Classification en Programmation Génétique

Alain Brunie-Taton, Antoine Cornuéjols
Équipe Inférence et Apprentissage
Laboratoire de Recherche en Informatique (LRI), UA 410 du CNRS
Université de Paris-sud, bâtiment 490, 91405 ORSAY (France)
e-mail: {brunie, antoine}@lri.fr

RÉSUMÉ : Cet article présente un nouvel algorithme de classification binaire, PGclass, capable de produire l'équation d'une hypersurface séparant les exemples de deux classes. Cet algorithme est fondé sur une utilisation de la Programmation Génétique permettant la recherche de l'équation d'une séparatrice adéquate entre les exemples d'apprentissage. A la différence des méthodes issues de la Reconnaissance des Formes ou des techniques connexionnistes, PGclass produit un résultat intelligible, tandis qu'il dépasse les limitations des méthodes classiques de l'IA, telles l'induction d'arbres de décision, en matière de complexité des séparatrices identifiables.

Un grand nombre d'expériences de classification binaire dans un espace à attributs numériques effectuées en variant systématiquement plusieurs paramètres : dimension, nombre de variables pertinentes pour la classification, complexité des surfaces de séparation, fonction de qualité utilisée (entropie, distance de Widrow-hoff), nombre et choix des primitives d'expression des individus, etc., montrent que la Programmation Génétique est une bonne méthode de classification produisant l'équation d'une séparatrice correcte pour des expressions complexes et de nombreuses variables non pertinentes.

Par ailleurs, pour la première fois, une étude systématique a été conduite pour évaluer et comprendre le rôle de la fonction de qualité dans la Programmation Génétique. Il en résulte que des effets significatifs sont liés à la pente, la dérivée seconde et la borne inférieure de cette fonction, ce qui correspond à différentes caractéristiques de la pression sélective s'exerçant sur les populations de solutions candidates.

MOTS-CLÉS : Apprentissage supervisé, Classification, Programmation Génétique, fonctions d'évaluation.

1. Introduction

L'une des tâches les plus étudiées en apprentissage est la **classification supervisée.** Le problème consiste à induire un critère de classification dans l'espace des exemples à partir d'une collection d'exemples classés. Ce critère peut prendre plusieurs formes : arbre de décision, surface séparatrice, etc. Il doit permettre la classification ultérieure d'exemples non classés.

Les travaux en Reconnaissance des Formes et sur les Réseaux de Neurones ont principalement porté sur des espaces d'exemples décrits par des attributs numériques avec la recherche de surfaces séparatrices entre classes. Excepté dans les cas de séparation linéaire des classes, le résultat est un critère de décision illisible car codé, dans la structure d'un réseau de neurones par exemple. Dans le cadre de l'apprentissage symbolique automatique, de nombreux systèmes de classification existent, tels ceux de la famille ID3 [Quinlan 86]. Ils produisent généralement des critères de décision assimilables à des règles qui sont donc lisibles, mais ils sont difficilement adaptables aux espaces décrits par des attributs numériques. Par ailleurs, et malgré des travaux récents sur la génération d'*arbres obliques* [Breiman 84, Heath 93, Murthy 93, Utgoff 90, Ma Jie 94], ils restent généralement limités à des classifications combinant des hyperplans parallèles aux axes, ou, au mieux, des combinaisons linéaires de ces axes, ce qui

entraîne souvent la production de règles de classification nombreuses et difficilement compréhensibles. Une méthode mariant les avantages de chaque approche : performance, intelligibilité des résultats, applicabilité à des espaces à la fois numériques et symboliques serait donc d'un grand intérêt.

La Programmation Génétique, dérivée des Algorithmes Génétiques, est une méthode de recherche d'expressions très générales pouvant se représenter sous forme d'arbres (par exemple des expressions Lisp). La recherche est guidée par un critère d'évaluation des expressions testées. En définissant un critère traduisant la qualité d'une partition d'exemples, il devient possible d'utiliser la Programmation Génétique pour des tâches de classification. Les avantages potentiels d'une telle méthode sont : sa grande généralité d'emploi, la complexité à priori arbitraire des expressions qu'elle peut produire, son applicabilité à des espaces aussi bien numériques que symboliques, et enfin, mais à nos yeux d'une grande importance, l'expressivité des solutions produites.

Cet article présente une expérimentation systématique, à l'aide du système PGclass, de l'utilisation de la Programmation Génétique pour des tâches de classification binaire (classification en deux classes, soit en exemples soit en contre-exemples). D'une part, les facteurs permettant de varier la difficulté des tâches de classification et donc d'évaluer les performances de la Programmation Génétique ont été méthodiquement changés : nombre total de descripteurs, nombre de descripteurs pertinents, complexité des fonctions séparatrices, nombre d'exemples d'apprentissage. D'autre part, les paramètres propres au fonctionnement de la Programmation Génétique : importance des populations de solutions candidates à chaque génération, méthode de sélection des "parents", proportions des différents opérateurs de reproduction, ensemble des primitives disponibles, ont également fait l'objet de multiples variations afin de déterminer les plages les plus favorables.

Il est ainsi apparu, lors des expériences conduites avec deux fonctions de qualité[1] : la fonction d'entropie utilisée dans ID3, la fonction de coût quadratique de Widrow-Hoff, que le choix de la fonction de qualité a une influence notable sur les performances. A notre connaissance, aucune étude systématique n'a jusqu'ici été publiée sur le rôle de la fonction de qualité dans les Algorithmes Génétiques, sans doute parce que, dans les tâches examinées, il n'existe pas une aussi grande variété de travaux et de critères de performance qu'en classification. C'est pourquoi, à la suite de ces observations, nous avons décidé de chercher à caractériser les facteurs bénéfiques pour la définition d'une fonction de qualité. C'est ce qui constitue le deuxième volet de ce travail.

La présentation de cet article est structurée de la manière suivante. Les principes de base de la Programmation Génétique ainsi que les paramètres les plus importants sont rappelés dans la section 2. La section suivante présente alors comment la Programmation Génétique peut être adaptée pour la classification, et le fonctionnement du système PGclass. Le principe des expériences réalisées et la synthèse des résultats obtenus sont également présentés dans cette section. La deuxième partie de ce travail, sur le rôle de la fonction de qualité, fait l'objet de la section 4. La section de conclusion résume les leçons que l'on peut tirer à ce stade des recherches.

---

[1] Nous utiliserons indifféremment dans cet article les termes de fonctions de qualité et de fonctions d'évaluation.

2. Programmation Génétique

2.1 Notions de base

La PG [Koza 92] est une forme d'Algorithme Génétique [Holland 75] utilisant une représentation sous forme d'arbres au lieu de chaînes de bits ou de caractères.

Comme dans les AGs, on exploite l'évolution darwinienne d'une population d'individus (c'est-à-dire solutions potentielles).

L'algorithme démarre avec une population de programmes générés aléatoirement et une mesure de qualité et utilise la sélection et la reproduction sexuée pour produire, à partir de la population courante (génération P(t)), une nouvelle population de programmes (génération P(t+1)), dont la qualité s'améliore généralement..

Une exécution de l'algorithme consiste habituellement en un nombre fixé de générations suivant ce modèle d'évolution simulée. Le résultat d'une exécution est habituellement le meilleur individu qui a réussi à émerger.

Les programmes, à chaque génération d'une exécution, sont créés soit par leur copie depuis la génération précédente (avec une probabilité proportionnelle à leur qualité (méthode de Monte-Carlo)), soit par croisement de 2 programmes parents et introduction des 2 programmes fils résultants dans la nouvelle génération, ou encore par mutation d'un programme parent[2].

---

1. Génération de P(1) et évaluation des individus ; t ← 1
2. Tant que le critère d'arrêt n'est pas vérifié pour P(t)
   - Tant que l'on n'a pas fini de générer la nouvelle population P(t+1), choisir une opération :
     (a) **Reproduction** : Un individu de la génération parents est sélectionné* pour être tout simplement *recopié* dans la génération des descendants.
     (b) **Croisement** : Appliqué à 2 individus de la population parents (sélectionné grâce à P_{sel}^{i} *), il produit 2 descendants. On choisit aléatoirement un point de coupure sur chacun des 2 individus et ensuite on *échange les 2 sous-arbres* (figure 1).

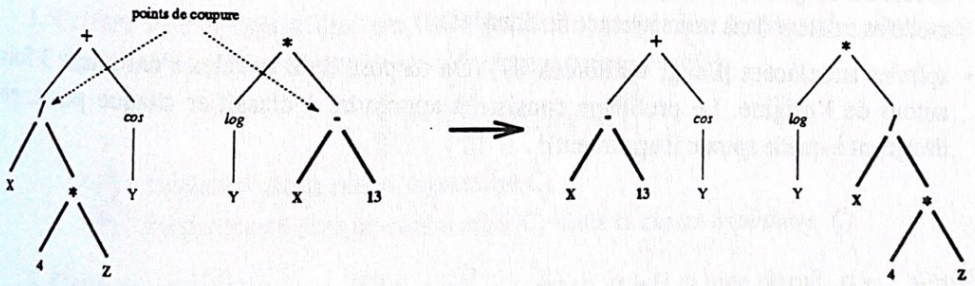

Figure 1 : Croisement de 2 formules.

(c) **Mutation** : En premier lieu on choisit un point de mutation (fonction ou terminal), puis le sous-arbre à ce point est remplacé par un autre *engendré aléatoirement* (figure 2).

---

[2] La nouvelle génération remplace l'ancienne.

• Sélection : $P^i_{sel} = \frac{f(i)}{\sum_{j \in P(t)} f(j)}$

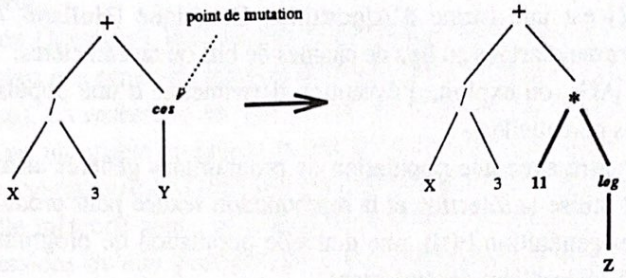

Figure 2 : Mutation d'une formule.

• Évaluation de P(t+1) ; t ← t+1

2.2 État de l'art

La Programmation Génétique est une nouvelle méthode qui n'a pas encore été le sujet d'importantes études. Peu de théories ont été élaborées (en comparaison des AGs), tandis que des investigations empiriques nombreuses (voir surtout Koza [Koza 92,94]) ont cherché à explorer les domaines d'application. Parmi celles-ci citons :

- la découverte de lois numériques,
- la robotique (la fourmi de Santa Fe [Langton], diriger une tondeuse à gazon),
- l'identification d'une fonction de réponse d'un système (identifier automatiquement des primitives -ADF[3]-).

Koza a fait quelques tentatives dans la classification [Koza 92] avec la PG.

- induction d'arbres de décision,
- induction de grammaires [Schalkoff 92] (Ce problème implique la reconnaissance de modèles existant dans une séquence de données),
- spirales entrelacées [Lang, Whitbrock 89] (On dispose de 2 spirales s'enroulant 3 fois autour de l'origine. Le problème consiste à apprendre à classifier chaque point, en désignant à quelle spirale il appartient)[4] .

[3] Définition automatique de fonctions [Koza 94].

[4] En général, Koza a essayé de montrer la versatilité de la PG en attaquant de très nombreux problèmes. Beaucoup n'ont pu été fouillés, c'est le cas des problèmes de classification. Le problème des spirales entrelacées montre que la PG permet de trouver des équations séparatrices qui sont délicates pour la plupart des méthodes de classification.

3. Classification en Programmation Génétique

3.1 Principes des expériences

Le problème peut se résumer ainsi; depuis un tableau de données correspondant à la base d'exemples de l'ensemble d'apprentissage (créé[5] auparavant grâce à la fonction que l'on désire apprendre par exemple), on essaie de retrouver la fonction, en faisant évoluer une population de solutions potentielles (ici une solution représente l'expression d'une surface séparatrice, elle est évaluée par le fait que les exemples sont cités au-dessus ($f(i) \geq 0$) ou au-dessous ($f(i) < 0$)). Il faut noter que les concepts représentables et apprenables ne sont pas restreints à des concepts impliquant une seule surface séparatrice, mais pourrait, par l'utilisation de primitives telles que ET, OU, IF, ..., inclure des concepts disjonctifs ou avec conditionnelles.

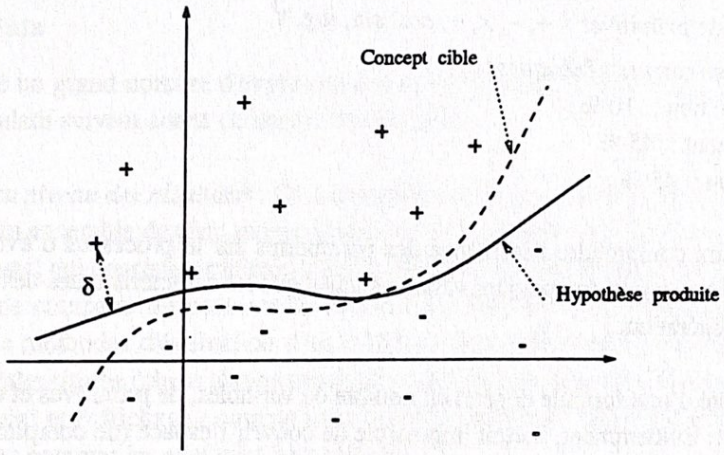

Figure 3 : Recherche de la séparatrice.

Paramètres de l'Algorithme :

- Fonctions d'*évaluation* :

1. **Critère entropique** utilisé par ID3 [Quilan 86] (minimiser le *désordre*, c'est-à-dire séparer au mieux les exemples positifs des exemples négatifs).

$$\sum_{j=1}^{2} P_j \cdot \left[\sum_{i=1}^{2} -P_i \cdot \log(P_i) \right]$$

   - P_j : fréquence de la *classe hypothèse* C_j,
   - P_i : fréquence de chaque *classe cible* C_i dans la *classe hypothèse* C_j.

2. **Coût quadratique** : $\sum_{i \in ens.\ d'appr.} (\delta_i + \alpha)^2$ (avec $\alpha = 0$ si bien classé, $\alpha = 1$ sinon[6])

   - δ_i correspond à la distance entre l'exemple i et la courbe (voir figure 3).

[5] On tire aléatoirement des valeurs pour les différentes variables, avec pour dernier champ : la classe (donnée par l'application de la fonction à considérer).
[6] Il s'agit d'une constante arbitraire qui dépend du domaine de définition des variables. Elle est prise égale à 1 ici,

- α permet de pénaliser un individu lorsqu'il se trouve du mauvais côté. Si l'on n'utilisait pas cette variable, le programme se contenterait de trouver le "barycentre" de la base de points.

Les valeurs des paramètres suivants correspondent à des valeurs utilisées habituellement et satisfaisantes en général.

- Taille de l'ensemble d'*apprentissage* : 200 exemples générés aléatoirement
- Taille de l'ensemble de *test* : 10.000 exemples générés aléatoirement
- Taille de la *population* : 500 (i.e. 500 fonctions de décision en compétition)
- Nombre de *générations* : 50 (pour chaque exécution)
- Nombre d'*exécutions* : 1000
- *Profondeur* maximale des arbres : 8
- Ensemble de *primitives* : $+, -, \times, \div, cos, sin, log, \sqrt{}$
- Taux des *opérateurs génétiques* :
  - reproduction : 10 %
  - croisement : 45 %
  - mutation : 45 %

Afin de mieux comprendre l'influence des paramètres sur le processus d'évolution, on a gardé une trace statistique de quelques variables telles que les caractéristiques des individus de génération en génération.

La complexité d'une formule dépend du nombre de variables, de primitives et de constantes qu'elle contient. Évidemment, il était impossible de couvrir l'espace (de complexité) complet durant notre étude. On a dû faire un compromis entre la difficulté de trouver la solution et la durée de la recherche. Ainsi, le but de ce travail n'était pas de tester la PG sur ses limites extrêmes pour un type donné de formules mais de couvrir plusieurs formules qui sont représentatives de différentes sortes de complexités.

On a donc travaillé avec plusieurs formules de diverses complexités :

- plusieurs dimensions (pour voir la progression des résultats en fonction du nombre de variables pertinentes ou pas), par exemple : $x+y$, $x+y+z$, $x+y+z+t$ ...

  On entend par variables pertinentes celles qui apparaissent dans la formule cible (que l'on désire apprendre), au contraire des variables non pertinentes qui, elles, ne servent à rien, et qui sont présentes uniquement pour tromper l'algorithme.
- formule compliquée au niveau de la forme de la séparatrice correspondante (ex : $x.y+\frac{y.z}{t}-\frac{y+t}{x}$).
- avec constantes[7] (ex : $a.x+b.y+c.z$).

[7] Valeurs entières générées aléatoirement entre [-127, 128] (par construction le programme peut engendrer n'importe quelle valeur réelle).

Au niveau du calcul de la fitness (qualité), on utilise bien évidemment la fonction d'évaluation choisie (*entropie, coût quadratique*). Cependant, afin de favoriser la recherche de solutions ayant la plus petite expression (c'est à dire mettre en exergue le critère de simplicité d'une formule), on a introduit un coefficient permettant de rendre la qualité proportionnelle aux nombres de noeuds dans l'arbre.

Exemple de simulation de l'Algorithme :

$$\frac{\log(z)}{z/y}+x-\sin(z) \;\rightarrow\; \frac{z/y}{\exp(x)}-\sqrt{\sin(z)} \;\rightarrow\; x\times\frac{z}{y}+\sqrt{x} \;\rightarrow\; y\times\frac{x}{z} \;\rightarrow\; \frac{x.y}{y+z}$$

On peut voir ici l'évolution du meilleur individu à différents moments (générations) d'une simulation.

3.2 Résultats

On a réalisé un grand nombre d'expériences avec les paramètres par défaut déjà cités. Le tableau récapitulatif suivant donne un aperçu des résultats.

Remarque au niveau des résultats[8] : On a évalué la qualité des solutions finales trouvées par PGclass avec un ensemble de tests indépendants et une autre fonction d'évaluation (identique pour tous les tests) qui consiste simplement à compter le nombre d'exemples bien classés. Il est ainsi possible de comparer directement les résultats des fonctions *entropie* et *distance* (alors qu'elles ont des méthodes d'évaluation d'un individu, bien différentes). Plus simplement, la simulation de l'algorithme (phase d'apprentissage) se fait avec la fonction d'évaluation normale (par ex : entropie) et l'affichage / comparaison des résultats (phase de tests) se fait grâce à la fonction simple : compter les exemples bien classés.

Format employé dans les tableaux :

	α →	pourcentage d'*exécutions* réussies (celles qui classent parfaitement ou presque la base d'exemples).
$\alpha(\beta/\chi)$:	β →	*moyenne* (sur 1000 exécutions) des meilleures fitness.
	χ →	*écart type* des meilleures fitness.
n* *formule*	même *formule*, mais avec des coefficients (ex : $x+y \rightarrow ax+by$).	
formule **(C)**	On permet l'introduction de **constantes** dans les formules générées.	

Exemple de lecture du tableau : pour le problème xy / $x+y$ (à 2 variables pertinentes) sans l'option création de constantes avec 5 variables (pertinentes + non pertinentes) on a un taux de réussite (trouver une fonction qui sépare parfaitement les exemples positifs des exemples négatifs) de 43,3% (sur 1000 exécutions), avec une moyenne de 87,3 (et un écart type de 12,1) pour le meilleur individu de chaque exécution.

[8] Idem pour la 2nde partie de l'article.

		Variables pertinentes + non pertinentes					
Var	**Problème**	**2**	**3**	**4**	**5**	**10**	**20**
2	x + y	100				100	100
	xy / (x+y)	98,3 (99,7/2,5)	80,4 (96,1/8,4)	59,3 (91/11,7)	43,3 (87,3/12,1)	16,1 (79,4/9,7)	
	xy/(x+y) *(C)*	93 (98,8/4,5)	80,5 (96,5/7,7)	72 (94,5/9,5)	59,9 (91,6/11,2)	31,2 (83,6/12)	
	2x - 3y + 4	51,1 (97,9/1,1)	35,2 (97,5/1,4)	32,9 (97,4/1,2)	25,3 (97,2/1,3)	21 (96,8/1,8)	
	2x-3y+4 *(C)*	77,2 (98,8/1,2)	72,3 (98,6/1,3)	66,1 (98,4/1,5)	59,5 (98,2/1,5)	42,6 (97,6/1,8)	
3	x + y + z						100
	xy / (y+z)		89,4 (97,8/6,8)	71,8 (93,3/11)	53,7 (88,7/12,8)	14,2 (77,6/9,7)	
	n° x+y+z *(C)*		42,2 (96,7/2,5)	38,5 (96,4/2,6)	33,2 (96,1/2,6)	26,2 (95,3/2,7)	
4	x +y +z +t					100	99,7 (/0,6)
	n°x+y..+t *(C)*			46,1 (97,1/2,2)	41,1 (96,8/2,3)	24,9 (95,8/2,6)	
5	x+y+z+t+u						99 (99,9/1,4)
7	x+y+z+ ..					98,7 (99,9/0,5)	94,5 (99,5/2,4)
10	x-y+z-t+ ..					46,2 (94,2/6,6)	12,9 (86,5/7,6)

tableau récapitulatif des résultats.

3.3 **Analyse**

L'analyse des résultats fait ressortir plusieurs observations :

- Les expériences réalisées avec utilisation du critère quadratique (non reportées dans le tableau) sur les problèmes à 2 variables pertinentes donnent des taux de généralisation d'environ 10% inférieur avec le critère quadratique, tandis que, en moyenne, la complexité des solutions trouvées (mesurée en nombre de noeuds et profondeur de l'arbre de représentation) est supérieure. L'utilisation du critère entropique semble donc préférable. Une des raisons de ce décalage pourrait être le fait que le critère quadratique est plus exigeant en cherchant la séparatrice passant au "milieu" des deux classes.
- L'algorithme est peu sensible quand on augmente le nombre de variables pertinentes (celles qui apparaissent dans la formule cible) jusqu'à une certaine limite que l'on peut situer aux alentours de 10 variables. Cela s'observe en descendant le long d'une colonne.
- Les variables non pertinentes (i.e. les variables qui ne servent à rien, sinon à tromper l'algorithme) influent approximativement linéairement sur les résultats, alors que la taille de l'espace de recherche augmente exponentiellement avec le nombre total de variables.

  On peut observer tout de même un phénomène intéressant ; les formules "simples" (au sens de l'arbre de représentation associé), comme $xy / y+z$, sont plus sensibles à l'augmentation des variables non pertinentes que des formules plus complexes avec constantes (par ex : $2x-3y+4$ *(C)*). Bien que au départ les performances sur $xy / y+z$ soient supérieures, la tendance s'inverse en ajoutant quelques variables non pertinentes.
- Lorsque l'algorithme peut engendrer directement les constantes (option (C)), la complexité de l'espace de recherche s'en trouve accrue, mais le nombre de chemins solution augmente également. C'est peut-être ce qui explique que les performances, moins bonnes avec peu de variables, dépassent celles obtenues sans l'option (C) lorsque le problème, avec plus de variables, devient plus difficile.

- La complexité d'une formule (par ex : $xy / y+z$ comparée à : $x+y+z$) affecte peu l'algorithme. A l'opposé, trouver les constantes approximant au mieux une formule simple avec constantes (par ex : $2x-3y+4$) semble plus difficile.
- Si l'on augmente le nombre de primitives possibles ($+, -, \times, \div$ [9] $\rightarrow$ $+, -, \times, \div, cos, sin, log, \sqrt{}$) utilisables par l'algorithme dans les noeuds de chaque arbre solution, on s'aperçoit d'une baisse moyenne des résultats.

4. Analyse du rôle de la fonction d'évaluation

4.1 Principe

Une des observations importantes qui ressort immédiatement de l'étude des résultats obtenus en classification concerne la très grande sensibilité des résultats au choix de la fonction d'évaluation. Ainsi, avons-nous constaté que l'utilisation de la fonction de coût quadratique conduit presque toujours à des performances de classification inférieures à celles obtenues en utilisant la fonction de gain entropique. Cette observation, ainsi que l'absence de travaux systématiques concernant l'influence du choix de la fonction d'évaluation, nous ont amenés à étudier les critères qui président à leurs performances.

Deux types principaux de comparaisons sont envisageables. D'une part, des *variations sur ce que mesure la fonction d'évaluation* : ainsi, par exemple, la fonction quadratique mesure en fait les "écarts" entre les données et la séparatrice, tandis que la fonction entropie ne prend en compte que le nombre d'exemples mal classés. D'autre part, des *variations sur la forme* que prend la fonction d'évaluation : sa pente, sa concavité, etc. Ce travail s'est attaché à une étude de ce deuxième type afin de déterminer d'abord les caractéristiques "syntaxiques" les plus favorables au fonctionnement de la Programmation Génétique. Ces caractéristiques ont en effet plus de chances d'avoir un champ de validité général, que les caractéristiques sémantiques plus limitées à un domaine ou une tâche particulière.

Dans les Algorithmes Génétiques, la fonction d'évaluation joue essentiellement un rôle de pression sélective favorisant ou éliminant une partie de la population en fonction de sa qualité. Il est très facile de faire varier cette pression sélective en jouant sur la pente, sur la concavité (dérivée seconde) de la fonction d'évaluation, et sur sa borne inférieure. C'est ce qui a été fait systématiquement en mesurant et en comparant les performances résultantes sur diverses tâches de classification. L'enjeu est à la fois d'examiner s'il existe des tendances significatives et si elles correspondent à des comportements génériques à travers les types de tâches.

4.2 Résultats

Les expériences réalisées ont porté sur des variations selon deux axes orthogonaux: d'une part la forme de la fonction d'évaluation, et, d'autre part, le type de séparatrices à approximer par l'algorithme.

---

[9] Le tableau récapitulatif avec les primitives ($+, -, \times, \div$) n'a pas été inséré dans l'article pour ne pas surcharger l'esprit du lecteur.

1. Fonctions d'évaluation

Les trois paramètres étudiés sont la pente, la concavité et l'asymptote horizontale de la courbe associée. Nous avons cherché à examiner leurs effets séparément, et en combinaison, d'où les 10 fonctions sélectionnées (représentées graphiquement dans la figure 5) :

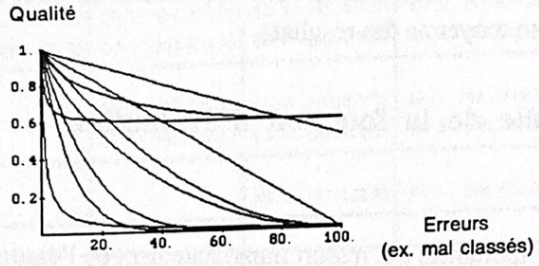

Figure 4 : Aperçu de l'ensemble des fonctions d'évaluation

- **Somme linéaire :** $\frac{nb\ exemples\ bien\ classés - nb\ total\ d'exemples/2}{nb\ total\ d'exemples/2}$

  Cette fonction est de concavité nulle et de borne inférieure = 0.

- **Somme affine :** $\frac{nb\ exemples\ bien\ classés}{nb\ total\ d'exemples}$

  Cette fonction est de concavité nulle et de borne inférieure = 0,5.

- **Ve 60 :** $1 - \frac{nb\ exemples\ bien\ classés}{60}$ pour *nb exemples mal classés* $\in [1, 60]$, sinon 0.

  Cette fonction est de concavité nulle et de borne inférieure = 0.

- **Entropie [0.59, 1] :** Fonction d'entropie ramenée dans l'intervalle [0.59, 1].
  Cette fonction est de concavité faible et de borne inférieure = 0.59, ce qui signifie que les individus peu performants ont une probabilité non négligeable d'être sélectionnés pour le calcul de la génération suivante.

- **Somme [0.59, 1] :** Fonction **somme** (voir ci-dessous) ramenée dans l'intervalle [0.59, 1]. Cette fonction est de concavité forte et de borne inférieure = 0,59.

- **Entropie :** Fonction d'entropie (voir section 3.1).
  Cette fonction est de concavité faible et de borne inférieure = 0.

- **Somme :** $\frac{1}{1+nb\ exemples\ mal\ classés}$

  Cette fonction est de concavité forte et de borne inférieure = 0.

- **Biz (de 2 à 10) :** $\left(\frac{nb\ total\ d'exemples - 2 \times nb\ exemples\ mal\ classés}{nb\ total\ d'exemples}\right)^i$, i pouvant varier de 2 à 10.

  Cette fonction est de concavité plus ou moins forte suivant la valeur de i, et de borne inférieure = 0.

Remarque : Les classes définies par l'algorithme sont étiquetées par la classe majoritaire des exemples couverts, en conséquence, il ne peut y avoir plus de la moitié des exemples mal classés. C'est pourquoi les graphiques de la figure 5 ont leur axe d'erreur allant de 0 à 100 alors que le nombre total d'exemples est de 200.

2. Types de séparatrices

Nous avons conduit des tests systématiques sur 5 fonctions séparatrices entre 2 classes en faisant varier leur complexité, en jouant sur le nombre de variables (pris entre 2 et 3), la présence de constantes ou non dans le jeu des opérateurs disponibles, et le nombre d'opérateurs impliqués.

Remarque : Certaines fonctions d'évaluation comportent l'option *cst* (voir section 3.2).

Nous avons repris les mêmes paramètres (pour l'Algorithme) que l'étude précédente (voir section 3.2).

4.3 Analyse

L'analyse des résultats montre les phénomènes suivants :

- Une borne inférieure élevée, c'est-à-dire donnant une possibilité importante aux moins bons individus de participer à la procréation de la génération suivante, est très préjudiciable aux performances, et ceci quelle que soit la concavité de la fonction concernée : **somme affine, entropie** [0.59, 1] ou **somme** [0.59, 1].
- Les bonnes performances relatives de la fonction **Ve 60**, montrent le bénéfice d'éliminer les plus mauvais individus pour la reproduction de la génération suivante. A contrario, l'existence d'une concavité semble expliquer les meilleurs performances des fonctions **Biz** et autres.
- La comparaison des performances des différentes fonctions **Biz**, montre un accroissement des performances générales de **Biz2** à **Biz6**, suivi d'une légère baisse pour **Biz10**. Cela laisserait supposer que la concavité de la fonction **Biz6** est proche d'un optimum. C'est ce qui est confirmé par l'examen des performances des fonctions **entropie** moins concave que **Biz2** et **somme** plus concave, et plus proche d'un "dirac" que la fonction **Biz10**. En résumé, plus on s'écarte de **Biz6** (d'un côté comme de l'autre) plus les performances baissent.
- Lorsque l'algorithme commence à converger dans une direction de recherche donnée, on considère qu'il *exploite* la direction qu'il croit être la bonne. Or, il est toujours possible qu'il se trompe, et doit donc toujours donner une chance d'*explorer* une nouvelle piste. On peut assimiler la concavité de notre courbe à pression sélective, et faire un parallèle entre cette concavité et le rapport entre exploitation/exploration. En effet, lorsque la concavité de la courbe est forte (**somme, Biz10**), l'algorithme aura tendance à exploiter les meilleurs individus. A contrario, quand la concavité est faible (**entropie, Biz2**), on aura tendance à travailler sur un plus grand nombre d'individus (dans une même génération) donc à

augmenter l'exploration. Ainsi un bon compromis entre exploitation et exploration pourrait être une concavité du type de **Biz6**.

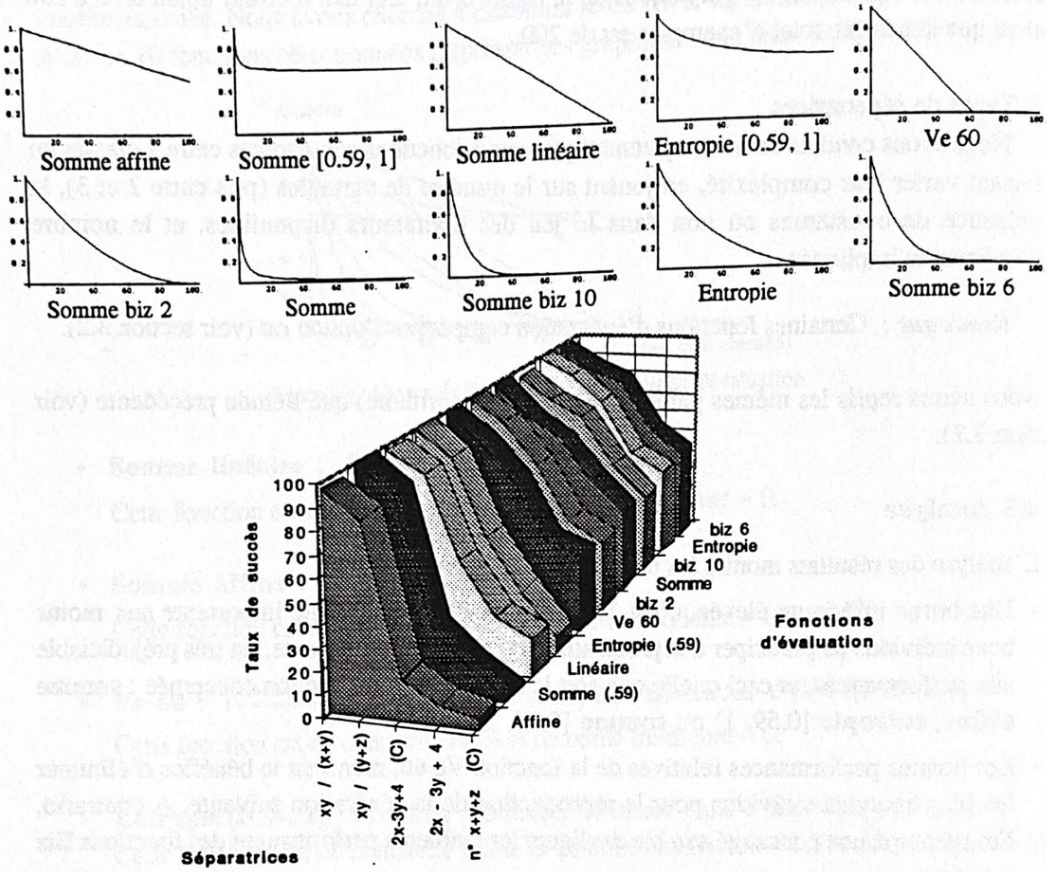

Figure 5 : Comparaison des fonctions d'évaluation

5. Conclusion

Les expériences réalisées avec le système PGclass montrent que la Programmation Génétique offre une alternative très intéressante aux techniques usuelles de classification, qu'elles soient issues de la Reconnaissance des Formes ou de l'Intelligence Artificielle. En effet, sur les nombreux problèmes étudiés, variant en particulier le nombre total de descripteurs, le nombre de descripteurs pertinents, et la complexité des séparatrices entre classes, la PG permet de trouver une solution (au moins)[10] jusqu'à 20 variables dont 7 pertinentes, et des solutions approchées sur des problèmes plus difficiles. Les résultats sont peu affectés par le nombre de descripteurs non pertinents, et baissent à peu près linéairement avec le nombre de variables pertinentes. La complexité du concept cible a par contre une influence légère sur les performances (qui est accentuée quand on lui adjoint des constantes), sans que celle-ci soit

[10] Sur 50 exécutions de 200 générations, soit environ ½ h de temps calcul sur une SPARC station, ou 3mns sur notre version parallèlisée de PGclass sur une machine à 8 noeuds.

encore bien comprise. Par ailleurs, en dehors des performances en classification généralement très satisfaisantes, il faut souligner l'intérêt de pouvoir "lire" facilement l'équation des séparatrices trouvées.

En retour, il est possible d'influer sur le fonctionnement du système en modifiant l'ensemble des opérateurs disponibles pour le rendre plus adapté au problème courant. Il est également possible de biaiser la fonction d'évaluation pour favoriser certains types d'expressions, par exemple les plus simples.

Finalement, l'étude conduite sur l'influence de la forme de la fonction d'évaluation semble montrer que des fonctions à concavité assez marquée, telle la fonction **Biz6**, correspondent au meilleur profil pour la pression sélective, et qu'il faut réduire à zéro les possibilités de reproduction des moins bons individus.

Les perspectives immédiates de ce travail concernent une étude plus exhaustive des types de séparatrices et de l'effet sur les performances de la PG. C'est aussi en liaison avec ce problème que l'expérimentation, sur des bases d'exemples symboliques et numériques qui introduisent d'autres types de concepts, doit être menée.

6. Références

[Breiman 84] L. Breiman, J. Friedman, R. Olhshen (1984), *Classification and Regression Trees*. Monterey, CA:Wadsworth and Brooks.

[Heath 93] D. Heath, S. Kasif, S. Salzberg (1993), *Introduction of Oblique Decision Trees*. IJCAI-93, Chambéry, France.

[Holland 75] J. H. Holland (1975), *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press.

[MA Jie 94] MA Jie (1994), *Découverte de Nouveaux Descripteurs dans ID3*. Rapport de Stage de D.E.A, Université de Paris-XI, Orsay.

[Koza 92] J. R. Koza (1992), *Genetic Programming*. Cambridge, MA: MIT Press.

[Koza 94] J. R. Koza (1994), *Genetic Programming II*. Cambridge, MA: MIT Press.

[Lang, Whitbrock 89] K. J. Lang, M. J. Whitbrock (1989), *Learning to tell two spirals apart*. Proc. of the 1988 Connectionist Models Summer School, Morgan Kaufmann.

[Moulet 93] M. Moulet (1993), *ARC: Découverte empirique de lois numériques ou ABACUS revu et corrigé*, Thèse.

[Murthy 93] S. Murthy, S. Kaisf, S. Salzberg, R. Beigel (1993), *OC1: Randomized Induction of Oblique Decision Trees*. Proc. of 11th Nat. Conf. on AI AAAI-93, Cambridge MIT press, pp323-327.

[Quinlan 86] J. R. Quinlan (1986), *Induction of Decision Trees*. Machine Learning Vol 1, n°1, Kluwer Academic Publisher.

[Quinlan 93] J. R. Quinlan (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann.