

# Méthodes d'ensemble en Inférence Grammaticale : une approche à base de moindres généralisés

Fabien Torre, Alain Terlutte

INRIA Mostrare et LIFL Grappa

**Résumé** : Nous présentons un cadre général pour la classification supervisée basé sur la notion de moindre généralisé (généralisation minimale d'un ensemble d'exemples). Nous montrons que le fait de définir cette opération dans l'espace d'hypothèses choisi met à notre disposition, sans effort supplémentaire, plusieurs algorithmes d'apprentissage supervisé à plusieurs classes, en particulier des méthodes d'ensemble (de *boosting* par exemple). Après avoir décrit cette architecture générique, nous expliquons son utilisation pour l'Inférence Grammaticale : le calcul du moindre généralisé y est un apprentissage à partir d'exemples positifs seuls comme ceux effectués par les algorithmes TSSI et ZR. Des classifications efficaces de séquences sont alors possibles par vote de plusieurs automates élémentaires. Des expérimentations valident notre approche et nous discutons des intérêts et implications de ces idées.

**Mots-clés** : Inférence Grammaticale, méthodes d'ensemble, moindres généralisés, classification supervisée.

## 1 Introduction

L'*Inférence Grammaticale* s'intéresse à l'identification de langages : il s'agit de trouver une grammaire ou un automate cible à partir de mots-exemples. L'Inférence Grammaticale est à ce titre une branche de l'apprentissage automatique, en particulier de l'apprentissage supervisé. Cela dit, beaucoup de développements lui sont très spécifiques, en particulier ses aspects théoriques : la nature des classifieurs (automates ou grammaires) partagés avec la théorie des langages, les modèles d'apprentissage (à la limite, par positifs seuls, etc.), la notion d'échantillon caractéristique, les résultats d'apprenabilité pour certaines familles de langages et les algorithmes d'apprentissage associés à ces preuves d'apprenabilité. Ces algorithmes constituent le volet pratique de l'Inférence Grammaticale. Dans le cas de l'induction d'automates à partir d'exemples positifs et négatifs, l'algorithme emblématique est RPNI (Oncina & García, 1992). Il utilise des opérations courantes comme la construction de l'automate PTA (*Prefix Tree Acceptor*) des exemples positifs et la fusion d'états contrôlée.

Cependant, nous voyons depuis quelques années une partie de la communauté se détourner de l'identification de langages pour travailler à la classification de séquences,

sans caractériser explicitement le langage sous-jacent dans un formalisme de la théorie des langages. Avec ce mouvement, les preuves d'apprenabilité laissent la place à des expérimentations exhibant des taux d'erreur, démarche classique en apprentissage automatique cette fois. Citons parmi ces travaux ceux qui s'appuient sur des techniques étrangères à l'Inférence Grammaticale comme les chaînes de Markov cachées (Callut & Dupont, 2006; Dupont, 2006) et d'autres qui ont une vision plus géométrique (Bonache *et al.*, 2007) et utilisent des méthodes à base de noyaux (Clark *et al.*, 2006). Dans ces nouveaux cadres, certains langages réputés difficiles en Inférence Grammaticale deviennent facilement apprenables, par exemple lorsqu'ils sont représentés comme des hyperplans dans des espaces aux dimensions pertinentes.

Ces travaux récents empruntent donc à l'apprentissage automatique des techniques pour classer des séquences, éventuellement en présence de bruit, et sans recourir à des notions familières de l'Inférence Grammaticale. Dans cet article, nous nous plaçons résolument à l'intersection de la classification supervisée et de l'Inférence Grammaticale et prétendons qu'il est possible d'intégrer des techniques à base d'automates dans un cadre générique ayant les avantages et la souplesse issues de la classification supervisée.

L'organisation de cet article est la suivante :

- la section 2 présente la notion de moindre généralisé, son utilisation pour élaborer une hypothèse correcte dans un cadre supervisé et enfin la mise en œuvre de ces hypothèses dans la construction de classifieurs complets ;
- ces algorithmes génériques sont instanciés dans la section 3 pour traiter de la classification de mots par automates moindres généralisés ; nous y verrons que l'Inférence Grammaticale nous fournit les algorithmes d'apprentissage dont nous avons besoin sous la forme d'apprenants par positifs seuls ;
- la section 4 présente des expérimentations et comparaisons de nos algorithmes à des implémentations classiques en Inférence Grammaticale, relativement aux avantages attendus de notre démarche ;
- enfin, la section 5 dresse un bilan de ce travail, le discute en termes d'originalité et avantages de la méthode, et en propose quelques perspectives.

## 2 Apprentissage à base de moindres généralisés

Cette section rappelle rapidement ce qu'est un moindre généralisé dans un cadre générique, c'est-à-dire sans faire de supposition sur les langages décrivant les exemples et les hypothèses. Nous montrons ensuite que la définition d'un moindre généralisé nous offre une large panoplie d'algorithmes, chacun structuré en trois niveaux :

- le premier niveau fournit l'opération permettant de calculer l'hypothèse moindre généralisée d'un ensemble d'exemples quelconque ;
- le deuxième prend en compte les classes des exemples pour produire des hypothèses correctes (c'est-à-dire qu'une hypothèse produite ne couvre pas d'exemples en dehors de sa propre classe) ;
- le dernier niveau permet l'apprentissage d'un classifieur complet, par combinaison de règles élémentaires apprises par le niveau précédent.

Nous allons décrire maintenant ces trois niveaux mais précisons immédiatement que les deux derniers niveaux sont purement génériques et que seul le premier niveau est à

définir en fonction des représentations choisies pour les exemples et les hypothèses. La section 3 décrira ce niveau pour le cas particulier des exemples-mots et des hypothèses-automates.

## 2.1 Calcul d'un moindre généralisé

Revenons rapidement sur la définition d'un problème de classification : doivent être donnés un espace  $\mathcal{E}$  permettant de décrire les exemples, un espace  $\mathcal{H}$  pour exprimer les hypothèses (c'est-à-dire les règles de classification) et enfin un test de subsomption entre une hypothèse et un exemple (pour décider si oui ou non une règle s'applique à un exemple). Depuis Feigenbaum (1977), il est admis que l'espace des exemples est inclus dans l'espace des hypothèses, si bien qu'un exemple est une hypothèse comme une autre et que la relation de subsomption structure l'espace des hypothèses.

Dès lors, il est aisé de définir un moindre généralisé d'un ensemble d'exemples : comme son nom l'indique, il s'agit d'une hypothèse maximale spécifiquement qui subsume tous les exemples donnés, *maximalement* signifiant qu'il n'existe pas d'hypothèse plus spécifique qui subsume ces mêmes exemples.

### Définition 1 (Moindre généralisé)

Étant donné un espace d'exemples  $\mathcal{E}$  et un espace d'hypothèses  $\mathcal{H}$  vérifiant  $\mathcal{E} \subseteq \mathcal{H}$ , étant donnée une relation de subsomption  $\succeq$  sur  $\mathcal{H} \times \mathcal{H}$ , étant donné un ensemble d'exemples  $E \subseteq \mathcal{E}$ , une hypothèse  $h \in \mathcal{H}$  est dite moindre généralisée de  $E$  si et seulement si :

- $\forall e \in E : h \succeq e$  ;
- il n'existe pas  $h'$  vérifiant  $(h \succ h') \wedge (\forall e \in E : h' \succeq e)$ .

Notons que cette définition n'implique pas l'unicité du moindre généralisé, il peut en exister plusieurs, incomparables entre eux par subsomption. Cependant, nous posons l'unicité du moindre généralisé dans les langages d'hypothèses que nous considérons comme *pré-requis nécessaire* à l'application de nos méthodes d'apprentissage.

Considérons à titre d'exemple le cas à deux dimensions. Il est possible d'y avoir des exemples-points ( $\mathcal{E}$ ), des hypothèses-rectangles ( $\mathcal{H}$ ) et un test de subsomption : une hypothèse-rectangle *subsume* un exemple si celui-ci est à l'intérieur du rectangle. Dans ce cas, le moindre-généralisé d'un ensemble de points-exemples est unique, il s'agit du plus petit rectangle qui contient tous ces points.

Nous le voyons, ce moindre généralisé offre un opérateur de type « plus petit pas » pour parcourir l'espace des hypothèses, indépendamment de la classe des exemples. Pour être parfaitement pratique, nous avons besoin d'un algorithme calculant le moindre généralisé d'un ensemble d'exemples, et si possible en version incrémentale. Autrement dit, des deux signatures possibles pour cet algorithme nous préférons la seconde :

- $\text{MG}(e_1, e_2, \dots, e_n \in \mathcal{E})$  returns  $h \in \mathcal{H}$  ;
- $\text{MG}(h_{n-1}, e_n)$  returns  $h \in \mathcal{H}$ .

Nous supposons donc disposer d'un opérateur incrémental non supervisé de généralisation défini sur l'espace des hypothèses ; l'étape suivante consiste à prendre en compte l'information de classes des exemples pour guider cet opérateur dans son parcours.

## 2.2 Moindres généralisés corrects

Dans une tâche supervisée, nous disposons des classes des exemples fournis pour l'apprentissage. Il est donc possible de calculer le moindre généralisé d'exemples d'une même classe, en prenant garde à ne pas y intégrer les exemples d'autres classes. Nous obtenons dans ce cas des *règles correctes*, mais peut-être *incomplètes*, par rapport à l'ensemble d'apprentissage. Le profil d'une telle méthode est :

$$\text{MGC} (E \subseteq \mathcal{E}, N \subseteq \mathcal{E}) \text{ returns } h \in \mathcal{H},$$

où  $E$  contient les exemples d'une même classe et  $N$  ceux des autres classes, la procédure elle-même est donnée par l'algorithme 1.

---

### Algorithm 1 MGC

---

**Entrées :**  $E = [e_1, \dots, e_n] \subseteq \mathcal{E}$  un ensemble *ordonné* de  $n$  exemples de la même classe,  $N$  un ensemble de contre exemples.

**Sortie :**  $h \in \mathcal{H}$  une généralisation de  $E$ , maximale correcte par rapport à  $E$  et  $N$ .

```

1:  $g = e_1$ 
2: for  $i = 2$  to  $n$  do
3:    $g' = \text{MG}(g, e_i)$  {Généralisation entre deux hypothèses}
4:   if  $(\forall e \in N : g' \not\preceq e)$  then
5:      $g = g'$  { $g'$  (correcte) devient la généralisation courante}
6:   end if
7: end for
8: return  $h(x) = \text{class}(E)$  si  $g \succeq x$ , 0 sinon (abstention)

```

---

Le principe de cet algorithme est d'ajouter un à un les exemples d'une même classe, chaque ajout devant être validé contre les exemples des autres classes. Il produit des hypothèses qui peuvent s'abstenir mais qui ne se trompent pas sur l'échantillon d'apprentissage. Il présente en outre des caractéristiques intéressantes : il nous permet de parcourir les hypothèses correctes de  $\mathcal{H}$ , par plus petit pas, de manière monotone (un exemple couvert à un instant du processus le reste ensuite) et complète (on ne peut pas ajouter au résultat un exemple de  $E$  de la même classe et non déjà subsumé sans perdre la correction). Notons également la sensibilité à l'ordre de présentation des exemples<sup>1</sup> : selon le classement des exemples dans  $E$ , MGC fournira des hypothèses différentes. Nous verrons que cette caractéristique n'est pas un défaut et que les méthodes d'ensemble en tireront avantage.

Muni de cet apprenant (qui nécessite uniquement de définir le test de subsomption et l'opération MG), il nous reste à lui faire produire plusieurs hypothèses et à combiner celles-ci pour créer un classifieur complet.

---

<sup>1</sup>Dans nos algorithmes, nous notons les ensembles avec des accolades, les séquences avec des crochets.

## 2.3 Apprentissage d'un classifieur complet

Nous passons en revue différentes méthodes d'apprentissage susceptibles d'intégrer notre algorithme MGC, avec différents objectifs :

- rapidité de l'apprentissage pour DLG ;
- compréhensibilité de la théorie construite pour GLOBO ;
- excellentes prédictions des classifieurs appris pour GLOBOOST et ADABOOST.

Précisons immédiatement que toutes ces méthodes peuvent traiter des problèmes avec un nombre quelconque de classes.

La première, DLG (Webb & Agar, 1992), de type *glouton*, se caractérise donc par sa rapidité. DLG consiste à apprendre les hypothèses sur des exemples qui ne sont pas déjà couverts et à arrêter dès que la couverture complète de l'ensemble d'apprentissage est atteinte (algorithme 2).

---

### Algorithm 2 DLG

---

**Entrées :**  $A$  un ensemble de  $n$  exemples  $(x_i, y_i)$ .

**Sortie :**  $H$  le classifieur final.

```

1:  $O = A$ ;  $t = 0$ 
2: while ( $O \neq \emptyset$ ) do
3:    $target$  = classe du premier exemple de  $O$ 
4:    $P = [x_i \in O | y_i = target]$ 
5:    $N = \{x_i \in A | y_i \neq target\}$ 
6:    $h_t = \text{MGC}(P, N)$  {Appel à l'algorithme 1}
7:    $O = O - \{x \in O : h_t \succeq x\}$ 
8:    $t = t + 1$ 
9: end while
10: return  $H(x) = \text{ArgMax}_k |\{h_t : h_t(x) = k\}|$ 

```

---

GLOBO (Torre, 2005) est conçu pour produire des classifieurs plus compréhensibles et plus prédictifs que DLG. Son principe est de produire autant d'hypothèses qu'il y a d'exemples disponibles en apprentissage (en mélangeant aléatoirement les exemples à chaque fois) puis d'opérer une couverture minimale pour ne garder qu'un minimum d'hypothèses permettant la couverture complète de l'ensemble d'apprentissage (voir l'algorithme 3). GLOBO fournit naturellement moins de règles que DLG, mais elles sont a priori plus générales et plus pertinentes : le tout en est plus facilement appréhendable par un expert humain.

Nous terminons avec les méthodes d'ensemble qui nous offrent des erreurs en généralisation faibles. Ces algorithmes font appel de manière répétée à un apprenant de base pour produire différentes hypothèses ; au moment de la prédiction, ces hypothèses sont combinées au sein d'un vote. L'intérêt de ces techniques de combinaison a été établi : quel que soit le mode de production des hypothèses, quelles que soient les modalités du vote final, l'erreur en généralisation observée est plus faible que celle de n'importe lequel des votants. Dans ces méthodes, l'importance de la diversité des hypothèses a été démontrée et c'est pourquoi MGC avec sa sensibilité à l'ordre de présentation des exemples nous semble être un bon candidat pour jouer le rôle d'apprenant de base.

---

**Algorithm 3** GLOBO

---

**Entrées :**  $A$  un ensemble de  $n$  exemples  $(x_i, y_i)$ .**Sortie :**  $H$  le classifieur final.

```

1:  $R' = \emptyset$ 
2: for  $i = 1$  to  $n$  do
3:    $P = [x_j | y_j = y_i \wedge i \neq j]$ 
4:    $N = \{x_j | y_j \neq y_i\}$ 
5:   mélanger  $P$  aléatoirement
6:    $h_i = \text{MGC}(x_i :: P, N)$  {Appel à l'algorithme 1 avec  $x_i$  en tête des positifs}
7:   ajouter  $h_i$  à  $R'$ 
8: end for
9:  $R = \emptyset$ 
10: while  $(\exists x_i, \forall h_j \in R, h_j \not\succeq x_i)$  do
11:    $h = \text{ArgMax}_{h_i \in R'} \{x_j : h_i \succeq x_j \wedge \nexists h_k \in R : h_k \succeq x_j\}$ 
12:   ajouter  $h$  à  $R$ 
13: end while
14: return  $H(x) = \text{ArgMax}_k \{h \in R : h(x) = k\}$ 

```

---

Les questions à régler pour mettre en œuvre une méthode d'ensemble sont : quel apprenant de base ? quel processus de production ? quel protocole de vote ? À la première question, nous répondons à nouveau par l'utilisation de moindres généralisés. Contrairement à GLOBO, nous ne cherchons plus à faire une sélection parmi les hypothèses produites, mais à toutes les faire voter. Les réponses possibles aux deux questions suivantes ont donné naissance à différentes méthodes, dont le *boosting*.

Il s'agit d'une technique de combinaison qui a des origines théoriques ancrées dans le modèle PAC et a abouti à l'algorithme ADABOOST (Freund & Schapire, 1995). Son principe est d'associer un poids à chaque exemple, puis de solliciter l'apprenant de base (dit *faible* dans ce cadre) sur ces exemples pondérés pour obtenir une hypothèse qui classe correctement les exemples de poids forts ; ensuite, les poids des exemples sont modifiés en fonction de cette hypothèse (les poids des bien classés sont diminués et ceux des mal classés augmentés) et le processus itéré. Pour *booster* des hypothèses produites par MGC, nous proposons AB-MG (algorithme 4) : il s'agit de ADABOOST dans sa version gérant les apprenants faibles pouvant s'abstenir (Schapire & Singer, 1999) et modifiée pour ordonner les exemples de la classe cible par poids décroissant avant l'appel à MGC.

Enfin, Dietterich (2000) propose de mettre en concurrence le *boosting* avec une production purement stochastique des hypothèses. L'idée de cette dernière est d'ajouter de l'aléatoire dans l'algorithme qui apprend les hypothèses. Dans le cas des moindres généralisés, une variabilité suffisante est obtenue par un mélange aléatoire des exemples, préalable à l'appel à MGC. Cette solution correspond à GLOBOOST (Torre, 2005), algorithme 5.

---

**Algorithm 4** AB-MG (version à deux classes)

---

**Entrées :**  $n$  exemples  $x_i$  et leurs classes  $y_i \in \{-1, +1\}$ ;  $T$  un nombre d'itérations à effectuer.

**Sortie :**  $H$  le classifieur final.

```

1: for  $i = 1$  to  $n$  do
2:    $w_i = 1/n$  {initialisation des poids des exemples}
3: end for
4: for  $t = 1$  to  $T$  do
5:    $target =$  la classe de l'exemple de poids le plus fort
6:    $P = [x_i | y_i = target]$ 
7:    $N = \{x_i | y_i \neq target\}$ 
8:   trier  $P$  par poids décroissant
9:    $h_t = \text{MGC}(P, N)$  {Appel à l'algorithme 1}
10:  for  $b \in \{-, 0, +\}$  do
11:     $W_b = \sum_{i: \text{sign}(y_i h_t(x_i)) = b} w_i$ 
12:  end for
13:   $\alpha_t = \frac{1}{2} \log \left( \frac{W_+ + \frac{1}{2} W_0}{W_- + \frac{1}{2} W_0} \right)$ 
14:   $Z_t = \sum_i^n [w_i \cdot \exp(-\alpha_t y_i h_t(x_i))]$ 
15:  for  $i = 1$  to  $n$  do
16:     $w_i = \frac{w_i \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ 
17:  end for
18: end for
19: return  $H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$ 

```

---



---

**Algorithm 5** GLOBOOST

---

**Entrées :**  $n$  exemples  $(x_i, y_i)$  et  $T$  un nombre d'itérations.

**Sortie :**  $H$  le classifieur final.

```

1: for  $t = 1$  to  $T$  do
2:    $target =$  classe choisie au hasard
3:    $P = [x_i | y_i = target]$ 
4:    $N = \{x_i | y_i \neq target\}$ 
5:   mélanger  $P$  aléatoirement
6:    $h_t = \text{MGC}(P, N)$  {Appel à l'algorithme 1}
7: end for
8: return  $H(x) = \text{ArgMax}_k |\{h_t : h(x) = k\}|$ 

```

---

### 3 Moindres généralisés en Inférence Grammaticale

Pour bénéficier des avantages de ces différentes méthodes, nous avons simplement à :

- choisir les espaces de description pour les exemples et les hypothèses ;
- définir un test de subsomption hypothèse/exemple ;
- prouver l’unicité du moindre généralisé et proposer un moyen de le calculer sous forme incrémentale  $MG(h, e)$ .

Dans le cadre de cet article, les exemples sont des mots et les hypothèses des automates, le test de subsomption est tout naturellement le test de reconnaissance d’un mot par un automate. Il reste à préciser la famille d’automates et son opération de moindre généralisé. Cela revient à demander pour cette famille un algorithme d’apprentissage à partir de positifs seuls avec la contrainte de proposer le plus petit langage incluant les exemples fournis. Nous considérons dans cet article trois familles de langages bien connues en Inférence Grammaticale dont deux disposent déjà de résultats d’apprenabilité à partir de positifs seuls : les  $k$ -TSS et les 0-réversibles (nous renvoyons le lecteur à [Dupont & Miclet \(1998\)](#) pour une étude approfondie de ces deux familles). La dernière correspond à des expressions régulières particulières que nous allons présenter.

#### 3.1 Les $k$ -TSS

Un langage *k-testable au sens strict* ( $k$ -TSS) peut se définir par la donnée de séquences de taille  $k$  qui n’ont pas le droit d’apparaître dans les mots du langage. Ce sont donc des langages très simples, à l’expressivité assez pauvre : en particulier, ils ne permettent pas de compter et donc d’identifier des langages qui posent des conditions sur les nombres d’occurrences des lettres de l’alphabet.

Cependant, un algorithme par exemples positifs seuls est disponible et celui-ci est réputé calculer le plus petit langage incluant l’échantillon, c’est-à-dire le moindre généralisé dans cette famille. Nous le proposons dans une version incrémentale nommée MG-TSSI (algorithme 6 dont la clef est que chaque état est représenté par les  $(k - 1)$  dernières lettres lues avant d’atteindre cet état).

#### 3.2 Les 0-réversibles

La famille des  $k$ -réversibles est souvent jugée plus intéressante que la précédente, en partie parce qu’elle est plus expressive : les langages  $(k - 1)$ -TSS sont tous  $k$ -réversibles. Il faut cependant noter qu’à  $k$  fixé, certains langages *finis* ne sont pas  $k$ -réversibles.

Dans cet article, nous nous restreignons à  $k = 0$ , c’est-à-dire aux 0-réversibles. Un calcul de moindre généralisé dans la famille des 0-réversibles est proposé dans ([Angluin, 1982](#)), il est facilement rendu incrémental pour devenir MG-ZR (algorithme 7 et figure 1). Cet algorithme consiste à ajouter le nouveau mot dans l’automate existant à la manière du PTA, c’est-à-dire qu’au besoin une nouvelle branche uniquement dédiée à reconnaître ce mot est créée. Puis, l’algorithme opère les fusions nécessaires pour n’avoir qu’un état final, des transitions déterministes et un automate-miroir également déterministe.

---

**Algorithm 6** MG-TSSI

---

**Entrées :**  $h = (Q, \Sigma, q_0, F)$  un automate  $k$ -TSS,  $w$  un mot et une valeur de  $k$

**Sortie :**  $h'$  un automate  $k$ -TSS généralisation minimale de  $h$  couvrant  $w$

```

1:  $q = q_0$ 
2: for  $i = 1$  to  $|w|$  do
3:    $v = q.w_i$  {concaténation du mot de  $q$  avec la  $i^e$  lettre de  $w$ }
4:   if ( $|v| = k$ ) then
5:      $v = v_2, \dots, |v|$ 
6:   end if
7:    $nq = v$ 
8:   ajouter  $nq$  à  $Q$  { $nq$  peut déjà être présent dans l'ensemble  $Q$ }
9:   ajouter  $(q, w_i, nq)$  à  $\delta$ 
10:   $q = nq$ 
11: end for
12: ajouter  $q$  à  $F$ 
13: return  $h' = (Q, \Sigma, \delta, q_0, F)$ 

```

---



---

**Algorithm 7** MG-ZR

---

**Entrées :**  $h = (Q, \Sigma, q_0, F)$  une hypothèse-automate 0-réversible,  $w$  un exemple-mot

**Sortie :**  $h'$  un automate 0-réversible, généralisation minimale de  $h$  reconnaissant  $w$

```

1:  $i = 1$ ;  $q = q_0$ 
   {suivi des états existants}
2: while  $\delta(q, w_i)$  is defined do
3:    $q = \delta(q, w_i)$ ;  $i = i + 1$ 
4: end while
   {création d'une nouvelle branche pour les lettres restantes}
5: while  $i \leq |w|$  do
6:   créer un état  $q'$  et l'ajouter à  $Q$ 
7:   ajouter  $(q, w_i, q')$  à  $\delta$ 
8:    $i = i + 1$ 
9: end while
10: ajouter  $q$  à  $F$ 
   {mise en œuvre des fusions}
11: fusionner tous les états de  $F$ 
12: repeat
13:   si  $\exists A, B \in Q$  et  $\exists l \in \Sigma$  tels que  $\delta(A, l) = \delta(B, l)$ , fusionner  $A$  et  $B$ 
14:   si  $\exists A, B, E \in Q$  et  $\exists l \in \Sigma$  tels que  $\delta(E, l) = \{A, B\}$ , fusionner  $A$  et  $B$ 
15: until no fusion
16: return  $h' = (Q, \Sigma, \delta, q_0, F)$ 

```

---

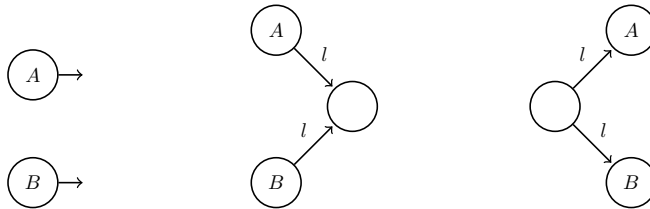


FIG. 1 – Les trois cas dans lesquels ZR fusionnent les états  $A$  et  $B$  ( $l \in \Sigma$ ).

### 3.3 Les monômes réguliers

Nous reprenons de (Fernau, 2005) une méthode de généralisation des mots en expressions régulières. Dans l'article original, l'algorithme est donné à travers un exemple portant sur l'échantillon  $S = \{ababb, aabb, ababa, abc\}$ . Le principe consiste à regrouper les lettres en blocs, à aligner ces blocs comme suit :

(a)	(b)	(a)	(bb)	
(aa)	(bb)			
(a)	(b)	(a)	(b)	(a)
(a)	(b)	(c)		

puis à les généraliser pour obtenir ici :

$$a^+b^+(ab^+(\epsilon|a)|\epsilon|c)$$

On note qu'il ne s'agit pas d'une stricte généralisation en colonne car la rencontre de lettres distinctes provoque la poursuite de l'expression sur des branches différentes.

Nous avons modifié cette méthode à deux niveaux :

- pour chaque bloc, nous observons dans l'échantillon d'apprentissage le nombre minimal et le nombre maximal d'occurrences de la lettre associée ; les quantificateurs  $+$  ou  $*$  ne sont jamais utilisés ;
- nous nous interdisons d'imbriquer les expressions régulières et le symbole de disjonction.

Notre calcul d'expressions régulières, nommé MG-REGEX, se réduit alors à généraliser les blocs colonne par colonne, les blocs de lettres différentes faisant apparaître le symbole *point* ( $.$ ) dénotant une lettre quelconque de l'alphabet. Sur l'exemple ci-dessus, MG-REGEX donne donc :

$$a^{\{1,2\}}b^{\{1,2\}}.\{0,1\}b^{\{0,2\}}a^{\{0,1\}}$$

Nous avons nommé les expressions régulières ainsi atteignables la classe des *monômes réguliers*.

## 4 Validation expérimentale

### 4.1 Mise en œuvre

Nous évaluons, à l'aide de l'implémentation VOLATA<sup>2</sup>, toutes les combinaisons possibles entre nos apprenants principaux (DLG, GLOBO, GLOBOOST, AB-MG) et nos apprenants par positifs seuls (MG-TSSI, MG-ZR, MG-REGEX).

Nous les comparons à des méthodes classiques en Inférence Grammaticale : RPNI (Oncina & García, 1992), TRAXBAR (Lang, 1992) et RED-BLUE (Lang *et al.*, 1998) de la famille EDSM (*Evidence Driven State Merging*). RPNI et TRAXBAR nécessitent de fixer une classe cible. Pour chaque problème, nous les avons exécutés sur chaque classe et retenu uniquement le meilleur résultat. Enfin, MG-TSSI exige de fixer une valeur  $k$  pour fonctionner. Nous avons déterminé ces valeurs par validation croisée sur les ensembles d'apprentissage et finalement  $k$  varie entre 2 et 5 dans nos expérimentations.

Les jeux de données considérés proviennent de l'*UCI repository* (Blake & Merz, 1998) : tic-tac-toe, badges, us-first-name, promoters et splice, ce dernier dans une version à deux classes déjà utilisée par Dupont (2006). Nous ajoutons à cette liste soccer, un problème produit à partir des résultats d'un championnat de football réel : il s'agit de séquences de résultats de deux équipes, chaque séquence étant étiquetée par le nom de l'une des équipes. Dans ce problème, l'alphabet est constitué des lettres 1,  $N$ , 2 (1 pour victoire,  $N$  pour match nul et 2 pour défaite). Nos méthodes peuvent traiter des problèmes avec un nombre quelconque de classes, mais nous nous sommes restreints ici à des problèmes à deux classes pour nous comparer aux autres méthodes sans avoir à modifier leurs implémentations.

Nous utilisons une validation croisée 10 fois et toutes les méthodes travaillent sur les mêmes découpages en blocs. Dans le cas des algorithmes stochastiques, chaque exécution est répétée 10 fois. Pour les méthodes d'ensemble (GLOBOOST et AB-MG), nous fixons à 1 000 le nombre d'hypothèses produites par apprentissage, sauf lorsque c'est MG-ZR qui est apprenant de base car celui-ci s'est révélé trop lent pour l'usage intensif que nous en faisons. Enfin, la mesure utilisée pour comparer les algorithmes est la moyenne des taux de bonne classification obtenus sur les ensembles de test.

### 4.2 Résultats

Les résultats de ces expérimentations sont donnés à la table 1. Pour chaque problème, un tableau donne les résultats de nos concurrents et un autre les résultats obtenus par nos méthodes. Dans chaque tableau, le meilleur résultat est indiqué en gras. Les valeurs manquantes n'ont pas pu être obtenues dans un temps raisonnable avec les implémentations à notre disposition.

Il apparaît que sur chaque problème, le meilleur système est l'un de ceux que nous avons proposés. En général, il s'agit soit de GLOBOOST, soit de AB-MG ce qui est bien ce à quoi nous nous attendions : les méthodes d'ensemble ont les meilleures performances en prédiction. Retrouver ce comportement connu est rassurant et nous conforte

---

<sup>2</sup><http://www.grappa.univ-lille3.fr/~torre/Recherche/Softwares/volata/>

TAB. 1 – Résultats chiffrés des expérimentations.

<b>tic-tac-toe</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	65.34 %	91.13 %	90.81 %	<b>93.89 %</b>	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	79.02 %	81.43 %	91.47 %	90.27 %
	MG-ZR	96.76 %	95.84 %	98.36 %	98.31 %
MG-REGEX	95.20 %	99.77 %	98.56 %	<b>99.61 %</b>	
<b>badges</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	<b>71.43 %</b>	62.24 %	57.48 %	61.09 %	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	73.61 %	72.24 %	72.69 %	73.47 %
	MG-ZR	71.43 %	71.43 %	71.43 %	-
MG-REGEX	98.30 %	99.93 %	95.10 %	<b>100.0 %</b>	
<b>promoters</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	50.00 %	- %	56.60 %	<b>63.02 %</b>	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	53.53 %	60.00 %	61.13 %	61.51 %
	MG-ZR	50.00 %	50.00 %	50.00 %	50.00 %
MG-REGEX	69.81 %	70.85 %	<b>86.23 %</b>	85.66 %	
<b>soccer</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	50.00 %	62.50 %	58.33 %	<b>63.06 %</b>	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	64.31 %	59.17 %	72.08 %	68.89 %
	MG-ZR	52.78 %	50.00 %	53.19 %	56.81 %
MG-REGEX	69.44 %	65.28 %	<b>73.33 %</b>	68.33 %	
<b>us-first-name</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	81.62 %	81.42 %	81.37 %	<b>82.83 %</b>	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	84.19 %	83.83 %	<b>89.50 %</b>	86.29 %
	MG-ZR	81.77 %	81.97 %	83.07 %	83.92 %
MG-REGEX	87.75 %	88.40 %	88.28 %	87.32 %	
<b>splice</b>	<i>Majorité</i>	RPNI	TRAXBAR	RED-BLUE	
	50.26 %	- %	<b>58.33 %</b>	54.65 %	
		DLG	GLOBo	GLOBOOST	AB-MG
	MG-TSSI	67.61 %	73.42 %	78.07 %	79.79 %
	MG-ZR	-	-	-	-
MG-REGEX	85.57 %	89.93 %	93.46 %	<b>94.91 %</b>	

TAB. 2 – Variations sur la taille d'un échantillon (tic-tac-toe).

Proportion en apprentissage	90 %	50 %	25 %	10 %
RED-BLUE	93.89 %	85.07 %	73.04 %	68.29 %
AB-MG + MG-TSSI	90.27 %	82.33 %	76.29 %	69.57 %
GLOBOOST + MG-ZR	98.36 %	98.00 %	93.20 %	66.96 %
GLOBO + MG-REGEX	99.77 %	99.23 %	93.38 %	72.28 %

dans l'idée d'appliquer notre approche générique à différents espaces de représentation et d'y obtenir à chaque fois des performances comparables.

Ajoutons que des expériences supplémentaires nous ont permis de constater un autre comportement attendu : les performances de GLOBOOST et AB-MG augmentent avec le nombre d'hypothèses produites (10, 100 et 1 000).

Parmi les apprenants par positifs seuls, c'est le plus souvent MG-REGEX qui apparaît dans la combinaison gagnante, MG-TSSI venant ensuite. MG-ZR est le plus mauvais : sur certains problèmes il est trop lent pour être utilisable et sur d'autres il ne permet pas de faire mieux que la décision majoritaire. Cela s'explique par le fait que la famille des 0-réversibles est assez riche et autorise un apprentissage quasi par cœur sur certaines données ; ainsi, il est souvent possible qu'un unique 0-réversible capture tous les exemples d'une classe, sans reconnaître de contre-exemples.

### 4.3 Influence de la taille des échantillons

Une dernière expérience consiste à évaluer la résistance des différentes méthodes face à des variations de la taille de l'échantillon donné en apprentissage. Précédemment, la validation croisée 10 fois nous faisait utiliser 90% des données disponibles, le reste étant gardé pour le test.

Les résultats avec des échantillons plus petits du problème tic-tac-toe, et pour une sélection de quelques combinaisons variées de méthodes, sont donnés à la table 2. Nous observons les performances des méthodes classiques chuter alors que nos systèmes maintiennent des résultats raisonnables, même avec peu de données. Nous pouvons noter qu'avec 25% des données, deux de nos méthodes assurent encore plus de 90% de bonnes prédictions alors que RED-BLUE est nettement en retrait.

## 5 Conclusion

### 5.1 Bilan

Notre objectif dans cet article est la classification de séquences : il s'agit au final de décider si un mot appartient à un langage ou non et donc d'approcher au mieux un langage cible, la réussite se mesurant en terme d'erreur en généralisation. Nous rejoignons en cela les travaux que nous avons déjà évoqués en introduction (Callut & Dupont, 2006; Dupont, 2006; Clark *et al.*, 2006; Bonache *et al.*, 2007). Pour notre part,

nous avons fait le choix d'intégrer des techniques de base en Inférence Grammaticale dans un cadre générique ayant les avantages et la souplesse issus de la classification supervisée. Cela passe par un algorithme apprenant à partir de mots positifs seuls, embarqué dans différents algorithmes génériques sophistiqués d'apprentissage supervisé multi-classes, chacun présentant une ou plusieurs caractéristiques intéressantes : rapidité de l'apprentissage, gestion d'un nombre quelconque de classes, compréhensibilité des classifieurs obtenus, erreur en généralisation faible, etc. Ainsi, nous bénéficions des résultats des deux domaines à la fois.

Nos expérimentations ont confirmé la pertinence de cette démarche et nous ont permis d'exhiber de bonnes performances pour nos techniques au point de surpasser les méthodes classiques, tout en nécessitant moins d'exemples. Nous avons en effet montré une certaine tolérance à la diminution de la taille des échantillons d'apprentissage. Les algorithmes classiques disposent souvent d'une preuve d'apprenabilité en présence d'un échantillon caractéristique mais ces mêmes algorithmes peuvent avoir des comportements déroutants si un seul exemple est manquant. Dans cette situation d'échantillon incomplet, nos techniques ont l'avantage de posséder une certaine stabilité. Enfin, ces expérimentations ont permis de retrouver les comportements de nos méthodes déjà constatés en attribut-valeur, ce qui valide d'une autre manière notre travail.

## 5.2 Discussion

Une caractéristique de notre travail est la production par nos apprenants de combinaisons (ou disjonctions) : pour chaque classe, nous obtenons plusieurs automates. Nous sommes convaincus que les cibles sous forme de disjonction nécessitent moins d'exemples qu'un automate équivalent mais unique. Un argument allant dans ce sens : RPNI cible un automate déterministe dont la taille peut croître de manière exponentielle par rapport à la version non déterministe (que nous exprimons par une disjonction).

Un intérêt supplémentaire des disjonctions est de nous permettre de sortir du langage cible fixé pour les apprenants par positifs seuls et donc de gagner en expressivité.

Mettons également en avant notre choix de caractériser chaque classe du problème. D'une part, nous partons d'algorithmes travaillant sur des positifs seuls et le simple fait de les plonger dans notre cadre générique nous permet au final d'exhiber un ensemble d'automates pour chacune des classes du problème, quel que soit leur nombre. D'autre part, tous ces automates constituent effectivement le classifieur appris et utilisé pour de nouveaux exemples. Chacun de ces automates est a priori plus simple que l'automate unique appris par RPNI par exemple. Un expert peut choisir d'observer l'automate qui a le poids le plus fort dans la combinaison et essayer de lui donner du sens.

Janodet *et al.* (2004) ont, avant nous, proposé une technique d'apprentissage de combinaisons d'automates, celle-ci a plusieurs facettes. Dans un premier temps est supposée l'existence d'un oracle fournissant une confiance sur les étiquettes données aux exemples en apprentissage et ADABOOST est modifié pour prendre en compte cette information. En particulier, une nouvelle règle de mise à jour des poids est proposée et fondée théoriquement. Cet ADABOOST avec oracle pourrait s'ajouter à notre liste de méthodes d'ensemble. Ensuite, ce nouvel algorithme est utilisé pour *booster* RPNI\*, une version de RPNI capable de gérer les données bruitées grâce à un critère particulier

pour contrôler la fusion des états. Nous ne pouvons pas reprendre RPNI\* dans notre cadre puisqu'il s'agit d'un algorithme complet et non pas d'un apprenant par positifs seuls. Cependant, l'idée de relâcher le critère de généralisation en présence de bruit peut être généralisé pour bénéficier à tous nos apprenants. Finalement, ce travail est donc une combinaison spécifique d'algorithmes et d'heuristiques mais les idées exprimées peuvent enrichir notre cadre générique à bien des égards.

### 5.3 Perspectives

Une caractéristique attendue et largement étudiée en attribut-valeur est la tolérance au bruit de classe. Nous avons évoqué la solution de *Janodet et al. (2004)* pour rendre RPNI résistant à ce type de bruit. Dans notre cadre, il serait possible de relâcher le critère absolu de la correction pour valider une généralisation (dans MGC, algorithme 1) en lui préférant l'*accuracy de Laplace (Clark & Boswell, 1991)*. Nous comptons expérimentalement pour mesurer l'intérêt de ce critère vis-à-vis de données bruitées mais également pour déterminer si ce critère amène une diversité des hypothèses profitable aux méthodes d'ensemble comme ADABOOST sur des données non bruitées.

Nous devons aussi explorer d'autres familles de langages que celles vues dans cet article, en particulier celles qui disposent de résultats d'apprenabilité par exemples positifs seuls. De plus, il est indispensable de trouver des familles de langages ayant un moindre-généralisé unique, condition *sine qua non* du bon fonctionnement de nos algorithmes. Dans cet article, nous nous sommes appuyés sur des résultats d'unicité déjà démontrés pour les *k*-TSS et les 0-réversibles mais cette unicité n'est pas garantie par définition, elle est à démontrer pour chaque classe de langages considérée. Nous avons pu constater sur quelques problèmes que les classes riches comme les 0-réversibles ne sont pas forcément les plus pertinentes dans notre cadre. Ainsi, même si des résultats négatifs sont connus pour l'apprentissage par positifs seuls de classes de langages importantes, nos travaux encouragent la recherche d'apprenants par positifs seuls pour des classes réduites.

### Remerciements

Nous sommes reconnaissants à Jérôme CHAMPAVÈRE et François COSTE de nous avoir fourni les implémentations de méthodes dont ils disposaient. Merci à eux.

### Références

- ANGLUIN D. (1982). Inference of reversible languages. *Journal of the ACM*, **29**(3), 741–765.
- BLAKE C. & MERZ C. (1998). UCI repository of machine learning databases [<http://www.ics.uci.edu/~mllearn/MLRepository.html>].
- BONACHE L. B., DE LA HIGUERA C., JANODET J.-C. & TANTINI F. (2007). Apprentissage des boules de mots avec des requêtes de correction. In *Proc. Conférence en Apprentissage CAp 2007*.

- CALLUT J. & DUPONT P. (2006). Sequence discrimination using phase-type distributions. In J. FÜRNKRANZ, T. SCHEFFER & M. SPILIOPOULOU, Eds., *ECML 2006, 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, p. 78–89 : Springer.
- CLARK A., FLORÊNCIO C. C. & WATKINS C. (2006). Languages as hyperplanes : Grammatical inference with string kernels. In J. FÜRNKRANZ, T. SCHEFFER & M. SPILIOPOULOU, Eds., *ECML 2006, 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, p. 90–101 : Springer.
- CLARK P. & BOSWELL R. (1991). Rule induction with CN2 : Some recent improvements. In *Proc. Fifth European Working Session on Learning*, p. 151–163, Berlin : Springer.
- DIETTERICH T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees : Bagging, boosting, and randomization. *Machine Learning*, **40**(2), 139–158.
- DUPONT P. (2006). Noisy sequence classification with smoothed markov chains. In *CAp 2006*, p. 187–201 : Presses Universitaires de Grenoble.
- DUPONT P. & MICLET L. (1998). *Inférence grammaticale régulière : fondements théoriques et principaux algorithmes*. Rapport interne RR-3449, INRIA.
- FEIGENBAUM E. A. (1977). The art of artificial intelligence : Themes and case studies of knowledge engineering. In R. REDDY, Ed., *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, p. 1014–1029 : Morgan Kaufmann.
- FERNAU H. (2005). Algorithms for learning regular expressions. In S. JAIN, H.-U. SIMON & E. TOMITA, Eds., *Algorithmic Learning Theory, 16th International Conference, ALT 2005, Singapore, October 8-11, 2005, Proceedings*, volume 3734 of *Lecture Notes in Computer Science*, p. 297–311 : Springer.
- FREUND Y. & SCHAPIRE R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, p. 23–37.
- JANODET C., NOCK R., SEBBAN M. & SUCHIER H.-M. (2004). Boosting grammatical inference with confidence oracles. In GREINER, Ed., *International Conference on Machine Learning (ICML04)*, p. 425–432.
- LANG K. J. (1992). Random dfa's can be approximately learned from sparse uniform examples. In *In Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, p. 45–52 : ACM Press.
- LANG K. J., PEARLMUTTER B. A. & PRICE R. A. (1998). Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *ICGI '98 : Proceedings of the 4th International Colloquium on Grammatical Inference*, p. 1–12, London, UK : Springer-Verlag.
- ONCINA J. & GARCÍA P. (1992). *Identifying regular languages in polynomial time*, In *Advances in Structural and Syntactic Pattern Recognition*, p. 99–108. World Scientific Publishing.
- SCHAPIRE R. E. & SINGER Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, **37**(3), 297–336.
- TORRE F. (2005). Globoost : Combinaisons de moindres généralisés. *Revue d'Intelligence Artificielle*, **19**(4-5), 769–797.
- WEBB G. I. & AGAR J. W. M. (1992). Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm. *Artificial Intelligence in Medicine*, **4**, 419–430.