Apprentissage de langages formels : théorie et pratique Stage de fin de licence

Jean-Marie Madiot
sous la direction de Rémi Eyraud

Juin, juillet 2009
LIF de Marseille, équipe BDAA CMI

Remerciements

Je remercie chaleureusement Rémi Eyraud qui a fait preuve de persévérance et de patience envers moi. Je remercie Alexander Clark, qui m'a épargné quelques heures d'implémentation. Je remercie aussi Amaury Habrard et toute l'équipe BDAA du CMI qui rendèrent ce stage possible et agréable.

Je remercie le LIF de Marseille pour ses séminaires et ses petits fours, ainsi que Loïc Blet et Lucien Capdevielle avec qui j'ai passé une bonne partie de mon temps à Marseille.

Table des matières

Introduction

L'apprentissage de langages naturels et un problème actuel, difficile notamment car il ne s'agit pas uniquement d'un problème formel, mais aussi linguistique. Les classes de langages que l'on sait apprendre (par exemple, les langages rationnels) sont pour l'instant trop peu expressives pour permettre d'appréhender les langues naturelles. On tente de nouvelles approches qui se distinguent de la hiérarchie de Chomsky, en gardant comme objectif l'apprentissage automatique.

Les CBFG [4] sont un type de grammaires formelles développé récemment. Ce formalisme relie la structure du langage généré à des caractéristiques observables, ce qui confère un moyen d'analyse à un algorithme d'apprentissage. On s'intéressera ici principalement à un algorithme d'apprentissage de langages qui utilise ce formalisme comme représentation de langages, on s'intéressera plus particulièrement aux performances de cet algorithme face à une classe de langages qui fait actuellement l'objet de recherche en apprentissage, celle des langages hors-contexte.

Les CBFG sont un concept nouveau et malgré les efforts de communication, il reste difficile d'appréhender complètement le formalisme, c'est pourquoi on commencera par la présentation de ce formalisme. Les définitions héritent d'un formalisme (les BFG) à peine plus ancien que les CBFG. Les ressemblances avec les grammaires hors-contexte sont trompeuses ; par exemple, les CBFG ne sont pas génératives alors qu'on utilise, à l'instar des grammaires hors-contexte, la flèche de production $\rightarrow$.

Les deux premières parties du rapport (*Définition et notations*, puis *CBFG*) consiste en l'étude bibliographique du stage, et plus particulièrement sur les éléments de l'état de l'art nécessaires à la lecture de la partie suivante. Celle-ci (*Expérimentations*) rapporte l'apport personnel effectué pendant le stage, c'est à dire la mise en place du protocole d'expérimentation et l'étude du comportement de l'algorithme d'apprentissage face à des données qu'il faut générer.

1 Définitions et notations

Un alphabet Σ est un ensemble de lettres. Un langage est un ensemble de mots de Σ^*. L'apprentissage d'un langage L consiste, à partir d'une suite de mots (w_1, w_2, ..., w_n, ...) de L (mots positifs) ou bien une suite de mots de Σ^* et d'étiquettes dans $\{0, 1\}$: (x_1, y_1), (x_2, y_2), ... (mots positifs et négatifs), à construire une représentation d'un langage aussi proche de L que possible.

1.1 Exemples positifs et négatifs

Nombre d'algorithmes d'apprentissage en général ont besoin à la fois d'exemples positifs et d'exemples négatifs, pour ne pas sur-généraliser. En effet, comment, sans exemple négatif, distinguer le langage listant tous les mots vus (apprentissage par cœur) du langage Σ^* de tous les mots ? (sur-généralisation). Ainsi, l'apprentissage par exemples positifs seuls reste une tâche difficile. Par exemple,

la simple classe des langages rationnels n'est pas ainsi apprenable. Alors qu'avec les exemples positifs et négatifs, RPNI [5] l'apprend polynomialement

Cependant on peut se demander si cette approche convient à imiter un apprentissage naturel (celui d'une langue maternelle), car en effet, un enfant ne semble pas avoir besoin d'exemples négatifs pour apprendre sa langue maternelle. Il ne dispose pas d'exemple de mauvaises phrases.

1.2 Utilisation d'un oracle

On se place alors dans le cadre d'un apprentissage de langages formels à partir d'exemples positifs seulement. Cependant, il est peu probable d'obtenir un résultat exact (C'est-à-dire le langage cible est exactement le langage atteint) car les résultats d'apprentissage à partir d'exemples positifs sont généralement basés sur la distribution des exemples, ce qui est difficile à appliquer aux langages. On autorise alors les requêtes d'appartenance au langage (Membership Queries) : « Est-ce que ce mot appartient au langage ? » ce qui se rapproche de l'apprentissage de langue maternelle.

L'algorithme d'apprentissage dispose donc d'une suite potentiellement infinie de mots du langages et d'un oracle pouvant déterminer si oui ou non un mot appartient au langage.

1.3 Inférence grammaticale

On se place dans le cadre de l'apprentissage d'une classe de langages formels par un algorithme qui fournit une grammaire, G. G n'est pas forcément une grammaire hors-contexte ni même une grammaire au sens de celles de la hiérarchie de Chomsky, mais d'une CBFG.

On note :
- G une grammaire, et $L(G)$ le langage qu'elle reconnaît.
- w_i le $i$ème mot de la suite d'apprentissage
- $\mathcal{O}$ un oracle
- $\mathcal{A}$ un algorithme d'apprentissage
- G_i la grammaire $\mathcal{A}((w_1, ..., w_i), \mathcal{O})$ générée par l'algorithme avec pour entrée l'oracle et les i premiers mots de la suite.

2 CBFG

2.1 Besoin

Dans le cadre de la linguistique, un problème récurrent est l'apprentissage de langages, et plus particulièrement l'apprentissage de langages naturels. La plupart des langages naturels ne sont pas des langages réguliers (au sens formel du terme), et il faut appréhender une plus grande classe de langages. De même, quelques structures des langages naturels mettent en évidence un caractère contextuel des langages naturels. On ne se restreint pas spécialement

aux langages hors-contexte, mais à une classe incomparable avec celle des langages hors-contexte.

Certains formalismes bien fondés théoriquement existent pour apprécier de telles classes mais ces formalismes ne se prêtent pas bien à l'apprentissage. De même, beaucoup de résultats d'apprentissage intéressants portent sur des langages difficilement caractérisable par la théorie des langages.

Pour l'apprentissage, il est nécessaire de s'appuyer sur des caractéristiques observables du langages. Ainsi, pour un solide fondement théorique, on voudrait lier ces caractéristiques observables à la représentation du langage. Ainsi, un tel langage peut être défini d'après ses caractéristiques observables, ce qui forme une classe de langages prometteuse du point de vue de l'apprentissage, tout en conservant une garantie théorique.

Le formalisme des CBFG permet ainsi de considérer les contextes des mots d'un langage en tant que structure du langage. En effet, les contextes sont une caractéristique facilement observable d'un langage.

2.2 Contextes

Les contextes sont une partie importante du formalisme des CBFG. On remarque d'abord certains résultats sur les contextes, qui permettront de mieux appréhender le formalisme. Par exemple, l'équivalence de contextes est compatible avec la concaténation, c'est donc une relation de congruence. Si u et u' ont les mêmes contextes et que v et v' ont aussi les mêmes contextes, alors uv et $u'v'$ également.

Lemme 1 *Pour tout langage L et pour tous mots u, u', v, v', si $C(u) = C(u')$ et $C(v) = C(v')$, alors $C(uv) = C(u'v')$.*

On a un résultat un peu plus fort :

Lemme 2 *Pour tout langage L et pour tous mots u, u', v, v', si $C(u) \subseteq C(u')$ et $C(v) \subseteq C(v')$, alors $C(uv) \subseteq C(u'v')$.*

Et on en déduit alors que pour tout ensemble de mots K :

Corollaire 3

$$C(w) \supseteq \bigcup_{\substack{u',v':\\ u'v'=w}} \bigcup_{\substack{u\in K:\\ C(u)\subseteq C(u')}} \bigcup_{\substack{v\in K:\\ C(v)\subseteq C(v')}} C(uv)$$

2.3 Définitions

Définition 4 *Une CBFG (Contextual Binary Feature Grammar) G est un quadruplet $\langle F, P, P_L, \Sigma\rangle$, où*

- *F est un ensemble de contextes*
- *P est un ensemble fini de règles $x \to yz$ où x, y, z sont des éléments de $\mathcal{P}(F)$*

– P_L *est un ensemble de règles* $x \to c$ *avec* $x \in \mathcal{P}(F)$ *pour chaque* c *dans* Σ

Une CBFG G définit ainsi récursivement une fonction f_G de Σ^* vers $\mathcal{P}(F)$:

$$f_G(\lambda) = \emptyset \tag{1}$$

$$f_G(a) = \bigcup_{(x \to a) \in P_L} x \qquad \text{si } |a| = 1 \tag{2}$$

$$f_G(w) = \bigcup_{u,v:uv=w} \bigcup_{\substack{x \to yz \in P: \\ y \subseteq f_G(u) \wedge \\ z \subseteq f_G(v)}} x \qquad \text{si } |w| > 1. \tag{3}$$

Les CBFG sont une application directe du formalisme des BFG (*Binary Feature Grammar* [4]) ; on choisit ici de ne présenter que les CBFG.

Les CBFG sont des BFG dont les *features* sont notés comme des contextes des mots du langage. Cependant, *features* n'ont pas de sens *a priori*. Les relier aux contextes n'a vraiment de sens que quand les mots qui possèdent un *feature* de contexte (a, b) apparaissent effectivement avec le contexte (a, b) dans le langage. On s'aide alors du corollaire 3 pour respecter la structure du langage.

Définition 5 *Étant donné un ensemble de contextes* F *et un langage* L*, on définit* F_L *(context feature map) ainsi :* $F_L(u) = \{(l, r) \in F \mid lur \in L\}$.

$F_L(u)$ est l'ensemble des contextes de F du mot u (dans L).

Définition 6 *Une CBFG* G *est* exacte *si, pour tout mot* u*,* $f_G(u) = F_{L(G)}(u)$*, c'est à dire si les éléments de* F *dans le langage représenté par la grammaire sont les contextes effectivement associés à* u *par la grammaire.*

Par exemple, si (λ, b) est dans F, et est bien un contexte de u, il faut pour cela que (λ, b) soit aussi dans $f_G(u)$, donc qu'il existe un découpage de u selon les règles de P pour obtenir ce contexte.

Le caractère exact de certaines CBFG donne un sens plus évident aux contextes : ils font alors partie intégrante de la structure du langage. Ainsi, pour les langages reconnaissable par une CBFG exacte, on peut espérer déduire d'informations visibles (les contextes) une structure qui permettra de caractériser les mots du langage. [4] montre que les langages reconnus par une CBFG exacte sont apprenables.

2.4 Différence avec les grammaires hors-contexte

Les CBFG possèdent une puissance que ne possèdent pas les grammaires hors-contexte.

Dans une grammaire hors-contexte, la règle de production $A \to BC$ signifie que si u et v peut être dérivés respectivement par B et C, alors uv peut être dérivé par A. Ensuite, un mot w est dans le langage s'il peut être dérivé par le non-terminal racine (S).

Dans une CBFG, les règles de production sont de la forme $x \to yz$, ou x, y, z sont des *ensembles* de contextes. La règle signifie que si u est associé à tous les contextes présents dans y et v tous ceux de z, alors uv est associé à tous les contextes présents des x. Ensuite, un mot w est dans le langage s'il est associé au contexte vide (λ, λ).

Ce formalisme permet, par rapport aux grammaires hors-contexte, de créer en quelque sorte des intersections en plus des unions (que les grammaires hors-contexte effectuent en ajoutant plusieurs règles de même partie gauche).

Par exemple, si a, b, c, d sont des contextes, on peut utiliser la règle suivante :

$$\{a\} \to \{b\} \cdot \{c, d\}$$

pour indiquer que, pour avoir le contexte a, un mot doit être composé d'un facteur ayant pour contexte b et d'un facteur ayant à la fois les contextes c et d. Ainsi, on peut se permettre grâce aux CBFG des conjonctions de conditions de contextes.

Par exemple, si A, C, D sont des ensembles de contextes correspondant aux sous-langages respectifs $a^*b^nc^n$, $a^nb^nc^*$, $\{d\}$, on peut procéder à une intersection en plus d'une concaténation ainsi :

$$S \to A \cup C \cdot D$$

pour obtenir le langage contextuel $a^nb^nc^nd$.

Les langages reconnus par une CBFG sont d'ailleurs exactement les langages reconnus par une *conjunctive grammar* [6], c'est à dire une grammaire hors-contexte à laquelle on rajoute un opérateur de conjonction & qui permet à la classe d'être stable par intersection. Il est important de noter qu'on s'intéresse surtout aux CBFG exactes qui, même si elles possèdent une partie de la puissance de la conjonction, forment une classe plus réduite.

On note ainsi qu'une union de contextes à gauche de la flèche correspond à une union de dérivation, et qu'une union de contextes à droite correspond à une intersection.

2.5 Parsing

Par définition, un mot w est dans le langage reconnu par G ssi $(\lambda, \lambda) \in f_G(w)$.

Pour savoir si un mot w est dans $L(G)$, il faut donc calculer $f_G(w)$. On le sépare alors en deux facteurs u et v de toutes les façons possibles et on combine $f_G(u)$ et $f_G(v)$ selon les règles de productions. En fait, comme pour le parsing de mots d'une grammaire hors-contexte, on calcule d'abord pour les facteurs de taille 1, puis de taille 2, jusqu'à arriver au seul facteur de taille $|w|$, w. L'algorithme de cette analyse ascendante pour les grammaires hors-contexte est l'algorithme CKY.

À grammaire G fixée, la complexité du calcul d'appartenance de w à $L(G)$ est donc en $O(|w|)^3$, comme pour les grammaires hors-contexte.

La combinaison de $f_G(u)$ et de $f_G(v)$ n'arrive que quand il existe une règle $x \to yz$ avec $y \subseteq f_G(u)$ et $z \subseteq f_G(v)$, dans ce cas, on ajoute x à $f_G(uv)$.

2.5.1 Exemple

Le langage $L = \{a^n b^n\}$ est reconnaissable par une CBFG exacte.
Avec
- $F = \{(\lambda, \lambda), (a, \lambda), (aab, \lambda), (\lambda, b), (\lambda, abb)\}$.
- $P_L = \{\{(\lambda, b), (\lambda, abb)\} \to a\}, \{(a, \lambda), (aab, \lambda)\} \to b\}\}$
- $P = \{$

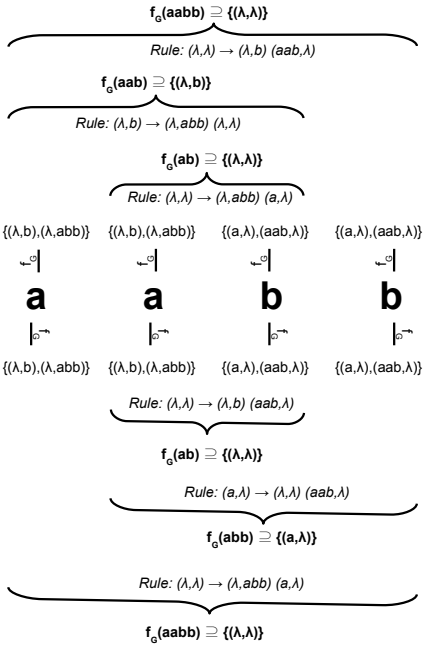
$$\{(\lambda, \lambda)\} \to \{(\lambda, abb)\}\{(aab, \lambda)\} \quad (4)$$
$$\{(\lambda, \lambda)\} \to \{(\lambda, b)\}\{(aab, \lambda)\} \quad (5)$$
$$\{(\lambda, \lambda)\} \to \{(\lambda, abb)\}\{(a, \lambda)\} \quad (6)$$
$$\{(\lambda, b)\} \to \{(\lambda, b)\}\{(\lambda, \lambda)\} \quad (7)$$
$$\{(a, \lambda)\} \to \{(\lambda, \lambda)\}\{(a, \lambda)\} \quad (8)$$

$\}$,
$G = \langle F, P_L, P, \{a, b\}\rangle$ reconnaît L.

On voit ici que dans toutes les règles de productions, les ensemble de contextes à droite des flèches sont des singletons. Il est donc possible, en remplaçant chaque singleton par un non-terminal arbitraire, et dupliquant les règles selon l'ensemble des contextes de gauche (dans P_L), d'extraire de cette CBFG une grammaire hors-contexte.

On note également que si les règles (5) et (7) ne sont pas indispensables pour reconnaître effectivement le langage $a^n b^n$, elles permettent à la CBFG d'être exacte.

Voici un exemple de *parsing* du mot *aabb*.



2.6 Classe de langages

La classe des langages reconnue par les CBFG est transversale à la hiérarchie de Chomsky. En effet, elle contient les langages réguliers ($(ab)^*$), les langages hors-contexte ($a^n b^n$), et certains langages contextuels (comme $a^n b^n c^n d$).

La classe des langages reconnue par les CBFG exactes est strictement plus petite. Par exemple, elle ne contient pas le langage $\{a^n b^m \mid n > m\} \cup \{a^n c\}$ [**?**].

Cependant ce contre-exemple n'invalide pas le fait de modéliser les langues naturelles par les CBFG exactes, puisque cette structure ne semble pas exister dans les langues naturelles.

Ainsi, la classe des langages hors-contexte n'est pas incluse dans celle des CBFG exactes. Ces classes sont d'ailleurs incomparables, puisque le langage des mots qui ont autant de a que de b et de c $\{w\,,\, |w|_a = |w|_b = |w|_c\}$ est reconnaissable par une CBFG exacte [4], et il n'est pas hors-contexte. (En effet, on peut l'intersecter avec le langage rationnel $a^*b^*c^*$ ce qui donnerait $a^n b^n c^n$, qui est l'exemple par excellence des langages non hors-contexte).

Par rapport aux précédents résultats d'apprenabilité, on remarque que la classe des CBFG exactes contient celle des *Very Simple Languages* [7] et celle des langages substituables [3]. La classe des CBFG exactes semble donc intéressante pour l'apprentissage.

2.7 Inférence

Pour apprendre une classe de langage on va inférer une CBFG, grâce aux mots positifs qui sont proposés dans l'échantillon d'apprentissage S, puis on combine l'ensemble des facteurs avec l'ensemble des contextes, pour former un nombre polynomial de mots en la taille de l'échantillon $|S|$, que l'on pourra proposer à l'oracle.

Définition 7 *Étant donné K, un ensemble de mots et F, un ensemble de contextes, soient :*
- *P_L, l'ensemble des $F_L(u) \rightarrow u$ avec $u \in K$, si $|u| = 1$*
- *P, l'ensemble des $F_L(uv) \rightarrow F_L(u)F_L(v)$, si $u, v, uv \in K$*

On définit alors $G_0(K, L, F) = \langle F, P, P_L, \Sigma \rangle$

On a construit la grammaire $G_0(K, L, F)$ a l'aide d'un oracle, d'un ensemble de mots K (pas forcément de L), et d'un ensemble de contextes. G_0 est une grammaire CBFG mais elle ne reconnaît pas forcément exactement L. On devine que le choix de F et K sur la pertinence de G_0 est important.

En fait, le langage reconnu par G_0 va augmenter quand K augmente. Ce premier résultat se déduit de la définition de f_{G_0}. En revanche, le langage $L(G_0)$ va diminuer selon F [4].

2.8 Algorithme d'apprentissage

L'algorithme d'apprentissage reste simple. Les requêtes d'appartenance sont effectuées par l'oracle $\mathcal{O}$.

Si D est un ensemble de mots, $Con(D)$ est l'ensemble des contextes des mots de D. $Sub(D)$ est l'ensemble des facteurs des mots de D. $Con(D) \odot Sub(D)$ est la combinaison cartésienne de ces deux ensembles.

Algorithme 1 : Algorithme d'apprentissage de CBFG

Données : Une suite de mots $S = \{w_1, w_2 \ldots, \}$, un oracle d'appartenance $\mathcal{O}$
Résultat : Une suite de CBFG $G_1, G_2, \ldots$
$K \leftarrow \emptyset$;
$F \leftarrow \{(\lambda, \lambda)\}$;
$D \leftarrow \emptyset$;
$G_0 = G_0(K, \mathcal{O}, F)$;
pour w_i **faire**
  $D \leftarrow D \cup \{w_i\}$;
  $S \leftarrow Con(D) \odot Sub(D)$;
  si $\exists w \in S$ *tel que* $w \in L(G_{i-1}) \setminus L$ **alors**
    $F \leftarrow Con(D)$;
  fin
  si $\exists w \in S$ *tel que* $w \in L \setminus L(G_{i-1})$ **alors**
    $K \leftarrow Sub(D)$;
    $F \leftarrow Con(D)$;
  fin
  Renvoyer $G_i = G_0(K, \mathcal{O}, F)$;
fin

L'algorithme, à chaque nouveau mot w_i, combine tous les facteurs et les contextes et propose les nouveaux mots obtenus à l'oracle et à la grammaire en cours.

- S'il existe un faux négatif (dans $L \setminus L(G_{i-1})$), il augmente F
- S'il existe un faux positif (dans $L(G_{i-1}) \setminus L$), il augmente F et K

Cet algorithme fonctionne en temps polynomial en la somme des tailles des exemples vus et nécessite un nombre polynomial de requêtes.

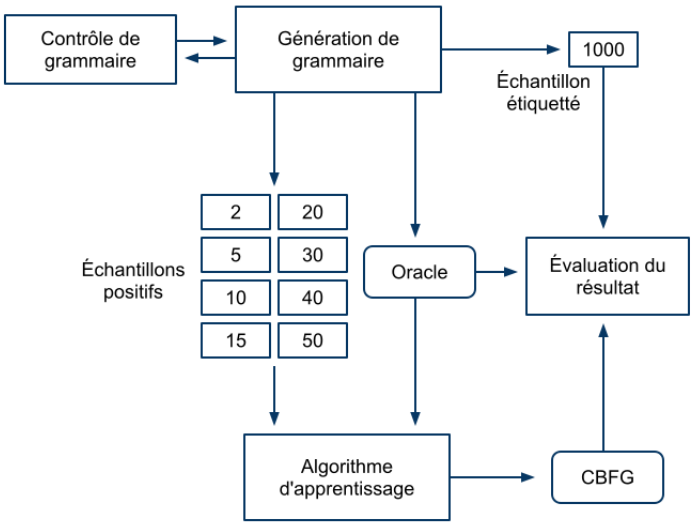
La convergence de cet algorithme est prouvée [4], mais on ignore pour l'instant son comportement sur des données réelles de langage. Ainsi, on décide d'observer cet algorithme sur des données générées automatiquement, avec la contrainte technique d'avoir à fournir un oracle.

3 Expérimentations

On s'intéresse à l'efficacité de l'algorithme d'apprentissage. Même si, d'après les résultats théoriques, un algorithme apprend exactement un langage avec des contraintes raisonnables sur les données, il est important de connaître son comportement empirique. En fonction du nombre d'exemples d'apprentissage, le langage appris évolue et il est intéressant de savoir si l'algorithme rejoint progressivement le langage cible ou si, tant qu'il n'est pas exactement le même, il en reste éloigné. C'est le comportement de RPNI : même s'il apprend exactement le langage, s'il n'a pas suffisamment d'exemples, le langage appris a de nombreuses différences avec le langage cible.

Ainsi on s'intéresse au comportement de l'algorithme d'apprentissage décrit plus avant. Il permet d'apprendre une classe de langage plus précise, celle des CBFG exactes.

3.1 Protocole d'expérimentation



On génère une grammaire hors-contexte G aléatoirement. Si elle n'est pas triviale selon quelques critères que l'on détaille par la suite, on génère alors plusieurs échantillons d'apprentissage S de tailles différentes $(2, 5, 10, 15, 20, 30, 40, 50)$ tel que qu'un échantillon plus grand est un sur-ensemble des échantillons plus petits. On pourra ainsi analyser le comportement d'une progression réelle.

On génère également un échantillon de test de 1000 mots positifs et négatifs, avec un taux minimal de positifs et de négatifs garanti, c'est à dire qu'il y a au moins 40% d'exemples positifs et 40% d'exemples négatifs dans chaque échantillon de test. C'est une contrainte arbitraire mais nécessaire pour évaluer le comportement sur-généralisant ou sous-généralisant de l'algorithme. Chaque échantillon de test ne contient pas de mots de l'échantillon d'apprentissage, car on sait que l'algorithme ne se trompe pas sur l'échantillon d'apprentissage.

Pour l'algorithme d'apprentissage, on génère un oracle $\mathcal{O}$, qui se transmet sous la forme d'une grammaire hors-contexte. Il s'agit de la grammaire G générée initialement.

L'algorithme d'apprentissage est alors lancé successivement sur l'entrée $(\mathcal{O}, S)$ pour chaque taille de S. La CBFG en sortie est alors testée sur l'échantillon de test et un résumé d'informations est sauvegardé, contenant le taux de réussite, le nombre de requêtes effectuées, le temps d'apprentissage, etc. On conserve suffisamment d'informations pour analyser correctement le comportement de l'algorithme.

3.2 Génération des données

3.2.1 Choix de la classe de test

Comme on revendique le choix des CBFG exactes dont la classe correspondante n'est pas incluse dans celle des langages hors-contexte, on aurait pu échantillonner les langages de test parmi les CBFG exactes. Cependant, il semble qu'il soit indécidable de déterminer l'exactitude d'une CBFG.

De plus, il est difficile de générer des mots d'une CBFG car ce formalisme n'est pas génératif, au contraire des grammaires hors-contexte. Même si cette contrainte est provisoirement levée car le procédé de sélection des exemples se fait par analyse syntaxique (parsing), la génération d'exemples d'une grammaire générative reste plus simple. Par exemple, on peut s'interroger sur la notion vue ci-après d'ensemble structurellement complet appliquée aux CBFG, mais elle reste en tout cas moins bien définie et plus difficile à générer que pour les grammaires hors-contexte.

Finalement, on choisit pour les expérimentations la classe des grammaires hors-contexte, car elle offre une variété intéressante au point de vue de l'apprentissage. En effet, il est toujours difficile d'apprendre un langage hors-contexte : les avancées au sein de cette classe consistent actuellement à étendre au fur et à mesure les sous-classes pour lesquelles on dispose d'un algorithme d'apprentissage.

De plus, l'utilisation et la génération de grammaires hors-contexte sont des processus connus et relativement faciles à implémenter, par rapport à la complexité de la classe représentée. Les résultats de décidabilité, d'expressivités sont connus. On dispose donc d'une connaissance générale de la classe avec laquelle on travaille et les résultats obtenus pourront plus facilement être comparés.

Le formalisme des grammaires hors-contexte est déjà très répandu, on travaille donc avec une classe de langages dont l'expressivité est reconnue. Une compétition d'apprentissage de langages, Omphalos [1], a eu lieu en 2004 et consistait en l'apprentissage de langages générés par des grammaires hors-contexte. On travaille donc avec la même classe de langages utilisée pour une compétition dans le cadre de l'ICGI (International Colloquium on Grammatical Inference).

3.2.2 Génération de grammaires

On préfère éviter de biaiser la génération de grammaire, quitte à obtenir un nombre non négligeable de grammaires triviales ou finies. Étant donnée la liste des terminaux et des non-terminaux, les sommets d'un graphes pour l'instant vide, on relie aléatoirement des paires et des triplets de sommets qui correspondent aux règles de production de la grammaire hors-contexte.

3.3 Analyse de grammaires

Pour éviter de tester l'algorithme sur des données peu intéressantes, comme des langages vides ou finis, on analyse la grammaire. Heureusement, déterminer si une grammaire hors-contexte G est vide ou même finie est un problème décidable.

Pour déterminer si une grammaire hors-contexte est finie, on part des règles de production terminales $A \to a$ et on parcours les règles $A \to BC$ pour arriver à S, le non-terminal racine, en se souvenant à chaque étape des terminaux, non-terminaux et règles empruntés. Il faut prendre garde à l'explosion combinatoire du nombre de terminaux/non-terminaux marqués pour chaque terminal/non-terminal, pour éviter un calcul excessivement lourd et gourmand en mémoire, à un tel point qu'il deviendrait impossible à terminer.

Ce qui est plus intéressant encore, c'est que l'analyse ainsi effectuée permet de déterminer quelles sont les règles, terminaux et non-terminaux utilisables depuis la racine S de la grammaire, pour pouvoir éliminer les éléments inutiles de la grammaire. Ainsi, on peut avoir une meilleure idée de la complexité de la grammaire cible lors de l'analyse des résultats. En effet, avoir une idée de la complexité d'une grammaire sans connaître le nombre de règles est finalement assez difficile à partir du moment où le langage qu'elle génère est infini.

En revanche, savoir si $L(G) = \Sigma^*$ est un problème indécidable. Cependant, comme le taux d'exemples négatifs dans l'échantillon de test est garanti, Σ^* ne sera jamais présenté à l'algorithme. En fait, Σ^* (avec $\Sigma^* \setminus \mathcal{L}$ où $\mathcal{L}$ est fini) est un des rares langages hors-contexte qui ne peut être généré avec les données étiquetées, le biais liées aux limitations dues aux taux minimums est donc limité.

3.4 Génération de mots

3.4.1 Ensemble structurellement complet

Définition 8 *Étant donné une grammaire hors-contexte G, un ensemble S de mots de Σ^* est structurellement complet si pour chaque règle r, il existe un mot w de S tel qu'il existe une dérivation de w qui utilise r.*

À partir de là, un échantillon d'apprentissage devrait clairement être un ensemble structurellement complet pour identifier exactement le langage. En effet, si une des règles n'est utilisée pour aucun des mots de l'échantillon, l'algorithme ne peut faire la différence avec la grammaire d'origine et la grammaire sans cette règle.

On décide alors d'ajouter aléatoirement dans les échantillons d'apprentissage une partie d'un ensemble structurellement complet prédéfini, de sorte que les échantillons de taille suffisante soient structurellement complet, sans forcément en disposer dans les échantillons de taille inférieure. De cette manière, il est certain qu'*au final*, un ensemble structurellement complet sera présenté à l'algorithme.

La taille de l'ensemble structurellement complet n'est pas bornée polynomialement en fonction de la taille de la grammaire. En effet, la grammaire suivante dont la taille est linéairement liée à n, n'admet que $\{a^{2^n}\}$ comme langage donc a un ensemble structurellement complet minimum de taille 2^n :

- $A_0 \to a$
- $A_1 \to A_0 A_0$
- $\dots$
- $A_{n-1} \to A_{n-2} A_{n-2}$
- $S \to A_{n-1} A_{n-1}$

Néanmoins, on peut déterminer un ensemble structurellement complet en un temps raisonnable. L'algorithme proposé ici est un algorithme d'exploration, comme les algorithmes d'analyse de grammaire vus précédemment.

Le principe utilisé est de mémoriser pour chaque non-terminal une liste de mots, chacun associés à une liste de non-terminaux qui peuvent être utilisés pour le dériver. Lors de la progression à travers une règle $A \to BC$ s'opère, on fusionne un à un les mots de B avec ceux de C, en fusionnant pour chaque les listes des non-terminaux. Pour éviter l'explosion combinatoire, on élimine arbitrairement certains couples (mot, liste de non-terminaux) pour conserver le même ensemble total de non-terminaux qui peuvent être empruntés. L'éliminination des couples est gloutone. Le choix optimal de ces couples semble NP-difficile.

3.4.2 Parsing

Pour générer des mots positifs, on tente dans un premier temps d'explorer le graphe des règles pour atteindre des mots. Cette méthode est générative et relativement rapide, en particulier dans le cas d'une grammaire avec des mots assez grands. Pour cela, on explore à partir des terminaux le graphe des règles jusqu'à atteindre la racine S. Il est aussi possible de partir de la racine et de suivre aléatoirement parmi les règles multiples correspondant à chaque non-terminal.

Cependant, cette méthode est plutôt lente et ne permet pas de générer des exemples négatifs. De plus, la distribution des mots obtenue seraient différente d'un échantillon dont la distribution a été choisie avant la connaissance de la grammaire.

On opte alors pour le parsing, l'analyse syntaxique de mots générés aléatoirement dans Σ^*. La génération de mots est aléatoire, d'une longueur qui s'agrandit progressivement au cours de la génération. On évite alors de rester bloqué à cause

d'un manque de mots positifs ou bien négatifs. Ensuite, l'analyse syntaxique est une analyse ascendante, encore grâce à l'algorithme CKY.

3.5 Implémentation

L'algorithme d'apprentissage a été implémenté en Java par Alexander Clark, sous la forme d'une console interactive dans laquelle on entre manuellement le noyau et les contextes qui serviront à la génération des mots. Ensuite, l'oracle est l'utilisateur décidant manuellement si oui ou non un mot présenté appartient au langage cible.

3.5.1 Automatisation de l'algorithme

En Java, on utilise le code source existant pour créer du bytecode capable de prendre un fichier en tant échantillon d'entraînement ainsi qu'un fichier en tant qu'oracle, stoqué sous la forme d'une BFG qui est dans ce cas également une grammaire hors-contexte. On peut alors assimiler les deux formats.

Le programme bytecode peut alors prendre en entrée trois fichiers, échantillons d'apprentissage et de test, et oracle, et produit en sortie un fichier de résultats.

3.5.2 Génération des données

La génération des grammaires puis de la génération des échantillons positifs puis étiquetés, ainsi que l'oracle a été implémentée en Objective Caml. D'une part pour générer des échantillons rapidement ; en effet, les programmes compilés en code natif avec `ocamlopt` sont plus rapide qu'exécutés grâce à une machine virtuelle Java. D'autre part, il est important qu'il y ait une trace dans des fichiers des échantillons d'apprentissage et de test, ainsi, le fait de se servir de fichier pour communiquer d'un processus à un autre n'est pas un inconvénient majeur. Enfin les outils d'analyse de grammaire et de génération d'échantillons, gourmands en manipulation de listes, de fonctions et même de types ont été plus facilement développés en programmation en partie fonctionnelle, sans compter une certaine habitude du langage.

3.5.3 Gestion des processus

On dispose alors d'un programme, qui prend en entrée des fichiers (échantillons et oracle), et qui donne en sortie les résultats de l'analyse et d'un autre qui génère à la demande ces échantillons et oracles.

Cependant, tant le processus d'apprentissage que le processus de génération des données est gourmand en ressources et en temps. Il faut parfois quelques heures à l'algorithme d'apprentissage pour apprendre sur toutes les tailles d'échantillon d'apprentissage et les tester sur l'échantillon de test. Il est nécessaire, afin de proposer des statistiques fiables, d'effectuer un grand nombre de tests. Il faut donc disposer de temps processeur et l'organiser. Il est donc nécessaire d'automatiser les tâches.

Langage On automatise d'une part l'expérience au niveau du langage. Un programme en langage de script, (en Bourne shell, celui d'Unix) lance la génération des échantillons. Une fois que les fichiers d'échantillons sont créés, le script lance successivement l'apprentissage sur les différentes tailles. L'intervention manuelle est alors de lancer l'expérience, à l'aide d'un identifiant d'expérience qui ne doit pas déjà exister.

Suite de langages Il n'est pas difficile de créer un processus qui lance successivement plusieurs autres avec des identifiants d'expériences uniques. Cependant, il arrive que le langage généré soit trop grand, ou trop petit, pour que le programme de génération de mots étiquettés ne parvienne pas à aboutir. Il est alors nécessaire d'établir un garde-fou automatique pour prévenir ce genre d'incident, ce qui fut fait grâce à la gestion d'exceptions (en continuation). Ainsi, on peut lancer un processus qui est censé tourner sans s'arrêter ni bloquer.

Processeurs Cependant, il est nécessaire que cette gestion soit effectuée à distance, en tirant parti d'une machine unix disposant de huit cœurs, donc potentiellement huit processus à tourner simultanément. Ainsi, toujours en bourne shell, on développe un programme qui mets à jour les données distantes des nouveaux bytecodes et programmes, et un autre qui lance plusieurs scripts à distance afin d'exploiter pleinement les capacités du calculateur. Cette méthode permet de lancer des processus sur une machine distante indépendante, qu'on peut faire tourner pendant plusieurs jours.

Les scripts se sont révélés suffisamment robustes pour entretenir les processus d'apprentissage pendant toute la durée attendue, c'est à dire environ un mois.

3.6 Résultats

Voici la courbe de la moyenne du pourcentage de reconnaissance exacte sur l'échantillon de test de 1000 mots, sur environ 700 expériences en fonction de la taille d'échantillon d'apprentissage.

On observe que l'algorithme a une progression de convergence de type exponentielle (en moyenne). À partir d'une taille d'échantillon d'apprentissage de 15 mots, on constate que le taux de reconnaissance est de plus de 90%.

Cependant, les résultats en moyenne sont seulement indicatif. La figure 1 est un peu trompeuse, car il s'agit d'une moyenne qui ne montre pas la répartition de résultats : certains langage engendrent un mauvais taux de réussite, alors que beaucoup d'autres sont très bien appris. En effet, avec 50 mots, environ la moitié des langages sont reconnus à 99% ou plus, et environ 10% des langages sont reconnus exactement. La donnée du pourcentage de langages qui sont reconnus avec plus d'un taux de réussite donné est plus descriptive (Figure 2).

On observe aussi qu'avec 50 mots en apprentissage, tous les langages sont reconnus à plus de 83%, et avec seulement 10 mots, 80% des langages sont appris à 80%. On rappelle que l'algorithme utilise un oracle.

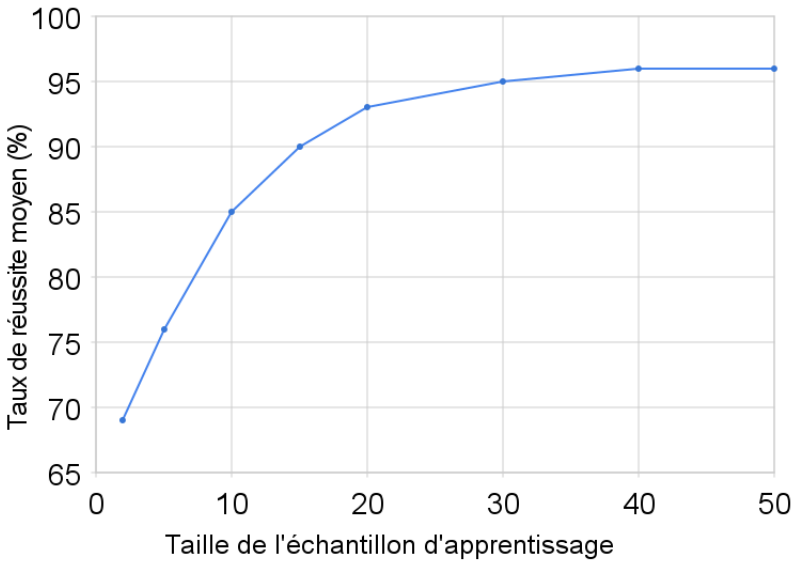


FIG. 1 – Évolution du taux de réussite moyen en fonction de $\#S$

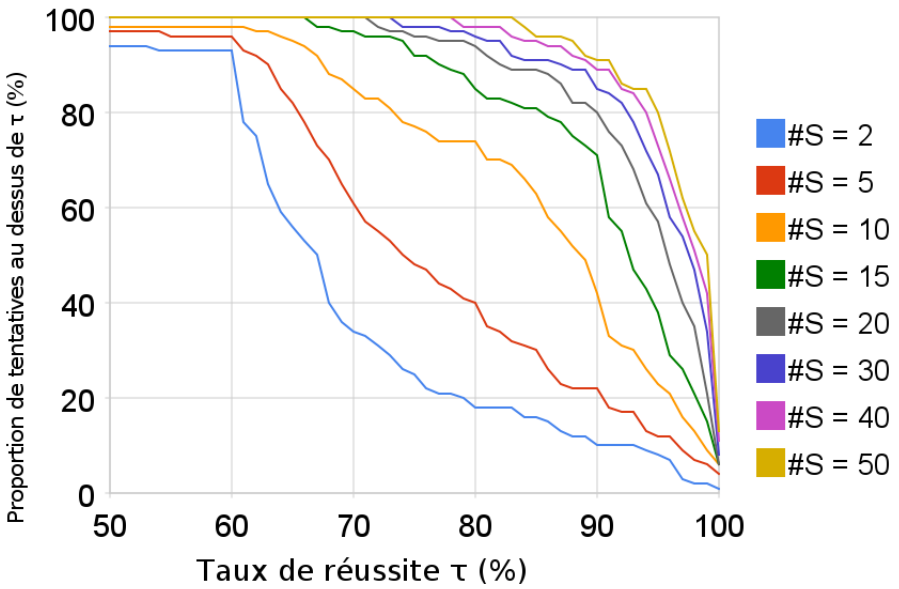


FIG. 2 – Proportion de langages appris en fonction d'un taux minimal d'apprentissage

On observe sur le tableau suivant le nombre de faux positifs, c'est à dire le nombre de mots étiquetés positifs par l'algorithme alors qu'il ne sont pas dans le langage, et le nombre de faux négatifs, étiquetés négatifs à tort :

$\#S$	faux positifs	faux négatifs	requêtes
2	2.3 %	28.1 %	19 053
5	3.3 %	20.0 %	83 559
10	2.4 %	12.1 %	123 394
15	2.6 %	6.5 %	297 041
20	1.6 %	4.6 %	527 242
30	1.1 %	3.6 %	1 148 859
40	0.69 %	3.0 %	2 092 354
50	0.55 %	2.5 %	2 619 895

L'algorithme est donc sous-généralisant, car on observe que le nombre de mots négatifs est nettement plus important que le nombre de mots positifs parmi l'échantillon de test de 1000 mots.

La sous-généralisation est un caractère recherché pour les algorithmes fonctionnant à partir d'exemples positifs. En effet, si un programme sur-généralise, le fait d'avoir des mots supplémentaires est difficilement exploitable, alors qu'un algorithme sous-généralisant qui reçoit des nouveaux exemples positifs s'en aide pour s'approcher du langage cible en augmentant.

Conclusion

L'algorithme exploitant le formalisme des CBFG est simple. Par exemple, il ne prend pas en compte le mot contre-exemple qui lui est fourni, mais plus généralement son existence et sa nature (faux positif ou faux négatif). Il est probablement possible d'améliorer les résultats en raffinant l'algorithme. En apprentissage, et même en inférence grammaticale, l'efficacité d'un algorithme est souvent améliorée grâce à des heuristiques.

Grâce aux données obtenues, il est peut-être possible d'améliorer l'algorithme en prenant en compte le fait qu'il semble plutôt sous-généralisant. Le nombre de requêtes est toutefois très important. C'est un problème gênant, mais l'algorithme testé est un algorithme naïf qui teste toute les possibilités de combinaisons contexte / facteur.

La comparaison avec d'autres algorithmes est difficile, car il existe peu d'algorithmes connus qui utilisent des exemples positifs en utilisant un oracle. En revanche, dans quelque temps, Zulu [2], une compétition du même type qu'Omphalos [1] mais pour des algorithmes à requêtes d'appartenance, va peut-être départager les algorithmes existants.

La première partie du stage était surtout bibliographique. Le formalisme des CBFG est déstabilisant et il faut un certain temps pour l'appréhender. Les objectifs du stage étaient soit de tenter de remplacer l'oracle, soit de déterminer avec plus de précision la classe de langages apprenable, soit d'expérimenter avec des données réelles de langage. Pour remplacer l'oracle, cependant, il faut d'abord étudier le comportement et l'efficacité de l'algorithme avec un oracle. L'étude avec des données réelles de langage est elle aussi difficile, car on a besoin

d'un oracle. La seconde partie du stage a concilié ces besoins, en étudiant le comportement de l'algorithme avec des données aléatoires.

Le travail en laboratoire fut enrichissant. On y découvre les difficultés des démarche administrative, les relations avec des réseaux d'excellence ou encore avec les parties extérieures du laboratoire. Par exemple, la journée du LIF a permis de découvrir certains champs de recherche des autres équipes du laboratoire.

Questions ouvertes

L'oracle, on l'a vu, peut être justifié d'un point de vue linguistique. Cependant, l'apprentissage automatique est une tâche qui se prête parfois difficilement à de un grand nombre de requêtes d'appartenances, par exemple, ce test ne peut être automatique, sinon on dispose *déjà* du résultat qu'on recherche.

On se demande alors s'il est possible de remplacer les requêtes d'appartenance (l'oracle) par une solution plus facile à implémenter. On pourrait penser à un test statistique, à partir de données déjà existantes.

On connaît assez bien la classe de langage reconnue par les CBFG, qui correspond à cette des *conjunctive grammars* [6]. Cependant, celle qui nous intéresse est celle des CBFG exactes. On sait qu'elle est incomparable avec celle des langages hors-contexte, mais elle reste encore assez floue.

Pour exploiter le concept des CBFG a des fins statistiques, il reste du travail à effectuer du côté du traitement de la langue, pour donner du sens à une dérivation par exemple. En effet, savoir si un mot est dans un langage est intéressant, mais donner une raison de classification, comme une dérivation, semble l'être encore plus.

Références

[1] Omphalos competition, 2004. http ://www.irisa.fr/Omphalos/.

[2] Zulu competition, 2010. http ://labh-curien.univ-st-etienne.fr/zulu.

[3] Alexander Clark and Rémi Eyraud. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8 :1725–1745, Aug 2007.

[4] Alexander Clark, Rémi Eyraud, and Amaury Habrard. A polynomial algorithm for the inference of context free languages. In *Proceedings of International Colloquium on Grammatical Inference*. Springer, September 2008.

[5] J. Oncina ; P. García. Inferring regular languages form positive and negative examples. In *Summer School on Machine Learning*, Urbino, Italia, august 1989.

[6] A. Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6 :4 :519–535, 2001.

[7] Takashi Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theor. Comput. Sci.*, 298(1) :179–206, 2003.