DES ALGORITHMES EVOLUTIONNAIRES POUR LA CLASSIFICATION AUTOMATIQUE

Fatima-Zohra Kettaf

LIMSI, Bâtiment 508, université ParisXI,
Orsay, bp 133, 91403 Orsay cedex

Jean-Pierre Asselin de Beauville

Laboratoire d'Informatique
Université de Tours (E3i)
64 Avenue Jean Portalis, 37200 Tours
(actuellement en détachement auprès de l'Agence universitaire de la Francophonie à Montréal, Canada)

Résumé : *Cet article traite des méthodes de classification à l'aide d'algorithmes évolutionnaires (algorithmes génétiques, stratégies d'évolution). Ces algorithmes sont connus pour leur capacité à échapper aux extrema locaux de la fonction optimisée (critère de classification). Nous proposons ici de nouveaux algorithmes de partitionnement utilisables en particulier lorsque le nombre de classes de la partition n'est pas fixé a priori, ils sont définis à partir de codages originaux des partitions et font appel à des opérateurs génétiques nouveaux. On étudie également la question difficile du choix du critère de classification à optimiser. Des tests permettent d'évaluer les méthodes proposées.*

Introduction

Récemment, plusieurs chercheurs ont utilisé l'optimisation stochastique pour partitionner un ensemble de données en un nombre de classes connu ou non. Klein et al [KLE89] ont utilisé le recuit simulé pour classer les données, d'autres leur ont préféré les algorithmes évolutionnaires : Stratégies d'Evolution (SE) [BAB94], Algorithmes Génétiques [CUC93], [KET96a]... Charon et al [CHA99] ont hybridé un algorithme génétique avec des méthodes d'exploration locale comme le tabou et le recuit simulé pour résoudre un problème de

partitionnement de graphes pondérés. Ces méthodes se distinguent par la méthode d'échantillonnage de l'espace de recherche des solutions. On peut les classer en trois catégories : la première basant le codage des partitions sur l'appartenance des objets aux classes, la deuxième plutôt sur les prototypes de classes (exemples : centroïdes de classes [BAB94][KET96a], paramètres de mélange de distributions [KET96b]) et la troisième sur la similitude entre observations (objets à classer). Nous n'allons aborder en détail ici que les deux premières approches, le lecteur intéressé pourra consulter [KET97] pour des compléments.

L'intérêt des méthodes proposées dans cet article réside essentiellement dans le fait que le nombre de classes de la partition n'est pas supposé connu a priori. Dans un tel contexte, il est bien connu que la classification des données est un problème difficile et mal résolu [DUB87].

Ce papier est divisé en cinq parties : après l'introduction, dans la deuxième section, on présente brièvement les algorithmes évolutionnaires : les algorithmes génétiques et les stratégies d'évolution. La troisième partie présente les deux nouveaux codages. Enfin, on donne des résultats de classification sur des données artificielles et naturelles et on compare les méthodes entre elles. Dans la conclusion on envisage des perspectives à ces travaux.

Les algorithmes évolutionnaires (AE)

On regroupe sous cette appellation, les algorithmes d'optimisation stochastique se basant sur les principes de l'évolution naturelle néo-darwinienne. On s'intéresse ici à deux types d'algorithmes : les algorithmes génétiques [HOL75] et les stratégies d'évolution [REC73]. Ils sont apparus pratiquement en même temps des deux côtés de l'atlantique (respectivement aux USA et en Allemagne). Le but d'un AE est de faire évoluer un ensemble de solutions appelé population (au lieu d'une seule solution comme c'est le cas pour le recuit simulé par exemple) vers une solution optimale (ou qui approche l'optimale).

•

1.1. Les Algorithmes Génétiques (AG)

Pour ces algorithmes, les points de l'espace de recherche (solutions) sont appelés "chromosomes " et sont codés par des chaînes de bits. La population, à chaque instant t, est appelée "génération ". L'application des différents opérateurs génétiques va permettre de

créer de nouveaux chromosomes (et en faire disparaître d'autres, la taille de la population étant toujours la même).

Les opérateurs génétiques **:** la sélection, le croisement et la mutation, sont le moyen de modifier les chromosomes entre deux générations afin de parcourir l'espace des solutions.

La sélection est un procédé par lequel un chromosome est recopié dans la nouvelle population en fonction des valeurs de la fonction à optimiser pour ce chromosome. Cela revient à donner aux chromosomes dont la valeur de fonction est grande une probabilité plus élevée de contribuer à la génération suivante. Elle peut être mise en œuvre sous forme algorithmique de différentes façons, la plus connue étant désignée par la méthode de "sélection de la roulette biaisée" [GOL90].

Après la sélection, on applique aux chromosomes de la population intermédiaire des opérateurs de croisement et de mutation. Le croisement simple ou à un point de coupure consiste premièrement à choisir un couple de chromosomes avec une probabilité P_c (paramètre de l'AG). Puis dans une deuxième étape, on coupe les chaînes représentatives des deux chromosomes en une position aléatoire Pos (identique chez les deux parents). Cela produit alors deux "segments têtes " et "deux segments queues ". Enfin, on permute les deux segments queues des parents pour obtenir ainsi deux enfants qui héritent de quelques caractéristiques de leurs parents.

La mutation est appliquée avec une faible probabilité P_m (paramètre de l'AG) après le croisement et consiste à modifier un gène du chromosome (inversion de bit). Cette procédure : sélection, croisement, et mutation est répétée jusqu'à ce qu'un critère d'arrêt soit vérifié (généralement un nombre maximum de générations).

1.2. Les Stratégies d'Evolution (SE)

Rechenberg [REC73] est l'un des premiers à utiliser les SE dans les problèmes d'optimisation continue. Schewefel [SCH81] en s'inspirant des algorithmes génétiques a introduit de nouvelles versions des SE. Tout comme pour les AG, les SE sont basées sur les trois principes de l'évolution naturelle : sélection, recombinaison (croisement), et mutation. La SE fait évoluer une population de solutions ; chaque solution est une chaîne de paramètres appelée individu de la population. La population est constituée d'un ensemble de parents noté U et

d'un ensemble d'enfants noté Θ. Les premières SE ne comportaient qu'un parent et un enfant à la fois dans la population, la mutation étant le seul opérateur, ajoutait à un paramètre une valeur aléatoire issue d'une distribution normale de moyenne nulle et d'écart type σ (fixé a priori). La sélection consistait à prendre le meilleur des deux (l'enfant et le parent) comme individu de la prochaine génération. Cette préférence est guidée par la fonction de qualité (fitness) de l'individu. Ce type de SE est appelée $(1+1)$-SE et se caractérise par l'absence de l'opérateur de croisement.

[SCH81] a introduit les SE à plusieurs membres, dans lesquelles plusieurs parents participent à la création d'un enfant. Pour ce faire, l'opérateur génétique de croisement a été emprunté aux AG. Ici deux individus sont choisis aléatoirement, et sont combinés pour générer un seul enfant. L'enfant muté remplace le plus mauvais des parents. Cette version est appelée $(\mu+1)$-SE. Cette dernière a ensuite été étendue aux versions $(\mu+\lambda)$ SE et (μ,λ)-SE. La différence entre ces deux versions étant que dans la première, seulement les μ meilleurs individus sont choisis parmi les parents et les enfants alors que dans la deuxième, les meilleurs ne sont choisis que parmi les enfants.

Soient deux individus I_1 et I_2 sélectionnés pour subir l'opération de recombinaison (croisement). On peut représenter chaque solution par les vecteurs suivants : $I_1=(z_1,\sigma_1)$ et $I_2=(z_2,\sigma_2)$, où z_i^j est le j^{ème} attribut du vecteur z_i représentant l'individu I_i et σ_i^j est l'écart type du bruit ($N_0(0,\sigma_i^j)$) utilisé par la mutation pour bruiter la j^{ème} composante de z_i.

Un opérateur possible de recombinaison est l'opérateur moyenne : l'enfant généré $I=(z,\sigma)$ est tel que : $z^i=\frac{1}{2}\left(z_1^i+z_2^i\right)$ et $\sigma^i=\frac{1}{2}\left(\sigma_1^i+\sigma_2^i\right),\ \forall i$

L'opérateur de mutation : soit $I=(z,\sigma)$. Cet individu et son écart type subissent les modifications suivantes pour générer un nouvel individu tel que :

$$\sigma^i=\sigma^i\exp\left(N_0(0,\Delta\sigma)\right),\ z^i=z^i+N_0\left(0,\sigma^i\right),\ \forall i$$

L'opérateur de sélection : Cet opérateur dépend du choix des versions de la SE : (μ,λ)-SE ou $(\mu+\lambda)$-SE.

2. Les codages de partitions et les opérateurs associés

2.1. Représentation classée ou par appartenance

Cucchiara [CUC93] est l'un des premiers à appliquer les AG au problème de la classification. Il considère un gène comme codant une classe et spécifie l'absence ou la présence de l'objet à l'intérieur de cette classe par un code binaire u_{ij} $\forall i = 1..n, j = 1..K$ (n= nombre d'objets à classer , K= nombre de classes).

Exemples de chromosomes codant des partitions :

Soit l'ensemble d'objets à partitionner $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ noté $X = \{1,2,3,4,5,6,7,8\}$. Deux partitions possibles de X et leurs codages associés seraient :

((1,6)(3,4,7)(2,5,8)) : ((10000100)(00110010)(01001001))

et ((1,6,3)(2,4,7)(5,8)) : ((10100100)(01010010)(00001001))

La représentation classée de Cucchiara est basée sur une hypothèse forte qui est la connaissance du nombre de classes. Nous avons étendu son utilisation au cas général où le nombre de classes K est autorisé à varier, ce qui donne lieu à des chromosomes de longueurs variables [HAR92]. Le chromosome est formé d'un ensemble de gènes (classes), de façon que l'ensemble constitue une solution valable. Un chromosome est autorisé à croître (par division des classes) ou à décroître (par fusion des classes) d'une façon dynamique à l'aide d'opérateurs génétiques de manière à toujours constituer une solution valable (par rapport à la définition mathématique d'une partition de X).

Exemples de chromosomes codant deux partitions distinctes du même ensemble X :

((1,6,3,4,7)(2,5,8)) : ((10110110)(01001001))

et ((1,6,3)(2,4,7)(5,8)) : ((10100100)(01010010)(00001001))

Les opérateurs génétiques nouveaux :

Le croisement à deux coupures : il s'agit d'un croisement qui échange entre deux chromosomes (de longueurs en bits l_1 et l_2) des parties délimitées par des points de coupure

différents chez les deux parents (resp. $coup_1$ et $coup_2$) [SRI95]. Ce croisement génère des chromosomes enfants de longueurs variées comme souhaité.

Soient les deux parents et les points de coupure suivants :

$((1,2,5,6)(3,4,7,8))$: $((11001100)\mathrm{I}_{coup_1}(00110011))$

$((1,5)(2,6)(3,4)(7,8))$: $((10001000)(01000100)\mathrm{I}_{coup_2}(00110000)(00000011))$

Les enfants obtenus sont :

$((11001100)(00110000)(00000011))$: $((1,2,5,6)(3,4)(7,8))$

$((10001000)(01000100)(00110011))$: $((1,5)(2,6)(3,4,7,8))$

Les coupures sont définies par les équations suivantes : $coup_1 = (r \bmod K_1) \times n$ et $coup_2 = (r \bmod K_2) \times n$, r où est un nombre entier généré aléatoirement et K_1, K_2 sont les nombres de classes des partitions des parents (générés aléatoirement entre deux et le nombre maximum de classes autorisé).

Il est clair que le croisement donne parfois des chromosomes non valables, ceci est dû à la contrainte d'appartenance (chaque objet doit appartenir à une classe et une seule). C'est pour cela que l'on a développé une procédure de correction qui maintient la cohérence dans les chromosomes.

La mutation : consiste à déplacer un objet de sa classe originale vers une autre classe, qui est choisie aléatoirement.

L'insertion : est une opération utilisée dans le but d'ajouter de nouvelles classes dans le chromosome. On choisit aléatoirement un point d'insertion qui doit se situer entre deux gènes (deux classes) et un nombre i de classes à ajouter. Les classes à ajouter sont initialement vides, l'opérateur d'insertion choisit aléatoirement des objets de la partition et les place dans les nouvelles classes.

La suppression : suit le processus inverse de celui de l'insertion, elle réduit le nombre de classe d'un chromosome (partition). On choisit aléatoirement une position de suppression située entre deux gènes et un nombre de classes à supprimer. Une fois les classes supprimées, les objets les constituant sont réaffectés aléatoirement aux classes restantes.

On propose dans ce qui suit deux autres opérateurs génétiques utilisant l'information de distance par rapport aux centres de classes. On les a nommés opérateurs génétiques dédiés à la classification.

Intégrer de la connaissance du domaine d'application dans les AE permet d'augmenter leur efficacité [MIC92]. Dans cette perspective, on propose de nouveaux opérateurs génétiques qui explorent l'espace des partitions à l'aide d'heuristiques de partitionnement. Les opérateurs précédents inséraient/supprimaient des gènes (classes) aléatoirement. Ici, l'introduction de connaissances a priori permet d'utiliser des heuristiques de base du processus de partitionnement pour reclasser les données (ISODATA [BAL64]).

L'opérateur d'insertion dédié à la classification : la probabilité d'insertion d'une classe est fixée a priori dans l'algorithme. Une fois la décision d'insertion prise pour un chromosome, une classe vide est insérée aléatoirement entre deux classes. Ce sont les objets les plus éloignés de leur centres de classe qui seront affectés à cette nouvelle classe. L'insertion est itérative et nécessite le re-calcul des centres de classe après chaque affectation.

L'opérateur de suppression de classes dédié à la classification : la probabilité de suppression n'est pas fixée a priori, elle est dynamique dans le sens où elle dépend de la partition codée par le chromosome, elle évolue en même temps qu'elle. Une classe peut être supprimée si la plus petite distance de son centre aux (K-1) autres centres est inférieure au seuil de fusion noté π (identique à celui utilisé dans le processus de fusion de l'algorithme ISODATA :"*Lumping threshold* "). Pour chaque classe C_i, on calcule cette distance que l'on note D_i :

$D_i = \min_{j \neq i} \left\| V^i - V^j \right\|_A$ où A est une métrique et où V^j est le centre de la classe C_j.

Le seuil de fusion π [KAN82] est calculé comme suit :

$\pi = \frac{\sqrt{p}}{3K} \sum_{i=1}^{K} D_i$, où p est la dimension de l'espace de représentation.

On note S l'ensemble des distances inférieures au seuil de fusion, la probabilité qu'un individu I (partition) subisse l'opération de suppression est alors : $P_{sup}(I) = \frac{card(S)}{K}$. Une fois l'individu sélectionné pour cette opération, il reste à définir le nombre de classes à supprimer et à préciser la ou (lesquelles) sera (seront) supprimée(s). On choisit pour la suite de ne supprimer au plus qu'une classe. Pour chaque classe $i = 1..K$, on calcule sa probabilité de suppression $P_{sup}(i)$ en fonction de son effectif et de sa distance aux autres classes comme suit : $P_{sup}(i) = P_{eff}(i) + P_D(i)$ où $P_{eff}(i) = (1 - \frac{n_i}{n})$ et $P_D(i) = 1 - \frac{D_i}{\max_{j \neq i} D_j}$, où n_i est l'effectif de la classe i

La procédure d'affectation des objets de la classe supprimée peut se faire de deux manières : la classe à supprimer est fusionnée avec la classe dont le centre était le plus proche, ou bien les objets sont affectés aux plus proches (K-1) centres avec re-calcul des centres après chaque affectation ou sans re-calcul. Nous avons choisi la deuxième solution.

Une fonction objectif originale : La fonction objectif d'un chromosome reflète la qualité de la partition qu'il code. Dans un contexte non supervisé où le nombre de classes n'est pas connu a priori, on doit bien entendu estimer le "bon" nombre de classes, soit le nombre de classes dans la population parente. Cette fonction objectif est un indice interne qui se base uniquement sur la matrice de proximité des données. Un tel indice est difficile à choisir car il dépend des paramètres du problème (dimension, nombre de classes...). L'utilisation de l'erreur quadratique donne des solutions triviales, car elle décroît quand le nombre de classes augmente.

Par ailleurs on pourra aussi utiliser l'indice de Davies-Bouldin [DAV79] :

Cet indice pour une partition en K classes se calcule de la façon suivante :

$$DB(K) = (1/K)\sum_{l=1}^{K} R_l$$

où $R_i = \max_{j \neq i} \{R_{ij}\}$ et $R_{ij} = \frac{e_i + e_j}{m_{ij}}$, m_{ij} est la distance euclidienne entre les centres des deux classes, et $e_i = \frac{\sum_{x \in C_i} \|x - V^i\|^2}{\sqrt{n_i}}$ $\forall i$.

Nous proposons ici un nouvel indice interne basé sur une fonction du rapport interclasses sur inertie totale.

Soit le rapport inertie interclasses sur inertie totale :

$fitness_{brut}(I) = \frac{\sum_{i=1}^{K} n_i \times \|V^i - V\|^2}{\sum_{x \in X} \|x - V\|^2}$, V^i est le centre de la classe i et V est le centre d'inertie de tout le nuage.

On cherche à maximiser ce rapport de façon à favoriser plutôt des solutions avec moins de classes d'effectifs faibles et les plus éloignées possible. On propose pour cela de pénaliser la fitness ci-dessus selon deux critères : premièrement, le pourcentage tx_1 de classes de la partition dont les effectifs sont inférieurs à un effectif minimal (un par exemple) et deuxièmement, le pourcentage tx_2 de classes dont les centres de classes sont assez rapprochés. On notera ces deux ensembles S1 et S2 , S1 étant l'ensemble des classes d'effectif <= 1 et S2 l'ensemble des paires de classes dont la distance mutuelle entre leurs centres est inférieure à un certain seuil.

Les proportions tx_1 et tx_2 sont définies par : $tx_1 = \frac{card(S_1)}{K}$ et $tx_2 = \frac{card(S_2)}{K(K-1)/2}$

$tx = \alpha tx_1 + (1-\alpha) tx_2$, ($\alpha$ égal à 0.5 donne la même importance aux deux pénalisations)

La fonction objectif d'un individu I sera réécrite de la façon suivante :

$fitness_{pen}(I) = fitness_{brut}(I) \times (1 - tx)^{\theta}$, où $\theta > 0$.

Le paramètre θ gouverne l'importance de la pénalisation.

2.2. Représentation par prototypes : centres de classes

Etant donné un ensemble X à partitionner en K classes, on représente ici les classes par un prototype (exemple : les centroïdes de classes). L'espace de recherche de l'algorithme évolutionnaire (en l'occurrence la SE) est donc l'espace des centroïdes de toutes les partitions possibles de X en K classes selon Babu et al [BAB94]. Un chromosome est donc la suite des coordonnées des K centroïdes de la partition qu'il code. Reprenons l'exemple du paragraphe précédent :

Soit X={1,2,3,4,5,6,7,8}, et soient deux partitions ((1,6)(3,4,7)(2,5,8)) et ((1,3,5),(4,7),(2,6,8)) de centroïdes respectifs V^1, V^2, V^3 pour la première et V'^1, V'^2, V'^3 pour la deuxième. Les deux chromosomes correspondants sont alors :

$$Z_1 = \left(V_1^1, V_2^1, V_1^2, V_2^2, V_1^3, V_2^3\right)$$

$$Z_2 = \left(V_1'^1, V_2'^1, V_1'^2, V_2'^2, V_1'^3, V_2'^3\right)$$

Où V_j^i est la j^{ème} coordonnée du centroïde V^i, la taille du chromosome est égale à $K \times p$ où p est la dimension de l'espace de représentation (ici 2).

On propose ici une extension du codage de Babu et al qui autorise les chromosomes à représenter des partitions de X avec des nombres de classes variés. La population de la SE est donc constituée de chromosomes de longueurs variables.

Dans la population initiale, on calcule le plus petit hypercube (dimension p) contenant les données, et pour chaque chromosome on génère un nombre aléatoire de classes K ($2 \le K \le K_{sup}$) et $K \times p$ valeurs aléatoires qui sont les coordonnées des centroïdes des classes.

Pour construire la partition correspondant à un chromosome Z, on procède comme dans l'algorithme des centres mobiles [DID82], c'est à dire en affectant les observations aux centroïdes les plus proches au sens de la distance euclidienne. On obtient ainsi une partition en un nombre c de classes inférieur ou égal à K, le chromosome pouvant contenir certains prototypes n'attirant aucune observation. On désigne ces gènes par "gènes non codants".

Les opérateurs génétiques nouveaux :

Les opérateurs génétiques utilisés dans cette SE sont : le croisement à deux coupures (identique à celui proposé dans 2.1., suivi d'une moyenne des écarts type σ), la mutation telle qu'elle est pratiquée dans les SE (voir 1.2.), les opérateurs aléatoires d'insertion et de suppression et les opérateurs dédiés à la classification.

L'insertion aléatoire utilisée pour augmenter le nombre de classes de la partition, choisit aléatoirement dans la population un chromosome *Donneur* noté P, une position dans ce dernier notée "donne " à partir de laquelle on copiera les gènes du donneur et une position "insert " dans le receveur (chromosome courant) à partir de laquelle on insèrera les gènes copiés. On ne copiera qu'un certain nombre de gènes "is" généré aléatoirement entre deux bornes (fixées a priori). La suppression suit le processus inverse : choix aléatoire d'une position de suppression et d'un nombre "d" de gènes à supprimer. Après l'application de ces deux opérations, il est nécessaire d'affecter des objets aux classes et de recalculer les centroïdes.

Opérateurs génétiques utilisant l'information de distance par rapport aux centres de classes

L'opérateur d'insertion dédié à la classification

Pour l'insertion, on choisit d'insérer un seul centre à la fois. L'opération consiste à éclater un centre dont la classe possède une inertie importante. Pour chaque individu, le seuil d'éclatement est égal à l'inertie intra-classes de la partition qu'il code divisée par $c-1$, où c est le nombre réel de classes ($c \leq K$).

On note S l'ensemble des classes C_j dont l'inertie est supérieure au seuil "seuil_éclatement":

$$S = \left\{ C_i, i = 1..c / \frac{1}{n_i} \sum_{x \in C_i} \left\| x - V^i \right\|_A^2 \geq seuil\_éclatement \right\}$$

La probabilité qu'une classe de S subisse un éclatement est calculée ainsi :

$$P_{eclat}(C_k) = \frac{\sum_{x \in C_k} \left\| x - V^k \right\|_A^2}{\sum_{j=1}^{c} \sum_{x \in C_j} \left\| x - V^j \right\|_A^2}$$

Soit un individu I sélectionné pour subir l'opération d'insertion, soient $(V^1,...,V^c)$ ses vrais centres et soit V^k le centre choisi pour être éclaté. Celui-ci sera effectivement éclaté en V_1^k et V_2^k le long de l'axe j qui porte la plus forte variance de la façon suivante :

$$V_{i1}^k = V_i^k, \ \forall i \neq j, i = 1...p$$

$$V_{i2}^k = V_i^k, \ \forall i \neq j, i = 1...p$$

$$V_{j1}^k = V_j^k + \beta \times sigma$$

$$V_{j2}^k = V_j^k - \beta \times sigma$$

avec coordonnées V_j^k, $\forall j = 1...p$ du centre de classe V^k.

Où *sigma* est la racine carrée de la variance de la classe selon la j^{ème} direction. β a été choisi égal à 0.6. On peut alors appliquer l'opération d'insertion à individu I selon la probabilité suivante :

$$P_{insert}(I) = \frac{card(S)}{c}$$

L'opérateur de suppression de classes dédié à la classification

L'opération de suppression d'un centre est similaire à celle utilisée dans la représentation précédente (1.2.), elle est appliquée aux vrais centres des classes.

Après avoir sélectionné un chromosome, et choisit le centre selon la probabilité de fusion des classes citée dans (1.2.), on supprime ce centre des c vrais centres. Les (c-1) centres vont remplacer les K centres représentatifs de la partition.

La fonction objectif

On utilise deux possibilités : d'une part une fonction du rapport inertie interclasses sur inertie totale pénalisé et d'autre part l'indice de Davies-Bouldin, tout comme dans (1.2.).

3. Résultats

Nous testons nos algorithmes de classification sur des données artificielles (fichiers : 1, 2, 4, 5, 7, 6, 8) que l'on a générées à partir de distributions normales (en faisant varier le nombre de classes, la moyenne, et l'écart type). Les données réelles sont constituées par les données classiques des iris (quatre attributs et trois classes)

Dans ce qui suit, on utilisera les abréviations suivantes :

In_P : rapport inertie interclasses sur inertie totale pénalisé (voir 2.1.).

DB : indice de Davies-Bouldin (voir 2.1.).

% b nb classes : le pourcentage du bon nombre de classes (combien de fois l'algorithme de classification trouve le bon nombre de classes).

R, J, nb : correspondent respectivement à l'indice de Rand, l'indice de Jaccard [HUB85] et le nombre de classes. Ils sont utilisés pour comparer deux partitions indépendantes, l'une fournie par notre algorithme de partitionnement (AG ou SE), l'autre représentant la solution du problème.

$R = \frac{a+d}{C_n^2}$ et $J = \frac{a}{a+b+c}$ où :

a est le nombre de paires d'objets appartenant aux mêmes groupes dans les deux partitions.

d est le nombre de paires qui sont dans des groupes différents dans les deux partitions.

b est le nombre de paires d'objets qui sont dans le même groupe dans la première et dans des groupes différents dans la deuxième ($c = C_n^2 - a - b - d$).

Donc plus R et J sont proches de 1, plus la partition trouvée sera proche de celle de référence.

3.1. Interprétation des résultats de l'AG avec le codage par appartenance

Algorithme AG1

L'algorithme AG1 utilise les deux opérateurs classiques d'insertion et de suppression définis au début du paragraphe 2.1 en plus des opérateurs de croisement et de mutation

Les résultats ci-dessous montrent que le critère DB a tendance à sous-estimer le nombre de classes, ses résultats sont bons quand les données sont réparties en un petit nombre de classes (2 ou 3) (voir fichiers 1 et 6) et deviennent de plus en plus mauvais quand le nombre de classes augmente (voir fichiers 2, 4, 5, 8). Le même comportement a été observé dans l'étude de Dubes [DUB87].

Le critère In_P (rapport de l'inertie interclasses sur inertie totale après modification) donne de meilleurs résultats. Il donne 100% de bons classements avec les fichiers 1, 2 et d'assez bons résultats avec le fichier 6 avec des indices de Rand de 0.86 et de Jaccard de 0.75. Par contre, il classe moins bien les fichiers 5, 7 et 8. Il rencontre des difficultés dans le classement des données iris : il sépare bien les données de la première classe mais fusionne les deux dernières, ce comportement est identique à celui de l'algorithme des nuées dynamiques sur ce même exemple. En effet, les iris sont difficiles à classer car les deux dernières classes présentent un recouvrement important.

Algorithme AG2

Dans cette deuxième version de l'AG on remplace les deux opérateurs d'insertion et de suppression par des opérateurs dédiés reposant sur les heuristiques de classification définis en 2.1. Pour le critère DB, l'ajout de ces procédures a amélioré les résultats de la classification (voir fichier 8 par exemple), un test de comparaison de moyenne à 90% a prouvé que la différence entre les deux versions (pour l'indice de Rand et pour le %b nb classes) était significative. L'utilisation conjointe du critère In_P et des heuristiques de classification améliore nettement les résultats. Les résultats pour les fichiers 5, 6, 7, 8 et iris montrent bien la capacité de ce critère à discriminer les classes ainsi que l'apport des heuristiques dans le guidage de l'AG vers des régions plus intéressantes de l'espace des partitions. Pour s'en rendre compte, il suffit de comparer les résultats de deux versions de l'AG avec ce même

critère sur les fichiers 5, 6, 7, 8 et iris. Le test de Student entre les deux versions a montré que la différence entre les moyennes était significative à 90%. La moyenne de Rand pour tous les fichiers était de 0.82 avec un écart type de 0.09 pour la version n°1, elle est passée à 0.93 avec un écart type de 0.08 pour la version n°2. La moyenne du pourcentage du bon nombre de classes pour tous les fichiers était de 36.25 avec un écart type de 43.07 dans la version n°1, elle est passée à 73.35 avec un écart type de 37.39 dans la version n°2. Les résultats de ce critère sur le fichier iris sont assez bons en terme de nombre de classes trouvé et de valeurs d'indices de Rand et de Jaccard.

3.2. Interprétation des résultats de la SE avec le codage par centroïdes

Algorithme SE1

Cet algorithme est basé sur les opérateurs classiques définis au début du paragraphe 2.2. (en particulier : Insertion et suppression aléatoires).

Les résultats ci-dessus sont proches de ceux obtenus par l'AG et montrent aussi bien la supériorité du critère In_P par rapport au critère DB.

Algorithme SE2

Dans cette deuxième version de la SE on remplace les deux opérateurs d'insertion et de suppression aléatoires par des opérateurs dédiés reposant sur les heuristiques de classification définis en 2-2. Nous basons leur utilisation sur les résultats de classification obtenus par SE1.

Au lieu d'enrichir la SE par des opérateurs génétiques supplémentaires intégrant des heuristiques de classification pour mieux la guider vers des solutions meilleures, nous proposons ici une approche plus spécialisée et qui consiste à "bénéficier " de l'étude du comportement intrinsèque des critères (SE1) pour aider la SE à choisir des opérateurs génétiques en fonction du critère utilisé (En fonction des résultats de classification obtenus par chaque critère, on décide d'utiliser plus fréquemment l'opérateur de suppression que celui de l'insertion ou vice versa). Par exemple, pour le critère DB, on a choisi d'utiliser les

opérateurs qui favorisent une augmentation du nombre de classes (insertion), puisqu'il avait tendance à sous estimer le nombre de classes. Les résultats de la SE avec ce critère (fichier2, fichier7) illustrent bien l'avantage de ce choix (voir les tableaux SE1, et SE2 pour ce critère et pour les fichiers 2 et 7). Malheureusement les résultats de la SE restent mauvais pour les fichiers 4, 5 et 8 où le nombre de classes est élevé (4 ou 5). Cet échec peut provenir du fait que nos opérateurs intégrant les heuristiques de classification n'ajoutent qu'une classe à la fois et l'éclatement résultant n'est pas toujours optimal. Un test de Student (avec un seuil de rejet de 10%) a montré que la différence entre les deux versions de la SE utilisant le critère DB était non significative pour l'indice de Rand (pas d'amélioration) mais significative pour le nombre de fois où l'algorithme trouve le bon nombre de classes. Le critère In_P obtient de meilleurs résultats (les opérateurs de fusion et de suppression sont appliqués d'une façon équivalente) sur les fichiers utilisés que le critère de DB ou les autres critères (Hubert et Huber modifié voir [KET97]). Les opérateurs de fusion et d'insertion améliorent les résultats de ce critère. Par exemple pour le fichier 4 le %b nb de classes est passé de 0% à 20%, pour le fichier 6 de 20% à 70%, pour le 7 de 60% à 100%. Même si pour le fichier 8 le « vrai » nombre de classes n'a jamais été trouvé, la moyenne du nombre de classes et le %b nb classes ont augmenté dans la deuxième version. Pour le fichier iris le %b nb classes est passé de 60% à 50% mais les indices de Rand et de Jaccard ont été améliorés. Le critère In_P ayant un comportement moyen, la différence entre les résultats des deux versions n'est pas significative que ce soit pour l'indice de Rand ou pour le % b nb classes. D'après ces résultats on remarque que l'effet de la fusion est plus "efficace" que celui de l'éclatement. En effet, il est plus facile de choisir de fusionner deux classes que d'en éclater une (plusieurs possibilités). Ceci explique la différence entre les résultats de l'augmentation du % b nb classes pour DB et In_P (fichiers 4, 5 et 8).

4. Conclusions et perspectives

Dans ce travail, nous nous sommes intéressés au problème de la classification automatique par partition et à l'estimation du nombre de classes par AE, nous avons implémenté des algorithmes qui recherchent en parallèle le nombre de classes et la classification appropriée. Pour cela nous avons utilisé un nouveau type de chromosomes : les chromosomes de longueur variable, et défini des opérateurs génétiques qui leur sont spécifiques. Nous nous sommes

heurtés au problème de la mesure de qualité d'une partition avec un nombre de classes inconnu et avons modifié certains critères de classification soit d'une façon explicite en les pénalisant dans leur formulation mathématique soit d'une façon implicite en biaisant leur comportement par compensation (choix d'opérateurs génétiques ou de probabilité d'application de ces opérateurs en fonction du critère utilisé). En effet, l'étude du comportement pur des critères avec les AE nous a permis d'acquérir (ou de confirmer) des connaissances a priori sur leur tendance (sous-estimation / surestimation du nombre de classes...). Cette "contrôlabilité" de nos AE ne s'est pas limitée au choix d'opérateurs classiques appropriés mais est aussi due à l'introduction de nouveaux opérateurs utilisant des heuristiques de classification. Les résultats obtenus ont montré l'apport de la supervision dans nos AE. Par ailleurs, nous avons comparé nos résultats de classification (AG avec le critère du rapport inertie interclasses sur inertie totale pénalisé) aux résultats de l'algorithme GMVE [JOL91]. Ils leur sont supérieurs en classement et en fréquence d'estimation du bon nombre de classes sur les fichiers utilisés. Pour tirer profit des capacités d'exploration globales de nos AE de classification, nous envisageons de les utiliser en hybridation avec des algorithmes classiques (moyennes mobiles...). L'AE fournit une solution qui sera le point de départ d'un algorithme de classification itératif.

Tableaux des résultats

Les résultats sont présentés sous forme de tableaux dans lesquels, les statistiques sont calculées (moyenne, écart type, valeur min, valeur max) sur 10 exécutions pour deux critères de classification (DB, In_P). Les mesures sont : indice de Rand, indice de Jaccard et % b nb classes. Pour chaque fichier (tableau), en haut à gauche, sont rappelés le nombre de classes entre parenthèses, et les %b nb classes pour les deux critères DB et In_P.

fic1(2C)0%0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	1	1	2	1	1	2
écart-type	0 01	0,01	0	0	0	0
valeur max	1	1	2	1	1	2
valeur min	0 98	0,96	2	1	1	2

100%100%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	1	1	2	1	1	2
écart-type	0	0	0	0	0	0
valeur max	1	1	2	1	1	2
valeur min	1	1	2	1	1	2

fic2(2C)0%0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,68	0,53	2	0,89	0,79	2 7
écart-type	0,18	0,11	0	0,11	0,2	0,48
valeur max	0,77	0,59	2	1	1	3
valeur min	0,34	0 32	2	0,77	0 56	2

30%100%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,77	0 58	2,33	1	1	2
écart-type	0,01	0,02	0,58	0	0	0
valeur max	0,78	0 59	3	1	1	2
valeur min	0 77	0,57	2	1	1	2

fic4(4C)0%/0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,39	0,28	2,1	0,83	0,59	2 7
écart-type	0,15	0 06	0,32	0,06	0 08	0,48
valeur max	0,6	0,37	3	0,87	0,66	3
valeur min	0,25	0,24	2	0,75	0,49	2

0%/0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,75	0,49	2	0,74	0 49	2
écart-type	0	0	0	0	0	0
valeur max	0,75	0,49	2	0,74	0,49	2
valeur min	0,75	0 49	2	0 74	0,49	2

fic5(5C)/0%/0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,31	0,21	2 1	0,72	0 39	2,75
écart-type	0,11	0,03	0 45	0,05	0,06	0,55
valeur max	0,49	0,27	4	0,858	0,53	4
valeur min	0,2	0 19	2	0,66	0,28	2

0%/0%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,66	0,35	2	0 83	0,5	4,4
écart-type	0,01	0,01	0	0 01	0,04	1,07
valeur max	0,67	0,37	2	0,84	0,54	6
valeur min	0 65	0,33	2	0,82	0 44	3

fichier6(2C)100%/90%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,65	0,57	2	0,86	0,75	2,15
écart-type	0,15	0,12	0	0 04	0,08	0 37
valeur max	0 9	0,82	2	0,92	0 86	3
valeur min	0 49	0,46	2	0,77	0 56	2

100%/20%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0 83	0,71	2	0 76	0,57	3,1
écart-type	0 01	0,02	0	0,06	0 12	0 74
valeur max	0,85	0,74	2	0 86	0,76	4
valeur min	0,81	0 69	2	0 67	0,4	2

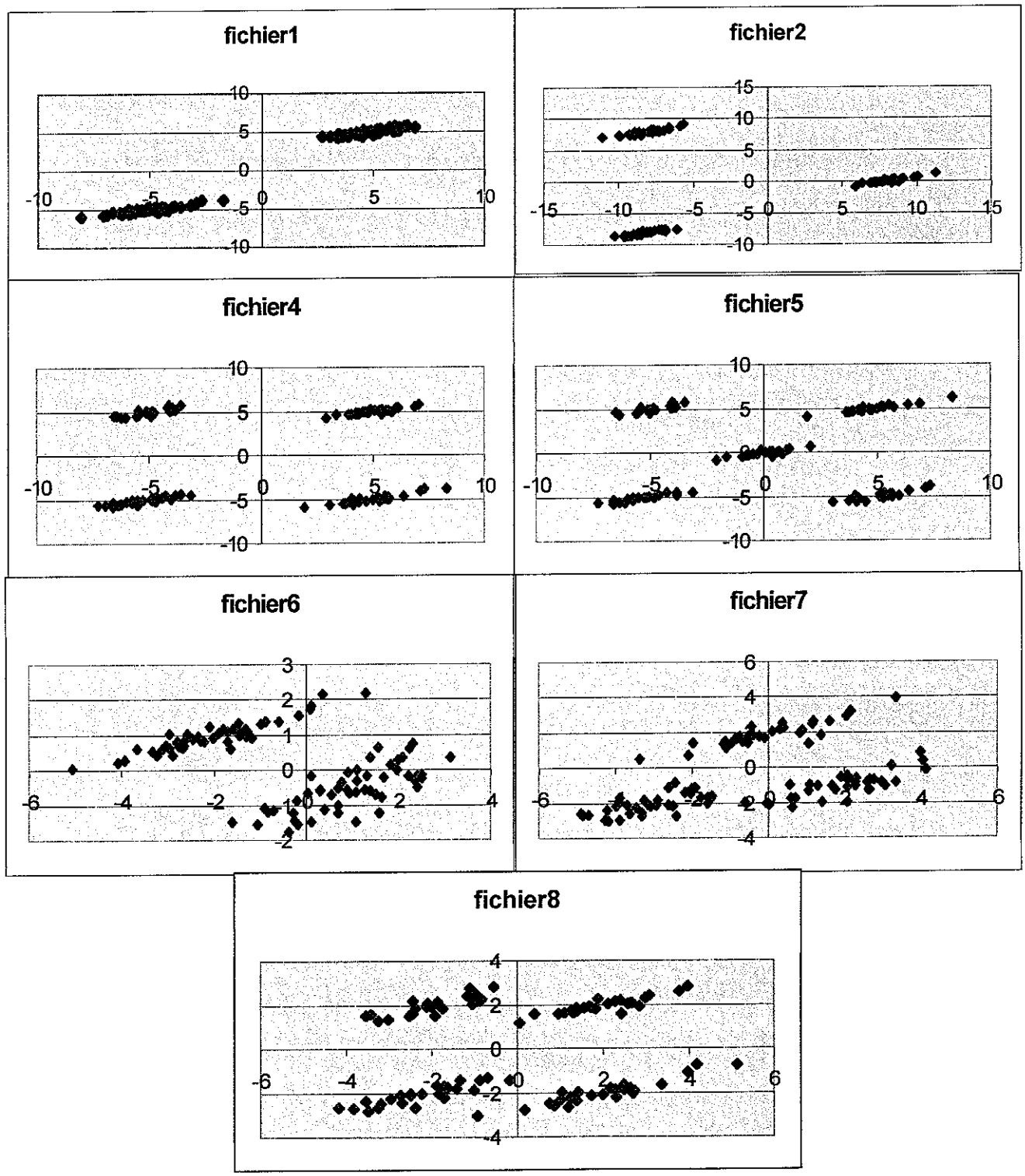
fic7(3C)0%/20%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,54	0,42	2	0,76	0 39	2,2
écart-type	0,17	0,11	0	0,03	0 03	0 42
valeur max	0 78	0,6	2	0,83	0 6	3
valeur min	0,33	0,32	2	0,74	0 48	2

0%/60%	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0 75	0,56	2	0 79	0,53	3,2
écart-type	0 01	0,01	0	0 05	0,06	0,63
valeur max	0 75	0,58	2	0 84	0,61	4
valeur min	0,73	0 54	2	0 66	0,43	2

fic8(4C)0%/10%	*R*	*J*	*nb*	*R*	*J*	*nb*	*0%/10%*	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,32	0,26	2	0 78	0,5	2,8	moyenne	0,74	0,53	2	0,75	0,45	2.6
écart-type	0,11	0,4	0	0,05	0,05	0 63	écart-type	0,01	0,01	0	0,01	0,04	0,52
valeur max	0,58	0,34	2	0 87	0,58	4	valeur max	0,75	0 49	2	0,76	0,49	3
valeur min	0,25	0 19	2	0,74	0,42	2	valeur min	0,74	0 48	2	0 74	0,42	2

Iris(3C 4D)0%/0%	*R*	*J*	*nb*	*R*	*J*	*nb*	*(3C,4D)0%6/0 %*	*R*	*J*	*nb*	*R*	*J*	*nb*
moyenne	0,63	0 45	2	0 75	0,55	2	moyenne	0,72	0 5	2	0 8	0,58	2 6
écart-type	0,16	0,07	0	0 01	0,01	0	écart-type	0,01	0 01	0	0,04	0,04	0,52
valeur max	0,73	0 52	2	0 76	0,57	2	valeur max	0,73	0 51	2	0 86	0,61	3
valeur min	0,33	0,33	2	0 74	0,53	2	valeur min	0,72	0,49	2	0,75	0 5	2

Représentations graphiques des fichiers de données



REFERENCES BIBLIOGRAPHIQUES

[BAB94] G. P. Babu, and M. N. Murty, "Clustering with evolution strategies ", Pattern recognition, vol 27, N°2 pp 321-329, 1994.

[BAL64] G. H. Ball et D. J. Hall, "Some fundamental concepts and synthesis procedures for pattern recognition preprocessors ", International Conference on Microwaves, Circuit Theory, and Information Theory, September, Tokyo, 1964.

[CHA99] I. Charon, O. Hurdy et C. O. Rubina, "Algorithmes génétiques hybridés pour un problème de classification automatique", SFC'99, 15-17 Septembre 1999, Nancy, France.

[CUC93] R. Cucchiara, "Analysis and comparison of different genetic models for clustering problem in image analysis ", Proceedings of Artificial Neural Nets and Genetic Algorithms Conference, R. F Albrecht, C. R Reeves and N. C. Steele (Eds.), Austria, 1993.

[DAV79] D. L. Davies and D. W. Bouldin, "A cluster separation measure", IEEE Transactions on Pattern Analysis ans Machine Intelligence PAMI 1, pp 224-227, 1979.

[DID82] E. Diday, J. Lemaire, J. Pouget et F. Testu, "Eléments d'analyse de données", ED. Dunod, 1982.

[DUB87] R. C. Dubes, "How many clusters are best?- An experiment", Pattern Recognition, vol 20, N°6, pp 645-663, 1987.

[GOL90] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms ", FOGA, pp 69-93, 1990.

[HAR92] I. Harvey, "The SAGA cross : the mechanisms of recombination for species with variable-length genotypes ", In R. Männer and B. Manderick, Eds., Parallel problem solving from nature, North-Holland, Amsterdam, pp 269-278, 1992.

[HOL75] J. Holland, "Adaptation in natural and artificial systems ", University of Michigan Press, Ann Arbor, 1975.

[HUB85] L. Hubert, P. Arabie, "comparing partitions", journal of classification, 2, pp 193-218, 1985.

[JOL91] J. M Jolion, P. Meer and S. Bataouche, "Robust clustering with applications in computer vision", IEEE trans. PAMI vol 13, n°8, pp 791-801, August, 1991.

[KAN82] A. Kandel, "Fuzzy techniques in pattern recognition ", John Wiley & Sons, Inc, 1982.

[KET96a] F. Z. Kettaf et J. P. Asselin de Beauville, "Genetic and Fuzzy based clusetering ", proceedings of the IFCS'96, Kobe, 1996.

[KET96b] F. Z. Kettaf et J. P. Asselin de Beauville, «Evolution strategie for clustering data drawn from a mixture model ", ICML'96 workshop on evolutionary computing and Machine Learning, Bari, 2-3 Juillet 1996.

[KET97] F. Z. Kettaf, "Contributions des algorithmes évolutionnaires au partitionnement des données ", Thèse de doctorat, Laboratoire d'Informatique, Université de Tours, 1997.

[KLE89] R. W Klein and R. Dubes, "Experiments in projection and clustering by simulated annealing ", Pattern Recognition, vol 22, pp 213-220, 1989.

[MIC92] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs ", Springer Verlag, 1992.

[REC73] Rechenberg, "Evolution strategies : Optimierung technischer systeme nach prinzipien deer biologischen evolution. ", Fromman Holzboog, verlag Stuttgart, 1973.

[SCH81] H. P. Schewefel, "Numerical optimization of computer models ", wiley, New York, 1981.

[SRI95] R. Srikanth, R. Goerges, N. Warsi, "A variable length genetic algorithm for clustering and classification ", Pattern Recognition Letters volume 16, N°8, August 1995.