marchés boursiers avant de les soumettre à un SVM.

Dans le cadre de l'apprentissage à la limite à partir de données bruitées systématiquement, nous ne connaissons pas l'ensemble de la présentation. Nous allons alors pour

chaque nouvel élément x de la présentation, essayer de voir si x appartient au langage ou non, et s'il peut permettre de montrer l'appartenance de précédents éléments de la présentation au langage. Une fois la présentation débruitée, les éléments restants sont alors uniquement des mots non bruités à partir desquels nous pouvons alors apprendre à la limite le langage cible. Nous allons nous intéresser ici uniquement au débruitage des données, l'apprentissage se faisant ensuite comme au chapitre précédent, à partir de données non bruitées.

Afin de pouvoir distinguer les éléments bruités des éléments non bruités, nous associons à chaque *présentation bruitée* $\boldsymbol{f} : \mathbb{N} \to X$, une fonction **estDuBruit** : $X \to \{0,1\}$ indiquant si un élément particulier de la présentation est du bruit ou non. Si **estDuBruit**$(x) = 0$ alors x n'est pas un élément bruité.

Exemple 35 *Apprendre un langage L à partir d'une présentation k-bruitée signifie apprendre L à partir de mots de $N_k(L)$, c'est-à-dire d'une présentation $\boldsymbol{f}$ telle que $\boldsymbol{f}(\mathbb{N}) = N_k(L)$. Pour cette présentation, la fonction **estDuBruit** vaut alors 0 sur les éléments de L, 1 sur ceux de $N_k(L) \setminus L$ et n'est pas définie sur les mots de $\Sigma^* \setminus N_k(L)$.*

Cela signifie que si pour tous les éléments x d'une présentation $\boldsymbol{f}$ de $N_k(L)$ nous arrivons à trouver la valeur de **estDuBruit**(x), alors nous pouvons construire une présentation non bruitée de L. De ce fait, nous pouvons définir ce qu'est le débruitage à la limite. C'est le fait de pouvoir décider pour chaque élément, à partir d'un certain rang de la présentation, si l'élément est du bruit ou non :

Définition 38 (Débruitable à la limite) *Soit $\boldsymbol{f}$ une présentation k-bruitée. S'il existe un algorithme $\theta : X \times \bigcup_{i\in\mathbb{N}}\{\boldsymbol{f}_i\} \to \{0,1\}$ tel que : $\forall x \in X, \exists n_x$ tel que $\forall m \geq n_x$ $\theta(x, \boldsymbol{f}_m) = \theta(x, \boldsymbol{f}_{n_x}) = \boldsymbol{estDuBruit}(x)$(= si $x \in N_k(L) \setminus L$ alors 1 sinon 0), alors $\boldsymbol{f}$ est débruitable à la limite.*

De même, soit $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$ une classe de présentations k-bruitées. Si toutes les présentations de $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$ sont débruitables à la limite par un même algorithme θ alors $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$ est débruitable à la limite.

En d'autres termes, s'il existe un algorithme tel que, pour chaque élément de n'importe quelle présentation, il existe un rang à partir duquel l'algorithme aura toujours la même valeur, celle de **estDuBruit**, alors la classe de présentations est débruitable à la limite.

Il est à noter que l'identification du bruit n'est pas monotone : nous pouvons avoir identifié certaines données comme étant bruitées et ne pas pouvoir (encore) le faire pour d'autres.

Dans la suite, nous ne considérons que l'apprentissage à partir de *texte k-bruité*. Dans ce cas, $\theta_k(u, \boldsymbol{f}_m) = 1$ indique alors le fait qu'au rang m de la présentation $\boldsymbol{f}$, l'algorithme estime que u est un mot bruité et donc ne fait pas partie de L.

Débruiter les données revient donc à savoir si elles appartiennent au langage cible ou non. Pour cela, nous allons avoir besoin de connaître les relations de proximité des données les unes par rapport aux autres, et notamment par rapport à celles qui appartiennent effectivement au langage.

En effet, dans une présentation bruitée, chaque mot u du langage a été bruité systématiquement, et donc tous les voisins de u (soit tous les mots jusqu'à une certaine distance de u) apparaissent dans la présentation. En conséquence, si à un certain moment, tous les mots de $N_k(u)$ sont apparus dans la présentation, alors nous pouvons être sûr que u appartient au langage.

Nous pouvons ainsi définir une fonction dite d'*intérieur* qui, pour un langage L, va garder tous les mots de L tels que leurs voisins sont inclus dans le langage :

Définition 39 (Intérieur) *Notons* $I(L)$ l'intérieur *de* L *:*

$$I(L) = \{w \in \Sigma^* : N_1(\{w\}) \subseteq L\}$$

Nous appelons le k-intérieur de L *la fonction définie par*

$$I_0(L) = L, \forall k \in \mathbb{N}^+ \; I_k(L) = I[I_{k-1}(L)]$$

Une définition équivalente du k-intérieur d'un langage est la suivante : $I_k(L) = \{w \in \Sigma^* : N_k(\{w\}) \subseteq L\}$. Ainsi, supposons que $\boldsymbol{f}_n$ soit débruitable à la limite, les éléments de $I_k(\boldsymbol{f}_n)$ sont ceux dont tous les voisins appartiennent à $\boldsymbol{f}_n$, donc les éléments non bruités de la présentation, ceux pour lesquels **estDuBruit** vaut 0.

Cette notion de k-intérieur est naturellement liée à celle de k-extérieur. Soit $E(L)$ *l'extérieur* de L : $E(L) = \{w \in \Sigma^* : N_1(\{w\}) \cap L \neq \emptyset\}$. Le k-extérieur d'un langage L est alors défini par $E_0(L) = L, \forall k \in \mathbb{N}^+ \; E_k(L) = E[E_{k-1}(L)]$. Nous pouvons alors montrer que $E_k(L) = N_k(L)$, donc que $E_k(L) = \{w \in \Sigma^* : N_1(\{w\}) \cap L \neq \emptyset\}$. L'intérieur et l'extérieur sont en fait des fonctions *c-duales*, c'est-à-dire telles que $\forall L, I_k(\bar{L}) = \overline{E_k(L)}$. Ces couples de fonctions c-duales font partie d'un cadre plus général qu'il n'est pas nécessaire de connaître pour notre étude. Le lecteur désireux d'en savoir plus trouvera cependant plus d'informations en annexe.

Les fonctions I_k et E_k permettent ainsi de supprimer et d'ajouter du bruit. La Figure 5.1 illustre le fonctionnement du k-extérieur et du k-intérieur.

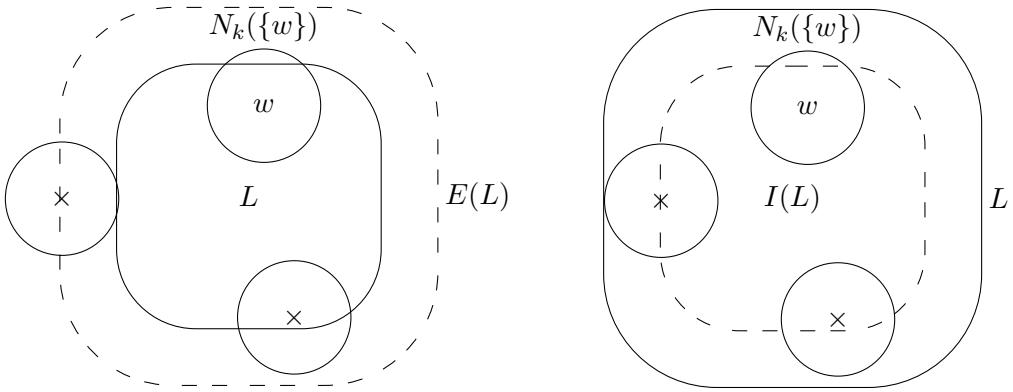

FIG. 5.1 – Illustration des fonctions d'extérieur et d'intérieur.

Supposons maintenant que $I_k(E_k(L)) = L$. Cela signifie que lorsque nous débruitons avec la fonction I_k le k-bruité de L, les éléments restant sont tous ceux et uniquement ceux de L. Ainsi, pour une présentation $\boldsymbol{f}$ de L, nous aurons $I_k(\boldsymbol{f}(\mathbb{N})) = L$, ce qui nous permettra bien d'effectuer un débruitage à la limite.

La condition de ce débruitage est que $I_k(E_k(L)) = L$. Si un langage vérifie cette propriété, nous dirons que L est un langage *fermé*[3] :

Définition 40 (Langage fermé) *On dit qu'un langage L est* k-fermé *si et seulement si $I_k(E_k(L)) = L$ et qu'une classe de langage est* k-fermée *si tous ses éléments sont k-fermés.*

Inversement, si $L = E_k(I_k(L))$, on dit que le langage est k-ouvert. Nous pouvons montrer que la définition d'un fermé peut aussi s'écrire sous une autre forme :

Lemme 8

$$L \text{ fermé} \Rightarrow \left(\forall x \in \Sigma^*, N_k(x) \subseteq E_k(L) \Rightarrow x \in L\right)$$

Démonstration :
Si L est un langage k-fermé, alors $L = I_k(E_k(L))$. Nous avons, quelque soit L, $L \subseteq I_k(E_k(L))$ (voir en annexe). Donc, L fermé $\Rightarrow \left(I_k(E_k(L)) \subseteq L\right)$. Or $I_k(E_k(L)) \subseteq L \Leftrightarrow \forall x \in \Sigma^*, x \in I_k(E_k(L)) \Rightarrow x \in L$ et $x \in I_k(E_k(L)) \Leftrightarrow N_k(x) \subseteq E_k(L)$ par définition. □

En d'autres termes, si pour n'importe quel mot, lorsque le bruité du mot est inclus dans l'extérieur d'un langage, celui-ci appartient au langage, alors le langage est fermé.

Exemple 36 *Soit $\overline{\boldsymbol{BOULE}}(\Sigma)$ l'ensemble défini par $\overline{\mathcal{B}_\Sigma} = \left\{\overline{B_r(o)} : r \in \mathbb{N}, o \in \Sigma^*\right\}$ où $\overline{B_r(o)} = \Sigma^* \setminus B_r(o)$. $\overline{\boldsymbol{BOULE}}(\Sigma)$ est donc la classe des complémentaires des boules.*

En annexe, nous montrons que $B_r(o) = E_k(I_k(B_r(o)))$, c'est-à-dire que les boules sont k-ouvertes. Or $\overline{B_r(o)} = \overline{E_k(I_k(B_r(o)))} = I_k(\overline{I_k(B_r(o))}) = I_k(E_k(\overline{B_r(o)}))$ par c-dualité. Comme $\overline{B_r(o)} = I_k(E_k(\overline{B_r(o)}))$, nous avons bien $\overline{B_r(o)}$ qui est un ensemble fermé.

Comme nous l'avons dit, les fonctions I_k et E_k nous permettent effectivement de mettre en œuvre un débruitage des données :

Théorème 21 *Soit N_k la fonction de bruit. Si $\boldsymbol{\mathcal{L}}$ est k-fermée alors $\boldsymbol{Pres}(\boldsymbol{\mathcal{L}})$ est k-débruitable à la limite.*

[3] La relation de voisinage que nous avons décrite fait en fait appel à la topologie. Cependant, les espaces topologiques sont définis à partir de trop nombreux axiomes qui empêchent la mise en place d'un débruitage à la limite ainsi que la mise en avant de langages fermés non triviaux. Les espaces mis en place sont alors des *espaces prétopologiques*. Un espace prétopologique $\mathbb{E}_k = (\Sigma^*, E_k \circ I_k, I_k \circ E_k))$ est l'espace Σ^* auquel sont associés les fonctions $E_k \circ I_k$ et $I_k \circ E_k$. Nous rappelons en annexe la définition et des propriétés des espaces prétopologiques. Un langage que nous qualifierons de k-fermé est en fait un langage fermé pour l'espace prétopologique $\mathbb{E}_k$.

Démonstration :
Considérons l'algorithme θ_k suivant : soient $\boldsymbol{f}$ une présentation k-bruitée d'un langage L et $u \in N_k(L)$; on pose $\theta_k(u, \boldsymbol{f}_p) = 0$ si $u \in I_k(\boldsymbol{f}_p)$ et 1 si $B_k(u) \not\subseteq \boldsymbol{f}_p$.

Soient $\boldsymbol{f}$ une présentation k-bruitée d'un langage L et $u \in N_k(L)$. Si $\mathbf{estDuBruit}(u) = 0$ alors $u \in L$ donc $B_k(u) \subseteq N_k(L)$ et comme $\boldsymbol{f}(\mathbb{N}) = N_k(L)$, il existe un rang n_u tel que $B_k(u) \subseteq \boldsymbol{f}_{n_u}$ et donc $u \in I_k(B_k(u)) \subseteq I_k(\boldsymbol{f}_{n_u})$. Par conséquent $\theta_k(u, \boldsymbol{f}_{n_u}) = 0 = \mathbf{estDuBruit}(u)$. Inversement, si $\mathbf{estDuBruit}(u) = 1$, alors $u \notin L$ et comme L est k-fermé alors $B_k(u) \not\subseteq N_k(L)$ par le Lemme 8, et donc $\forall p \in \mathbb{N}, B_k(u) \not\subseteq \boldsymbol{f}_p$. Par conséquent, $\forall p \in \mathbb{N}, \theta(u, \boldsymbol{f}_p) = 1 = \mathbf{estDuBruit}(u)$. □

Ce théorème nous permet alors de déduire le corollaire :

Corollaire 1 $\mathcal{BOULE}(\Sigma)$ *est débruitable à la limite à partir d'*Informateur.

Démonstration :
Les boules étant des langages k-ouverts, leurs complémentaires sont donc k-fermés. Ces derniers sont donc débruitables à la limite à partir de Texte k-bruité. Il suffit alors de prendre uniquement les exemple négatifs de la présentation (correspondant aux exemples positifs du complémentaire), de les débruiter, et de prendre le complémentaire des exemples négatifs débruités comme exemples positifs. □

Bien que ce résultat soit positif, force est de constater que le débruitage peut être très long. Afin d'accélérer le processus, il peut être utile d'ajouter au préalable plus de bruit. Ajouter du bruit peut paraître étrange, néanmoins il permet d'obtenir le résultat suivant :

Théorème 22 *Soit N_k la fonction de bruit. Si $\boldsymbol{\mathcal{L}}$ est j-fermée et si $j \geq k$ alors $\boldsymbol{\mathcal{L}}$ est k-débruitable à la limite.*

Démonstration :
Il suffit de considérer l'algorithme $\theta_k(x, \boldsymbol{f}_p) = 0$ si $x \in I_k(E_{j-k}(\boldsymbol{f}_p))$ et 1 sinon puis de reprendre la preuve du Théorème 21. Plus intuitivement, soit $\boldsymbol{f}$ une présentation k-bruitée de L. Pour tout p nous définissons $g_p = E_{j-k}(\boldsymbol{f}_p)$. Comme $\boldsymbol{f}$ est une présentation de $N_k(L)$, g est une présentation de $N_j(L)$. De plus L est un ensemble j-fermé donc d'après le Théorème 21, g est j-débruitable à la limite et donc $\boldsymbol{f}$ est k-débruitable à la limite. □

Ajouter du bruit n'est pas une technique nouvelle. Par exemple, dans [GC97] Granvalet et Canu injectent du bruit aux entrées d'un réseaux de neurones pour améliorer la robustesse du réseau aux imprécisions des entrées, ainsi que pour éviter le surapprentissage. Toutefois, il n'existe pas à notre connaissance de travaux en inférence grammaticale utilisant cette technique.

En outre, ajouter du bruit ne permet clairement pas d'apprendre à la limite des classes qui ne sont pas apprenables à partir de présentations non bruitées. En revanche, il peut permettre comme nous l'avons dit, de débruiter (et donc d'apprendre) plus vite :

Exemple 37 *Soit la classe des boules centrées sur λ, considérons la boule $B_2(\lambda)$. Soit $\boldsymbol{f}$ une présentation 1-bruitée de cette boule commençant par $\boldsymbol{f}_4 = \{b, abb, aaa, baa\}$. Si nous voulons débruiter directement, nous obtenons $I_1(\boldsymbol{f}_4) = \emptyset$, et il faudra encore d'autres exemples pour arriver à dégager des éléments non bruités. En revanche, si nous ajoutons du bruit de niveau 1 et que nous débruitons, nous obtenons $I_2(E_1(\boldsymbol{f}_4)) = \{\lambda, a, b, aa\}$ ce qui suffit à retrouver $B_2(\lambda)$, la plus petite boule contenant $I_2(E_1(\boldsymbol{f}_4))$. De plus, si le prochain élément qui apparaît est aab, alors nous obtenons encore $I_1(\boldsymbol{f}_5) = \emptyset$ lorsque nous n'ajoutons aucun bruit et $I_2(E_1(\boldsymbol{f}_5)) = B_2(\lambda)$ en ajoutant un peu de bruit.*

Enfin, la plupart des langages ne sont naturellement pas totalement débruitables à la limite. Néanmoins, en combinant ajout et suppression de bruit, nous pouvons nous ramener à une classe de langages $\mathcal{L}$' à partir de laquelle il est possible de *déduire* la classe $\boldsymbol{\mathcal{L}}$.

Exemple 38 *Soit la classe des boules $\boldsymbol{\mathcal{BOULE}}(\Sigma)$. Rappelons que cette classe n'est pas fermée. Soit $L = B_r(o)$. Nous avons alors $I_{j+k}(E_j(N_k(L))) = I_{j+k}(E_{j+k}(L))$ qui contient une approximation de L, c'est-à-dire, L plus éventuellement quelques mots (par exemple, $bbbaaa \in I_1(E_1(B_4(aabb)))$ mais $bbbaaa \notin B_4(aabb)$). Or dans $I_{j+k}(E_{j+k}(L))$, il existe un couple $(a^n v, b^n v)$ où $a^n v$ et $b^n v$ sont respectivement le plus petit et le plus grand mot des mots les plus longs de L. Ces mots nous permettent de déduire $r = n$ et $o = v$, donc d'identifier $L = B_r(o)$. Par conséquent, nous avons un algorithme permettant d'identifier indirectement $\boldsymbol{\mathcal{BOULE}}(\Sigma)$ après un débruitage approximatif des données.*

Les boules étant ainsi débruitables à la limite, il ne reste plus qu'à les identifier à la limite à partir des données débruitées. Elles sont donc identifiables à la limite à partir de données bruitées si elles le sont à partir de données non bruitées.

5.2 Identification à la limite des boules

Nous venons de voir deux techniques d'apprentissage de $\boldsymbol{\mathcal{BOULE}}(\Sigma)$ en situations bruitées permettant chacune de se ramener à une identification à la limite à partir de données non bruitées. Dans cette section, nous montrons que les hypothèses faites sont correctes. Nous étudierons dans un premier temps l'apprentissage en temps Mc polynomial, puis l'apprentissage en temps Ipe polynomial et enfin celui en temps Cs polynomial.

Au préalable, nous rappelons que s'il n'y a pas de contraintes de polynomialité, l'identification à la limite de boules est aisée. En effet, il « suffit » d'attendre que la boule entière soit présente dans les données pour ensuite pouvoir l'identifier grâce au couple $(a^r o, b^r o)$ avec r maximal. Nous en déduisons donc :

Théorème 23 *Les classes $\boldsymbol{\mathcal{BB}}(\Sigma)$ et $\boldsymbol{\mathcal{BOULE}}(\Sigma)$ sont identifiables à la limite.*

Démonstration :
La Proposition 3 du chapitre précédent stipule que pour chaque boule, il existe un

unique couple de la forme $(a^i u, b^i u)$ avec $|u| + i = |o| + r$ tel que i soit maximum : le couple $(a^r o, b^r o)$. Il suffit alors d'attendre que ce couple apparaissent pour retourner l'hypothèse (o, r).

Une seconde façon de formuler des hypothèses est de vérifier la présence d'un témoin de minimalité pour une boule $B_r(o)$ au sein de la présentation. Si un tel témoin existe, l'hypothèse retournée sera alors (o, r) tant qu'un témoin pour une boule plus grande n'est pas trouvé. Lorsque toutes les données seront présentes, le témoin de la boule cible sera alors présent et l'hypothèse retournée sera alors correcte. □

En fait, il n'est pas utile d'attendre nécessairement le couple $(a^r o, b^r o)$. Le fait que ce couple soit intéressant vient du fait que $lcs(a^r o, b^r o) = \{o\}$, donc que o soit l'unique plus long sous-mot commun du couple. Supposons maintenant que X soit un ensemble quelconque. Si parmi les plus longs mots de X, il existe un couple ayant les mêmes propriétés, il suffit seulement de quelques autres conditions pour que nous puissions en déduire que la boule que nous cherchons est $B_r(o)$. Nous pouvons alors trouver des témoins de minimalité pour les boules au sens de la Définition 18 du Chapitre 2

Proposition 4 *Soit un ensemble $X = \{x_1, \dots x_n\}$ de mots. Notons X^{max} et X^{min} les mots de longueur maximum et minimum dans X. X est un témoin de minimalité pour la boule $B_r(o)$ s'il contient des mots u, v, w tels que :*

1. $u, v \in X^{max}$,
2. $w \in X^{min}$,
3. $|u| - |w| = 2r$,
4. $lcs(u, v) = \{o\}$,
5. $|o| = |u| - r$ *et*
6. $X \subseteq B_r(o)$.

En d'autres termes, supposons que X contienne un témoin de minimalité $\{u, v, w\}$ pour la boule $B_r(o)$. Si $X \subseteq B_{r'}(o')$, alors soit $r' > r$, soit ($r' = r$ et $o' = o$). En d'autres termes, $B_r(o)$ est la plus petite boule par rapport au rayon contenant X.

Démonstration :
Nous avons $d(u, w) \geq |u| - |w| = 2r$. De plus, comme $\{u, w\} \subset X \subseteq B_{r'}(o')$, nous en déduisons que $d(u, w) \leq 2r'$ (puisque $2r'$ est le *diamètre* de $B_{r'}(o')$). Nous avons donc $r \leq r'$. Si $r < r'$, le lemme est vrai.

Supposons donc que $r' = r$. Nous avons alors $d(u, w) = 2r \leq d(u, o') + d(o', w)$. Comme $\{u, w\} \subset X \subseteq B_{r'}(o')$, nous avons $d(u, o') \leq r$ et $d(o', w) \leq r$, donc $d(u, o') = d(o', w) = r$. Comme $|u| - |o'| \leq d(u, o') = r$, nous obtenons $|o'| \geq |u| - r = |o|$. Inversement, $|o'| - |w| \leq d(o', w) = r$, donc $|o'| \leq |w| + r = |u| - 2r + r = |o|$ et $|o'| = |o|$. si $o' \neq o$, comme $lcs(u, v) = \{o\}$ et $|o| = |o'|$, nous en déduisons que $o' \notin lcs(u, v)$, donc soit $o' \not\preceq u$ soit $o' \not\preceq v$. Supposons donc sans perte de généralité que $o' \not\preceq u$. Nous avons alors, $d(o', u) > |u| - |o'| = r$, ce qui est impossible puisque $u \in B_{r'}(o')$. Nous en concluons que $o' = o$. □

De plus, il existe un algorithme polynomial en $\|X\|$ pour vérifier si l'ensemble X est le témoin de minimalité pour une boule, et qui calcule cette boule $B_r(o)$ le cas échéant. En effet, il suffit :

- d'extraire X^{max} et X^{min} (1 et 2),
- de vérifier si la différence des longueurs entre ces deux ensembles est paire, ce qui permet de trouver r (3),
- de chercher $u, v \in X^{max}$ admettant un plus petit sous-mot commun unique, ce qui permet de trouver o (4),
- de tester les conditions (5) et (6).

Le seul point délicat ici est bien entendu la vérification en temps polynomial de l'unicité de $lcs(u, v)$. En effet, le nombre de sous-mots communs à u et v peut être supérieur à $1,442^n$ (avec $n = |u| = |v|$), et si nous cherchons à connaître tous les emplacements possibles des sous-mots communs ce nombre monte à $\frac{3}{\pi n}2,598^n$ [Gre03].

Néanmoins, Rick propose dans [Ric00] une structure de données du nom de Lcs-graphe permettant de stocker un ensemble de sous-mots communs. Greenberg reprend cette structure dans [Gre02], et présente un moyen de construire le Lcs-graphe de deux mots u et v en un temps $\mathcal{O}(|u| \times |v|)$. Une fois le graphe construit, nous pourrions alors facilement lister les sous-mots communs à u et v. Toutefois, ce nombre peut être exponentiel. De plus, nous ne désirons pas connaître tous les sous-mots de tous les couples de X : dès que nous savons que le cardinal de $lcs(u, v)$ est au moins égal à 2, nous savons que le couple (u, v) n'est pas un couple qui nous intéresse. Donc même s'il existe $1,442^{|u|}$ sous-mots communs, nous nous arrêterons à partir du deuxième. Par conséquent, le point (4) précédent est réalisable grâce au Lcs-graphe en un temps $\mathcal{O}(|X|^2 \times |u| \times |v|)$.

Il existe donc des ensembles (au moins un par boule) tels que nous pouvons construire une boule de rayon r contenant tous les mots de cet ensemble et être sûr que tout autre boule contenant ces mots possède un rayon strictement plus grand que r.

Afin de préparer l'identification sous contrainte de polynomialité, nous présentons maintenant un algorithme permettant l'identification des bonnes boules grâce au témoin de minimalité (Algorithme 5). Le fonctionnement de l'algorithme est le suivant : nous vérifions si le Texte est un témoin de minimalité pour une boule $B_r(o)$ à chaque nouvelle donnée de la présentation. Si tel est le cas, nous faisons l'hypothèse (o, r). Dans le cas contraire nous retournons une boule dont nous savons pertinemment qu'elle n'est pas la bonne (nous l'appellerons boule *de sécurité*), mais dont le rayon est suffisamment grand pour contenir toutes les données de la présentation vue juste qu'à présent.

L'Algorithme 5 identifie alors bien $\mathcal{BB}(\Sigma)$ à la limite car si $B_r(o)$ dénote la boule cible, alors l'algorithme verra un jour les mots de la boule $u = a^r o$ et $v = b^r o$ ainsi qu'un mot w de longueur $|o| - r$ qui répondent tous les trois aux exigences du témoin de minimalité pour $B_r(o)$.

5.2.1 Apprentissage en temps Mc polynomial

Considérons maintenant l'identification en temps Mc polynomial. L'Algorithme 5 nous permet de ne faire qu'un nombre minimum de changements d'hypothèse :

Algorithme 5 : Identification des bonnes boules à partir de texte

Données : Un texte $\boldsymbol{f} = \{x_1, x_2, \ldots\}$
Résultat : Une séquence de boules (o, r)

```
1  lire(x_1);
2  c ← x_1;
3  renvoyer (x_1, 0);
4  tant que vrai faire
5  |  lire(x_i);
6  |  si f_i est un témoin de minimalité pour B_r(o) alors
7  |  |  renvoyer (o, r)                          // boule valide
8  |  sinon
9  |  |  si c ∉ X^max alors
10 |  |  |  c ← un mot de X^max;
11 |  |  finsi
12 |  |  renvoyer (c, |c|)                        // boule de sécurité
13 |  finsi
14 fintq
```

Théorème 24 $\boldsymbol{\mathcal{BB}}(\Sigma)$ *est identifiable à la limite à partir de* Texte *en temps* Mc *polynomial.*

Démonstration :
L'Algorithme 5 est un algorithme identifiant à la limite $\boldsymbol{\mathcal{BB}}(\Sigma)$ à partir de Texte en temps Mc polynomial.

En effet, nous avons déjà vu que l'Algorithme 5 identifie à la limite $\boldsymbol{\mathcal{BB}}(\Sigma)$ à partir de Texte. Il ne reste donc qu'à compter le nombre de changements d'hypothèse que fait l'algorithme.

L'algorithme peut changer d'avis de plusieurs façons : en faveur d'une boule *valide*, ou en faveur d'une boule *de sécurité*. Le changement d'hypothèse en faveur d'une boule valide ne se fait que vers une boule valide de rayon plus grand que celui de la boule valide précédente. Il en est de même pour les boules de sécurité.

Plus précisément, soit $T = (x_1, s_1)(x_2, s_2)(x_3, s_3)\ldots$ la trace d'une exécution de l'algorithme sur une présentation $\boldsymbol{f}$. Supposons de plus que la boule cible soit $B_r(o)$. La trace de l'algorithme est une succession de représentations de boule. Chaque (x_i, s_i) est donc soit la représentation d'une boule (o, r) venant d'un témoin de minimalité, soit celle d'une boule de sécurité $(c, |c|)$.

Intéressons-nous tout d'abord aux boules valides. Soient (o_i, r_i) une représentation issue d'un témoin de minimalité, et j le plus petit rang tel que $j > i$ et (o_j, r_j) provienne aussi d'un témoin de minimalité. Deux cas sont alors possibles :

1. $\boldsymbol{f}(i+1) \in B_{r_i}(o_i)$: le témoin de minimalité $(u_i, v_i, w_i, o_i, r_i)$ ayant permis de construire $B_{r_i}(o_i)$ est toujours un témoin de minimalité pour $\boldsymbol{f}_{i+1}$ (par définition du témoin). Donc, par la Proposition 4, nous en déduisons que $j = i+1$ et que $(o_j, r_j) = (o_i, r_i)$, c'est-à-dire qu'aucun changement d'hypothèse n'a eu lieu.

2. $\boldsymbol{f}(i+1) \notin B_{r_i}(o_i)$: par construction $\boldsymbol{f}_i$ est inclus dans $B_{r_j}(o_j)$. La Proposition 4 nous permet de déduire que soit $r_i < r_j$, soit $(o_j = o_i$ et $r_j = r_i)$. Les deux boules étant différentes (puisque $\boldsymbol{f}(i+1) \in (B_{r_i}(o_i) \oplus B_{r_j}(o_j)))$, nous avons nécessairement $r_i < r_j$.

Par conséquent, chaque fois que **Alg** change d'hypothèse en faveur d'une nouvelle boule valide, le rayon est incrémenté d'au moins 1 par rapport à la boule valide précédente. Le nombre de représentations de boules valides différentes retournées par l'algorithme sera donc au plus r, c'est-à-dire le rayon de la boule cible. Le nombre de MC de **Alg** *en faveur* d'une boule valide est donc inférieur à r.

Si nous nous intéressons maintenant au nombre de changements d'avis en faveur d'une boule de sécurité, l'étude de la trace de l'exécution de l'algorithme nous permet à nouveau de borner ce nombre. Soient deux boules de sécurité $(c_i, |c_i|)$ et $(c_j, |c_j|)$. Supposons que $i < j$. Alors, $|c_i| \leq |c_j|$ puisque c_i et c_j sont des mots de longueur maximale dans, respectivement, $\boldsymbol{f}_i$ et $\boldsymbol{f}_j$. De plus, si $|c_i| = |c_j|$ alors $c_i = c_j$. Le nombre de boules de sécurité différentes le long de T est alors borné par $2r$ (puisque $\forall x \in B_r(o), |o| - r \leq |x| \leq |o| + r$). De plus, le nombre de MC *en faveur* d'une boule de sécurité est inférieur à $3r$, c'est-à-dire, r MC pour passer d'une boule valide à une boule de sécurité, et $2r$ MC pour passer d'une boule de sécurité à une autre boule de sécurité.

Le nombre total de MC est donc borné par $4r$. Enfin, le temps de mise à jour est polynomial puisque la seule chose que fait l'algorithme est de vérifier si oui ou non $\boldsymbol{f}_i$ est un témoin de minimalité. Or nous avons vu au chapitre précédent que cette vérification pouvait se faire en temps polynomial. □

La Figure 5.2 montre un exemple de ce que vaut le rayon des boules issus d'un témoin, celui des boules de sécurité, et celui des boules hypothèses. La courbe des rayons des boules issus d'un témoin sera donc borné par r, celle des boules de sécurité par $2r$, et la courbe du rayon des boules hypothèses sera comprise entre ces deux courbes.

Lorsque l'algorithme aura convergé, la courbe des rayons des boules hypothèses sera confondue avec celle des boules témoins. Le nombre de MC peut également être calculé en comptant le nombre de variations de la courbe *hypothèse*. Par exemple, à chaque fois qu'elle décroît, un nouveau témoin a été trouvé, et l'algorithme a effectué un changement d'hypothèses.

Il est à noter que l'algorithme n'a en fait pas besoin d'être consistant et donc de changer d'avis pour des boules de sécurité. Toutefois, nous allons le réutiliser pour l'apprentissage en temps IPE polynomial.

L'apprentissage en temps MC polynomial pourrait donc très bien se faire uniquement avec r changements d'hypothèse au maximum : si la boule hypothèse n'est plus consistante avec les nouvelles données, il suffit d'attendre d'avoir un nouveau témoin de minimalité pour une boule plus grande. En d'autres termes, il suffit de rester le long de la courbe *témoin*.

Le fait de n'avoir pas besoin d'être consistant nous permet également de montrer que $\boldsymbol{\mathcal{BB}}(\Sigma)$ est identifiable à partir d'INFORMATEUR :

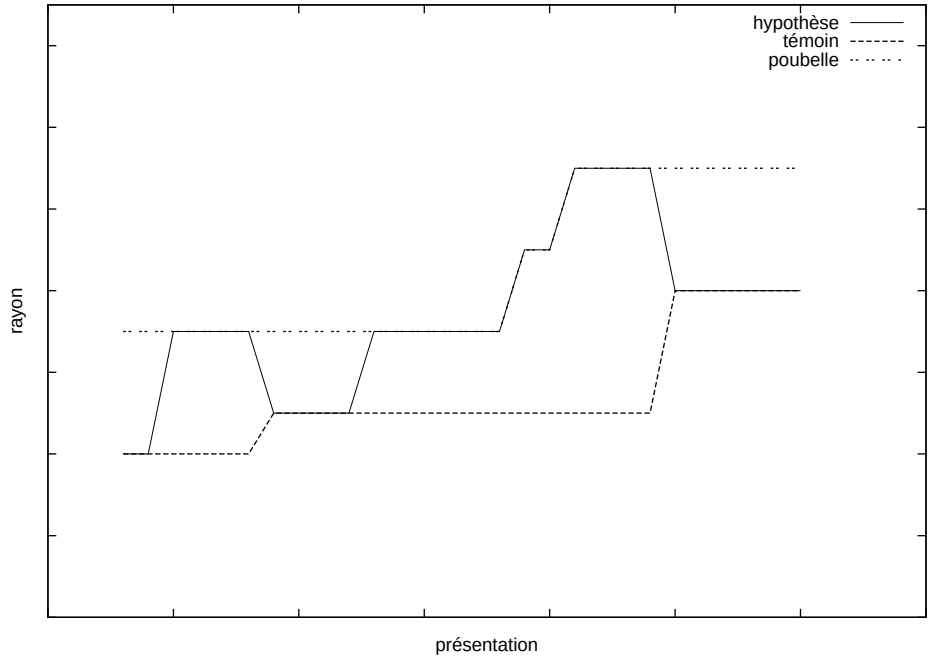

FIG. 5.2 – Rayon des boules hypothèses, des boules de sécurité et des boules issus des témoins, le long d'une présentation.

Théorème 25 $\mathcal{BB}(\Sigma)$ *est identifiable à la limite à partir de* Informateur *en temps* Mc *polynomial.*

Démonstration :
L'Algorithme 5 identifie à la limite $\mathcal{BB}(\Sigma)$ à partir de Texte en temps Mc polynomial (Théorème 25). Or l'algorithme n'a pas besoin d'être consistant avec les données pour pouvoir identifier polynomialement. Pour chaque exemple négatif qu'il reçoit, il lui suffit donc de retourner la dernière hypothèse qu'il vient de faire. Ainsi, il identifie toujours à la limite, il a un temps de mise a jour qui reste polynomial, et le nombre de changements d'hypothèse est toujours borné par quatre fois le rayon de la boule cible. □

La vérification de l'existence du témoin de minimalité d'un ensemble de données positives pour une boule en temps polynomial permet donc d'identifier à la limite $\mathcal{BB}(\Sigma)$ en temps Mc polynomial. Il est à noter que cet algorithme peut être adapté à l'identification de n'importe quel classe de langage, si tant est que nous puissions vérifier l'existence d'un témoin de minimalité[4].

Concernant les mauvaises boules, nous pouvons montrer que $\mathcal{BOULE}(\Sigma)$ n'est pas identifiable à partir de Texte. En effet, il existe des présentations telles que n'importe quel algorithme doit faire un nombre de changements d'hypothèse supérieur au rayon de la boule :

Théorème 26 $\mathcal{BOULE}(\Sigma)$ *n'est pas identifiable à la limite en temps* Mc *polynomial à partir de* Texte.

Démonstration :

Raisonnons par l'absurde et supposons que nous ayons un apprenant **Alg** et un polynôme $p()$ tels que $\forall G \in \mathcal{G}, \forall f \in \mathbf{Pres}, \#\text{Mc}(f) \leq p(\|G\|)$.

Soit n un entier suffisamment grand, et considérons la sous-classe cible $B_k(\lambda)$ avec $k \leq n$. Pour chaque cible, nous construisons une présentation f^k en utilisant **Alg** de façon interactive.

À chaque étape i, **Alg** produit une hypothèse H_i, et nous devons calculer un nouveau mot $f^k(i+1)$. Pour cela, si $i = 0$ nous retournons λ. Sinon, il y a deux cas :

1. Si $H_{i-1} = B_k(\lambda)$, alors nous retournons le plus petit mot de $B_k(\lambda)$ qui n'est pas apparu dans f^k_{i-1} (si aucun mot ne vérifie cette contrainte, nous retournons λ). En d'autres termes, si l'hypothèse est correcte, nous retournons le plus petit mot de la boule qui n'a pas été vu (λ sinon) afin que la présentation soit correcte (c'est-à-dire qu'elle contienne tous les mots de la boule).
2. Si $H_{i-1} \neq B_k(\lambda)$, alors nous retournons a^{j+1} si $H_{i-1} = B_j(\lambda)$ avec $j = \max\{|u| : u \in f^k_{i-1}\}$, et λ sinon. En d'autres termes, si l'hypothèse n'est pas la bonne, mais qu'elle couvre tous les éléments de la présentation, nous présentons un mot plus long que tous ceux qui ont été vus jusqu'à présent, λ sinon.

[4]Par exemple, c'est cette méthode qui a été utilisée pour identifier les automates en temps Mc polynomial.

Chaque présentation $\boldsymbol{f}^k$ est alors une présentation (TEXTE) correcte de la cible $B_k(\lambda)$. En d'autres termes, nous avons bien $\boldsymbol{f}^k(\mathbb{N}) = B_k(\lambda)$.

Posons $m(k) = \min\{i \in \mathbb{N} : \boldsymbol{f}^k(i) = a^k\}$. Pour chaque k, $\boldsymbol{f}^k$ et $\boldsymbol{f}^{k+1}$ coïncident sur les mêmes $m(k)$ valeurs initiales. Alors $\boldsymbol{f}^n$ peut être réécrit en : $\lambda, \ldots, \lambda, a, \ldots, a^j, \ldots, a^n, \ldots$ avec : $\forall 0 < j \leq n, \forall i \in \{m(j-1), .., m(j)-1\}, \boldsymbol{f}^n(i) = \boldsymbol{f}^j(m(j-1)) = \boldsymbol{f}^j(i) = a^{j-1}$, et **Alg** change d'avis juste avant de recevoir le nouvel exemple a^i et fait donc au minimum n changements d'hypothèses. Cela prouve que pour tout polynôme $p()$, il existe un entier n tel que $\#\text{MC}(\boldsymbol{f}^n) > p(\log n)$. □

En revanche, $\boldsymbol{\mathcal{BOULE}}(\Sigma)$ devient identifiable si la présentation contient des exemples négatifs. En effet, grâce aux exemples négatifs, nous pouvons vérifier qu'il n'existe pas de boules possédant un rayon plus grand que celui de la boule hypothèse :

Théorème 27 $\boldsymbol{\mathcal{BOULE}}(\Sigma)$ *est identifiable à la limite en temps* MC *polynomial à partir d'*INFORMATEUR.

Démonstration :

Nous montrons qu'il existe un apprenant qui vérifie les données jusqu'à être sûr qu'il n'existe qu'une boule consistante avec les données et fait donc un unique changement d'avis.

Soient $B_r(o)$ la boule cible et $\langle X_+, X_- \rangle$ des exemples tels qu'il existe un mot u pour lequel :

1. $a^k u, b^k u \in X_+$,
2. tous les sur-mots de $a^k u$ et de $b^k u$ de longueur $|u| + 1 + k$ sont dans X_- et
3. si $u \neq \lambda$, pour chaque sous-mot v de u de longueur $|u| - 1$, il existe un sur-mot de v de longueur $|u| + k$ dans X_-.

Il n'existe alors qu'une seule boule vérifiant ces conditions. En effet, étant donnée une boule $B_r(o)$, ces exemples existent toujours, et vérifier si un tel mot est dans X est en $\mathcal{O}(\|X\|)$.

En outre, toutes les opérations d'édition dans un chemin minimal pour transformer o en $a^k u$ et $b^k u$ sont des insertions. En effet, la condition 2 nous permet de déduire que $a^k u$ et $b^k u$ sont des mots de la frontière supérieure de la boule. Nous en déduisons par la Proposition 5 que $o \preceq a^k u$ et $o \preceq b^k u$. Ainsi, $o \preceq u$.

Enfin, puisque pour chaque sous-mot w de u il existe un sur-mot de longueur $|u| + k$ dans X_-, aucun sous-mot propre de u ne peut être le centre.

Nous en déduisons alors que $u = o$ et donc que $k = r$. Évidemment, les conditions requises seront vraies à un certain moment de la présentation. □

5.2.2 Apprentissage en temps IPE polynomial

Intéressons-nous maintenant à l'identification de $\boldsymbol{\mathcal{BB}}(\Sigma)$ à partir de TEXTE en temps IPE polynomial. Si nous étudions l'Algorithme 5, nous pouvons constater que l'existence

des boules de sécurité permet à l'algorithme de rester consistant tout au long de l'inférence. Cela nous permet alors de déduire que les boules sont apprenables en temps IPE polynomial puisqu'elles le sont en temps MC polynomial :

Théorème 28 $\mathcal{BB}(\Sigma)$ *est identifiable à la limite à partir de* TEXTE *en temps* IPE *polynomial.*

Démonstration :
L'Algorithme 5 est un algorithme consistant. En effet, pour chaque nouvelle donnée de la présentation n'appartenant pas à une boule valide, **Alg** change d'avis pour une boule de sécurité, boule qui contient toutes les données positives de la présentation : $\boldsymbol{f}_i \subseteq \Sigma^{\leq |c|} \subseteq B_{|c|}(c)$. Il ne fera alors que deux sortes d'erreurs : lorsque la boule de sécurité n'est plus suffisamment grande, ou lorsqu'un témoin de minimalité ne tient plus.

Par conséquent, puisque l'algorithme est consistant et qu'il identifie les boules à la limite en temps MC polynomial, le Lemme 1[5] nous permet de déduire que $\mathcal{BB}(\Sigma)$ est aussi identifiable à la limite à partir de TEXTE en temps IPE polynomial. □

En revanche, les boules ne sont pas apprenables à partir d'INFORMATEUR en faisant un nombre polynomial d'IPE. En effet, si tel était le cas, nous pourrions les apprendre avec des requêtes d'équivalence, suivant la technique utilisée par Pitt dans [Pit89] pour montrer que les AFD ne sont pas identifiables à la limite à partir d'INFORMATEUR en temps IPE polynomial (corollaire 6, page 25).

Théorème 29 $\mathcal{BB}(\Sigma)$ *n'est pas identifiable à la limite à partir d'*INFORMATEUR *en temps* IPE *polynomial.*

Démonstration :
Raisonnons donc par l'absurde, et supposons qu'il existe un algorithme **Alg** tel que **Alg** identifie $\mathcal{BB}(\Sigma)$ à la limite à partir d'INFORMATEUR en temps IPE polynomial.

Alg fait donc une suite d'hypothèses $B_1, B_2, \ldots$ jusqu'à trouver la boule cible. Laissons maintenant de côté toutes les hypothèses qui ne résultent pas en une erreur implicite de prédiction. En d'autres termes, nous gardons donc uniquement les hypothèses telles que **Alg** fasse une IPE sur la prochaine donnée de la présentation. Cette suite d'hypothèses correspond alors à une identification avec des requêtes d'équivalence puisqu'après chaque hypothèse faite par l'algorithme un contre-exemple est fourni (ou, si l'hypothèse est correcte, l'inférence prend fin).

Un algorithme **Alg2** apprenant à partir d'EQ pourrait donc se servir de l'algorithme **Alg** en lui fournissant tous les contre-exemples qu'il reçoit et en faisant comme requête la dernière hypothèse de **Alg**. **Alg** faisant un nombre polynomial d'IPE, le nombre de requêtes d'équivalence de **Alg2** serait donc lui aussi polynomial.

Si $\mathcal{BB}(\Sigma)$ était identifiable à la limite à partir d'INFORMATEUR en temps IPE polynomial, elle le serait également à partir de requêtes d'équivalence uniquement. Or, ce

[5] Si **Alg** identifie la classe $\mathcal{G}$ à la limite en temps MC polynomial et qu'il est consistant, alors **Alg** identifie $\mathcal{G}$ en temps IPE polynomial.

n'est pas le cas, d'après le Théorème 16, donc $\mathcal{BB}(\Sigma)$ n'est pas identifiable à la limite à partir d'Informateuren temps Ipe polynomial. □

Il est donc ainsi faux de penser que de disposer d'exemples négatifs en plus des exemples positifs aide à l'apprentissage. Ceci n'est pas surprenant : c'est en effet le cas pour l'apprentissage de certains langages hors-contexte. La principale raison de ce résultat est que le complémentaire d'une boule n'est pas une boule (de même que le complémentaire d'un langage hors-contexte n'est pas hors-contexte). Par conséquent, les exemples négatifs pénalisent ici plus qu'ils n'aident puisqu'il est difficile de trouver la plus petite boule consistante avec un ensemble de mots (Théorème 15).

Concernant les mauvaises boules, les bonnes boules n'étant pas identifiables à la limite à partir d'Informateur en temps Ipe polynomial, $\mathcal{BOULE}(\Sigma)$ ne peut pas l'être non plus. De même, pour l'apprentissage à partir de Texte, avec le même raisonnement que pour le Théorème 26, nous pouvons montrer que $\#\text{Ipe}(\boldsymbol{f}^n) > p(\log n)$ et en déduire :

Théorème 30 $\mathcal{BOULE}(\Sigma)$ *n'est pas identifiable à la limite en temps* Ipe *polynomial à partir de* Texte, *ou d'*Informateur.

5.2.3 Apprentissage en temps Cs polynomial

Nous terminons cette étude par l'apprentissage en temps Cs polynomial.

Reprenons encore une fois l'Algorithme 5 en laissant de côté les boules de sécurité. Les témoins de minimalité utilisés servent alors aussi d'ensembles caractéristiques. En effet, la taille d'un témoin de minimalité est bien polynomiale en la taille de la représentation d'une boule. De plus, dès que ce témoin est inclus dans la présentation, alors l'algorithme retourne toujours la même représentation, qui est celle de la boule cible. Cela répond alors bien à tous les points de la Définition 21 du Chapitre 2 d'ensemble caractéristique polynomial.

Ainsi, que ce soit à partir de Texte ou d'Informateur, le témoin de minimalité est un ensemble caractéristique qui fait converger l'Algorithme 5 :

Théorème 31 $\mathcal{BB}(\Sigma)$ *est identifiable à la limite en temps* Cs *polynomial à partir de* Texte *ou d'*Informateur.

Par contre, dans le cas des mauvaises boules, l'apprentissage ne peut se faire en temps polynomial. En effet, reprenons la classe des boules centrées sur λ et considérons les boules $B_r(\lambda)$ et $B_{r+1}(\lambda)$. Nécessairement, afin de pouvoir différencier ces deux boules, l'ensemble caractéristique de $B_{r+1}(\lambda)$ aura au moins un mot de Σ^{r+1} que l'ensemble caractéristique de $B_r(\lambda)$ ne contiendra pas. Ce mot étant de longueur $r+1$, l'ensemble caractéristique de $B_{r+1}(\lambda)$ sera donc de taille exponentielle en $\log r$ ce qui ne peut pas être polynomial en la taille $|o| + r$ de la représentation de la boule :

Théorème 32 $\mathcal{BOULE}(\Sigma)$ *n'est pas identifiable à la limite en temps* Cs *polynomial à partir de* Texte *ou d'*Informateur.

Les mauvaises boules ne sont donc apprenables que sans contraintes de polynomialité et en temps MC polynomial à partir d'INFORMATEUR. À l'inverse, les bonnes boules sont apprenables dans la plupart des paradigmes d'apprentissage à la limite, à l'exception de l'apprentissage à la limite en temps IPE polynomial à partir d'INFORMATEUR.

Conclusion

Les bonnes boules de mots étant identifiables à la limite dans la plupart des paradigmes, les techniques vues au début de ce chapitre peuvent donc s'appliquer. Contrairement aux AFD, $\mathcal{BB}(\Sigma)$ est donc identifiable à la limite à partir de données bruitées systématiquement en utilisant soit une technique de réduction, soit une technique de débruitage à la limite.

Les boules n'étant pas PAC apprenables, ni apprenables à partir de MQ et EQ dans un contexte non bruité, il ne nous reste plus qu'à nous intéresser à leur identification à partir de requêtes de correction lorsque l'oracle peut répondre approximativement.

Chapitre 6

Apprentissage actif des boules en situations bruitées

Nous terminons cette étude par l'apprentissage des boules de mots en situations bruitées à partir de requêtes. Les boules n'étant pas apprenables à partir de MQ et EQ, nous nous concentrons sur leur apprentissage à partir de requêtes de correction.

Nous proposons dans un premier temps un algorithme d'apprentissage linéaire des boules à partir de CQ_{EDIT}. Nous prouvons ensuite que sous certaines conditions, la complexité de l'algorithme peut être réduite à une complexité logarithmique. Nous terminons alors en montrant que l'algorithme identifiant les boules à partir de CQ_{EDIT} est assez résistant face à un oracle répondant approximativement aux requêtes de correction.

6.1 Apprentissage à partir de CQ_{EDIT}

L'identification de $\mathcal{BB}(\Sigma)$ à partir de requêtes de correction est assez technique et nous allons devoir procéder par plusieurs étapes. Tout d'abord, nous allons nous intéresser à la caractérisation des corrections retournées par l'oracle. Mais afin de mieux comprendre pourquoi les CQ_{EDIT} semblent intéressantes pour l'apprentissage de boules, prenons l'exemple de l'identification de disque dans $\mathbb{R}^2$ (voir Figure 6.1).

Si nous désirons identifier un disque dans le plan ou une boule dans l'espace euclidien, nous pouvons procéder de la façon suivante :

- Demander à l'oracle la correction d'un point ;
- Tant que l'oracle répond que le point appartient à la boule, demander la correction d'un autre point, en allant toujours dans la même direction, de plus en plus loin ;
- Une fois que l'on a obtenu un point A hors de la boule et sa correction A', recommencer[1] pour avoir un point B et sa correction B' ;
- Le centre O est alors donné par l'intersection des droites (AA') et (BB') ;
- Le rayon est la longueur des segments $[OA']$ et $[OB']$

[1] Il suffit en fait de se déplacer légèrement de A en ne restant pas sur la droite (AA'), par exemple, sur une droite perpendiculaire à (AA') passant par A (ce n'est pas le cas de la Figure 6.1 pour plus de lisibilité).

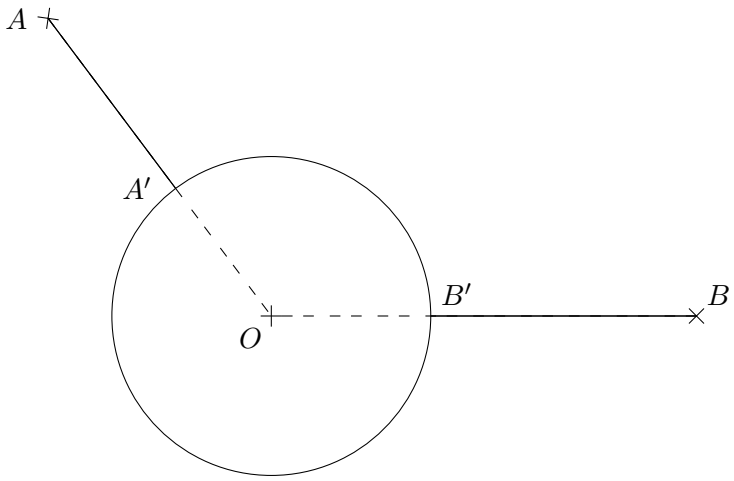

FIG. 6.1 – Apprentissage d'un disque dans le plan en utilisant des requêtes de corrections.

6.1.1 Une caractérisation des corrections

Trouver une boule dans un espace euclidien à n dimensions se fera toujours de la même façon, avec peu de requêtes de correction. Nous pouvons alors espérer qu'une méthode identique existe pour l'apprentissage des boules de mots. Toutefois, plusieurs problèmes vont alors se poser : quand nous munissons Σ^* de la distance d'édition, nous obtenons un espace métrique mais nous ne disposons pas d'espace vectoriel. Autrement dit, c'est comme si nous nous ramenions à l'apprentissage des boules dans le plan mais sans règle, seulement avec un compas. De plus, nous avons vu au chapitre précédent que les boules de mots sont très différentes des boules du plan (non homogène, non convexe, *etc.*). En particulier, plusieurs corrections peuvent être possible pour une même requête, les corrections pouvant même avoir des longueurs différentes :

Exemple 39 *Soit la boule $B_1(ba)$. Nous avons alors $Cq(a^4) = \{aa, aba, baa\}$, soit trois corrections possibles, toutes de longueurs différentes.*

L'identification n'en sera alors que plus ardue car dans le cas d'oracle malveillant, celui aura la possibilité de choisir la correction qui nous aide le moins (comme un mot dont nous savons déjà qu'il appartient à la boule cible par exemple).

De même, si dans le plan il suffit d'obtenir la correction de deux mots hors de la boule, cela n'est plus suffisant dans Σ^* :

Exemple 40 *Soient les corrections suivantes : $Cq(a) = ab, Cq(b) = bb, Cq(a^5) =$*

$aabaa, Cq(b^5) = babbb, Cq((ab)^5) = (ab)^4$. *Même si l'on dispose d'autant d'informations, il existe plusieurs boules telles que les réponses à ces requêtes sont justes :* $B_2(aabb)$, $B_2(abab)$ *ou encore* $B_2(abba)$.

Cependant, tout comme dans l'espace euclidien, nous pouvons montrer que les corrections sont des mots de la frontière de la boule :

Lemme 9 *Soit* $B_r(o)$ *une boule et* m *un mot tel que* $m \notin B_r(o)$. *Alors un mot* z *est une correction possible de* m *si et seulement si* $(d(o,z) = r$ *et* $d(z,m) = d(o,m) - r)$.

Démonstration :
Posons $k = d(o,m)$ et considérons une dérivation de o en m de longueur minimale : $o \xrightarrow{k} m$. Comme $m \notin B_r(o)$, nous avons $k > r$, donc le long de la dérivation précédente, il existe un mot w_0 tel que $o \xrightarrow{r} w_0 \xrightarrow{k-r} m$. Définissons l'ensemble $W = \{w \in \Sigma^* \mid d(o,w) = r \text{ et } d(w,m) = k-r\}$. Clairement, $w_0 \in W$, donc $W \neq \emptyset$. De plus, $W \subseteq B_r(o)$. Enfin, notons Z l'ensemble des corrections possibles de m et montrons que $Z = W$.

Soit $z \in Z$ et $w \in W$. Si $d(z,m) > d(w,m)$, alors w est un mot de $B_r(o)$ qui est strictement plus proche de m que z, ce qui contredit le fait que z soit une correction possible de m. Supposons que $d(z,m) < d(w,m)$. Nous avons $d(o,m) \leq d(o,z) + d(z,m)$, donc $d(o,z) \geq d(o,m) - d(z,m) > d(o,m) - d(w,m)$. Or $d(o,m) = k$ et $d(w,m) = k-r$, donc $d(o,z) > r$, ce qui est impossible puisque $z \in Z \subseteq B_r(o)$. Donc $d(z,m) = d(w,m) = k - r$. En conséquence, tout mot $w \in W$ est à la même distance de m qu'une correction $z \in Z$. Donc $W \subseteq Z$. De plus, nous avons $d(o,m) \leq d(o,z) + d(z,m)$, donc $k \leq d(o,z) + k - r$, c'est-à-dire, $d(o,z) \geq r$. Comme $z \in B_r(o)$, nous en déduisons $d(o,z) = r$. Comme nous avons aussi $d(z,m) = k - r$, $Z \subseteq W$. □

En d'autres termes, le Lemme 9 permet de dire que les corrections d'un mot w par rapport à une boule $B_r(o)$ se situent à l'intersection du cercle de centre o et de rayon r (soit l'ensemble $\{w \in \Sigma^* : d(o,w) = r\}$) avec le segment $[o,w]$ (c'est-à-dire avec l'ensemble $\{u \in \Sigma^* : d(o,u) + d(u,w) = d(o,w)\}$).

6.1.2 Les mots de longueur maximum sur la frontière

Une fois que l'on sait que les corrections sont à la frontière, nous pouvons nous attaquer au problème sous un nouvel angle. En effet, si nous connaissons un mot de la frontière de la boule de longueur maximum, nous allons pouvoir essayer de trouver le couple de mots[2] $(a^r o, b^r o)$ et en déduire la boule cible.

Nous allons donc commencer par distinguer certains mots des boules, ceux qui ont un nombre maximum de lettres :

Définition 41 (Frontière supérieure) *On appelle frontière supérieure d'une boule* $B_r(o)$, *notée* $B_r^{max}(o)$, *l'ensemble des mots de* $B_r(o)$ *qui sont de longueur maximum :*

$$B_r^{max}(o) = \{z \in B_r(o) | \forall w \in B_r(o), |w| \leq |z|\}.$$

[2] Voir Chapitre 4 Proposition 3.

Exemple 41 *Soit* $\Sigma = \{a, b\}$ *et* $B_1(ba) = \{a, b, aa, ba, bb, aba, baa, bab, bba\}$. *Nous avons* $B_1^{max}(ba) = \{aba, baa, bab, bba\}$.

$B_r^{max}(o)$ est un ensemble remarquable car tous les mots qui le composent sont construits à partir du centre o en faisant r insertions. Aussi, disposant d'un mot $w \in B_r^{max}(o)$, il « suffit » de trouver les lettres qui ont été insérées dans o pour construire w, puis de les effacer pour retrouver o. Il n'est ainsi pas nécessaire de *construire* le centre o de toute pièce :

Proposition 5 $w \in B_r^{max}(o)$ *si et seulement si* $o \preceq w$ *et* $d(o, w) = |w| - |o| = r$.

Démonstration :
Supposons $o \preceq w$ et $d(o, w) = |w| - |o| = r$. Alors $w \in B_r(o)$. Soit w' un mot tel que $|w'| > |w|$. Alors, par la Proposition 1, $d(o, w') \geq |w'| - |o| > |w| - |o| = r$. La distance entre w' et le centre étant plus grande que le rayon, $w' \notin B_r(o)$. Par conséquent, $w \in B_r^{max}(o)$.

Réciproquement, soit $w \in B_r^{max}(o)$. Prenons sans perte de généralité la lettre $a \in \Sigma$. Soit le mot $a^r o$. De toute évidence, nous avons $d(o, a^r o) = r$. Le mot $a^r o$ appartient donc à $B_r(o)$. Comme $w \in B_r^{max}(o)$, nous en déduisons que $|w| \geq |a^r o| = |o| + r$. Ainsi, par la Proposition 1, $d(o, w) \geq |w| - |o| \geq r$. D'un autre côté, $r \geq d(o, w)$ puisque $w \in B_r^{max}(o)$. Nous en déduisons donc que $d(o, w) = |w| - |o| = r$, et donc que $o \preceq w$, par la Proposition 1. □

6.1.3 Trouver le centre à partir d'un mot de la frontière supérieure

Parmi les mots de $B_r^{max}(o)$, certains jouent un rôle spécial. En effet, $a^r o$ est obtenu à partir de o en faisant r insertions, donc il appartient à $B_r^{max}(o)$. De plus, si nous connaissons r, il suffit d'effacer les r premiers a pour trouver o. Nous prétendons qu'avec des requêtes de correction, nous pouvons obtenir $a^r o$ à partir de n'importe quel mot $w \in B_r^{max}(o)$ en faisant des échanges de lettres. C'est le but de la procédure EXTRACTION_DU_CENTRE (voir l'Algorithme 6).

Algorithme 6 : EXTRACTION_DU_CENTRE

Données : Un mot $w = a_1 \dots a_n \in B_r^{max}(o)$, le rayon r
Résultat : Le centre o de la boule $B_r(o)$

1 $x \leftarrow a_n$ `// x est une lettre arbitraire`
2 **pour** $i = 1$ **à** n **faire**
3 | Supposons $w = a_1 \dots a_n$, soit $w' = x a_1 \dots a_{i-1} a_{i+1} \dots a_n$;
4 | **si** $Cq(w') =$ OUI **alors**
5 | | $w \leftarrow w'$
6 | **finsi**
7 **finpour**
8 Supposons $w = a_1 \dots a_n$, **retourner** $a_{r+1} \dots a_n$;

Regardons un exemple d'exécution de l'algorithme :

Exemple 42 *Soit la boule* $B_2^{max}(bb) = \{aabb, abab, abba, abbb, baab, baba, babb, bbaa, bbab, bbba, bbbb\}$. *Prenons le mot* $abba$ *et un rayon* $r = 2$. *L'algorithme* EXTRACTION_DU_CENTRE *transforme, à chaque itération, la lettre d'indice* i *en un* a *qu'il place au début du mot, puis soumet le mot obtenu à l'oracle.*

i	w	w'	$Cq(w')$	w *change*
1	$\underline{a}bba$	$abba$	OUI	*oui*
2	$a\underline{b}ba$	$aaba$	$abba$	*non*
3	$ab\underline{b}a$	$aaba$	$abba$	*non*
4	$abb\underline{a}$	$aabb$	OUI	*oui*

Ainsi, EXTRACTION_DU_CENTRE *s'arrête sur* $w = aabb$ *et retourne alors* $o = bb$ *(comme* $r = 2$*).*

Nous montrons maintenant que l'algorithme que nous venons de voir est correct et fonctionne en temps polynomial en la taille de la représentation de la boule :

Lemme 10 *Pour tout mot* $w \in B_r^{max}(o)$, *sachant* r, EXTRACTION_DU_CENTRE *retourne* o *en temps polynomial, avec* $\mathcal{O}(|o| + r)$ *requêtes de correction.*

Démonstration :
Montrons tout d'abord que l'opération d'échange de lettres est correcte. Soient $w = x_1 \dots x_n \in B_r^{max}(o)$ et $w' = ax_1 \dots x_{i-1}x_{i+1} \dots x_n$ pour $1 \leq i \leq n$. S'il existe une dérivation de réécriture $o \xrightarrow{r} w$ où la lettre x_i de w est le produit d'une insertion dans o, alors supprimer x_i et faire l'insertion d'un a en tête de o conduit à un mot w' qui satisfait $o \preceq w'$ et $|w'| = |w|$. Par la Proposition 1, nous avons $d(o, w') = |w'| - |c| = |w| - |c| = r$, donc par la Proposition 5, $w' \in B_r^{max}(o)$ et $Cq(w') = \text{OUI}$.

Par contre, s'il n'existe aucune dérivation de réécriture où x_i est le produit d'une insertion, alors x_i est une "vraie" lettre de o. Dans ce cas, supprimer x_i et ajouter un a en tête de o conduit à un mot w' tel que $o \not\preceq w'$. Par la Proposition 1, nous avons $d(o, w') > |w'| - |o|$ et comme $|w'| = |w|$, nous avons $d(o, w') > r$. Donc $w' \notin B_r^{max}(o)$ et $Cq(w') \neq \text{OUI}$.

À chaque itération, si la lettre est issue d'une insertion elle est placée en début de mot. Si la lettre est issue du centre, elle reste à sa place. À la fin de l'algorithme, les r lettres issues des insertions se retrouveront donc en début de mot, la fin étant composée des lettres de o, soit un mot de la forme $a^r o$.

Enfin, l'algorithme parcourt le mot w de $B_r^{max}(o)$ en effectuant une requête à chaque lettre, soit $|w| = |o| + r$ requêtes de correction. □

À partir d'un mot de la frontière supérieure (et du rayon), nous pouvons donc trouver le centre et donc la boule.

6.1.4 Trouver un mot de la frontière supérieure

Afin de pouvoir utiliser l'algorithme EXTRACTION_DU_CENTRE, il nous faut trouver un mot de la frontière supérieure ainsi que le rayon de la boule. Une première idée serait de demander la correction du mot vide, puis la correction d'un mot de plus en plus grand. Une fois que la correction est plus petite que la requête, le mot appartient-il à la frontière supérieure ? La réponse est non. La recherche d'un mot de longueur maximal est de nouveau plus complexe qu'il n'y paraît :

Exemple 43 *Mettons-nous à la place de l'apprenant. Soit la correction* $Cq(b^3a^3) = b^2a^3$. *La correction étant plus petite que la requête, nous pourrions penser que les mots les plus longs sont de longueurs* 5. *Pour plus de sûreté, nous demandons la correction d'un mot plus long* $Cq(b^4a^3) = b^2a^4$. *La correction étant plus longue que la précédente, nous continuons :* $Cq(b^{10}a^3) = bbbabba$. *Cette fois-ci la correction est beaucoup plus petite. Pour être vraiment sûr nous demandons la correction d'un mot vraiment très long :* $Cq(b^{100}a^3) = bbbabba$. *La différence de longueur étant vraiment importante, et la correction obtenue étant identique à la précédente, nous pouvons penser que la longueur des mots de la frontière supérieure est de* 7. *Malheureusement, la boule cible était* $B_4(aabb)$ *et donc les mots les plus longs de la boule sont de longueur* 8.

Cet exemple illustre bien le fait que nous ne pouvons pas nous fier à notre intuition pour trouver un mot de $B_r^{max}(o)$. Toutefois, il existe des mots qui, s'ils sont suffisamment grands, force les requêtes de correction à ne faire que des suppressions pour trouver un mot de la boule. La correction obtenue est alors un mot de la frontière supérieure :

Lemme 11 *Supposons que l'alphabet contienne n lettres :* $\Sigma = \{a_1, \ldots, a_n\}$. *Alors toute correction* w *du mot* $v = (a_1 \ldots a_n)^k$ *avec* $k \geq |o| + r$ *est un mot de* $B_r^{max}(o)$.

Démonstration :
Soit U l'ensemble des corrections possibles de v. Montrons que $U = B_r^{max}(o)$. Comme $v = (a_1 \ldots a_n)^k$ avec $k \geq |o| + r$, nous avons $o \preceq v$, donc par la Proposition 1, $d(o, v) = |v| - |o|$. Par ailleurs, soit $w \in B_r^{max}(o)$. Par la Proposition 5, nous avons $o \preceq w$ et $d(o, w) = |w| - |o| = r$. De plus, comme $v = (a_1 \ldots a_n)^k$ avec $k \geq |o| + r$, nous avons $w \preceq v$, donc par la Proposition 1, $d(w, v) = |v| - |w| = |v| - |o| - r = d(o, v) - r$. Comme $d(o, w) = r$ et $d(w, v) = d(o, v) - r$, nous en déduisons $B_r^{max}(o) \subseteq U$ par le Lemme 9.

Soit maintenant $u \in U$. Par le Lemme 9, nous avons $d(o, u) = r$. Si $o \preceq u$, alors $u \in B_r^{max}(o)$ par la Proposition 5. Si $o \not\preceq u$, alors par la Proposition 1, $d(o, u) > |u| - |o|$, c'est-à-dire, $|u| < |o| + r$. Mais alors, $d(u, v) \geq |v| - |u| > |v| - |o| - r = d(w, v)$ pour tout $w \in B_r^{max}(o)$, ce qui contredit $u \in U$. Donc $U \subseteq B_r^{max}(o)$. □

Demander la correction d'un mot de la forme $(a_1 \ldots a_n)^k$ avec k suffisamment grand permet donc d'être sûr d'avoir un mot de $B_r^{max}(o)$. Le problème reste donc de trouver un tel k. Pour cela, nous allons utiliser le lemme suivant qui stipule que si nous demandons à l'oracle de corriger un mot fait d'une grande quantité de a, la correction qu'on obtient nous donne des indications précieuses sur le rayon et le nombre d'occurrences de a dans le centre :

Lemme 12 *Considérons la boule $B_r(o)$, une lettre $a \in \Sigma$ et un entier $j \in \mathbb{N}$ tels que $a^j \notin B_r(o)$. Soit $w = Cq(a^j)$. Si $|w| < j$, alors $|w|_a = |o|_a + r$.*

Démonstration :
Soient $a^j \notin B_r(o)$ et $w = Cq(a^j)$. Par le Lemme 9, nous avons $d(o, w) = r$ et $d(w, a^j) = d(o, a^j) - r$. Comme $|w| < |a^j|$, le calcul de $d(w, a^j)$ consiste en (1) changer toutes les lettres de w qui ne sont pas des a, soit $|w| - |w|_a$ substitutions, et (2) compléter le mot obtenu par des a pour atteindre a^j, soit $j - |w|$ insertions de a. Nous en déduisons alors que $d(w, a^j) = |w| - |w|_a + j - |w| = j - |w|_a$. Calculons maintenant $d(o, a^j)$. Clairement, si $|o| \leq |w| < j$, alors nous pouvons utiliser les mêmes arguments que précédemment pour obtenir $d(o, a^j) = j - |o|_a$. Enfin, puisque $d(w, a^j) = d(o, a^j) - r$, nous en déduisons que $j - |w|_a = j - |o|_a - r$, c'est-à-dire, $|w|_a = |o|_a + r$.

Supposons maintenant que $|o| > |w|$. Le même argument ne tient plus puisque $|o| \geq |a^j|$. Des suppressions sont donc nécessaires pour calculer $d(o, a^j)$. Cependant, nous allons montrer que ce cas est impossible. En effet, considérons la dérivation $o \xrightarrow{r} w$. Puisque $|o| > |w|$, il y a au moins une suppression le long de cette dérivation. À la place de supprimer cette lettre, supposons qu'on la remplace par un a et que l'on ne change rien d'autre. Ceci nous amène à une nouvelle dérivation $o \xrightarrow{r} w'$ (ou $o \xrightarrow{r-1} w'$ si la lettre supprimée était un a) avec $|w'| = |w| + 1$ et $|w'|_a = |w|_a + 1$. De plus, $d(o, w') \leq r$, donc $w' \in B_r(o)$. Finalement, comme $|w| < j$, nous obtenons $|w'| \leq j$, donc, avec les mêmes arguments que précédemment, seules les substitutions et les insertions sont nécessaires pour calculer $d(w', a^j)$. Plus précisément, nous avons $d(w', a^j) = (|w'| - |w'|_a) + (j - |w'|) = -|w|_a - 1 + j = d(w, a^j) - 1$, et donc $d(w', a^j) < d(w, a^j)$. Comme $w' \in B_r(o)$, w ne peut être une correction de a^j. □

Demander la correction d'un mot de la forme a^j avec j grand permettra donc d'obtenir une correction dont nous pouvons tirer de l'information. Supposons en effet que l'alphabet soit $\Sigma = \{a_1, \ldots, a_n\}$ et soit $j_1, \ldots, j_n \in \mathbb{N}$ de grand entiers. Si nous définissons $k = \sum_{i=1}^{n} |Cq(a_i^{j_i})|_{a_i}$, alors le Lemme 12 donne $k = \sum_{i=1}^{n} (|o|_{a_i} + r) = |o| + |\Sigma| \cdot r \geq |o| + r$. Nous pouvons alors utiliser cette valeur k dans la Proposition 11 et obtenir un mot $w = Cq\left((a_1 \ldots a_n)^k\right) \in B_r^{max}(o)$. De plus, nous avons $|w| = |o| + r$ et $k = |o| + |\Sigma| \cdot r$. Donc, nous en déduisons que le rayon est $r = (k - |w|)/(|\Sigma| - 1)$.

6.1.5 Un algorithme d'identification des boules à partir de requêtes de correction

À ce point, nous pouvons donc résumer notre stratégie d'apprentissage de la boule $B_r(o)$ avec un alphabet $\Sigma = \{a_1, \ldots, a_n\}$ de la façon suivante :

1. Pour chaque lettre a_i, l'apprenant demande la correction de a_i^j avec j suffisamment grand pour que la correction soit plus petite que j.
2. Il calcule alors $k = \sum_{i=1}^{n} |Cq(a_i^{j_i})|_{a_i}$ et demande la correction w du mot $v = (a_1 \ldots a_n)^k$.
3. À partir de k et w, il déduit r.
4. Il utilise ensuite EXTRACTION_DU_CENTRE avec w et r afin d'obtenir o.

L'apprenant est donc capable d'identifier des boules à partir de requêtes de correction en un nombre polynomial de requêtes (voir l'Algorithme 7 IDENTIFICATION_BOULE et le Théorème 33).

Algorithme 7 : IDENTIFICATION_BOULE

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

```
1  j ← 1;
2  k ← 0;
3  pour i = 1 à n faire
4  |  tant que Cq(a_i^j) = OUI ou |Cq(a_i^j)| ≥ j faire
5  |  |  j ← 2 · j;
6  |  fintq
7  |  k ← k + |Cq(a_i^j)|_{a_i};
8  finpour
9  w ← Cq((a_1 a_2 ... a_n)^k);
10 r ← (k − |w|)/(|Σ| − 1);
11 o ← EXTRACTION_DU_CENTRE(w, r);
12 retourner (o, r)
```

Voici un exemple de l'exécution de l'algorithme :

Exemple 44 *Considérons la boule $B_2(bb)$ sur l'alphabet $\Sigma = \{a, b\}$. Notre algorithme commence par chercher des corrections de a^j et b^j avec j suffisamment grand. Nous pouvons par exemple observer : $Cq(a) = \text{OUI}, Cq(a^2) = \text{OUI}, Cq(a^4) = aabb, Cq(a^8) = abba, Cq(b^8) = bbbb$. D'où $k = |abba|_a + |bbbb|_b = 2 + 4 = 6$.*

Ensuite, $Cq\left((ab)^6\right) = Cq(abababababab) = abba$, par exemple, et nous pouvons donc en déduire $r = (6-4)/(2-1) = 2$. Enfin EXTRACTION_DU_CENTRE$(abba, 2)$ *retourne bb. Donc* IDENTIFICATION_BOULE *retourne $(bb, 2)$.*

Théorème 33 *Pour n'importe quelle boule $B_r(o)$,* IDENTIFICATION_BOULE *retourne sa représentation (o, r) en utilisant $\mathcal{O}(|\Sigma| + |o| + r)$ requêtes de correction.*

Démonstration :
La correction de l'algorithme IDENTIFICATION_BOULE découle de la suite de lemme de cette section. Concernant la polynomialité de l'algorithme, les requêtes interviennent à trois endroits :

1. les corrections des mots a_i^j (ligne 4) sont au nombre de $\mathcal{O}(|\Sigma| + \log(|o| + r))$ (soit $\mathcal{O}(\log(|o| + r))$ pour avoir la plus grande correction plus une requête par lettre, soit $|\Sigma|$ CQ_{EDIT} supplémentaires) ;
2. la correction du mot $(a_1 a_2 \ldots a_n)^k$ (ligne 9) ;
3. les corrections utilisées par l'algorithme EXTRACTION_DU_CENTRE (ligne 11) pour retrouver le centre, soit d'après la Proposition 10 $\mathcal{O}(|o| + r)$ CQ_{EDIT}.

Au total, l'algorithme a alors besoin de $\mathcal{O}(|\Sigma| + |o| + r)$ requêtes de correction. □

Il en résulte le corollaire :

Corollaire 2

- *Soit $q()$ un polynôme. L'ensemble de toutes les $q()$-bonnes boules $B_r(o)$ est identifiable avec un algorithme qui utilise $\mathcal{O}(|\Sigma| + |o| + q(|o|))$ requêtes de correction et un temps polynomial.*
- *L'ensemble des bonnes boules $B_r(o)$ est identifiable avec un algorithme qui utilise $\mathcal{O}(|\Sigma| + |o| + r)$ requêtes de correction et un temps polynomial.*

Contrairement aux AFD les boules sont donc apprenables en utilisant des CQ_{EDIT}. Ce résultat est somme toute mitigé. D'une part, nous ne savons pas montrer que la complexité obtenue est une borne inférieure. D'autre part, si l'on reprend l'exemple du plan, l'apprentissage de boules dans $\mathbb{R}^2$ est de complexité logarithmique. En fait, la complexité des boules de mots est telle qu'il est difficile de pouvoir utiliser à bon escient l'information contenue dans une correction (contrairement à $\mathbb{R}^2$).

De ce fait, nous pouvons aussi construire un algorithme qui, à partir d'un exemple (c'est-à-dire d'un mot de la boule), identifie la boule avec des MQ seulement ! En effet, une fois que l'on a obtenu un mot de la frontière supérieure de la boule, l'algorithme IDENTIFICATION_BOULE n'utilise les CQ_{EDIT} que pour tester l'appartenance du nouveau mot à la boule. Il s'en sert donc comme de MQ. Ainsi, il suffit de partir d'un mot quelconque de la boule et de faire le plus d'insertions possibles de toutes les lettres de l'alphabet entre chaque lettre de l'exemple pour pouvoir construire un mot de la frontière supérieure de la boule[3]. Ensuite, à partir de ce mot, il suffit d'utiliser une modification de l'algorithme EXTRACTION_DU_CENTRE pour trouver les mot $a^r o$ et $b^r o$ (comme nous ne connaissons pas encore r). Nous en déduisons alors facilement (o, r).

La complexité de ce nouvel algorithme est alors polynomiale et non plus linéaire, mais cela montre que les CQ_{EDIT} ne sont peut-être pas utilisées à leur « juste » valeur, contrairement à l'exemple de l'apprentissage dans le plan. De plus, si comme souvent en pratique c'est un humain qui joue le rôle de l'oracle, une complexité polynomiale peut alors être trop coûteuse. Nous allons donc essayer dans la section suivante de réduire cette complexité.

Au préalable nous allons nous intéresser aux mauvaises boules dont la situation est moins claire. En effet, prenons la classe contenant toutes les boules ayant pour centre le mot vide : $\{B_n(\lambda) : n \in \mathbb{N}\}$. Afin de pouvoir identifier une boule, il est nécessaire de faire une requête en dehors de la boule. La requête ainsi que sa correction sont alors des mots dont les longueurs sont supérieures ou égales à $|o| + r$, c'est-à-dire supérieures ou égales à r. La taille de la représentation de ces boules étant $|o| + \log r = \log r$, ces mots ont donc des tailles exponentielles en $\log r$...

[3] Techniquement, soit w un mot de la boule. Si l'on ne peut insérer aucune lettre à w (et ce quelque soit la position de l'insertion), alors nous pouvons montrer que $w \in \mathcal{B}_r^{max}(o)$. De plus, n'importe quel mot de la boule est un sous-mot d'au moins un mot de $\mathcal{B}_r^{max}(o)$. De ce fait, si $w \notin \mathcal{B}_r^{max}(o)$ alors il existe bien une position dans le mot telle que l'on peut faire une insertion et obtenir un mot w' plus long et plus proche de $\mathcal{B}_r^{max}(o)$ que de w.

Nous pouvons alors distinguer deux cas : soit nous autorisons le fait que le nombre de requêtes de correction autorisées puisse dépendre de la plus longue correction donnée par l'oracle, auquel cas les boules sont identifiables, soit nous ne l'autorisons pas, et alors elles ne le sont pas.

6.2 Apprentissage des boules avec un nombre logarithmique de requêtes de correction

Si nous regardons attentivement IDENTIFICATION_BOULE, nous constatons que trouver un mot de $B_r^{max}(o)$ se fait en $\mathcal{O}(|\Sigma| + \log(|o| + r))$ requêtes de correction alors que l'algorithme général est en $\mathcal{O}(|\Sigma| + |o| + r)$. L'utilisation de l'algorithme EXTRACTION_DU_CENTRE nous fait alors perdre cette complexité logarithmique. Afin de pouvoir apprendre les boules avec un nombre logarithmique de CQ_{EDIT}, il faut donc se passer de cette partie de l'apprentissage. Cela peut être fait dans un contexte légèrement différent : en utilisant des poids différents pour les opérations d'édition. De plus, nous supposerons que l'alphabet est de taille au moins 3 pour des raisons que nous verrons plus tard.

Jusqu'à présent, nous avons utilisé la distance d'édition telle que définie par Levenshtein[4] [Lev65]. Toutefois, en pratique, certains travaux utilisent des poids différents pour chaque opération d'édition. Dans ce cadre, chaque dérivation $w \xrightarrow{k} w'$ a un poids qui est la somme des poids des opérations d'éditions effectuées le long de la dérivation. La *distance d'édition pondérée* est alors le poids minimum des différentes dérivations transformant w en w'.

Chaque combinaison de poids impliquera alors un algorithme différent lors du recours aux requêtes de correction. Notre but étant de montrer que l'on peut réduire le nombre de CQ_{EDIT} utilisées, nous supposerons que :

1. le poids de chaque insertion et de chaque suppression vaut 1 (comme avant),
2. le poids de chaque substitution vaut un nombre irrationnel ρ compris strictement entre 0 et 1.

Par exemple, le poids de la substitution peut être $\rho = \frac{\pi}{4}$ ou $\rho = \frac{\sqrt{2}}{3}$.

Il est intéressant de noter qu'un faible coût pour l'opération de substitution est fréquent en linguistique. Par exemple, des travaux en phonologie font cette hypothèse dans le but de forcer les alignements de *segments* phonétiquement similaires [AH03].

Quoiqu'il en soit, le fait que le poids ne soit pas un nombre rationnel peut sembler déroutant. Nous verrons qu'en fait, du point de vue de l'apprenant, il n'est pas nécessaire de calculer des distances pondérées (contrairement à l'oracle). Le fait que ρ ne soit pas une fraction n'est donc pas un problème.

De plus, la Proposition 1 est toujours vraie[5] grâce aux poids des insertions et suppressions qui valent toujours 1. Enfin, le fait d'avoir un ρ irrationnel nous permet d'établir

[4] Nous appellerons cette distance la distance d'édition *standard*

[5] Même avec ce poids irrationnel, nous avons toujours $d(w, w') \geq ||w| - |w'||$ et $d(w, w') = ||w| - |w'||$ si et seulement si ($w \preceq w'$ ou $w' \preceq w$)

de fortes propriétés :

Proposition 6 *Soient $o, w, w' \in \Sigma^*$ trois chaînes. Les affirmations suivantes sont équivalentes :*

1. *Il existe une dérivation de w vers w' de poids minimum qui utilise $x \in \mathbb{N}$ insertions et suppressions, et $y \in \mathbb{N}$ substitutions.*
2. *$d(w, w') = x + \rho y$.*
3. *Toutes les dérivations de w vers w' de poids minimum utilisent $x \in \mathbb{N}$ insertions et suppressions, et $y \in \mathbb{N}$ substitutions.*

En conséquence, si $d(o, w) = d(o, w')$, alors toutes les dérivations de o vers w et de o vers w' utilisent le même nombre d'insertions et suppressions, ainsi que le même nombre de substitutions.

Démonstration :

3. $\Longrightarrow$ 1. Trivial.
1. $\Longrightarrow$ 2. Puisque les poids des insertions et suppressions valent 1, que le poids des substitutions vaut ρ, et que la dérivation est de poids minimum, nous avons $d(w, w') = x + \rho y$.
2. $\Longrightarrow$ 3. Considérons une autre dérivation de w vers w' de poids minimum qui utilise $x' \in \mathbb{N}$ insertions et suppressions, et $y' \in \mathbb{N}$ substitutions. Alors nous avons $d(w, w') = x' + \rho y' = x + \rho y$, donc $x - x' = \rho(y' - y)$. Comme ρ est irrationnel et x, x', y, y' sont des entiers, nécessairement $y' - y = 0$ et $x - x' = 0$, donc $x' = x$ et $y' = y$.

□

Cette propriété ne serait plus vraie si ρ était un nombre rationnel. Par exemple, si $\rho = \frac{1}{2}$ alors deux substitutions coûteraient autant qu'une insertion. Cela pourrait alors conduire à deux dérivations de même poids mais dont le nombre d'opérations est différent.

6.2.1 Les nouvelles boules et corrections

Assez logiquement, changer la distance d'édition change aussi les boules. Par exemple, en utilisant la distance d'édition standard, nous avons $B_1(abb) = \{ab, bb, aab, aba, abb, bbb, aabb, abab, abba, babb, abbb\}$. Mais l'utilisation de la distance pondérée avec $\rho = \frac{\sqrt{2}}{4} \simeq 0,3536$ ajoute les mots $\{aaa, bab, bba\}$ à la boule. Nous pouvons alors également nous demander s'il est pertinent de conserver un entier pour définir le rayon des boules. La réponse est oui : en effet, si tel n'est pas le cas, les boules $B_r(o)$ et $B_{r+\frac{\rho}{2}}(o)$ représenteraient le même ensemble de mots. En d'autres termes, le Théorème 14, qui montre l'unicité de la représentation, ne serait plus valide. À l'inverse, si l'on garde un rayon entier alors, comme la Proposition 1 continue d'être vraie, le Théorème 14 continue lui aussi d'être valide.

En ce qui concerne les corrections, leurs propriétés deviennent tortueuses à cause de la pondération. En particulier, le Lemme 9 affirme que l'ensemble des corrections

possibles de n'importe quel mot $v \notin B_r(o)$ est exactement $\{u \in \Sigma^* : d(o,u) = r \text{ and } d(u,v) = d(o,v) - r\}$. Ce résultat n'est à présent plus vrai en utilisant la distance d'édition pondérée. En effet, considérons la boule $B_1(abb)$ avec $\rho = \frac{\sqrt{2}}{4}$. Alors la correction u du mot baa est dans $\{aaa, bab, bba\}$. nous avons alors $d(abb,u) = 2\rho < 1$ et $d(u,baa) = \rho > d(abb,baa) - 1 = 3\rho - 1$. En d'autres termes, une correction n'est plus nécessairement sur le cercle qui délimite la boule.

Néanmoins, nous pouvons montrer qu'il existe une caractérisation plus complexe des corrections possibles :

Lemme 13 *Soient la boule $B_r(o)$ et un mot $v \notin B_r(o)$. Étant donné n'importe quel $\alpha \in \mathbb{R}$, nous définissons*

$$C_\alpha = \{u \in \Sigma^* : d(o,u) = \alpha \text{ et } d(u,v) = d(o,v) - \alpha\}.$$

Tous les C_α non vides définissent des « arcs de cercles » de mots concentriques autour du centre o paramétrés par α. Soit α_0 le rayon du plus grand arc de cercle inclus dans la boule :

$$\alpha_0 = \max_{0 \leq \alpha \leq r} \{\alpha : C_\alpha \neq \emptyset\}.$$

L'ensemble des corrections de v est alors exactement C_{α_0}.

Démonstration :
La preuve est la même que celle du Lemme 9, sauf que W est remplacé par C_{α_0} et r est remplacé par α_0. Le point crucial est que W peut être vide avec la distance d'édition pondérée alors que C_{α_0} ne peut pas, par définition. □

6.2.2 Les mots de longueur maximum

Intéressons-nous à présent à nouveau à l'apprentissage des boules. Comme dans la Section 6.1, nous allons étudier les mots de longueur maximum de $B_r(o)$ puisqu'ils contiennent beaucoup d'informations. En effet, nous allons montrer que comme pour le Lemme 12, si l'on demande la correction w d'un mot fait de beaucoup de a, alors $|w|_a = |o|_a + r$. En plus, dans ce nouveau contexte, nous obtenons directement $w \in B_r^{max}(o)$. Toutefois, il nous faut connaître le polynôme $q()$ pour lequel $B_r(o)$ est une bonne boule.

Lemme 14 *Soit $q()$ un polynôme. Considérons la $q()$-bonne boule $B_r(o)$, une lettre $a \in \Sigma$ et un entier $j \in \mathbb{N}$ tels que $a^j \notin B_r(o)$. Soient $u = Cq(a^j)$ et $v = Cq(a^{j+q(j)})$. Si $|u| < j$, alors $v \in B_r^{max}(o)$ et $|v|_a = |o|_a + r$.*

Cette sous-section vise à prouver ce lemme en utilisant deux propositions intermédiaires :

Proposition 7 *Considérons la boule $B_r(o)$, une lettre $a \in \Sigma$ et un entier $j \in \mathbb{N}$ tels que $a^j \notin B_r(o)$. Soit $u = Cq(a^j)$. Si $|u| < j$, alors $|o| < j$.*

Démonstration :
Montrons que $|o| \leq |u|$. Comme $|u| < j$, le résultat sera immédiat. Supposons alors que $|u| < |o|$ et considérons une dérivation $o \xrightarrow{k} u$ de poids minimum $d(o,u) = x + \rho y$. Comme $|o| > |u|$, il existe au moins une suppression le long de cette dérivation. Supposons qu'au lieu de supprimer cette lettre, nous la changeons par a et que le reste de la dérivation soit inchangé. Cela conduit alors à une nouvelle dérivation $o \xrightarrow{k} u'$ (ou $o \xrightarrow{k-1} u'$ si la lettre supprimée est un a) avec $|u'| = |u| + 1$ et $|u'|_a = |u|_a + 1$. De plus, $d(o,u') \leq (x-1) + \rho(y+1) = d(o,u) - 1 + \rho$. Comme $d(o,u) \leq r$, nous en déduisons que $d(o,u') < r$, ainsi $u' \in B_r(o)$. Finalement, comme $|u| < j$, nous avons $|u'| \leq j$, donc seulement des substitutions et des insertions sont nécessaires pour calculer $d(u,a^j)$ et $d(u',a^j)$. Plus précisément, nous avons $d(u',a^j) = (j - |u'|) + \rho(|u'| - |u'|_a) = (j - |u| - 1) + \rho(|u| - |u|_a) = d(u,a^j) - 1$, et donc $d(u',a^j) < d(u,a^j)$. Comme $u' \in B_r(o)$, u ne peut pas être une correction a^j, ce qui contredit l'énoncé du lemme. Nous avons donc $|u| \geq |o|$, et donc $|o| < j$. □

Proposition 8 *Considérons la boule $B_r(o)$, une lettre $a \in \Sigma$ et un entier $\ell \in \mathbb{N}$ tels que $a^\ell \notin B_r(o)$. Soit $v = Cq(a^\ell)$. Si $|o| + r < \ell$, alors $v \in B_r^{max}(o)$ et $|v|_a = |o|_a + r$.*

Démonstration :
Comme $|o| + r < \ell$, nous avons $|o| < \ell$, donc le calcul de $d(o,a^\ell)$ utilise nécessairement $\ell - |o|$ insertions de a ainsi que $|o| - |o|_a$ substitutions par des a. Définissons une dérivation de référence de o vers a^ℓ, où les $\ell - |o|$ insertions sont effectuées en premier, à l'avant de o, puis les $|o| - |o|_a$ substitutions par des a dans o : $o \xrightarrow{\ell-|o|} a^{\ell-|o|}o \xrightarrow{|o|-|o|_a} a^{\ell-|o|}a^{|o|} = a^\ell$. Comme $\ell - |o| > r$, le mot $a^{\ell-|o|}o$ n'est pas dans la boule, donc la dérivation passe par le mot $a^r o$ avant d'atteindre $a^{\ell-|o|}o$. En d'autres termes, la dérivation de référence ressemble à : $o \xrightarrow{r} a^r o \xrightarrow{\ell-|o|-r} a^{\ell-|o|}o \xrightarrow{|o|-|o|_a} a^\ell$. Considérons maintenant le Lemme 13. Comme $d(o,a^r o) = r$ et $d(a^r o, a^\ell) = \ell - |o| - r + \rho(|o| - |o|_a) = d(o,a^\ell) - r$, nous avons $C_r \neq \emptyset$. Donc, comme $v = Cq(a^\ell)$, nous en déduisons que $d(o,v) = r$ et $d(v,a^\ell) = d(o,a^\ell) - r$. Nous affirmons que seulement des insertions de a peuvent apparaître le long de la dérivation $o \xrightarrow{r} v$. En effet, nous avons $d(o,v) = r \in \mathbb{N}$, donc par la Proposition 6, aucune substitution n'apparaît. De plus, aucune suppression n'apparaît non plus puisque n'importe quelle dérivation minimale de o vers a^ℓ utilise seulement des insertions de a et des substitutions par des a. En conséquence, $v \in B_r^{max}(o)$ et $|v|_a = |o|_a + r$. □

Démonstration :
[du Lemme 14] Par la Proposition 7, nous avons $|o| < j$. Ainsi, $|o| + r \leq |o| + q(|o|)$. De plus, comme $|o| < j$ et tous les coefficients de $q()$ sont entiers, nous en déduisons que $|o| + r < j + q(j)$. Donc en remplaçant ℓ par $j + q(j)$ dans la Proposition 8 nous obtenons le résultat. □

6.2.3 Apprendre les boules logarithmiquement

En conséquence du Lemme 14, la correction d'un long mot composé de a conduit à un mot de $B_r^{max}(o)$. Mais plus intéressant, supposons que l'alphabet contienne au moins trois lettres, par exemple $a, b, c, \ldots$ Alors soient $u_a = Cq(a^j)$ avec $|u_a| < j$, et $v_a = Cq(a^{j+q(j)})$. Grâce au Lemme 14, v_a est obtenu à partir de o avec r insertions de a. Ainsi, toutes les lettres dans v_a, mises à part les occurrences de a, sont les lettres de o et apparaissent dans le bon ordre.

Plus formellement, soit E_a la fonction qui efface toutes les occurrences de la lettre $a \in \Sigma$ dans un mot :

1. $E_a(\lambda) = \lambda$,
2. $E_a(a.z) = E_a(z)$, et
3. $E_a(b.z) = b.E_a(z)$, pour tout $b \neq a$.

Alors, pour chaque lettre $a \in \Sigma$, $E_a(v_a) = E_a(o)$.

Exemple 45 *Considérons la boule $B_1(o)$ avec $o = badac$. Si les corrections des mots a^ℓ, b^ℓ et c^ℓ (avec ℓ suffisamment grand) sont $v_a = badaca$, $v_b = babdac$ et $v_c = badcac$, alors $E_a(v_a) = E_a(o) = bdc$, $E_b(v_b) = E_b(o) = adac$ et $E_c(v_c) = E_c(o) = bada$.*

Par conséquent, nous pouvons facilement déduire o en *alignant* les mots $E_a(o)$, $E_b(o)$ and $E_c(o)$:

Exemple 46 *En reprenant la même boule ainsi que les mêmes corrections qu'à l'exemple précédent, nous pouvons retrouver le centre de la boule :*

$E_a(o)$	b	$\cdot$	d	$\cdot$	c
$E_b(o)$	$\cdot$	a	d	a	c
$E_c(o)$	b	a	d	a	$\cdot$
o	b	a	d	a	c

Cette procédure n'utilise pas de nouvelle requête de correction et est exécutée en temps $\mathcal{O}(|o|)$ ce qui est bien plus efficace que EXTRACTION_DU_CENTRE. Notons que si $|\Sigma| > 3$, seulement trois corrections sont nécessaires comme le montre l'exemple. Nous obtenons finalement l'Algorithme 8 et le Théorème 34.

Théorème 34 *Supposons $|\Sigma| \geq 3$.*

- *Soit $q()$ un polynôme à coefficients dans $\mathbb{N}$. La classe de toutes les $q()$-bonnes boules $B_r(o)$ est identifiable avec un algorithme qui utilise $\mathcal{O}(\log(|o| + q(|o|)))$ requêtes de correction et un temps polynomial.*
- *La classe $\mathcal{BB}(\Sigma)$ est identifiable avec un nombre logarithmique $\mathcal{O}(\log |o|)$ de requêtes de correction et un temps polynomial.*

En conclusion, si le poids des substitution est un irrationnel inférieur à 1 alors nous pouvons réduire considérablement la complexité de l'apprentissage. Nous allons montrer maintenant que l'algorithme d'identification que nous avons utilisé résiste assez bien au bruit dans le sens où une simple heuristique *a posteriori* suffit à retrouver les boules cibles.

Algorithme 8 : IDENTIFICATION_BOULE_PONDÉRÉE

Données : L'alphabet $\Sigma = \{a_1, \ldots, a_n\}$ avec $n \geq 3$, et le polynôme $q()$
Résultat : La représentation (o, r) de la $q()$-bonne boule cible $B_r(o)$

$j \leftarrow 1$;
pour $i = 1$ **à** 3 **faire**
  tant que $Cq(a_i^j) =$ OUI ***ou*** $|Cq(a_i^j)| \geq j$ **faire**
    $j \leftarrow 2 \cdot j$;
  fintq
  $v_i \leftarrow Cq\left(a_i^{j+q(j)}\right)$;
  $e_i \leftarrow E_{a_i}(v_i)$;
finpour
$o \leftarrow$ ALIGNE(e_1, e_2, e_3);
$r \leftarrow |v_1| - |o|$;
retourner (o, r)

6.3 Apprentissage face à un oracle faillible

Nous avons fait dans les sections précédentes une hypothèse forte concernant le modèle de l'apprentissage actif. En effet, nous avons supposé que l'oracle était un oracle parfait. Cela correspond à une supposition tout à fait raisonnable lorsque l'on ne fait que des mathématiques et que l'on a pour objectif d'être dans une configuration favorable dans laquelle les résultats négatifs résultent de la complexité de la tâche et non de la nature de l'oracle.

Cependant, récemment, le cadre de l'apprentissage actif a été accepté comme un cadre plausible pour de vraies applications. En effet, nous faisons face à de plus en plus de données non étiquetées. Choisir quelles données doivent recevoir l'attention d'un expert (qu'il soit un humain ou une machine) est une question difficile. Les sessions d'apprentissage interactif, où l'algorithme d'apprentissage demande des informations spécifiques pendant son exécution, sont des alternatives intéressantes pour traiter de tels problèmes.

Un exemple typique est le système SQUIRREL [CGLN07] qui induit un *wrapper web* à travers des interactions avec un utilisateur humain. Un autre cas est celui des tests *hardware* [HHNS02] : les spécifications du logiciel correspondent à l'oracle, lequel peut alors autoriser le contrôle si les items construits obéissent aux spécifications. Dans les deux cas l'oracle est faillible : dans le second parce que l'équivalence est faite par un échantillonnage.

Enfin, une troisième situation dans laquelle l'apprentissage actif peut être utile correspond à celle de rendre intelligible une boîte noire apprise par une méthode d'apprentissage automatique statistique. En effet, même si les hyper-plans [CFWS06] ou les réseaux de neurones récurrents [GLT01] sont difficiles à interpréter, nous pouvons essayer d'utiliser les modèles numériques appris comme des oracles dans un algorithme d'apprentissage actif dont les résultats seraient des classifieurs basés sur des règles [dlH06b].

6.3.1 Face à un oracle faillible

L'algorithme IDENTIFICATION_BOULE a été construit pour être utilisé dans des conditions idéales, c'est-à-dire dans le cas où nous supposions que l'oracle était une machine parfaite : ses réponses étaient tellement précises que nous avions pu scrupuleusement les caractériser (voir Lemme 9). Cependant, comme nous venons de le dire, en pratique un oracle est souvent un expert, c'est-à-dire un être humain, ou est simulé par un échantillonnage. Dans de pareils cas, la supposée perfection n'est plus vraie. En effet, calculer la correction de $(bab)^{127}$ pour la boule $B_{217}((babb)^{32})$ est probablement hors de capacités cognitives de tout être humain. Notre algorithme ne devrait donc pas croire aveuglément les réponses qu'il obtient puisqu'elle peuvent être approximatives. Dans cette section, nous allons montrer à travers une série d'expérimentations que notre algorithme résiste à des réponses approchées (c'est-à-dire imprécises, bruitées).

6.3.2 Définition de l'oracle approximatif

Nous allons construire maintenant un oracle approximatif qui puisse ressembler à un être humain. Considérons un mot w et une boule $B_r(o)$. Soit $Cq_h(w)$ la réponse de l'oracle approximatif à la requête ; et $Cq(w)$ la réponse de celle qui serait retournée par un oracle parfait (c'est donc la même réponse que faite au chapitre précédent).

Premièrement, nous supposerons qu'un expert peut facilement déterminer si un exemple satisfait un concept ou non. Ainsi, ici, il sait si w appartient à $B_r(o)$ ou pas. Nous supposerons donc que si $Cq(w) =$ OUI, alors $Cq_h(w) =$ OUI. Deuxièmement, ce qui est vraiment dur pour l'expert est de *calculer* la meilleure correction de w quand le mot n'appartient pas à la boule, et plus précisément, un mot qui est *aussi près de w que possible*. Une fois encore, $Cq_h(w)$ sera certainement à l'intérieur de la boule plutôt qu'à la frontière.

Prenons maintenant un peu de recul. Soit $X = d(w, Cq_h(w)) - d(w, Cq(w))$ une mesure de la distance entre une correction approximative et une correction parfaite. Intuitivement un *bon* oracle approximatif ne fera que peu de mauvaises corrections. Ainsi, elles seront souvent telles que $X = 0$, quelques fois telles que $X = 1$, moins fréquemment telles que $X = 2$,... Pour formaliser cette idée, nous introduisons un paramètre de confiance $0 < p \leq 1$, appelé *niveau de pertinence de l'oracle*, qui traduit la qualité des réponses de l'oracle, et qui utilise une distribution géométrique :

$$Pr(X = k) = (1 - p)^k p, \forall k \in \mathbb{N}.$$

Par conséquent, avec une probabilité $(1-p)^k p$, la correction $Cq_h(w)$ d'un mot w sera dans la boule cible et à distance[6] k de $Cq(w)$. La distribution géométrique permet aussi de connaître l'espérance de la distance entre les corrections des deux types d'oracle : $E(X) = (1/p) - 1$. Donc lorsque l'oracle est plutôt précis, disons $p = 0,8$, alors la distance moyenne entre l'oracle approximatif et un oracle parfait est faible : $E(X) = 0,25$. À l'inverse, un expert avec des capacités de calcul limitées, qui aurait un niveau

[6]Si aucun mot de la boule n'est à distance k (c'est le cas si k>2r) alors la correction est un des mots le plus loin de la vraie correction.

de pertinence de $p = 0,4$, donnera souvent des corrections imprécises : en moyenne à distance $1,5$ de la vraie correction. La Figure 6.2 schématise les corrections que peut donner un oracle approximatif.

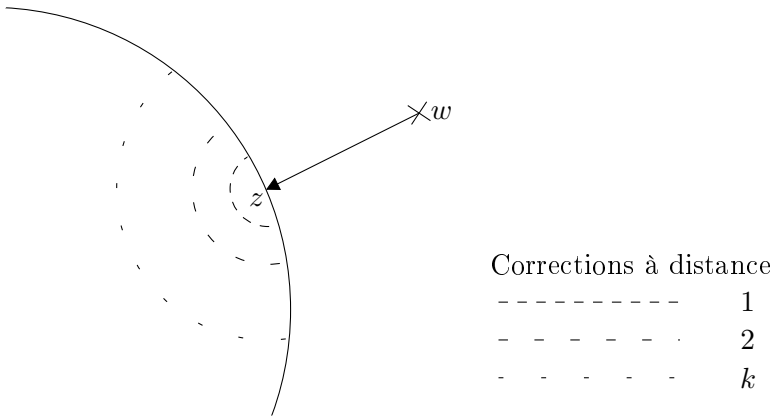

FIG. 6.2 – Réponses que peut donner un oracle approximatif à une requête w hors d'une boule : avec probabilité p la correction z sera la bonne. Avec probabilité $(1-p)p$ elle sera à distance 1, avec probabilité $(1-p)^k p$ elle sera à distance k.

Notre modèle d'oracle approximatif est simple : nous ne supposons pas par exemple que l'oracle dispose d'une mémoire. Ainsi, en soumettant chaque requête deux fois de suite, nous pourrions obtenir deux corrections. Nos requêtes n'étant pas persistantes, nous aurions alors la possibilité de pouvoir corriger la correction ! Quoiqu'il en soit, notre but n'est pas ici de construire le meilleur algorithme d'apprentissage possible, mais d'étudier le comportement de IDENTIFICATION_BOULE et plus particulièrement sa résistance face à des réponses approximatives. De ce point de vue, notre modèle, bien que basique, est largement suffisant.

6.3.3 Comportement de l'algorithme face à un oracle approximatif

Suivant le Théorème 33, IDENTIFICATION_BOULE devine systématiquement les boules cibles avec l'aide d'un oracle parfait. Bien entendu, il échouera en face d'un oracle approximatif. Dans le but d'évaluer la résistance d'IDENTIFICATION_BOULE aux corrections approximatives, nous allons conduire l'expérience suivante. Nous tirons au hasard un ensemble de cent boules $B_r(o)$ telles que $|o|+r = 200$. Plus précisément, nous faisons varier le rayon r de 10 à 190 par pas de 20. Pour chaque rayon, nous tirons aléatoirement 10 centres o de longueur $200 - r$. Ensuite, pour chaque niveau de pertinence $0,5 \leq p \leq 1$, nous demandons à IDENTIFICATION_BOULE d'apprendre ces boules, et nous calculons le pourcentage des boules qu'il a été capable de retrouver, pourcentage que nous appellerons la *précision de l'algorithme*. Nous montrons le résultat dans la

Figure 6.3. Nous pouvons remarquer que IDENTIFICATION_BOULE est capable d'identifier environ 75% des boules face à un niveau de pertinence de $p = 0,9$. En revanche, comme nous pouvions nous y attendre, avec des niveaux de pertinence plus bas, ses performances chutent considérablement (15% pour $p = 0,5$).

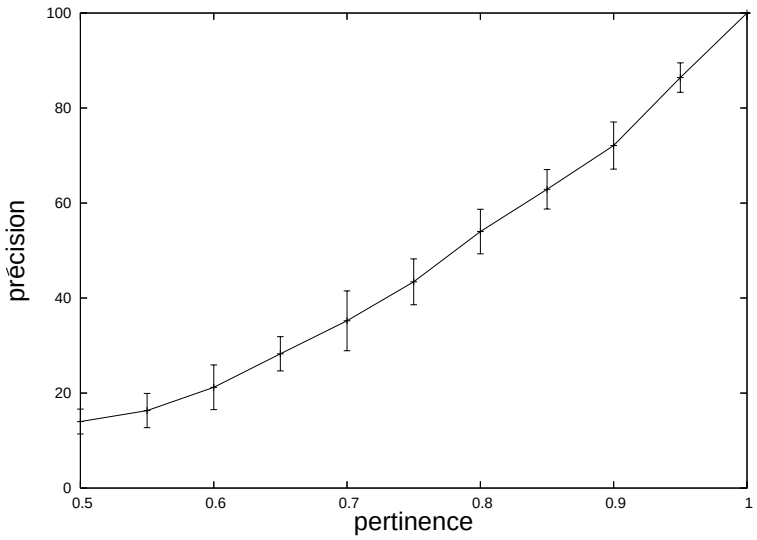

FIG. 6.3 – Précision de IDENTIFICATION_BOULE face à un oracle approximatif en fonction du niveau de pertinence p. Chaque point est une moyenne calculée sur 100 boules.

Nous montrons aussi dans la Figure 6.4 la distance moyenne entre les centres des boules cibles et les centres des boules que l'algorithme a « mal » appris. Nous pouvons alors constater que ces distances ne sont pas si importantes : même avec une pertinence de $p = 0,5$, la distance est inférieure à 3. La dernière courbe, Figure 6.5, montre la différence entre les rayons des boules cibles et des boules apprises. La courbe est grossièrement identique à la précédente.

6.3.4 Amélioration de la précision grâce à des heuristiques *a posteriori*

Nous avons vu que IDENTIFICATION_BOULE était capable d'assimiler les approximations de l'Oracle jusqu'à un certain niveau de pertinence. De plus, les centres et les rayons retournés par l'algorithme sont en général assez proches de ceux de la cible. Il est donc raisonnable de penser que l'on peut améliorer la précision de l'algorithme en explorant le voisinage du centre appris par des modifications d'édition locales. Ce genre d'approche a été fait une des premières fois dans [Koh85], et une étude approfondie peut être trouvée dans [MHJC00].

Supposons que la boule apprise soit $B_k(u)$ et soit $\mathcal{Q}$ l'ensemble de toutes les corrections retournées par l'oracle durant le processus de IDENTIFICATION_BOULE. L'heuristique est composée de deux étapes :

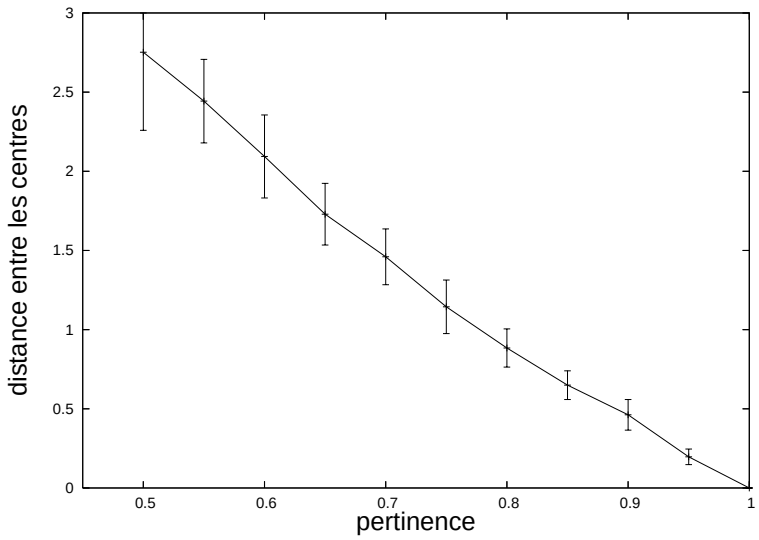

FIG. 6.4 – Distance moyenne (et écart-type) entre les centres des boules cibles et les centres des boules apprises par IDENTIFICATION_BOULE.

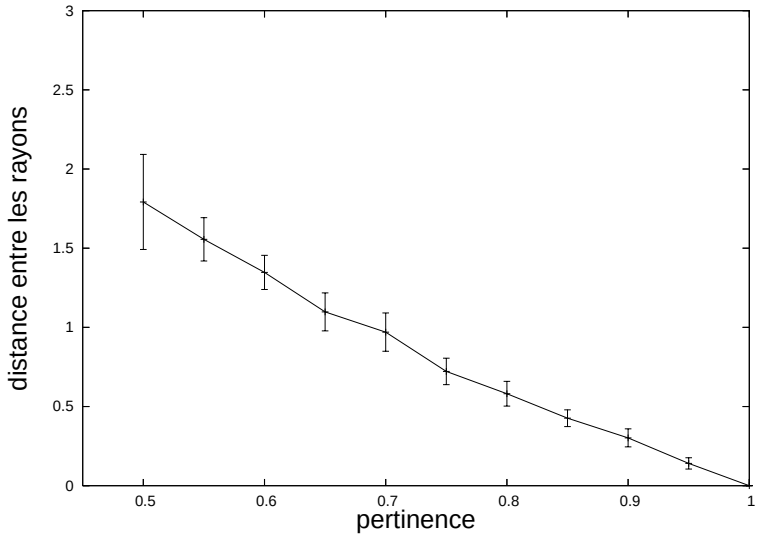

FIG. 6.5 – Différence moyenne (et écart-type) entre les rayons des boules cibles et les rayons des boules apprises par IDENTIFICATION_BOULE.

1. Nous testons chaque voisin u' (à distance 1) de u, le centre appris, et nous regardons si c'est un meilleur centre par rapport à $\mathcal{Q}$, c'est-à-dire, s'il existe $k' \in \mathbb{N}$ tel que $k' < k$ et $\mathcal{Q} \subseteq B_{k'}(u')$. En d'autres termes, nous cherchons si un des voisins de u peut être le centre d'une boule plus petite, contenant les corrections de $\mathcal{Q}$. Nous gardons alors les représentations (u', k') des plus petites boules.
2. À partir de cet ensemble, nous gardons les paires (u', k') qui maximisent le nombre de corrections (de $\mathcal{Q}$) à distance k'. De cette façon, nous aurons le plus possible de corrections sur le bord de la nouvelle boule. S'il reste plusieurs boules possibles nous retournons l'une d'entre elles au hasard.

Cette heuristique sera très bonne à chaque fois que le centre de la boule apprise u se trouvera à distance 1 du centre de la boule cible. Cependant, dès que cette distance augmente, l'algorithme se trompera à nouveau. Dans le but d'améliorer cette heuristique, nous pouvons itérer le processus et construire une seconde heuristique qui va répéter la recherche locale décrite précédemment jusqu'à convergence, c'est-à-dire jusqu'à ce que le rayon de la boule ne puisse plus décroître.

Afin de montrer que les boules apprises par IDENTIFICATION_BOULE peuvent être corrigées *a posteriori*, nous comparons, dans une série d'expérimentations, la précision de l'algorithme sans traitement ultérieur, avec l'heuristique à deux étapes, et avec la seconde heuristique qui attend la convergence. À nouveau, nous fixons $|o| + r = 200$ et nous faisons varier le rayon de 10 à 190. Pour chaque rayon, nous tirons cinquante centres au hasard de longueur $200 - r$. Nous faisons alors varier le niveau de pertinence de $0,5$ à 1. Pour chaque paire (pertinence, rayon), nous demandons à IDENTIFICATION_BOULE de retrouver les cinquante boules et nous notons la précision (c'est-à-dire le nombre de boules effectivement retrouvées sur le nombre de boules total). Afin de réduire la variance due aux approximations de l'oracle, nous répétons l'expérience dix fois en utilisant les mêmes boules. La précision moyenne est montrée Figure 6.6.

Nous pouvons remarquer que quelque soit le niveau de pertinence, si nous utilisons l'heuristique attendant la convergence, les résultats sont toujours au moins aussi bons que ceux de l'heuristique en deux étapes, laquelle étant toujours meilleure que l'algorithme sans heuristique. Cependant, les heuristiques n'améliorent pas toujours la précision de l'algorithme : elle dépend du ratio entre le rayon de la boule cible et la longueur du centre. Afin de voir cela plus en détails, nous avons extrait deux coupes transversales, montrées dans les Figures 6.7 et 6.8, où l'on fixe le rayon.

La Figure 6.7 montre la précision d'IDENTIFICATION_BOULE pour des boules cibles telles que $r = 170$ et $|o| = 30$. Dans ce cas, nous gagnons très peu avec l'heuristique. Ceci est peut être en partie dû au fait que la taille de l'ensemble $\mathcal{Q}$, qui est utilisé comme contrôle de l'heuristique, est incroyablement plus petite que le volume des boules cibles. En d'autres termes, l'heuristique n'est pas suffisamment guidée par $\mathcal{Q}$ vers les cibles, parce que $\mathcal{Q}$ n'est pas assez informatif.

D'un autre côté, nous montrons Figure 6.8 la précision pour des boules cibles telles que $r = 10$ et $|o| = 190$. Notre heuristique améliore alors grandement la précision par rapport à l'algorithme sans traitement ultérieur, et ce, quelque soit le niveau de pertinence de l'oracle. Le bénéfice est encore plus important si le niveau de pertinence

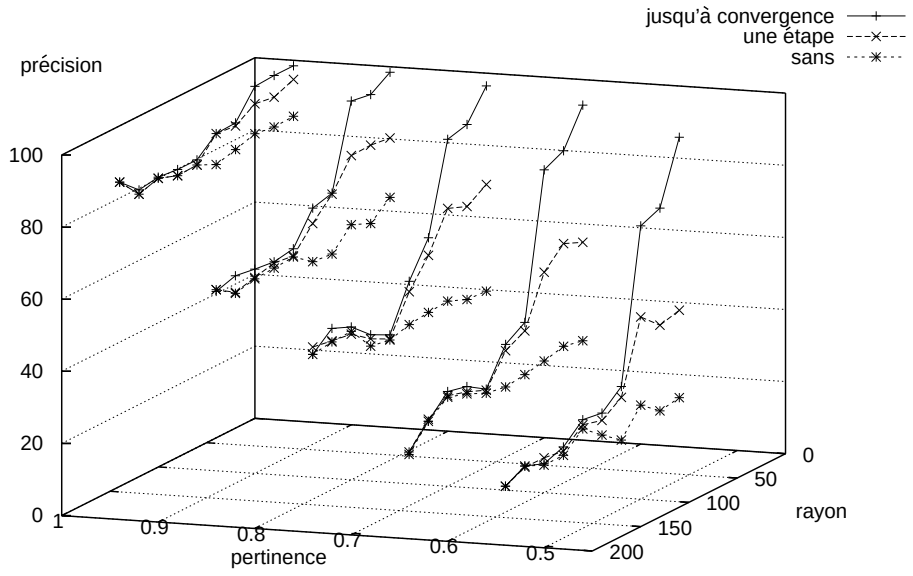

FIG. 6.6 – Précision de IDENTIFICATION_BOULE avec et sans heuristique en fonction de la pertinence et du rayon lorsque $|o| + r = 200$. Pour chaque paire (pertinence, rayon), la moyenne est calculée sur cinquante boules.

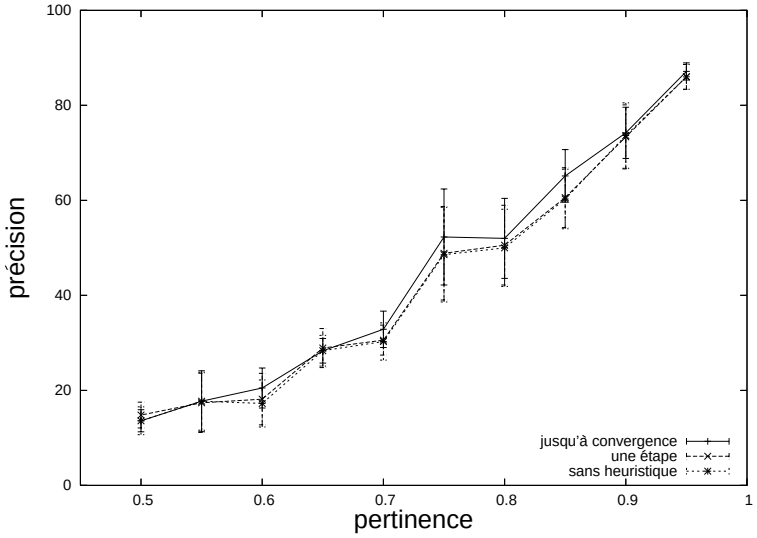

FIG. 6.7 – Précision de IDENTIFICATION_BOULE lorsque $|o| + r = 200$ pour $r = 170$. Pour chaque pertinence, nous calculons la moyenne sur cinquante boules. Afin de réduire la variance, l'expérience est répétée dix fois.

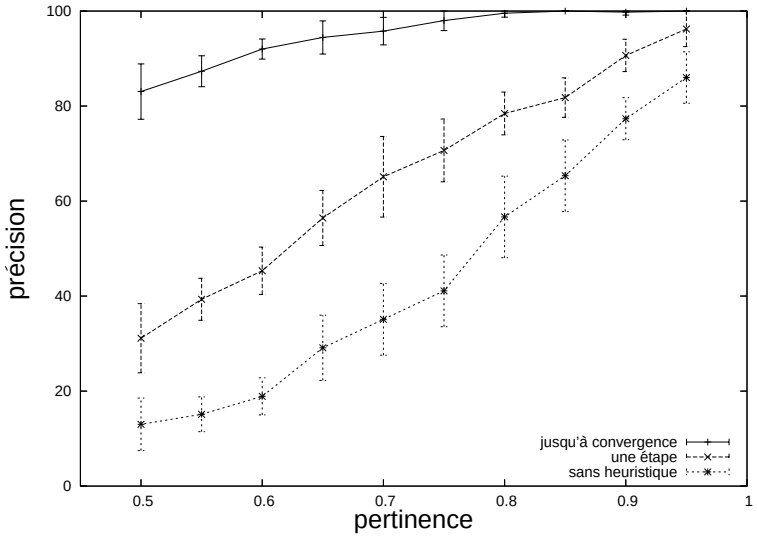

FIG. 6.8 – Précision de IDENTIFICATION_BOULE lorsque $|o|+r = 200$ pour $r = 10$. Pour chaque pertinence, nous calculons la moyenne sur cinquante boules. Afin de réduire la variance, l'expérience est répétée dix fois.

est mauvais. Par exemple, quand $p = 0,6$, l'heuristique avec convergence hausse la précision de 12% à 86%.

Ainsi, dans ce cadre, sans amélioration de l'algorithme, IDENTIFICATION_BOULE produit des boules tellement proche des cibles que de simples modifications d'éditions locales suffisent à améliorer sa précision. Le nombre de boules retrouvées peut alors sans doute être augmenté en se concentrant davantage sur l'amélioration de l'algorithme d'identification des boules.

Conclusion

L'identification des boules de mots à partir de requêtes de correction est possible en temps linéaire, voir logarithmique lorsque la distance d'édition est une distance pondérée.

De plus l'algorithme utilisé peut servir de base à l'apprentissage des boules de mots lorsque l'oracle répond approximativement aux requêtes de correction. Il suffit en effet de modifier légèrement le centre et le rayon de la boule apprise pour identifier correctement la boule cible. Nous insistons sur le fait que nous n'avons pas cherché à construire le meilleur algorithme d'apprentissage face à un oracle répondant approximativement. Nous avons voulu montrer que les propriétés des boules de mots sont telles que de simples heuristiques *a posteriori* suffisent à améliorer l'algorithme pour un contexte bruité.

Conclusion et perspectives

L'apprentissage de langages formels en situations bruitées est un problème important. En effet, les acquisitions de données comportent des risques de corruption. Que ce soit un défaut de capteur ou bien une erreur de recopie humaine, les données d'apprentissage ne correspondent bien souvent pas exactement au langage cible d'origine.

L'apprentissage des langages rationnels a fait l'objet de nombreux travaux, mais ce n'est que récemment que la question de leur identification à partir de données bruitées a commencé à être étudiée, et ce de façon sporadique. Les algorithmes « classiques » d'identification comme Rpni [OG92] n'étant pas conçus pour apprendre dans un tel contexte ne peuvent qu'échouer dans leur tâche.

S'il a fallu si longtemps pour que des algorithmes traitant de ce problème voient le jour, c'est sans nul doute parce qu'il est difficile. De plus, nous avons montré que les résultats d'apprentissage de ces travaux ne sont pas vraiment convaincants. Les algorithmes proposés ne fonctionnent en effet que sur de petits automates et avec un faible niveau de bruit. Le passage à l'échelle n'est donc pas possible. Les langages rationnels, base de la hiérarchie de Chomsky, ne sont donc pas apprenables à partir de données bruitées, et il en sera de même pour les langages de plus haut niveau.

Nous avons par conséquent proposé une nouvelle classe de langages : la classe des boules de mots [dlHJT08]. Bien que d'apparence anodine, les propriétés intrinsèques suggèrent une réelle richesse du point de vue de la complexité combinatoire. La question de savoir si une boule de mots peut être représentée par un automate ayant un nombre polynomial d'états en la longueur du centre et le rayon en est une preuve.

Partant du constat que le bruit statistique n'est pas adapté à une identification à la limite, nous avons également introduit un nouveau type de bruit, le bruit systématique. Nous avons alors étudié l'identification à la limite des boules de mots à partir de données bruitées systématiquement [TdlHJ06]. Les boules de mots sont ainsi apprenables grâce à deux procédés différents : le premier étant une technique de réduction, technique déjà utilisée pour montrer que des problèmes sont au moins aussi durs que d'autres ; le second consistant à débruiter les données à la limite. Dans ce dernier cadre, nous avons alors pu montrer que l'ajout de plus de bruit peut amener à identifier plus rapidement. Si ceci a déjà été utilisé en apprentissage, c'est à notre connaissance la première fois en inférence grammaticale.

Le second apport de ce manuscrit, transversal à la thèse principale de l'identification à partir de données bruitées, concerne l'apprentissage actif. Dans la lignée des travaux de Beccera-Bonache, Bibire et Dediu [BBBD05], nous avons introduit une nouvelle

définition de requêtes de correction, basée sur la distance d'édition [BBdlHJT08]. Notre définition se distingue des autres par l'oracle qui n'aide pas l'apprenant, et surtout par le fait que les corrections sont sensées : l'oracle ne retourne plus une correction basée sur le suffixe, mais une correction la plus proche possible de la requête, à la manière des moteurs de recherche, ou comme le ferait une mère pour corriger son enfant.

Un algorithme d'apprentissage des boules de mots à l'aide de requêtes de correction a alors été présenté. Dans le cadre de l'apprentissage à partir de données bruitées, nous avons étudié son comportement face à un oracle pouvant répondre approximativement aux requêtes. La définition des boules ainsi que leurs propriétés font qu'il est facile d'ajouter une simple heuristique *a posteriori* pour identifier la plupart des bonnes boules et ce même lorsque l'oracle répond souvent incorrectement. Les résultats de ces expériences tendent ainsi également à montrer que les boules de mots sont des candidats idéaux pour un apprentissage en situations bruitées réussi.

Nous pensons avoir montré l'apprentissage à partir de données bruitées sous un nouvel angle en suggérant que la hiérarchie de Chomsky n'est pas pertinente dans ce contexte. Les travaux théoriques ayant des résultats positifs d'identification présentent des classes de langages transversales à cette hiérarchie. Les solutions que nous avons apportées, le bruit systématique et les boules de mots, abondent aussi dans ce sens. Bien évidemment, il reste beaucoup de travail à faire : quels sont, parmi les autres langages topologiques que nous avons proposés, ceux qui se comportent aussi bien que les boules ? Peut-on construire une hiérarchie de langages selon la difficulté de leur apprentissage en situations bruitées ? Les perspectives sont nombreuses.

Un premier travail pourrait ainsi être de chercher d'autres classes identifiables à partir de données bruitées systématiquement, de dégager des points communs à ces classes afin de mieux appréhender ce qui rend une classe résistante au bruit. Un second concerne les boules de mots : est-il possible de représenter une boule par un automate ayant un nombre polynomial d'états ?

Le bruit systématique peut également faire l'objet d'une étude plus approfondie : ses fortes contraintes (tout le voisinage d'un mot doit apparaître dans la présentation) feront certainement que certaines classes de langages ne seront pas apprenables alors qu'elles le seraient en relâchant un peu la définition du bruit. Par exemple, les conditions pourraient être que seule une certaine quantité, comme par exemple les trois quarts du voisinage au lieu de tout le voisinage, pourrait être nécessaire pour définir le bruité d'un mot.

Concernant le débruitage à la limite, il n'est pas nécessaire, comme nous l'avons suggéré, que la classe soit une classe de langages fermés pour pouvoir apprendre à partir de données bruitées : un débruitage partiel des données peut suffire si elles correspondent à un unique langage. Prenons par exemple la classe des boules de mots sans leur centre. Le fait de bruiter le langage fera perdre l'information qu'il « manque » un mot dans le langage. Cependant, le débruitage permettra de trouver la boule et d'en déduire le langage cible, la boule sans le centre. Une étude peut également avoir lieu sur les fonctions d'intérieur et d'extérieur : d'autres fonctions c-duales peuvent être définies induisant alors d'autres langages ouverts et fermés.

Enfin, des directions de recherche peuvent concerner les requêtes de correction. Nous savons déjà que les automates ne sont pas identifiables, mais que les boules de mots le sont. Caractériser l'ensemble des classes de langages identifiables à partir de requêtes de correction semble une piste intéressante. L'algorithme présenté permettant d'identifier les boules de mots en un nombre logarithmique de requêtes peut certainement être amélioré. Les contraintes sont telles que la distance d'édition est pondérée et que l'alphabet comporte au moins trois lettres. Il serait intéressant de savoir quelles sont les autres pondérations permettant une telle complexité, et si ce résultat est possible avec un alphabet de deux lettres. Le modèle d'oracle approximatif est lui aussi améliorable : tel que défini, ses réponses aux requêtes de correction ne sont pas persistantes. Un apprenant posant plusieurs fois la même requête pourrait alors corriger les corrections de l'oracle, *etc.* Nous avons également fait l'hypothèse que les corrections sont toutes à l'intérieur du langage cible et qu'il répond selon une loi géométrique. De futurs travaux pourraient ainsi avoir lieu sur d'autres modèles d'oracle. Un oracle pourrait corriger en donnant un mot à l'extérieur du langage. Si cela n'est pas pertinent pour les boules (il est facile de vérifier qu'un mot appartient à la boule), certains langages peuvent ne pas permettre cette liberté. Un autre modèle d'oracle pourrait être construit tel que la correction varie en fonction de la distance de la requête par rapport au langage : plus la requête est loin, plus l'oracle peut se tromper dans sa réponse.

Annexe

Sur les prétopologies

Nous reprenons ici quelques définitions du Manuel de prétopologie et ses applications [Bel93], puis nous définissons un espace prétopologique adapté à l'étude de Σ^* et nous en étudions les propriétés dans le cadre du débruitage à la limite.

Pour débruiter les données, nous devons savoir si elles appartiennent au langage cible ou non, c'est-à-dire pouvoir décider si une donnée est du bruit. Pour cela, nous allons avoir besoin de connaître les relations de proximité des données les unes par rapport aux autres, et notamment par rapport à celles qui appartiennent effectivement au langage. Cette notion de « voisinage » fait naturellement appel à de la topologie. Cependant, pour notre problème, la topologie classique et son grand nombre d'axiomes sont trop contraignants. Nous allons donc utiliser les espaces prétopologiques qui visent à définir des topologies possédant moins d'axiomes. Ces espaces prétopologiques s'articulent autour de deux fonctions dites *c-duales* :

Définition 42 (c-dualité) *Notons c le* complémentaire *: soit U un ensemble, $\forall A \in \mathcal{P}(U), c(A) = U \setminus A = \bar{A}$. Deux applications e et i de $\mathcal{P}(U)$ dans $\mathcal{P}(U)$ sont* c-duales *si et seulement si elles vérifient $i = c \circ e \circ c$ (ou de façon équivalente $e = c \circ i \circ c$).*

En d'autres termes, soit L un ensemble définit sur Σ^*, si i et e sont c-duales, on a $e(L) = \overline{i(\bar{L})}$ ou encore, $\overline{e(L)} = i(\bar{L})$. Les fonctions i et e jouent souvent un rôle d'érosion et d'extension.

Définition 43 (Espace prétopologique) (U, i, e) *définit un* espace prétopologique*, si et seulement si :*

1. *i et e sont c-duales,*
2. *$i(U) = U$, (ou encore $e(\emptyset) = \emptyset$, comme $U = i(U) = \overline{e(\bar{U})} = \overline{e(\emptyset)}$)*
3. *$\forall L \in \mathcal{P}(U), i(L) \subset L$ (ou encore $L \subset e(L)$).*

La notion de topologie peut donc être définie comme étant un cas particulier de prétopologie. Une topologie est un espace prétopologique tel que $\forall A, B \in \mathcal{P}(U), e(A \cup B) = e(A) \cup e(B)$ et $e(e(A)) = e(A)$. Avec les outils de la prétopologie, nous pouvons donc modéliser des processus d'extension $L = e^0(L) \subset e(L) \subset e[e(L)] \subset \ldots \subset e^n(L) \subset \ldots \subset U$ et d'érosion $L = i^0(L) \supset i(L) \supset i[i(L)] \supset \ldots \supset i^n(L) \supset \ldots \supset \emptyset$, ce qui n'est pas le cas en topologie à cause de l'idempotence des applications e et i.

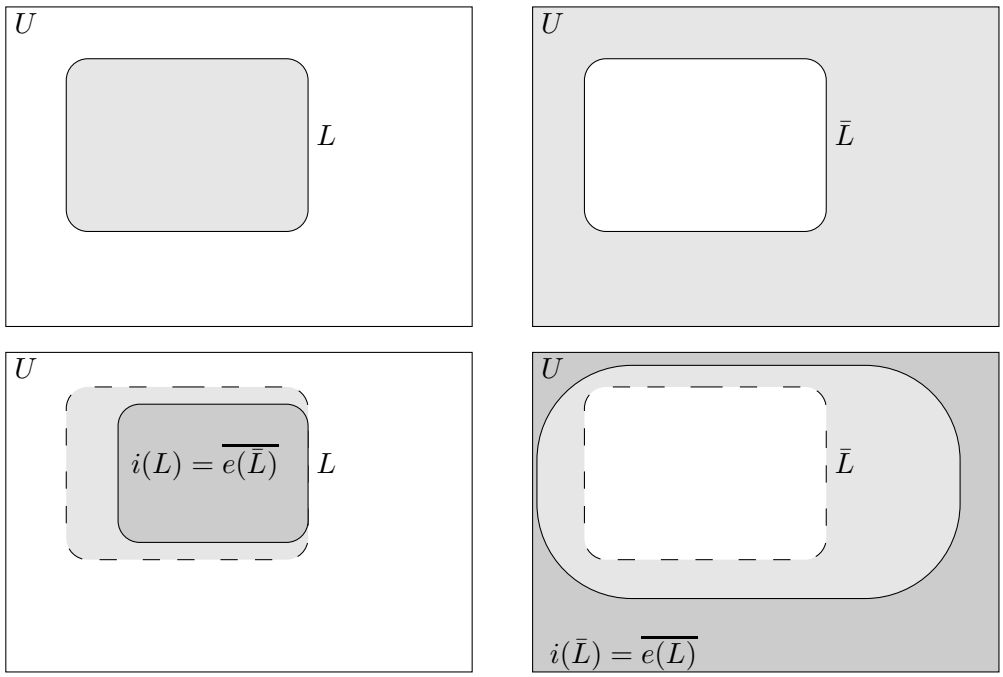

FIG. 9 – Principe de c-dualité de deux fonction i et e. Les figures de gauche montrent que l'érosion d'un langage correspond au complémentaire de l'extension de son complémentaire. De même, on peut voir sur les figures de droite que l'érosion du complémentaire d'un langage est égale au complémentaire de l'extension du langage.

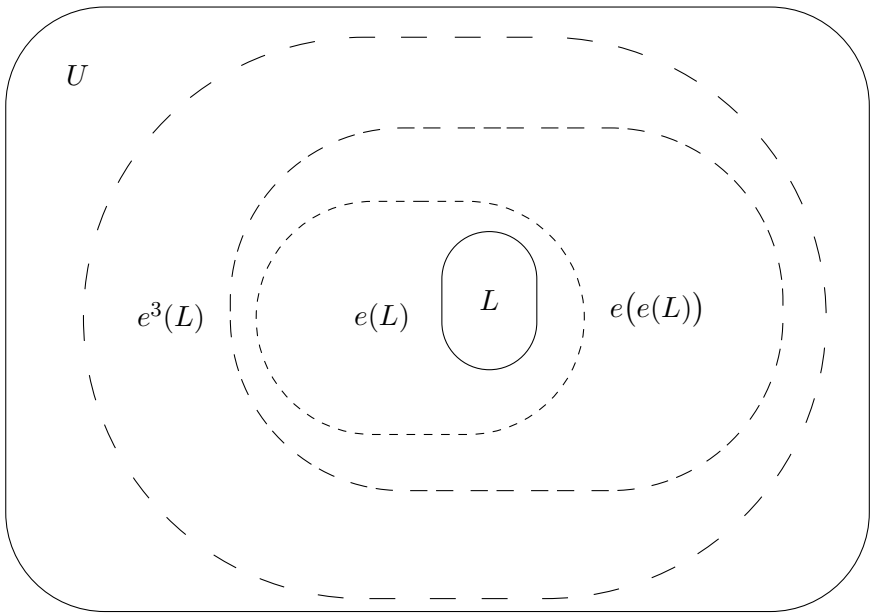

FIG. 10 – Processus d'extension d'une prétopologie (U, i, e).

Définition 44 (Ensembles fermés et ouverts) *Soit (U, i, e) un espace prétopologique. K est un* ensemble fermé *de U si et seulement si $e(K) = K$, et L est un* ensemble ouvert *de U si et seulement si $i(L) = L$. Une classe de langages $\mathcal{L}$ est* fermée *si et seulement si $\forall L \in \mathcal{L}$, L est un ensemble fermé et est* ouverte *si et seulement si $\forall L \in \mathcal{L}$, L est un ensemble ouvert.*

Après ces rappels sur les prétopologies, nous allons pouvoir définir des fonctions i et e grâce auxquelles nous construirons des espaces prétopologiques adaptés à notre étude. Nous rappelons pour cela que la distance utilisée (et notamment pour la fonction de bruit N) est la distance d'édition.

Définition 45 (Intérieur et extérieur) *Notons $I(L)$* l'intérieur *de L : $I(L) = \{w \in \Sigma^* : N_1(\{w\}) \subseteq L\}$ et $E(L)$* l'extérieur *de L : $E(L) = \{w \in \Sigma^* : N_1(\{w\}) \cap L \neq \emptyset\}$.*

Nous appelons le k-intérieur de L *la fonction définie par*

$$I_0(L) = L, \forall k \in \mathbb{N}^+ \; I_k(L) = I[I_{k-1}(L)]$$

et le k-extérieur de L *la fonction définie par*

$$E_0(L) = L, \forall k \in \mathbb{N}^+ \; E_k(L) = E[E_{k-1}(L)].$$

Le 1-intérieur d'un langage L correspond alors à tous les mots dont le voisinage est inclus dans le langage (le voisinage d'un mot w étant les mots à distance 1 de w). Par exemple, le 1-intérieur du langage $L = \{\lambda, a, b, aa, ab, ba, aaa, bbb\}$ est $I_1(L) = \{\lambda, a\}$.

De même, le 1-extérieur d'un langage L correspond à tous les mots dont le voisinage intersecte L. Le 1-extérieur de $L = \{\lambda, a\}$ est $E_1(L) = \{\lambda, a, b, aa, ab, ba\}$.

On peut montrer que la définition du k-intérieur et du k-extérieur peut être modifiée pour prendre en compte le k-bruité de chaque mot du langage :

Proposition 9 $I_k(L) = \{w \in \Sigma^* : N_k(\{w\}) \subseteq L\}$

Démonstration :
Notons $I^k(L) = \{w \in \Sigma^* : N_k(\{w\}) \subseteq L\}$. La preuve se fait par récurrence sur k :
(k=0) $I^0 = \{w \in \Sigma^* : N_0(\{w\}) \subseteq L\} = \{w \in \Sigma^* : \{w\} \subseteq L\} = I_0(L)$
(k=n) $I_{n+1}(L) = I(I_n(L)) = I(I^n(L))$ (par hypothèse). On a alors $I(I^n(L)) = \{w \in \Sigma^* : N(\{w\}) \subseteq I^n(L)\}$. Or $u \in I(I^n(L)) \Leftrightarrow u \in I^{n+1}(L)$. En effet :

$$\begin{aligned} u \in I(I^n(L)) &\Leftrightarrow \forall v \in N(\{u\}), v \in I^n(L) \\ &\Leftrightarrow \forall v \in N(\{u\}), N_n(\{v\}) \subseteq L \\ &\Leftrightarrow N_n(N(\{u\})) \subseteq L \\ &\Leftrightarrow u \in I^{n+1}(L) \end{aligned}$$

On a alors $I(I^n(L)) = \{w \in \Sigma^* : N(\{w\}) \subseteq I_n(L)\} = I(I_n(L)) = I_{n+1}(L)$.

□

De la même façon, en remarquant que $E_k = N_k$, on peut montrer que et $E_k(L) = \{w \in \Sigma^* : N_k(\{w\}) \cap L \neq \emptyset\}$. On en déduit également que l'extérieur d'une boule est une boule, c'est-à-dire, $E_k(B_r(o)) = B_{r+k}(o)$. En revanche, l'intérieur d'une boule n'est pas une boule :

Proposition 10 $B_{r-1}(o) \subseteq I(B_r(o))$

Démonstration :

$\subseteq$ Soient $w \in B_{r-1}(o)$ et $y \in \Sigma^*$ tel que $y \in N(\{w\})$. Alors $d(w, o) \leq r - 1$ et $d(w, y) \leq 1$. On a donc $(r-1)+1 \geq d(w, o) + d(w, y) \geq d(o, y)$, et $y \in B_r(o)$. On en déduit que $N(\{w\}) \subseteq B_r(o)$ et $B_{r-1}(o) \subset I(B_r(o))$
$\not\supseteq$ Le mot *bbbaaa* appartient à $\in I(B_5(aabb))$. Cependant, $bbbaaa \notin B_4(aabb)$.

□

Ces notions ne sont pas sans rappeller celles des Rough Sets [Paw90] utilisés notamment par Satoshi Kobayashi et Takashi Yokomori [KY95], les *lower approximation* et *upper approximation* d'un ensemble, avec lesquelles elles partagent un grand nombre de propriétés sur les intersections et les unions, ainsi que les *closet sets* de de Brecht, Kobayashi, Tokunaga et Yamamoto [dBY06]. Les espaces prétopologiques ont en effet les propriétés suivantes :

Proposition 11 *Les affirmations suivantes sont vraies :*

1. $I_k(L) \subseteq I(L) \subseteq L \subseteq E(L) \subseteq E_k(L)$

2. $I_k(\emptyset) = E_k(\emptyset) = \emptyset$
3. $I_k(\Sigma^*) = E_k(\Sigma^*) = \Sigma^*$
4. $E_k(L \cup L') = E_k(L) \cup E_k(L')$
5. $I_k(L \cap L') = I_k(L) \cap I_k(L')$
6. $L \subseteq L' \Rightarrow I_k(L) \subseteq I_k(L')$ *et* $E_k(L) \subseteq E_k(L')$
7. $I_k(L) \cup I_k(L') \subseteq I_k(L \cup L')$
8. $E_k(L \cap L') \subseteq E_k(L) \cap E_k(L')$
9. $I_k(\bar{L}) = \overline{E_k(L)}$
10. $E_k(\bar{L}) = \overline{I_k(L)}$

De part la définition des fonctions d'intérieur et d'extérieur, on pourrait s'attendre à ce que $E = I^{-1}$. Ce n'est cependant pas le cas :

Exemple 47 *Soit* $L = \{\lambda, a, b, aa, ab, ba, aaa, bbb\}$. *Nous avons vu que* $I_1(L) = \{\lambda, a\}$. *De plus,* $E_1(\{\lambda, a\}) = \{\lambda, a, b, aa, ab, ba\} \neq L$.

De même, soit $L = \{a, b\}$. *On a alors* $E_1(L) = \{\lambda, a, b, aa, ab, ba, bb\}$ *et* $I_1(E_1(L)) = \{\lambda, a, b\} \neq L$.

De plus, l'intuition voudrait que l'on prenne $i = I_k$ et $e = E_k$ comme fonction d'intérieur et d'extérieur. Cependant, définies comme ceci, l'extension et l'érosion sont trop importantes pour dégager des fermés et ouverts intéressants et non triviaux. Par exemple, si pour beaucoup de boules, $I(E(B_r(o))) = B_r(o)$, ce n'est pas toujours le cas comme le montre la Proposition 10.

Nous allons alors prendre $e = I_k \circ E_k$ et $i = E_k \circ I_k$. Nous montrons maintenant que ces deux fonctions remplissent bien les propriétés attendues.

Proposition 12 $I_k \circ E_k$ *et* $E_k \circ I_k$ *sont c-duales dans* Σ^*, *c'est-à-dire :*

$$\forall L \in \mathcal{P}(\Sigma^*), I_k(E_k(L)) = \overline{E_k(I_k(\overline{L}))}.$$

Démonstration :
I_k and E_k sont c-duales : $I_k(\overline{L}) = \{w \in \Sigma^* : N_k(\{w\}) \subseteq \overline{L}\} = \{w \in \Sigma^* : N_k(\{w\}) \cap L = \emptyset\} = \overline{E_k(L)}$. Donc $E_k(I_k(\overline{L})) = \overline{I_k(\overline{I_k(\overline{L})})} = \overline{I_k(E_k(L))}$. □

Théorème 35 $\mathbb{E}_k = (\Sigma^*, E_k \circ I_k, I_k \circ E_k)$ *définit un espace prétopologique, c'est-à-dire vérifie :*

1. $I_k \circ E_k$ *et* $E_k \circ I_k$ *sont c-duales,*
2. $E_k(I_k(\Sigma^*)) = \Sigma^*$,
3. $\forall L \in \mathcal{P}(\Sigma^*), E_k(I_k(L)) \subset L$

Démonstration :

1. Par la Proposition 12.
2. Trivial.
3. $x \in E_k(I_k(L)) \Rightarrow N_k(\{x\}) \cap I_k(L) \neq \emptyset \Rightarrow \exists y \in I_k(L) : d(x,y) \leq k$.
   $d(x,y) \leq k \Rightarrow x \in N_k(\{y\})$ et $y \in I_k(L) \Rightarrow N_k(\{y\}) \subseteq L$ donc $x \in L$ et $E_k(I_k(L)) \subseteq L$.

□

La fonction E_k, *respectivement* I_k, nous permettent ainsi d'ajouter du bruit à L, *respectivement* d'enlever du bruit. Nous pouvons alors les utiliser dans notre cadre de débruitage à la limite. De plus, les propriétés de la Proposition 11 restent pour la plupart vraies :

Proposition 13 *Les affirmations suivantes sont vraies :*

1. $E_k(I_k(L)) \subseteq L$
2. $L \subseteq I_k(E_k(L))$
3. $E_k(I_k(\emptyset)) = E_k(I_k(\emptyset)) = \emptyset$
4. $E_k(I_k(\Sigma^*) = I_k(E_k(\Sigma^*) = \Sigma^*$
5. $E_k(I_k(L)) \cup E_k(I_k(L')) \subseteq E_k(I_k(L \cup L'))$
6. $I_k(E_k(L \cap L')) \subseteq I_k(E_k(L)) \cap I_k(E_k(L'))$
7. $L \subseteq L' \Rightarrow E_k(I_k(L)) \subseteq E_k(I_k(L'))$ *et* $I_k(E_k(L)) \subseteq I_k(E_k(L'))$
8. $E_k(I_k(L \cap L')) \subseteq E_k(I_k(L)) \cap E_k(I_k(L'))$
9. $I_k(E_k(L)) \cup I_k(E_k(L')) \subseteq I_k(E_k(L \cup L'))$

Démonstration :
(1) $x \in E_k(I_k(L)) \Rightarrow N_k(\{x\}) \cap I_k(L) \neq \emptyset \Rightarrow \exists y \in I_k(L) : d(x,y) \leq k$. $d(x,y) \leq k \Rightarrow x \in N_k(\{y\})$ et $y \in I_k(L) \Rightarrow N_k(\{y\}) \subseteq L$ donc $x \in L$ et $E_k(I_k(L)) \subseteq L$.
(2) Par application de la c-dualité et du complémentaire.
Les autres points découlent de la Proposition 11. □

De plus, on peut montrer que dans $\mathbb{E}_k = (\Sigma^*, E_k \circ I_k, I_k \circ E_k)$, les boules sont des langages ouverts :

Théorème 36 *Dans* $(\Sigma^*, E \circ I, I \circ E)$, $\forall o \in \Sigma^*, \forall k \geq 1, B_r(o)$ *est un ensemble ouvert de* Σ^*, *c'est-à-dire* $B_r(o) = E(I(B_r(o)))$.

Démonstration :

$\supseteq$ $E(I(B_r(o))) \subseteq B_r(o)$: par la Proposition 13
$\subseteq$ $B_r(o) \subseteq E(I(B_r(o)))$:

$$\begin{aligned} x \in B_r(o) &\Rightarrow \exists y : x \in N(\{y\}) \wedge N(\{y\}) \subseteq B_r(o) \\ &\Rightarrow \exists y : d(x,y) \leq 1 \wedge y \in I(B_r(o)) \\ &\Rightarrow N(\{x\}) \cap I(B_r(o)) \neq \emptyset \\ &\Rightarrow x \in E(I(B_r(o))) \end{aligned}$$

□

De même, on peut montrer que $B_r(o) = E_j(I_j(B_r(o)))$ (si $j < r$).

Liste des publications

Les travaux présentés dans ce manuscrit ont donné lieu à différentes publications nationales et internationales :

- L'identification à la limite à partir de données bruitées des boules de mots a été présentée aux conférences CAp 2006 et ICGI 2006. Nous y introduisons les concepts de bruit systématique, de débruitage à la limite, et d'espaces prétopologiques. Nous montrons alors comment apprendre les boules de mots à partir de données bruitées systématiquement.

[CAp2006] Frédéric Tantini, Colin de la Higuera, and Jean-Christophe Janodet. Identification à la limite de langages dans le cadre d'un bruit systématique. Dans les actes de *8ème Conférence francophone sur l'Apprentissage automatique (CAp)*, pages 203–218, Trégastel, 2006. PUG.

[ICGI2006] Frédéric Tantini, Colin de la Higuera, and Jean-Christophe Janodet. Identification in the limit of systematic-noisy languages. In *Proceedings 8th International Colloquium on Grammatical Inference (IGCI)*, pages 19–31, Tokyo, 2006. LNCS 4201.

- Les requêtes de correction ont fait l'objet de plusieurs publications : elles ont été introduites lors des conférences CAp 2007 puis ECML 2007. Elles ont fait l'objet d'un papier long dans la revue JMLR.

[CAp2007] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Apprentissage des Boules de Mots avec des Requêtes de Correction. Dans les actes de *9ème Conférence francophone sur l'Apprentissage automatique (CAp)*, pages 55–70, Grenoble, 2007. CEPADUES.

[ECML2007] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings with correction queries. In *Proceedings 18th European Conference on Machine Learning (ECML)*, pages 18–29, Warsaw, 2007. LNCS 4701.

[JMLR2008] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings from edit corrections. *Journal of Machine Learning Research*, 9 :1841–1870, 2008.

- Enfin, nous avons proposé une étude transversale de différents paradigmes d'apprentissage en comparant l'apprentissage des automates finis déterministes avec celui des boules de mots, bonnes et mauvaises. Ce travail a été présenté en français

lors de la conférence CAp 2008 puis lors de la conférence ICGI 2008

[CAp2008] Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Apprentissage de langages à partir de ressources bornées : le cas des AFD et des boules de mots. Dans les actes de *10ème Conférence francophone sur l'Apprentissage automatique (CAp)*, pages 55-70, Porquerolles, 2008. CEPADUES.

[ICGI2008] Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning languages from bounded resources: the case of the DFA and the balls of strings. In *Proceedings 9th International Conference on Grammatical Inference (ICGI)*, pages 43–56, Saint-Malo, 2008. LNCS 5278.

Bibliographie

[Ada79] Douglas Adams. *The Hitchhiker's Guide to the Galaxy*. London: Pan Books, 1979.

[ADKF70] Vladimir L. Arlazarov, Efim A. Dinic, Mikhail A. Kronrod, and Igor A. Faradzev. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, 11(5) :1209–1210, 1970.

[AH03] Adam Albright and Bruce Hayes. Rules vs. analogy in English past tenses: a computational/experimental study. *Cognition*, 90 :119–161, 2003.

[AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings 24th ACM Symposium on Theory of Computing (STOC)*, pages 444–454, New York, 1991. ACM Press.

[AK94] Dana Angluin and Martins Krikis. Learning with malicious membership queries and exceptions (extended abstract). In *Proceedings 7th Annual ACM Conference on Computational Learning Theory (COLT)*, pages 57–66, 1994.

[AKST97] Dana Angluin, Martins Krikis, Robert H. Sloan, and György Turán. Malicious omissions and errors in answers to membership queries. *Machine Learning*, 28(2-3) :211–255, 1997.

[AL87] Dana Angluin and Philip D. Laird. Learning from noisy examples. *Machine Learning*, 2(4) :343–370, 1987.

[ALK99] Donald A. Adjeroh, Moon-Chuen Lee, and Irwin King. A distance measure for video sequences. *Computer Vision and Image Understanding*, 75(1-2) :25–45, 1999.

[All92] Lloyd Allison. Lazy dynamic-programming can be eager. *Information Processing Letters*, 43(4) :207–212, 1992.

[AM97] Naoki Abe and Hiroshi Mamitsuka. Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, 29 :275–301, 1997.

[Ang78] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39 :337–350, 1978.

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45 :117–135, 1980.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2) :87–106, 1987.

[Ang88a] Dana Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, March 1988.

[Ang88b] Dana Angluin. Queries and concept learning. *Machine Learning*, 2 :319–342, 1988.

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5 :121–150, 1990.

[Ang01] Dana Angluin. Queries revisited. In *Proceedings 12th International Conference on Algorithmic Learning Theory (ALT)*, pages 12–31, Washington, 2001. LNCS 2225.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference: theory and methods. *ACM computing surveys*, 15(3) :237–269, 1983.

[AS94] Dana Angluin and Donna K. Slonim. Randomly fallible teachers: learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1) :7–26, 1994.

[ASA01] Hiroki Arimura, Hiroshi Sakamoto, and Setsuo Arikawa. Efficient learning of semi-structured data from queries. In *Proceedings 12th International Conference on Algorithmic Learning Theory (ALT)*, pages 315–331, Washington, 2001. LNCS 2225.

[BB06] Leonor Beccera-Bonache. *On the learnability of mildly context-sensitive languages using positive data and correction queries.* PhD thesis, University of Tarragona, 2006.

[BBBD05] Leonor Beccera-Bonache, Cristina Bibire, and Adrian Horia Dediu. Learning DFA from corrections. In Henning Fernau, editor, *Proceedings Workshop on Theoretical Aspects of Grammar Induction (TAGI)*, WSI-2005-14, pages 1–11. Technical Report, University of Tubingen, 2005.

[BBdlHJT07] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings with correction queries. In *Proceedings 18th European Conference on Machine Learning (ECML)*, pages 18–29, Warsaw, 2007. LNCS 4701.

[BBdlHJT08] Leonor Becerra-Bonache, Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning balls of strings from edit corrections. *Journal of Machine Learning Research*, 9 :1841–1870, 2008.

[BDGW94] José L. Balcázar, Josep Díaz, Ricard Gavaldà, and Osamu Watanabe. The query complexity of learning DFA. *New Generation Computing*, 12 :337–358, 1994.

[Bel93] Z. Belmandt. *Manuel de prétopologie et ses applications.* Hermès, 1993.

[BJS06] Marc Bernard, Jean-Christophe Janodet, and Marc Sebban. A discriminative model of stochastic edit distance in the form of a conditional

transducer. In *Proceedings 8th International Colloquium on Grammatical Inference (IGCI)*, pages 240–252, Tokyo, 2006. LNCS 4201.

[BL97] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artifical Intelligence*, 97(1-2) :245–271, 1997.

[BM00] Eric Brill and Robert C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 286–293, 2000.

[BNC03] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14) :2357–2366, 2003.

[BYN99] Ricardo A. Baeza-Yates and Gonzalo Navarro. Faster approximate string matching. *Algorithmica*, 23(2) :127–158, 1999.

[BYN02] Ricardo A. Baeza-Yates and Gonzalo Navarro. New and faster filters for multiple approximate string matching. *Random Structures and Algorithms*, 20(1) :23–49, 2002.

[Cas90] Francisco Casacuberta. Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7) :691–695, 1990.

[CFWS06] Alexander Clark, Christophe Costa Florêncio, Chris Watkins, and Mariette Serayet. Planar languages and learnability. In *Proceedings 8th International Colloquium on Grammatical Inference (IGCI)*, pages 148–160, Tokyo, 2006. LNCS 4201.

[CGLN07] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1) :33–67, 2007.

[CHL01] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithmique du texte*. Vuibert, 2001.

[Cho57] Noam Chomsky. *Syntactic structure*. Mouton, 1957.

[CJS01] John Case, Sanjay Jain, and Arun Sharma. Synthesizing noise-tolerant language learners. *Theoretical Computer Science*, 261(1) :31–56, 2001.

[CK68] Arthur C. Clarke and Stanley Kubrick. 2001 : l'odyssée de l'espace, 1968.

[CM07] Graham Cormode and Shan Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(1), 2007.

[CM08] Antoine Cornuéjols and Laurent Miclet. *Apprentissage Artificiel - Concepts et Algorithmes, 2nde édition*. Eyrolles, 2008.

[CO94] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proceedings 2nd International Colloquium on Grammatical Inference and Applications (IGCI)*, pages 139–150, Alicante, 1994. LNCS 862.

[Cou06] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings 5th International Conference on Computers and Games (CG)*, pages 72–83, Turin, 2006. LNCS 4630.

[CSS99] Sung-Hyuk Cha, Yong-Chul Shin, and Sargur N. Srihari. Approximate stroke sequence string matching algorithm for character recognition and analysis. In *Proceedings 5th International Conference on Document Analysis and Recognition (ICDAR)*, pages 53–56, Bangalore, 1999. IEEE Computer Society.

[CV98] Pedro Cruz and Enrique Vidal. Learning regular grammars to model musical style: comparing different coding schemes. In *Proceedings 4th International Conference on Grammatical Inference (ICGI)*, pages 211–222, Ames, 1998. LNCS 1433.

[dBY06] Matthew de Brecht and Akihiro Yamamoto. Mind change complexity of inferring unbounded unions of pattern languages from positive data. In *Proceedings 17th International Conference on Algorithmic Learning Theory (ALT)*, pages 124–138, Barcelona, 2006. LNCS 4264.

[DdG96] François Denis, Cyrille d'Halluin, and Rémi Gilleron. PAC learning with simple examples. In *Proceedings 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, LNCS 1046, pages 231–242, 1996.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998.

[Den01] François Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1) :37–66, 2001.

[dlH97] Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27 :125–138, 1997.

[dlH05a] Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9) :1332–1348, 2005.

[dlH05b] Colin de la Higuera. Complexity and reduction issues in grammatical inference. Technical Report ISSN 0946-3852, Universität Tübingen, 2005.

[dlH06a] Colin de la Higuera. *Data complexity in Pattern Recognition*, chapter Data complexity in Grammatical Inference. Advanced Information and Knowledge Processing. Springer Verlag, 2006.

[dlH06b] Colin 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.

[dlHC00] Colin de la Higuera and Francisco Casacuberta. Topology of strings: median string is NP-complete. *Theoretical Computer Science*, 230 :39–48, 2000.

[dlHJT08] Colin de la Higuera, Jean-Christophe Janodet, and Frédéric Tantini. Learning languages from bounded resources: the case of the DFA and the

balls of strings. In *Proceedings 9th International Conference on Grammatical Inference (ICGI)*, pages 43–56, Saint-Malo, 2008. LNCS 5278.

[dlHOV96] Colin de la Higuera, Jose Oncina, and Enrique Vidal. Identification of DFA: data-dependent versus data-independent algorithm. In *Proceedings 3rd International Conference on Grammatical Inference (ICGI)*, pages 313–325, Montpellier, 1996. LNCS 1147.

[dlHPT04] Colin de la Higuera, Frédéric Piat, and Frédéric Tantini. Learning stochastic finite automata for musical style recognition. In *Proceedings 7th International Conference on Grammatical Inference (ICGI)*, pages 345–346, Athens, 2004. LNCS 3264.

[DM98] Pierre Dupont and Laurent Miclet. Inférence grammaticale régulière : fondements théoriques et principaux algorithmes. Technical Report RR-3449, IRISA, 1998.

[DMV94] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? In *Proceedings 2nd International Colloquium on Grammatical Inference and Applications (IGCI)*, pages 25–37, Alicante, 1994. LNCS 862.

[Dup96] Pierre Dupont. Incremental regular inference. In *Proceedings 3rd International Conference on Grammatical Inference (ICGI)*, pages 222–237, Montpellier, 1996. LNCS 1147.

[EM97] Nadia El-Mabrouk. On the size of minimal automata for approximate string matching. Technical Report 19, Institut Gaspard Monge, Université de Marne-la-Vallée, France, 1997.

[FGMP94] Michael Frazier, Sally A. Goldman, Nina Mishra, and Leonard Pitt. Learning from a consistently ignorant teacher. In *Proceedings 7th Annual ACM Conference on Computational Learning Theory (COLT)*, pages 328–339, New Brunswick, 1994. ACM.

[GC97] Yves Grandvalet and Stéphane Canu. Adaptive noise injection for input variables relevance determination. In *Proceedings 7th International Conference on Artificial Neural Networks (ICANN)*, pages 463–468, Lausanne, 1997. LNCS 1327.

[GG08] Robert Grossman and Yunhong Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In *Proceedings 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 920–927, Las Vegas, 2008. ACM.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLT01] Lee Giles, Steve Lawrence, and Ah C. Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine Learning*, 44(1) :161–183, 2001.

[GM92] Sally A. Goldman and David Mathias. Learning k-term DNF formulas with an incomplete membership oracle. In *Proceedings 5th Annual ACM Conference on Computational Learning Theory (COLT)*, pages 77–84, 1992.

[GM96] Sally A. Goldman and David Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2) :255–267, 1996.

[Gol67] Mark E. Gold. Language identification in the limit. *Information and Control*, 10(5) :447–474, 1967.

[Gol78] Mark E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37 :302–320, 1978.

[Gre02] Ronald I. Greenberg. Fast and simple computation of all longest common subsequences. *CoRR*, cs.DS/0211001, 2002.

[Gre03] Ronald I. Greenberg. Bounds on the number of longest common subsequences. *CoRR*, cs.DM/0301030, 2003.

[GSVG94] Pedro García, Encarna Segarra, Enrique Vidal, and Isabel Galiano. On the use of the morphic generator grammatical inference (MGGI) methodology in automatic speech recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 4 :667–685, 1994.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[Ham50] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2) :147–160, 1950.

[HBS03] Amaury Habrard, Marc Bernard, and Marc Sebban. Improvement of the state merging rule on noisy data in probabilistic grammatical inference. In *Proceedings 14th European Conference on Machine Learning (ECML)*, pages 169–180, Cavtat-Dubrovnik, 2003. LNCS 2837.

[HHNS02] Andreas Hagerer, Hardi Hungar, Oliver Niese, and Bernhard Steffen. Model generation by moderated regular extrapolation. In *Proceedings 5th International Conference on Fundamental Approaches to Software Engineering (FASE)*, pages 80–95, Heidelberg, 2002. LNCS 2306.

[HMGS05] Yu Hu, Irina Matveeva, John Goldsmith, and Colin Sprague. The SED heuristic for morpheme discovery: a look at Swahili. In *Proceedings 2nd Workshop of Psychocomputational Models of Human Language Acquisition (PMHLA)*, pages 28–35, 2005.

[HPRW96] Lisa Hellerstein, Krishnan Pillaipakkamnatt, Vijay Raghavan, and Dawn Wilkins. How many queries are needed to learn? *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, 43(5) :840–862, 1996.

[Ind04] Piotr Indyk. Approximate nearest neighbor under edit distance via product metrics. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 646–650, New Orleans, 2004. SIAM.

[JABC03] Xiaoyi Jiang, Karin Abegglen, Horst Bunke, and János Csirik. Dynamic computation of generalised median strings. *Pattern Analysis and Applications*, 6(3) :185–193, 2003.

[JLMZ02] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2) :371–388, 2002.

[JPHP00] Anil K. Jain, Salil Prabhakar, Lin Hong, and Sharath Pankanti. Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing*, 9(5) :846–859, 2000.

[Kel55] John L. Kelley. *General Topology*. Van Nostrand, 1955.

[Kin08] Efim B. Kinber. On learning regular expressions and patterns via membership and correction queries. In *Proceedings 9th International Conference on Grammatical Inference (ICGI)*, pages 125–138, Saint-Malo, 2008. LNCS 5278.

[KMR+94] Michael J. Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. On the learnability of discrete distributions. In *Proceedings 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 273–282, 1994.

[Koh85] Teuvo Kohonen. Median strings. *Information Processing Letters*, 3 :309–313, 1985.

[KV89] Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings 21st ACM Symposium on Theory of Computing*, pages 433–444, 1989.

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.

[KY95] Satoshi Kobayashi and Takashi Yokomori. On approximately identifying concept classes in the limit. In *Processing 6th International Conference on Algorithmic Learning Theory (ALT)*, pages 298–312, Fukuoka, 1995. LNCS 997.

[Lan99] Kevin J. Lang. Faster algorithms for finding minimal consistent DFAs. Technical report, NEC Research Institute, 1999.

[Lem00] Kjell Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, University of Helsinki, Finland, November 2000.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8) :707–710, 1965. Original in *Doklady Akademii Nauk SSSR* 163(4) : 845–848 (1965).

[LPC98] Kevin J. Lang, Barak A. Pearlmutter, and François Coste. The Gowachin automata learning competition.
http ://www.irisa.fr/gowachin/, 1998.

[LPP98] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Proceedings 4th International Conference on Grammatical Inference (ICGI)*, pages 1–12, Ames, 1998. LNCS 1433.

[LTZ96] Daniel Lopresti, Andrew Tomkins, and Jiangying Zhou. Algorithms for matching hand-drawn sketches. In *Proceedings 5th International Workshop on Frontiers in Handwriting Recognition (FHR)*, pages 233–238, 1996.

[LU00] Kjell Lemström and Esko Ukkonen. Including interval encoding into edit distance based music comparison and retrieval. In *Proceedings Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science (AISB)*, pages 53–60, 2000.

[Luc04] Simon Lucas. Learning Dfa from noisy data competition. http ://cswww.essex.ac.uk/staff/sml/gecco/noisydfa.html, 2004.

[LV91] Ming Li and Paul M. B. Vitányi. Learning simple concepts under simple distributions. *SIAM Journal of Computing*, 20 :911–935, 1991.

[Mae91] Maurice Maes. Polygonal shape recognition using string-matching techniques. *Pattern Recognition*, 24(5) :433–440, 1991.

[Mai77] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25 :322–336, 1977.

[MBD05] Laurent Miclet, Sabri Bayoudh, and Arnaud Delhay. Définitions et premières expériences en apprentissage par analogie dans les séquences. In *Conférence francophone sur l'apprentissage automatique (CAp)*, pages 31–48, Nice, 2005. PUG.

[Mel95] Borivoj Melichar. Approximate string matching by finite automata. In *Proceedings 6th International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 342–349. LNCS 970, 1995.

[MHJC00] Carlos D. Martínez-Hinarejos, Alfons Juan, and Francisco Casacuberta. Use of median string for classification. In *Proceedings 15th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 2903–2906, 2000.

[MOV94] Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1) :9–17, 1994.

[MP80] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1) :18–31, 1980.

[MP83] William J. Masek and Mike Paterson. In *How to compute string-edit distances quickly*, pages 337–349. Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, 1983.

[MS01] Yasuhito Mukouchi and Masako Sato. Refutable language learning with a neighbor system. In *Proceedings 12th International Conference on Algorithmic Learning Theory (ALT)*, pages 267–282, Washington, 2001. LNCS 2225.

[MS02] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *Proceedings 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 251–263, Siguenza, 2002. LNCS 2473.

[MSMO03] Francisco Moreno-Seco, Luisa Micó, and Jose Oncina. A modification of the LAESA algorithm for approximated k-NN classification. *Pattern Recognition Letters*, 24(1-3) :47–53, 2003.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1) :31–88, 2001.

[Ofl96] Kemal Oflazer. Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1) :73–89, 1996.

[OG92] Jose Oncina and Pedro García. Identifying regular languages in polynomial time. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Series in Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[OG94] Kemal Oflazer and Cemaleddin Güzey. Spelling correction in agglutinative languages. In *Proceedings 4th Applied Natural Language Processing Conference (ANLP)*, pages 194–195, Stuttgart, 1994.

[OR07] Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *Journal of the ACM*, 54(5), 2007.

[OS01] Arlindo L. Oliveira and João P. Marques Silva. Efficient algorithms for the inference of minimum size DFAs. *Machine Learning*, 44(1) :93–119, 2001.

[OS06] Jose Oncina and Marc Sebban. Learning stochastic edit distance: application in handwritten character recognition. *Pattern Recognition*, 39(9) :1575–1587, 2006.

[Pap94] Christos M. Papadimitriou. *Computational Complexity*. Addison–Wesley, New York, 1994.

[Paw90] Zdzislaw Pawlak. Theory of rough sets: a new methodology for knowledge discovery (abstract). In *Proceedings International Conference on Computing and Information (ICCI)*, page 11, Niagara Falls, 1990. LNCS 468.

[PH97] Rajesh J. Parekh and Vasant Honavar. Learning DFA from simple examples. In *Proceedings Workshop on Automata Induction, Grammatical Inference, and Language Acquisition (ICML)*, pages 116–131, 1997.

[Pit89] Leonard Pitt. Inductive inference, DFA's, and computational complexity. In *Analogical and Inductive Inference*, number 397 in LNAI, pages 18–44. Springer-Verlag, 1989.

[PR96] Fernando C. N. Pereira and Michael Riley. Speech recognition by composition of weighted finite automata. *CoRR*, cmp-lg/9603001, 1996.

[PV88] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4) :965–984, 1988.

[PW88] Leonard Pitt and Manfred K. Warmuth. Reductions among prediction problems: on the difficulty of predicting automata. In *3rd Conference on Structure in Complexity Theory*, pages 60–69, 1988.

[Ric00] Claus Rick. Simple and fast linear space computation of longest common subsequences. *Information Processing Letters*, 75(6) :275–281, 2000.

[RJM03] Juan R. Rico-Juan and Luisa Micó. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. *Pattern Recognition Letters*, 24(9-10) :1417–1426, 2003.

[RY96] Eric S. Ristad and Peter N. Yianilos. Learning string edit distance. *CoRR*, cmp-lg/9610005, 1996.

[Sak90] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76 :223–242, 1990.

[Sak91] Yasubumi Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters*, 37(5) :279–284, 1991.

[Sak93] Yasubumi Sakakibara. Noise-tolerant occam algorithms and their applications to learning decision trees. *Machine Learning*, 11 :37–62, 1993.

[San92] David Sankoff. Edit distances for genome comparisons based on non-local operations. In *Proceedings 3rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 121–135, Tucson, 1992. LNCS 644.

[SG86] Jeffrey C. Schlimmer and Richard H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3) :317–354, 1986.

[SHY+07] Xueshen Sui, Qinghua Hu, Daren Yu, Zongxia Xie, and Zhongying Qi. A hybrid method for forecasting stock market trend using soft-thresholding de-noise model and svm. In *Proceedings 11th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC)*, pages 387–394, Toronto, 2007. LNCS 4482.

[SJ03] Marc Sebban and Jean-Christophe Janodet. On state merging in grammatical inference: a statistical approach for dealing with noisy data. In *Proceedings 20th International Conference on Machine Learning (ICML)*, Washington, 2003. AAAI Press.

[SJT04] Marc Sebban, Jean-Christophe Janodet, and Frédéric Tantini. Blue*: a blue-fringe procedure to learn DFA from noisy data. A parallel event to the Genetic and Evolutionary Computation Conference, 2004.

[SJY02] Marc Sebban, Jean-Christophe Janodet, and Aziz Yahiaoui. Effets de la suppression des mots bruités en inférence grammaticale. In *Conférence francophone sur l'apprentissage automatique (CAp)*, pages 311–314, Orléans, 2002. PUG.

[SK83] David Sankoff and Joseph B. Kruskal. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley Publishng Company, Reading, MA, 1983.

[SM02] Klaus U. Schulz and Stoyan Mihov. Fast string correction with Levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1) :67–85, 2002.

[SMT02] Masako Sato, Yasuhito Mukouchi, and Mikiharu Terada. Refutable/inductive learning from neighbor examples and its application to decision trees over patterns. In *Final Report of the Japanese Discovery Science Project*, pages 201–213. LNCS 2281, 2002.

[SN98] Lorenza Saitta and Filippo Neri. Learning in the "real world". *Machine Learning*, 30(2-3) :133–163, 1998.

[SS07] Dana Shapira and James A. Storer. Edit distance with move operations. *Journal on Discrete Algorithms*, 5(2) :380–392, 2007.

[Ste97] Frank Stephan. Noisy inference and oracles. *Theoretical Computer Science*, 185 :129–157, 1997.

[Tak88] Yuji Takada. Grammatical inference for even linear languages based on control sets. *Information Processing Letters*, 28(4) :193–199, 1988.

[TC04] Frank Thollard and Alexander Clark. Pac-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, 5 :473–497, May 2004.

[TDdlH00] Frank Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proceedings 17th International Conference on Machine Learning (ICML)*, pages 975–982, Stanford University, 2000. Morgan Kaufmann 2000.

[TdlHJ06] Frédéric Tantini, Colin de la Higuera, and Jean-Christophe Janodet. Identification in the limit of systematic-noisy languages. In *Proceedings 8th International Colloquium on Grammatical Inference (IGCI)*, pages 19–31, Tokyo, 2006. LNCS 4201.

[Tir08] Cristina Tirnauca. A note on the relationship between different types of correction queries. In *Proceedings 9th International Conference on Grammatical Inference (ICGI)*, pages 213–223, Saint-Malo, 2008. LNCS 5278.

[TK07] Cristina Tirnauca and Timo Knuutila. Polynomial time algorithms for learning k-reversible languages and pattern languages with correction queries. In *Proceedings 18th International Conference on Algorithmic Learning Theory (ALT)*, pages 264–276, Berlin, 2007. LNCS 4754.

[Ukk83] Esko Ukkonen. On approximate string matching. In *Proceedings of the 1983 International Fundamentals of Computation Theory Conference (FCT)*, pages 487–495, Borgholm, 1983. LNCS 158.

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3) :100–118, 1985.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11) :1134–1142, 1984.

[War89] Manfred K. Warmuth. Towards representation independence in *pac*-learning. In *Proceedings International Workshop on Analogical and Inductive Inference (AII)*, pages 78–103. LNAI 397, 1989.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21 :168–178, 1974.

Table des figures

Liste des Algorithmes

Résumé

L'inférence grammaticale s'intéresse à l'apprentissage automatique de langages formels. Ces derniers sont organisés en plusieurs classes formant la hiérarchie de Chomsky. Parmi elles, les langages réguliers, reconnus par des automates finis déterministes, forment la classe la plus « simple » à apprendre : l'apprentissage des automates a largement été étudié et a donné naissance à plusieurs algorithmes d'inférence grammaticale.

Toutefois, un problème concernant les données est devenu crucial : celui du bruit. Des propositions d'algorithmes ont vu le jour pour essayer de résoudre ce problème, mais nous montrons que les résultats ne sont toujours pas satisfaisants, y compris pour les langages réguliers. Or, puisqu'ils forment la base de la hiérarchie de Chomsky, ce sont toutes les classes de la hiérarchie qui ne peuvent être apprises en situations bruitées.

Aussi, nous proposons une nouvelle classe de langages qui semble ne pas souffrir de ce handicap : celle des boules de mots. Nous démontrons que cette classe, de prime abord peu orthodoxe mais utilisée dans de nombreuses applications comme la correction orthographique ou la recherche de plus proches voisins, reste identifiable à la limite même lorsque les données d'apprentissage subissent l'influence d'un bruit non statistique.

De plus, nous introduisons les requêtes de correction basées sur la distance d'édition et nous présentons un algorithme d'apprentissage des boules de mots à partir de telles requêtes. Nous montrons expérimentalement que de simples heuristiques *a posteriori* suffisent à le rendre résistant lorsque l'oracle répond approximativement à de telles requêtes. Ceci justifie encore une fois la robustesse des boules de mots au bruit.

Contrairement aux idées reçues, le bruit n'est donc pas une malédiction en inférence grammaticale : les langages à base de distance offrent de nouvelles perspectives.

Abstract

Grammatical Inference is concerned with learning formal languages. Formal languages are organised into classes, regular languages being the components of the simplest class. A lot of work has been done studying regular languages and learning algorithms have been successfully developped.

However, a crucial point must be tackled now: noisy data. Some propositions have been made in this direction but we show that none are entirely convincing, even for regular languages. Therefore, as they are the foundations of the Chomsky hierarchy, no element of the hierarchy is learnable from noisy data.

In this work, we propose to study a new class of languages which does not seem to suffer of this drawback: the balls of strings. We prove that this unusual class, which is in fact already used in works such as spelling correction or nearest neighbor search, is still learnable even from non-statistical noisy data.

Moreover, we introduce the edit correction queries, based on edit distance, and propose an algorithm learning balls of strings with such queries. If the oracle is allowed to answer approximate corrections, we show that simple heuristics are enough to design a robust algorithm, giving further evidence that balls of strings can be learned from noisy data.

Contrary to popular belief, noise is not a curse in grammatical inference: distance-based languages open new perspectives.