République Algérienne Démocratique et Populaire
Ministre de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE MENTOURI DE CONSTANTINE
FACULTE DES SCIENCES DE L'INGENIEUR
DEPARTEMENT D'INFORMATIQUE

N° d'ordre :…..
Série :…..

MEMOIRE

Présenté pour l'obtention du diplôme de
Magister en Informatique

Approches bio-inspirées pour la reconnaissance de formes

Présenté par :

Mr. Deneche Abdelhakim

Dirigé par :

Pr. Meshoul .S

Soutenu le : .. /.. /2006

بسم الله الرحمان الرحيم

"و في الأرض آيات للموقنين ۞ و في أنفسكم أفلا تبصرون"

سورة الذاريات 19-20

"سنريهم آياتنا في الآفاق و في أنفسهم حتى يتبين لهم أنه الحق أولم يكف بربك أنه على كل شيء شهيد"

سورة فصلت 53

صدق الله العظيم

الحمد لله الذي هدانا لهذا وما كنا لنهتدي لولا أن هدانا الله

A monsieur et madame Batouche pour m'avoir transmis leur passion pour la recherche.

Aux enseignants pour avoir gracieusement partagé leur savoir avec moi.

A ma famille pour tout l'amour qu'elle me porte.

A mes amis, parce que c'est mes amis.

Au professeur S. M. Garrett pour m'avoir donné les informations dont j'avais besoin.

A tous ceux qui de prés ou de loin ont contribué à ce projet.

A tous les anticorps qui ont volontairement participé à ce travail.

A tous je dis merci

A ma famille que j'aime…

Résumé

Dans le cadre de ce magistère, nous nous proposons de concevoir un système de reconnaissance des chiffres imprimés. La reconnaissance des formes est un vaste champ de recherche, indispensable à tout système intelligent d'aide à la décision. En nous inspirant des systèmes immunitaires nous essayons de capturer plusieurs propriétés utiles à la reconnaissance, comme la mémorisation et l'adaptation. Les deux algorithmes développés ont pour base l'algorithme d'apprentissage bio inspiré CLONCLAS. Le premier algorithme proposé se nomme FCC pour Fast Clonal Classification. Cet algorithme utilise une nouvelle formule pour le calcul des clones qui est plus proche du modèle réel. La fonction affinité a elle aussi été optimisée rendant la complexité de l'apprentissage totalement indépendante de la taille de l'ensemble d'entraînement. Nous avons aussi ajouté un module de sélection d'attributs pour réduire encore plus la complexité. Le second algorithme nommé C^3 (pour Clonal Classification + Clustering) est une combinaison de la sélection clonale et du clustering. L'algorithme ainsi obtenu exploite des caractéristiques du système naturel qui lui permettent d'être plus performant face aux images non normalisées.

Abstract

In this work, we address an interesting but difficult problem: printed digit recognition. Pattern recognition is a wide research field, very useful for building intelligent decision making systems. Taking inspiration from natural immune systems, we try to grab useful properties such as automatic recognition, memorization and adaptation. We developed two new algorithms devoted to the learning stage of the recognition system. Both algorithms are based upon CLONCLAS, a bio inspired algorithm for pattern recognition. The first algorithm, named Fast Clonal Classification, proposes three improvements that reduce the complexity of CLONCLAS. The number of clones is calculated using a new formula that is closer to the natural system. Affinity calculation has been optimized to let the learning stage perform fast even when the training set is big. The attributes are reduced by using a multi objective algorithm. The second algorithm, named C^3 (for Clonal Classification + Clustering), handles non normalized digits by using a property of natural immune systems combined to clustering algorithms. This algorithm performs well compared to CLONCLAS.

Table des matières :

Table des figures :

Introduction générale :

L'ordinateur a toujours eu comme principal objectif l'automatisation des tâches effectuées par les êtres humains. Soit pour libérer l'homme des travaux répétitifs, ou bien éviter les erreurs de calcul ou encore accélérer les traitements. Certaines activités, considérées comme banales par la plupart des individus, sont pourtant très difficiles à reproduire par un ordinateur. Par exemple, la reconnaissance faciale est très difficile à automatiser, alors que les enfants dès leur plus jeune âge sont capables de reconnaître les personnes à leur visage.

La reconnaissance des formes est la discipline qui tente d'automatiser les mécanismes de reconnaissance utilisés par les êtres vivants en général, et par les êtres humains en particulier. Malgré plusieurs décennies de recherche, aucune solution générale n'a pu être développée. Il est devenu évident que les performances optimales ne peuvent être atteintes qu'en construisant des solutions spécifiques. La reconnaissance des formes peut être vue comme un problème de classification. Par exemple, un système de reconnaissance de chiffres manuscrits va donner pour chaque forme qu'il reçoit en entrée, sa classe la plus pertinente (0, 1, …,9). La reconnaissance des formes est aussi un problème d'apprentissage, le système doit généralement apprendre tout seul comment distinguer entre les formes des différentes classes.

De façon générale un système de reconnaissance des formes peut être divisé en quatre modules **[Woznica et Menal, 03]**: la préparation des données, l'apprentissage, la classification et le post traitement. Résoudre un problème de reconnaissance revient à construire un système qui donne les meilleures performances pour l'application ciblée. Cela implique l'optimisation de chacun des modules du système pour donner les meilleures solutions possibles, et comme les modules ne sont pas indépendants les uns des autres, le choix et l'optimisation d'un module influence les autres modules. Par exemple, le choix du classificateur influence le choix du module d'apprentissage utilisé.

Durant les dernières décennies, plusieurs méthodes et algorithmes ont été développés pour chacun des modules de reconnaissance. Chaque méthode étant plus adaptée à un certain type de problèmes. Au fur et à mesure que la puissance des machines croit, de nouvelles applications pour la reconnaissance voient le jour, pour lesquels il faut souvent développer de nouvelles solutions. La complexité de certains problèmes est telle que, les méthodes algorithmiques standards ne sont pas applicables. De nouvelles voies ont alors été explorées par les chercheurs, parmi lesquelles **le calcul bio inspiré**.

L'informatique bio inspirée est la discipline qui consiste à étudier les systèmes naturels afin d'y trouver des solutions pour des problèmes informatiques qui ne peuvent être résolus par les méthodes classiques. En effet, les chercheurs ont noté beaucoup de similitudes entre les problèmes que rencontrent les systèmes naturels et les problèmes informatiques. Un exemple d'algorithmes bio inspirés sont les réseaux de neurones, qui tentent de reproduire les mécanismes inhérents au cerveau humain afin de résoudre des problèmes d'apprentissage, ou

encore les algorithmes génétiques qui s'inspirent des mécanismes naturels assurant la diversité des espèces vivantes.

Les systèmes immunitaires sont des systèmes naturels qui présentent des caractéristiques très utiles à la reconnaissance des formes. Non seulement ils peuvent faire la différence entre les cellules du corps et les molécules étrangères, mais en plus ils peuvent mémoriser ces molécules pour y répondre de façon plus efficace lorsqu'elles sont rencontrées de nouveau. En plus, la réponse est spécifique à l'agent infectieux, ce qui indique que les systèmes immunitaires peuvent faire la distinction entre les molécules étrangères.

Plusieurs algorithmes ont été inspirés des systèmes immunitaires, et utilisés dans des domaines variés tels que la sécurité informatique **[Esponda et Forrest, 03]**, la détection de pannes **[Aicklin et Cayzer, 02]**, le datamining **[Secker et al, 03]**, l'optimisation mono et multi objective **[Coello et Cortez, 05]**, ainsi que pour la reconnaissance des formes **[White et Garrett, 03]**. Ces algorithmes peuvent être classés en trois grandes familles selon le mécanisme naturel à l'origine de l'implémentation : la sélection négative, la sélection clonale et les réseaux immunitaires. La sélection négative s'inspire des mécanismes qui permettent au système naturel de faire la différence entre les molécules non dangereuses et les molécules dangereuses. La sélection clonale s'inspire des mécanismes de mémorisation, et les réseaux immunitaires s'inspirent de la théorie du même nom.

Dans le cadre de ce mémoire, nous allons nous intéresser à l'application des principes des systèmes immunitaires naturels pour l'apprentissage d'un système destiné à la reconnaissance des chiffres imprimés. Dans les systèmes immunitaires, les mécanismes qui ressemblent le plus à l'apprentissage sont ceux responsables de la mémorisation des molécules étrangères, c'est-à-dire les mécanismes de sélection clonale. Les premières tentatives pour appliquer la sélection clonale à la reconnaissance des formes sont celles de de Castro et von Zuben, qui ont développé CLONALG pour la reconnaissance de chiffres imprimés **[de Castro et von Zuben, 02]**. CLONALG ne représente cependant qu'une preuve de concept et plusieurs limitations empêchent son utilisation pour des applications réelles. White et Garrett ont proposé CLONCLAS **[White et Garrett, 03]** comme une amélioration de CLONALG, et bien que certaines limitations aient été éliminées, le système résultant reste pénalisé par une complexité importante et de faibles performances face aux données non normalisées.

Nous allons, dans ce travail, proposer deux algorithmes d'apprentissage basés sur CLONCLAS. Le premier algorithme, nommé **FCC** (pour Fast Clonal Classification) est comme son nom l'indique, une réponse aux problèmes de complexité de CLONCLAS. Pour cela, une nouvelle formule de calcul du nombre de clones, une optimisation de la fonction de calcul des affinités (le cœur même de CLONCLAS), et un module de sélection d'attributs, vont permettre non seulement de réduire la complexité de façon significative, mais aussi les besoins en mémoire de l'algorithme. Le second algorithme, nommé $\mathbf{C^3}$ (pour Clonal

Classification + Clustering) permet d'améliorer les performances du système face aux données non normalisées, en utilisant les principes du clustering.

Ce mémoire est organisé comme suit : le chapitre I permettra au lecteur non initié de se familiariser avec la reconnaissance des formes, mais vu l'immensité du domaine ce chapitre ne peut dépasser la modeste introduction. Le chapitre II traite des systèmes immunitaires naturels et des principales familles d'algorithmes qui s'en inspirent. Le chapitre III donne une description détaillée de CLONCLAS et des résultats qu'il a obtenu, puis introduit FCC ainsi que les améliorations apportées. Le chapitre IV décrit l'algorithme **C^3**, développé en quatre variantes, puis une comparaison est faite entre ces méthodes en terme de performances et de limitations. Nous terminerons par une conclusion générale qui résume les travaux effectués ainsi que les perspectives de recherche.

1 La Reconnaissance des formes :

1.1 Introduction :

L'un des principaux objectifs de l'ordinateur est d'automatiser les taches habituellement effectuées par l'homme. La plupart des activités humaines reposent sur une faculté importante de notre cerveau : pourvoir distinguer entre les formes. Par exemple, reconnaître une pomme d'une orange ; un bon diagnostic d'un mauvais, ou encore un sous-marin d'une baleine. Pour pouvoir automatiser ce genre de tâches, l'ordinateur doit obligatoirement acquérir la faculté de reconnaître les formes.

Watanabe **[Watanabe, 85]** a défini une forme comme : « *l'opposé du chaos ; c'est une entité vaguement définie, à laquelle ont peut associer un nom* ». En des termes informatiques, une forme est un ensemble de valeurs, appelés *attributs*, auxquels est associé un nom (ou étiquette), qui est leur *classe***.** Plusieurs formes peuvent avoir la même classe, on dit alors que ce sont les *exemples* ou *réalisations* de la classe.

Le problème que cherche à résoudre la reconnaissance des formes est d'associer une classe à une forme inconnue (qui n'a pas encore de classe associée). On considère souvent la Reconnaissance des formes comme un problème de classification : trouver la fonction qui affecte à toute forme inconnue sa classe la plus pertinente. Elle est partie intégrante de tout système intelligent destiné à la prise de décision **[Theodoridis et Koutroumbas, 03]**.

1.1.1 Applications typiques de la reconnaissance des formes :

- **Marketing :** La reconnaissance des formes est souvent utilisée pour classer les consommateurs selon les produits qu'ils sont susceptibles d'acheter. Elle est aussi utilisée par les sociétés de vente pour classer les clients selon qu'ils soient de bons ou mauvais payeurs, ou encore selon qu'ils vont oui ou non passer à la concurrence **[Chung et al, 03]**.
- **Finances :** les systèmes de reconnaissance des formes sont utilisés pour la détection de transactions bancaires frauduleuses ainsi que la prédiction des banqueroutes **[Kaastra et Boyd, 96]**.
- **Usinage :** la qualité des produits dépend souvent de paramètres ajustables, et les relations exactes entre la qualité et les valeurs des paramètres n'est pas claire. Les systèmes de reconnaissance des formes sont utilisés pour classer les paramètres selon la qualité des produits qu'ils sont susceptibles de générer. Ils permettent ainsi de réduire le nombre d'essais ce qui fait gagner du temps et de l'argent **[Gupta et al, 95]**.
- **Energie :** les systèmes de reconnaissance des formes sont utilisés pour prévoir la consommation électrique (réduite, normale, élevée), permettant ainsi aux clients de

réduire si nécessaire leur consommation, et aux producteurs de mieux gérer leurs unités de production **[Hippert et al, 01]**.

- **Lecture automatisée :** les systèmes de reconnaissance des formes permettent de numériser les anciens documents ainsi que les archives, non pas sous la formes d'images, mais plutôt sous une forme textuelle **[Munich et Perona, 02]**.
- **Sécurité :** la reconnaissance vocale et rétinienne sont un exemple d'applications typiques de la reconnaissance des formes pour l'authentification. La vérification des signatures est aussi très populaire **[Yang et al, 02]**.

1.1.2 Schéma général d'un système de Reconnaissance des formes :

La majorité des systèmes de Reconnaissance des formes ont le schéma de fonctionnement suivant **[Woznica et Menal, 03]** (figure 1.1).

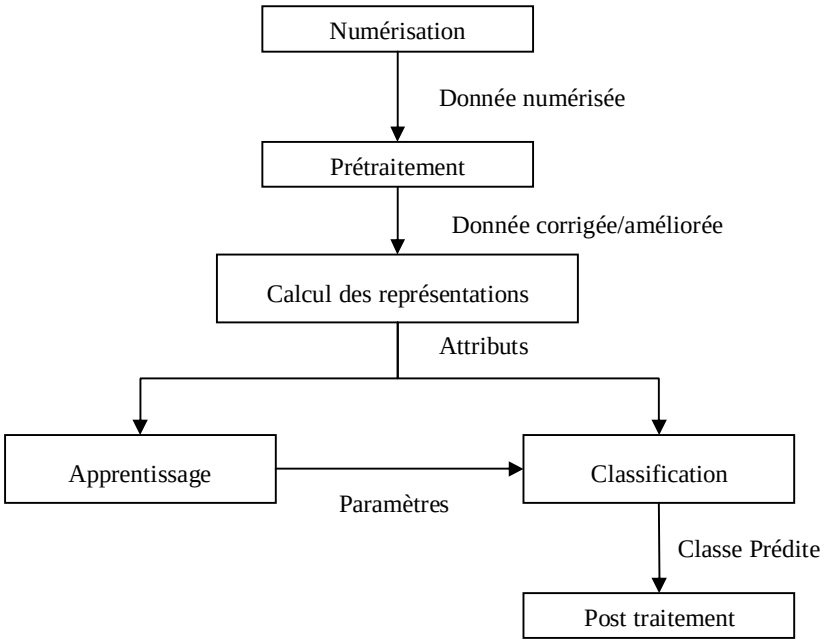

Figure 1-1 Schéma général d'un système de reconnaissance des formes

1. **Préparation des données :**
    a. **Numérisation** : À partir des informations du monde physique, construire une représentation des données directement manipulable par la machine. Des capteurs (microphone, caméra, instruments de mesure) convertissent les signaux reçus du monde réel en une représentation numérique discrète. L'espace résultant, appelé **espace de représentation**, a une dimension **r** très grande lui permettant de disposer du maximum d'informations sur les formes numérisées.
    b. **Prétraitement** : Consiste à sélectionner dans l'espace de représentation l'information nécessaire au domaine d'application. Cette sélection passe souvent par l'élimination du bruit, la normalisation des données, ainsi que par la suppression de la redondance. Le nouvel espace de représentation a une dimension **r'** très inférieure à **r** mais demeure un espace de grande dimension et contient des informations encore assez primitives.
    c. **Calcul des représentations** : Il s'agit de la phase finale de la préparation des données. Elle fournit un certain nombre de **caractéristiques** ou **paramètres** (les fameux attributs) en utilisant des algorithmes de sélection et/ou d'extraction d'attributs. Les attributs étant limités en nombre, l'**espace des paramètres** ainsi obtenu est de dimension **p** très petite par rapport à **r'**.
2. **Apprentissage :** L'apprentissage ou entraînement, est une partie importante du système de reconnaissance. Le classificateur étant généralement une fonction paramétrique, l'apprentissage va permettre d'optimiser les paramètres du classificateur pour le problème à résoudre, en utilisant des données d'entraînement. Lorsque les données d'entraînement sont préalablement classées, l'apprentissage est dit **supervisé**, sinon il est **non supervisé [Theodoridis et Koutroumbas, 03]**.
3. **Classification** : cette phase est le noyau de la Reconnaissance des formes. En utilisant les modèles (paramètres) obtenus lors de l'apprentissage, le classificateur assigne à chaque forme inconnue sa ou ses formes les plus probables.
4. **Post traitement** : cette phase a pour but de corriger les résultats de la classification en utilisant des outils spécifiques au domaine d'application. Par exemple pour un système de reconnaissance de textes manuscrits, le classificateur se charge de classer chaque caractère séparément, alors que le post traitement applique un correcteur orthographique sur tout le texte pour valider et éventuellement corriger le résultat de la classification. Bien que facultative, cette phase permet d'améliorer considérablement la qualité de la reconnaissance.

1.1.3 Notions essentielles :

Avant d'aborder les étapes de la reconnaissance, nous allons définir quelques notions essentielles à la bonne compréhension du chapitre. Pour plus de détails consulter **[Theodoridis et Koutroumbas, 03]**.

Définition 1 : Population

Une forme est constituée d'un ensemble d'attributs qui représentent la description de la forme. L'ensemble des formes est **la population**. Prenons pour exemple le domaine médical : Etablir un diagnostic signifie être capable d'associer le nom d'une maladie à un certain nombre de symptômes présentés par le malade. Les malades forment la population, les symptômes sont les descriptions des malades, et les maladies sont les classes.

On dispose pour chaque description d'un ensemble d'attributs qui prennent leurs valeurs dans des domaines ($D_1, D_2, \mathbf{K}, D_n$). Décrire un élément de la population consiste alors à attribuer une valeur à chacun de ses attributs. L'espace de description est alors le produit cartésien $D_1 \times D_2 \times \mathbf{K} \times D_n$.

Définition 2 : fonctions de description et de classement

Soit une population **P**, un ensemble de descriptions **D**, et un ensemble de classes **{1,…,c}**. **La fonction de description** $X : P \to D$ est la fonction qui associe une description à tout élément de la population. On suppose qu'il existe un classement $Y : P \to \{1, \mathbf{K}, c\}$ qui associe une classe à tout élément de la population. Y étant inconnue, le but du système de reconnaissance est de trouver **la fonction de classement** $C : D \to \{1, \mathbf{K}, c\}$ telle que $C \mathbf{o} X = Y$, ou de manière plus réaliste, telle que $C \mathbf{o} X$ soit une bonne approximation de Y.

Définition 3 : erreur de classification

Pour pouvoir comparer les fonctions de classification on utilise l'**erreur de classification** ou **erreur réelle** : soit C une fonction de classification, l'erreur $E(d)$ pour une description d est la probabilité qu'un élément de la population P de description d soit mal classé. L'erreur de classification $E(C)$ est la moyenne pondérée des erreurs sur les descriptions d.

Définition 4 : erreur apparente

En pratique, comme la fonction de classification est construite à partir d'un échantillon d'exemples (l'ensemble d'entraînement), on utilise plutôt l'**erreur apparente** : soit S un échantillon et C une procédure de classification, le taux d'erreur apparent sur S est :

$$E_{app}(C) = \frac{err}{|S|} \qquad (1.1)$$

Où **err** est le nombre d'exemples de S qui sont mal classés par C.

Il faut noter que l'erreur réelle est indépendante de l'échantillon alors que l'erreur apparente est mesurée sur l'échantillon. Le problème est donc de concevoir des méthodes qui vont générer des procédures de classification d'erreur apparente petite tout en assurant une erreur réelle petite. Seulement voila, nous sommes confrontés aux difficultés suivantes :

- L'erreur apparente est, en général, une version très (trop) optimiste de l'erreur réelle.
- L'espace de toutes les fonctions de **D** dans **1,..., c** est de taille considérable et il est impossible d'explorer complètement cet espace.

En plus, utiliser le même échantillon pour construire le classificateur et juger de sa validité conduit souvent à un **sur apprentissage** : le système apprend trop bien les exemples qu'on lui donne (l'échantillon) et n'est plus capable de traiter correctement des données inconnues.

Afin de contrôler ce risque et, de manière générale, pouvoir connaître la qualité du classificateur, on découpe la base en 2 parties. L'une pour l'apprentissage, et le reste pour le contrôle. Il faut noter que si l'ensemble d'apprentissage est petit, le classificateur résultant aura un faible pouvoir de généralisation (la capacité à classifier correctement de nouvelles données). D'un autre coté, si l'ensemble de vérification est petit, alors la confidence de l'erreur estimée sera faible.

En plus du taux d'erreurs, une autre mesure utile est le **taux de rejets**. Supposons qu'une forme est prés de la frontière de décision entre deux classes. Bien que les règles de décision puissent classifier cette forme correctement, la classification sera faite avec une faible confidence. Une meilleure alternative serait de rejeter les formes douteuses au lieu de les classifier. Le rejet réduit le taux d'erreurs ; plus le taux de rejets est grand, plus le taux d'erreurs est petit. Le choix d'un bon taux de rejets est basé sur les coûts liés aux décisions de rejet et aux décisions erronées.

1.2 Préparation des données :

Lors de la préparation des données, les données réelles sont d'abord numérisées par différents types de capteurs (scanner, caméra, …). Vient ensuite une phase de **prétraitement** qui a pour objectif d'améliorer la qualité des données numérisées. La phase de prétraitement dépend fortement du domaine d'application, par exemple pour les systèmes de reconnaissance de texte les opérations de prétraitement les plus usuelles sont la correction de l'inclinaison du texte (ou de skewing), l'amélioration de la qualité (élimination du bruit), ainsi que la correction de l'orientation du texte **[Tzanakou, 00]**. Vient ensuite la phase de **calcul des**

représentations qui va se charger de représenter les données sous la forme d'ensembles d'attributs. Souvent, d'autres traitements sont nécessaires avant la phase de calcul des représentations. Par exemple, pour les systèmes de reconnaissance de texte, la segmentation est une étape cruciale de la phase de préparation des données.

1.2.1 Segmentation

Remarquez que dans nos activités quotidiennes, nous tendons toujours à écrire les chiffres de façon connectée. Pour pouvoir reconnaître ces chiffres, nous devons d'abord les séparer les uns des autres.

En se basant sur la longueur des chaînes numériques, les méthodes de segmentation peuvent être divisées en deux classes. Les $1^{ères}$ doivent séparer des chaînes de chiffres de longueur inconnue, les $2^{èmes}$ doivent segmenter des chaînes de chiffres d'une longueur spécifique. Une grande variété d'applications appartient à cette classe, comme par exemple, les codes postaux ou encore les dates. Bien que la connaissance de la longueur de la chaîne réduit la complexité du problème, il ne devient pas pour autant simple à résoudre.

Les algorithmes de segmentation peuvent être divisés en trois catégories **[El Nager et Al Hadj, 2005]** : les algorithmes basés régions, les algorithmes basés contours et les algorithmes basés sur la reconnaissance. Les algorithmes basés régions commencent par identifier les régions appartenant au fond de l'image, puis extraient quelques primitives comme les vallées et les boucles, qui sont utilisées pour construire le chemin de segmentation. Les méthodes basées contours analysent les contours des chiffres connectés pour trouver des primitives de structure tel que les coins, la distance entre le contour du haut et celui du bas, les boucles et les arcs. Les méthodes basées sur la reconnaissance font appel à un algorithme de reconnaissance pour vérifier la validité de la segmentation, elles sont donc gourmandes en temps de calcul et leurs performances dépendent fortement de la robustesse de l'algorithme de reconnaissance utilisé.

Dans **[El Nager et Al Hadj, 05]**, les auteurs ont proposé un algorithme pour la segmentation des chaînes de deux chiffres. L'algorithme assume que les chiffres sont connectés au plus en un seul point et utilise une approche multi agents. Deux agents coopèrent pour trouver l'endroit où doit se faire la séparation entre les deux chiffres, l'un se charge de la partie haute de l'image alors que l'autre se charge de la partie basse. Chaque agent utilise une combinaison de primitives pour trouver un point candidat. Les agents doivent ensuite négocier pour décider du point de coupure à utiliser. Les auteurs ont rapporté que la coopération des agents donnait des résultats bien meilleurs que si les agents étaient utilisés séparément. En intégrant cette méthode de segmentation dans un système de reconnaissance des formes, les performances ont atteint les 96% de bonnes classifications.

1.2.2 Calcul des représentations :

Le but principal de la phase de calcul des représentations est de décrire chaque forme avec le plus petit ensemble d'attributs pertinents **[Theodoridis et Koutroumbas, 03]**. Il y a deux raisons principales pour garder le nombre d'attributs aussi petit que possible : les coûts de mesure et la précision de la classification. Un petit ensemble d'attributs bien choisis simplifie aussi bien la représentation des formes que les classificateurs qui sont construits autour. Conséquemment, le classificateur résultant sera plus rapide et utilisera moins de ressources mémoires. Plus encore, il a souvent été observé en pratique que l'ajout d'attributs peut dégrader les performances du classificateur, si le nombre d'exemples d'entraînement utilisés pour la conception du classificateur est petit par rapport au nombre d'attributs **[Raudys et Jain, 91]**.

Bien que la relation entre le taux d'erreurs, le nombre d'exemples d'entraînement et le nombre d'attributs soit très difficile à établir, il est généralement recommandé d'utiliser un ensemble d'entraînement d'une taille d'au moins 10 fois le nombre d'attributs. Plus le classificateur devient complexe et plus ce rapport devient large **[Jain et al, 00]**.

D'un autre coté, la réduction du nombre d'attributs peut conduire à une perte dans la discrimination du système résultant et ainsi une réduction de la précision du classificateur. Watanabe indique que les attributs doivent être choisis prudemment, puisqu'il est possible de rendre deux formes arbitraires similaires en les encodant avec suffisamment d'attributs redondants **[Watanabe, 85]**.

Il est important de faire la différence entre *sélection d'attributs* et *extraction d'attributs*. Le terme sélection d'attributs fait référence aux algorithmes qui sélectionnent le meilleur sous ensemble d'attributs, à partir de l'ensemble d'attributs de départ. Les méthodes qui créent de nouveaux attributs à partir de transformations ou combinaisons des attributs d'origine sont appelées algorithmes d'extraction d'attributs. En général, l'extraction d'attributs précède la sélection d'attributs : d'abord les attributs sont extraient des données initiales, puis une partie de ces attributs est sélectionnée. Le choix entre extraction d'attributs et sélection d'attributs dépend du domaine de l'application ainsi que de l'ensemble d'entraînement disponible.

La sélection d'attributs permet de faire des économies dans les coûts des mesures (puisque certains attributs sont éliminés) et les attributs restants gardent leur interprétation physique originale. D'un autre coté, les attributs transformés générés par l'extraction d'attributs peuvent fournir de meilleures qualités discriminantes, mais ces nouveaux attributs peuvent ne pas avoir de signification physique claire.

1.2.2.1 Extraction d'attributs :

L'objectif principal de l'extraction d'attributs est de simplifier le travail du classificateur et d'améliorer les performances du système. Pour certains problèmes, les attributs ne permettent pas de faire facilement la distinction entre les formes des différentes classes. Ceci conduit à l'utilisation de classificateurs complexes qui sont coûteux en temps de calculs et en données d'entraînement, sans pour autant garantir les meilleures performances.

Considérons par exemple les données de la **figure 1.2.** En sachant que l'objectif du classificateur est de définir la frontière entre les deux classes **[Theodoridis et Koutroumbas, 03]**, il devient évident qu'avec cette représentation c'est très difficile.

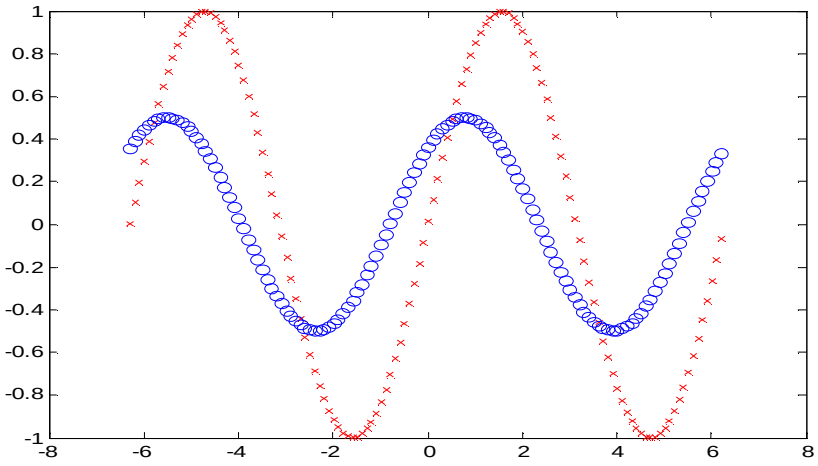

Figure 1-2 Données avant l'extraction d'attributs

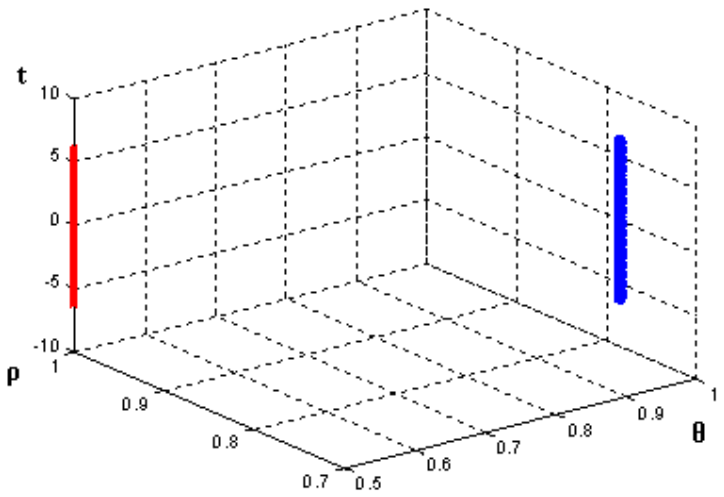

Figure 1-3 Données après l'extraction d'attributs

Supposons maintenant que nous avons des connaissances à priori sur la distribution des données dans chaque classe. On pourra alors représenter chaque exemple par l'équation $r\sin(t+q)$ où **ρ, t** et **θ** sont les nouveaux attributs de l'exemple. Les données de notre problème deviennent alors de la forme montrée à la **figure 1.3**.

On remarque alors qu'il est devenu plus facile de tracer la frontière entre les deux classes, ce qui devrait faciliter le travail du classificateur.

Devijver & Killer définissent l'extraction de primitives **[Devijver et Kittler, 82]** comme « le problème d'extraire à partir d'un ensemble de données l'information qui est la plus utile à la classification, dans le sens où elle minimise la variabilité des formes dans une même classe (within class pattern variability) tout en maximisant la variabilité des formes entre classes différentes (between class pattern variability) ».

Les primitives extraites doivent être invariantes par rapport aux distorsions attendues ainsi qu'aux variations possibles des formes à reconnaître **[Trier et al, 96]**. En plus, comme on l'a vu précédemment, le nombre d'attributs par forme est limité par la taille de l'ensemble d'entraînement utilisé. Dans la pratique, le choix d'une méthode d'extraction d'attributs est un problème en soi. Non seulement la méthode choisie influence les performances du système, mais en plus elle limite ou dicte la nature du classificateur utilisé. Cependant, on peut toujours combiner plusieurs classificateurs (comme nous le verrons plus loin), chacun utilisant un sous ensemble des attributs disponibles, nous pourrons alors utiliser des attributs multi formats.

1.2.2.1.1 Moments de Zernike

Les moments de Zernike **[Hse et Newton, 04]** sont un exemple d'attributs calculés à partir des pixels de l'image, et qui sont invariants par rapport aux rotations ainsi qu'aux réflexions. Bien que les moments d'ordres élevés capturent plus de détails de l'image, ils sont cependant plus sensibles au bruit. Lorsqu'on calcule les moments d'ordre **N**, on utilise tous les moments d'ordre **n≤N**.

Pour les images binaires, les moments de Zernike sont calculés comme suit :

$$Z_{nm} = \frac{n+1}{p}\sum_{x}\sum_{y} f(x,y)R_{nm}(r)\exp(-imq) \tag{1.2}$$

Où

$$R_{nm}(r) = \sum_{s=0}^{\frac{n-|m|}{2}} (-1)^s \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!} r^{n-2s} \tag{1.3}$$

Et

n entier positif qui vérifie $n-|m|$ pair, $|m| \leq n$

ρ longueur du vecteur partant de l'origine de l'image jusqu'au pixel (x, y)

θ angle entre **ρ** et l'axe des x dans le sens inverse des aiguilles d'une montre

Dans **[Hse et Newton, 04],** les auteurs ont utilisé les moments de Zernike dans une application de reconnaissance de sketchs manuscrits. Pour rendre les moments calculés invariants aux translations et aux mises à l'échelle, les images sont d'abord centrées sur leurs centroïdes puis redimensionnées à une taille de 100x100 pixels. Les auteurs ont comparé les performances obtenues en utilisant trois classificateurs différents, et différentes valeurs d'ordre pour les moments calculés. Ils ont montré que pour l'ensemble d'entraînement utilisé, un ordre de 8 était suffisant pour obtenir de très bonnes performances lors de la classification à savoir 96.9%.

1.2.2.2 Sélection d'attributs :

Le problème de la sélection d'attributs est défini comme suit : étant donné un ensemble **d** d'attributs, choisir un sous ensemble de taille **m** qui conduit à la plus petite erreur de classification **[Jain et al, 00]**. Soit **Y** un ensemble d'attributs donné, avec une cardinalité **d**. Et soit **X** un sous ensemble sélectionné $X \subseteq Y$. Soit **J(X)** la fonction critère de la sélection d'attributs pour l'ensemble **X**. on assume que des valeurs élevées pour **J** indiquent un meilleur sous ensemble d'attributs. En plus de la stratégie de recherche l'utilisateur doit spécifier une fonction critère **J(.)**.

Une approche directe pour résoudre le problème de la sélection d'attributs consiste en 1) examiner tous les $\binom{d}{m}$ sous ensembles de taille **m**, et 2) choisir le sous ensemble qui donne la plus grande valeur pour **J (.)**. Cependant, le nombre de sous ensembles à examiner augmente de façon exponentielle, rendant la recherche exhaustive non faisable même pour des valeurs modérées pour **m** et **d**. Par exemple, pour garantir l'optimalité d'un sous ensemble de 12 attributs d'un total de 24 attributs, approximativement 2.7 millions de sous ensembles doivent être examinés.

Comme la sélection d'attributs se fait généralement de manière off line. Le temps d'exécution de l'algorithme peut ne pas être aussi critique que l'optimalité de l'ensemble obtenu. Mais, bien que cela soit vrai pour les ensembles d'attributs de taille modérée, plusieurs applications récentes, particulièrement dans le datamining et la classification de documents, impliquent des milliers d'attributs. Dans un cas pareil, le temps d'exécution de l'algorithme peut rapidement devenir très contraignant.

Plusieurs méthodes de sélection d'attributs ont été proposées dans la littérature, ces méthodes peuvent être classées en deux groupes distincts selon la nature du critère de sélection utilisé **[Hall et Holmes, 03]**:

- Les fonctions de sélection dites « enveloppantes » (ou « wrapper » en anglais) : ces méthodes évaluent la qualité de la sélection en utilisant l'erreur de classification fournie par le classificateur utilisé, ce qui peut nécessiter le recours à l'apprentissage ;

- Les fonctions de sélection dites « filtrantes » : elles utilisent quand à elles des caractéristiques générales des données pour évaluer la qualité de la sélection d'attributs. Ce qui leur permet d'opérer de façon indépendante du classificateur utilisé, donc plus rapidement que les approches enveloppantes.

Dans **[Morita et al, 02]**, les auteurs ont utilisé la sélection d'attributs pour améliorer les performances d'un système de reconnaissance de texte manuscrit destiné au traitement automatisé des chèques bancaires. En utilisant une approche filtrante, le critère de sélection a été choisit en partant de la constatation suivante : un bon ensemble d'attributs doit permettre d'avoir des groupements de formes distincts les uns des autres, en d'autres termes il doit faciliter l'application d'un algorithme de clustering (voir la classification non supervisée). Le système développé utilise une fonction très connue dans le domaine du clustering à savoir l'indice de Davies-Bouldin (ou DB Index) **[Davies et Bouldin, 79]**.

Pour la classe **Ci** la distance intra classe correspondante est donnée par la formule

$$S_i = \frac{1}{|C_i|} \sum_{x \in C_i} \left\{ \| x - Z_i \| \right\} \tag{1.4}$$

Où $|C_i|$ est le nombre d'éléments dans la classe **Ci**, et **Zi** est le centroïde de la classe **Ci**.

La distance inter classes **Ci** et **Cj** est $d_{ij} = \| Z_i - Z_j \|$

On calcule alors $R_i = \max_{j, j \neq i} \left\{ \frac{S_i + S_j}{d_{ij}} \right\}$

Le DB Index est donné par la formule suivante

$$I_{DB} = \frac{1}{D} \frac{1}{K} \sum_{i=1}^{K} R_i \tag{1.5}$$

Où **D** est le nombre d'attributs sélectionnés, **K** est le nombre de groupements trouvés.

Les auteurs ont alors appliqué une méthode d'optimisation multi objective qui permet de minimiser simultanément le DB Index et le nombre d'attributs sélectionnés, obtenant ainsi l'ensemble d'attributs le plus petit qui améliore la qualité des regroupements disponibles.

1.3 Principales approches pour la classification:

On peut dire que la classification est le cœur de la reconnaissance des formes parce que c'est l'étape où se décide la classe de la forme à reconnaître. Un classificateur peut être vu comme une fonction $f\left(x|q_1, q_2, \mathbf{K}, q_n\right) = c_x$, où **x** est la forme à reconnaître, $\mathbf{c_x}$ est la classe trouvée par le classificateur et $\boldsymbol{\theta_i}$ sont les paramètres du classificateur, leur signification dépend du type de classificateur.

D'un point de vue géométrique, on peut dire qu'un classificateur est une fonction qui divise l'espace des formes en plusieurs régions, chaque région contient les formes d'une certaine classe. La classe d'une forme inconnue dépend alors de la région dans laquelle elle se trouve.

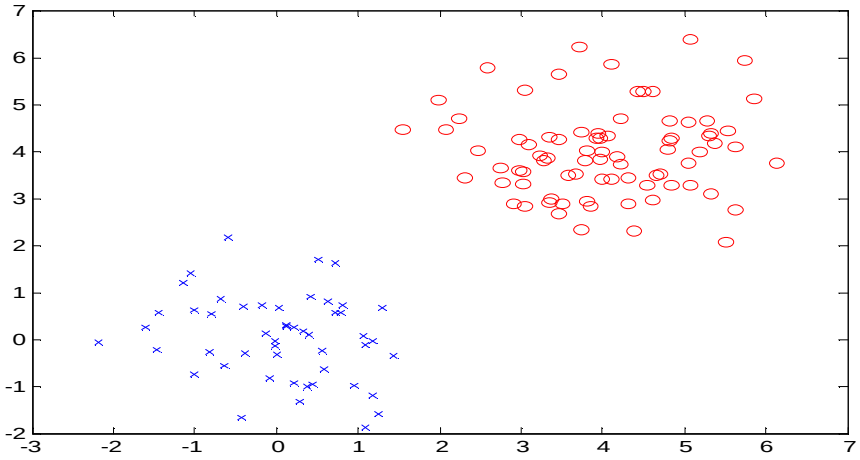

Figure 1-4 Données linéairement séparables

Considérons l'exemple de la **figure 1.4**, nous voulons construire un classificateur qui classifie correctement les croix et les cercles affichés. Le classificateur doit diviser l'espace des formes en deux régions Rx et Ro tel que Rx ne contient que des croix et Ro uniquement des cercles. Nous pouvons utiliser une simple ligne (**figure 1.5**), dont l'équation est:

$$ax_1 + bx_2 = c \qquad (1.6)$$

Où $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ sont les coordonnées du point **x** et **a, b, c** sont les paramètres de la droite. La fonction de classification peut alors s'écrire :

$$f(x|a,b,c) = \begin{cases} Rx & si \quad ax_1 + bx_2 > c \\ Ry & si \quad ax_1 + bx_2 < c \end{cases} \tag{1.7}$$

La ligne $ax_1 + bx_2 = c$ représente la frontière de décision **[Theodoridis et Koutroumbas, 03]**. Dans la pratique, plus un point est proche de cette frontière et moins il a de chances d'être classé correctement.

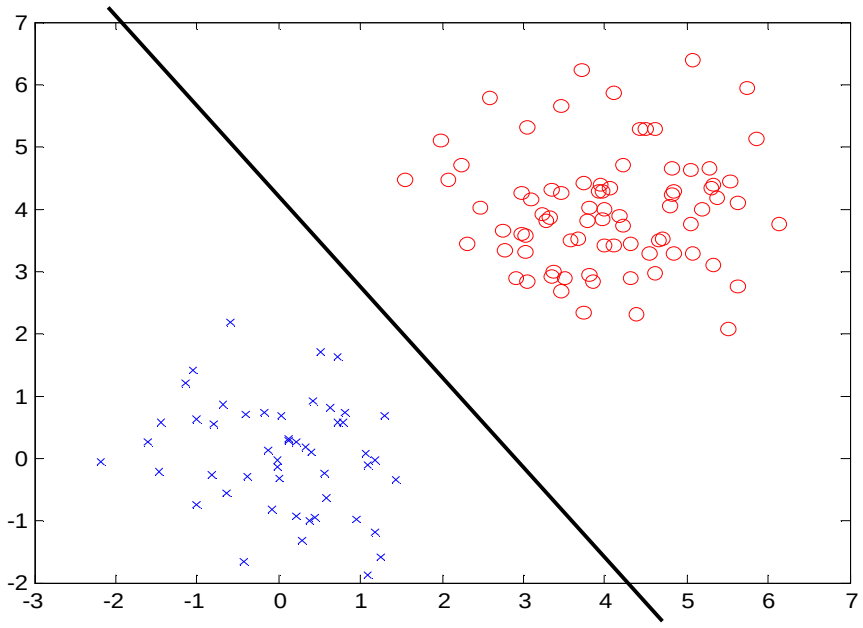

Figure 1-5 Frontière linéaire

Les problèmes pour lesquelles on peut construire des classificateurs en utilisant les lignes (en 2 dimensions), les plans (en 3 dimensions) et de façon générale les hyperplans sont dits *linéairement séparables* **[Theodoridis et Koutroumbas, 03]**. Lorsque les données se présentent sous une forme pour laquelle on ne peut tracer de ligne pour séparer les classes (**figure 1.5** par exemple) on dit qu'ils sont *non linéairement séparables*. Pour ce genre de problèmes, des classificateurs plus complexes doivent être utilisés.

Pour certains problèmes, la classification des formes est un problème très difficile à résoudre, même pour un être humain. Considérons par exemple les formes de la **figure 1.6**, à quelle classe appartient le point pointé par la flèche ?

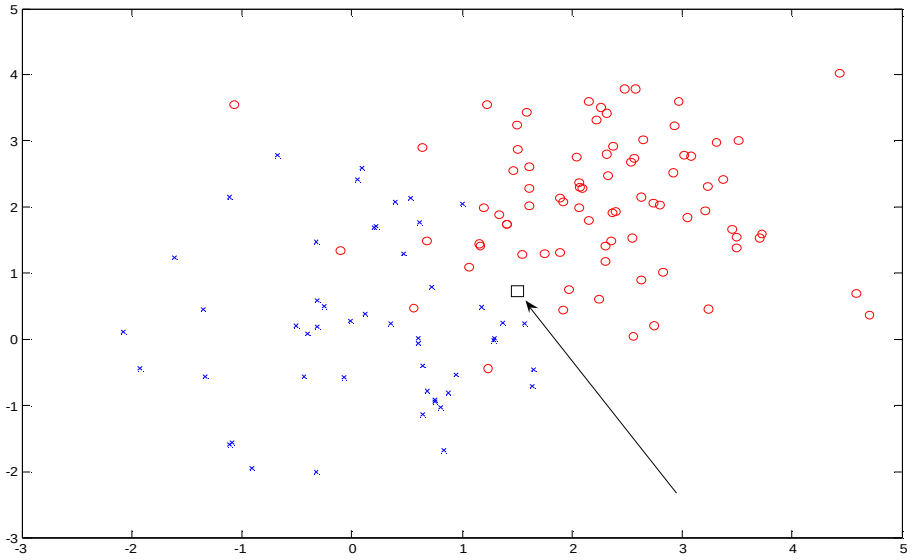

Figure 1-6 Données non linéairement séparables

Pour ce genre de problèmes, il serait intéressant que le classificateur renvoi en plus de la classe trouvée, la confidence qu'il a dans cette classification. Selon la valeur de cette confidence le système pourra, si la valeur de confidence est en dessous d'un certain de seuil, soit rejeter la forme sans la classer, soit appliquer un classificateur d'un autre type.

1.3.1 Approche basée sur les prototypes :

C'est l'approche la plus simple et la plus intuitive : les formes similaires sont assignées à la même classe. Donc, dès qu'une bonne mesure de similarité a été choisie, les formes peuvent être classées en utilisant quelques prototypes par classe. Le choix de la mesure et des prototypes est crucial pour le succès de la classification. Par exemple, pour le « nearest mean classifier » **[Skurichina et al, 02]**, le choix des prototypes est simple et robuste : chaque classe de formes est représentée par la moyenne de toutes les formes d'entraînement de la classe.

Un autre exemple de classificateur est le Nearest Neighbour Classifier ou classificateur du plus proche voisin **[Liu et Nagakawa, 01]**, son principe est le suivant :

Soit **D** un ensemble de formes d'entraînement. Chaque forme m est représentée par un vecteur d'attributs x^m et une classe c^m. Pour classer un nouveau vecteur x :

- Calculer la dissimilarité (par ex. distance euclidienne) entre le point x et chaque vecteur d'entraînement

$$d^m = d\left(x, x^m\right) \tag{1.8}$$

- Trouver le point d'entraînement x^{m^*} le plus proche de x ;
- Assigner la classe $c(x) = c^{m^*}$

Bien que cet algorithme soit très simple, cela ne l'empêche pas d'être très performant et d'être utilisé dans plusieurs applications. Cependant, plusieurs limitations sont à noter **[Barber, 03]**:

- Comment mesurer la distance entre les points ? Typiquement l'on utilise la distance euclidienne, mais ce n'est pas toujours approprié. Considérons la situation de la **figure 1.7** où la distance euclidienne peut conduire à une classification indésirable, en classifiant le point d'interrogation à la classe 2 ;

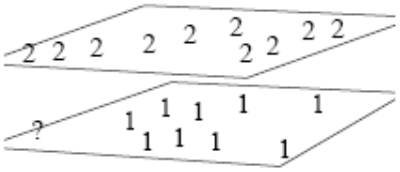

Figure 1-7 Problème de classification [Barber, 03]

- Dans la version simplifiée de l'algorithme nous devons stocker la totalité des exemples d'entraînement pour effectuer les classifications. Cependant, il est clair que d'une façon générale, seul un sous ensemble des exemples d'entraînement est nécessaire à la classification. On peut y remédier en supprimant de l'ensemble d'entraînement les vecteurs qui n'affectent pas (ou peu) la classification.
- Sensibilité aux outliers : Un outlier (ou singularité) est un exemple d'entraînement avec une classe différente de celles des exemples environnants. Il peut être le résultat d'une erreur lors de l'acquisition des données. Ce point ne devrait pas participer à la classification.

Une alternative au Nearest Neighbour classifier est le kNearest Neighbour Classifier **[Webb, 02]**, qui comme sont nom l'indique utilise non pas le plus proche voisin pour la classification mais les k plus proches voisins. La forme inconnue reçoit la classe la plus représentée par les k plus proches voisins de la forme. Cette méthode permet de réduire la sensibilité du système face aux outliers. Les performances de ce classificateur dépendent fortement du paramètre k, une méthode pour choisir k serait d'essayer plusieurs valeurs et de prendre celle qui donne les meilleures performances sur un ensemble test différent de l'ensemble d'entraînement.

1.3.2 Approche statistique :

Les systèmes de Reconnaissance des formes basés sur l'approche statistique ont été utilisés avec succès dans plusieurs applications commerciales. Des concepts bien connus de la théorie des statistiques sont utilisés pour établir les frontières de décision entre les classes des formes.

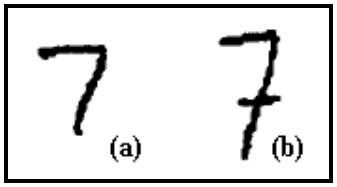

Figure 1-8 Exemple de chiffre manuscrit

Considérons un système de Reconnaissance des formes qui se limite à la reconnaissance des chiffres '7' écrits par une certaine personne ***A***. Bien que probablement, chaque chiffre écrit diffère un peu des précédents, on peut cependant dire que, par exemple, les chiffres 7 de la **figure 1.8 (a)** ont une plus grande probabilité d'être écrits par ***A*** que ceux de la **figure 1.8 (b)**. En se basant sur ces informations 'à priori' concernant la personne ***A***, il serait intéressant de construire une fonction de probabilités $P_A(x)$ qui donne pour chaque forme **x**, la probabilité qu'elle ait été écrite par la personne ***A***.

La nature de la fonction de probabilités peut être établie soit en utilisant des connaissances à priori sur le problème à résoudre, ou bien en utilisant l'apprentissage. Par exemple pour une distribution gaussienne, la fonction de probabilités s'écrit :

$$P_A(X) = \frac{1}{\sqrt{2p s^2}} e^{-\frac{1}{2s^2}(x-m)^2} \qquad (1.13)$$

Où m est la moyenne, et s^2 la variance.

Dans le cas général où nous avons plusieurs classes. Chaque classe **k** a sa propre fonction de probabilités $P(x|classe = k)$ qui donne la probabilité d'avoir la forme **x** en sachant que sa classe est **k**. Comme nous nous intéressons à la classification, nous voulons plutôt répondre à la question suivante : Sachant que nous avons une forme **x*** quelle est la probabilité qu'elle soit de classe **k** ?

En d'autres termes qu'elle est la valeur de $P(classe = k|x^*)$?

Pour évaluer cette probabilité, plusieurs règles existent, la plus utilisée est sans doute la règle de Bayes **[Webb, 02]**:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{1.14}$$

Dans notre cas cela donne :

$$P(classe = k | x^*) = \frac{P(x^* | classe = k) \cdot P(classe = k)}{P(x^*)} \tag{1.15}$$

Comme le dénominateur de l'équation précédente est indépendant de la classe, nous classifions **x*** par la classe **k** qui vérifie :

$$P(x^* | classe = k) P(classe = k) > P(x^* | classe = l) P(classe = l) \text{ Pour tout } l \neq k \tag{1.16}$$

Dans la pratique l'on utilise le logarithme des fonctions de probabilités, parce que c'est plus stable numériquement :

$$\log(P(x^* | classe = k)) + \log(P(classe = k)) > \log(P(x^* | classe = l)) + \log(P(classe = l)) \quad \text{Pour tout } l \neq k \tag{1.17}$$

La valeur $P(classe = k)$ est dite probabilité à priori, et peut être estimée en utilisant la fréquence relative de l'occurrence de chaque classe dans l'ensemble d'entraînement. Par exemple, si nous avons un ensemble d'entraînement composé de 50 formes de classe 1 et 100 formes de classe 2 alors :

$$P(classe = 1) = \frac{50}{150} = \frac{1}{3}$$

$$P(classe = 2) = \frac{100}{150} = \frac{2}{3}$$

Ces classificateurs sont dits classificateurs naïfs de Bayes. Bien qu'ils soient simples, les classificateurs naïfs de Bayes ont connu un grand succès, principalement à cause des raisons suivantes :

- Ils sont très simples à implémenter ;
- L'entraînement et la classification se font rapidement ;
- Il est facile de traiter les attributs manquants.

La limitation principale de ces classificateurs est le fait que pour construire un classificateur efficace, le concepteur doit généralement avoir des informations à priori sur la distribution des données dans l'espace. Si la forme des fonctions de densités est connue (par ex. Gaussienne), mais que certain paramètres de ces fonctions (par ex. la moyenne) sont manquants, alors nous avons un problème de décision paramétrique. Une stratégie courante pour ce type de problèmes est de remplacer les paramètres manquants par leurs valeurs estimées lors de l'entraînement. Si la forme des densités conditionnelles est inconnue, alors nous avons un problème non paramétrique. Dans ce cas, nous devons soit estimer les fonctions de densités, soit construire directement la frontière de décision en se basant sur les données d'entraînement.

L'apprentissage se fait par **estimation de densité [Webb, 02]**. Le but est de trouver la fonction de probabilité qui a le plus de chances de générer les exemples d'entraînements. En d'autres termes, si ***X*** est un ensemble d'exemples d'entraînement, le but de l'apprentissage est de trouver la fonction de densité de probabilités $p(x)$ tel que $p(X)$est maximale. La fonction de densité de probabilité étant généralement définie par un ensemble de paramètres θ (moyenne et variance par exemple) l'apprentissage revient à trouver les paramètres θ qui maximisent $p(X|q)$.

L'approche statistique a été appliquée, entre autres, au problème de reconnaissance faciale **[Moghaddam et al, 2000]**. Les auteurs ont défini une mesure de similarité statistique basée sur la règle de Bayes, et ont utilisé cette mesure pour calculer la similarité entre la face à reconnaître et les classes existantes. La mesure de similarité a été définie ainsi : La distance entre deux images I_1 et I_2 est égale à

$$S(I_1, I_2) = P(\Delta \in \Omega_I) = P(\Omega_I|\Delta) \tag{1.18}$$

Avec $\Delta = I_1 - I_2$ est la différence d'intensité entre deux images et $P(\Omega_I|\Delta)$est la probabilité à posteriori.

Les auteurs ont défini deux classes de variations des intensités : Ω_I ou variations *intra personnelles* (correspondant, par exemple, aux différentes expressions faciales d'un même individu) et Ω_E ou variations *extra personnelles* (correspondant aux variations entre des individus différents). En assumant que les deux classes ont des distributions Gaussiennes, la similarité S peut être écrite comme suit :

$$S(I_1, I_2) = \frac{P(\Delta|\Omega_I)P(\Omega_I)}{P(\Delta|\Omega_I)P(\Omega_I) + P(\Delta|\Omega_E)P(\Omega_E)} \tag{1.19}$$

Où la probabilité $P(\Omega)$peut être choisie de façon à refléter des conditions spécifiques lors de l'exécution ou à partir d'une autre source de connaissances à priori sur les images à classer.

1.3.3 Approches bio inspirées :

1.3.3.1 Réseaux de neurones :

Le but de cette approche est de développer des machines capables de mémoriser des expériences puis de les analyser pour prendre des décisions à propos de situations nouvelles. Les réseaux de neurones se focalisent principalement sur l'apprentissage adaptatif et la mémorisation.

Le cerveau humain a plusieurs qualités très intéressantes d'un point de vue informatique **[White, 04]** :

- **Une grande capacité de stockage :** 10^{11} neurones avec une capacité estimée de 10^{14} jonctions synaptiques, soit approximativement 10^4 fois la capacité d'un ordinateur moyen ;
- **Traitement en parallèle :** le cerveau gère simultanément plusieurs activités ;
- **Structure flexible :** les poids des connections changent au fil du temps. La formations de nouvelles connections peut être l'une des méthodes d'apprentissage du cerveau ;
- **Tolérance aux fautes :** bien que les connections entre les cellules nerveuses peuvent s'altérer à mesure que les cellules meurent et sont remplacées, cela n'empêche pas le cerveau de fonctionner correctement.

Modéliser le cerveau humain par un réseau de neurones n'est pas une tâche facile, principalement à cause des raisons suivantes:

- Il est difficile de déterminer la structure du réseau de neurones, c'est-à-dire comment sont connectés les neurones entre eux. Dans le cerveau, les connections changent constamment, et les connections initiales semblent être gouvernées par des facteurs génétiques;
- Les poids des connections et les seuils des nœuds changent constamment. Ce problème a été le sujet de recherches poussées et a été en grande partie résolu comme suit : Etant donné une certaine entrée, si le réseau commet une erreur, on pourra déterminer quels neurones étaient actifs avant l'erreur. On modifie alors les poids et les seuils de façon appropriée pour réduire cette erreur. Ce problème est généralement une généralisation du problème précédant (déterminer la structure). Si ce problème est résolu l'autre l'est aussi, mais il se peut qu'il y ait de meilleures méthodes pour trouver la structure.

Pour réduire ces difficultés, le problème a été divisé en sous problèmes. Différents types de réseaux de neurones ont été proposés, chaque type a ses propres contraintes sur les poids et les connections. Par exemple, si un neurone **A** a une connexion vers un neurone **B**, le

neurone **B** ne peut pas avoir de connexion vers **A**. les types de connections possibles sont l'architecture du réseau. A chaque fois que le système commet une erreur, certains poids des connections, et seuils doivent être modifiés pour compenser cette erreur. Les règles qui gouvernent ces modifications sont l'algorithme d'apprentissage. Différents types de réseaux de neurones peuvent avoir différents algorithmes d'apprentissage.

Des les systèmes naturels, l'information est généralement traitée par une série de couches. Par exemple, pour la vision humaine, la lumière est d'abord captée par la rétine qui la transforme (traite) en signaux qui sont ensuite envoyés au cerveau pour d'autres traitements. Les réseaux neuronaux artificiels tentent de s'inspirer de ce comportement.

Un neurone ou unité de traitement renvoi une valeur déterministe en fonction de la valeur des ses entrées. Dans ce sens, les réseaux de neurones artificiels sont des représentations graphiques de fonctions **[Barber, 03]**. La forme la plus simple des réseaux de neurones est le **perceptron [Theodoridis et Koutroumbas, 03]**. Un exemple de perceptron est donné à la **figure 1.9**.

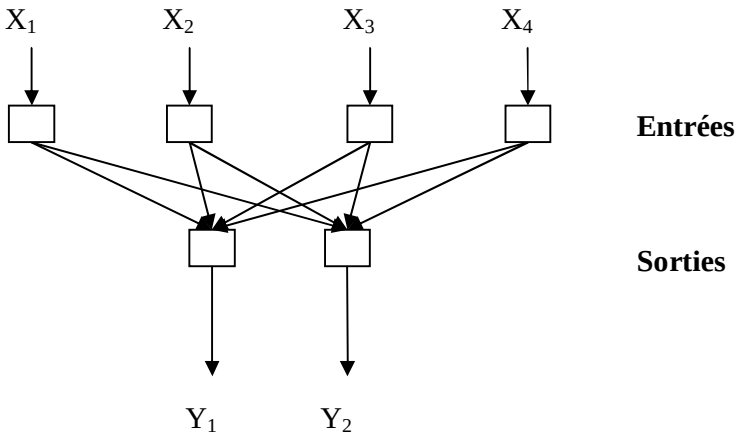

Figure 1-9 Exemple de perceptron

Les sorties $\mathbf{y_i}$ du réseau sont fonction de ses entrées $\mathbf{x_i}$, où $\mathbf{w_{ij}}$ sont les poids des connections et $\boldsymbol{\mu_i}$ est le biais, **g( )** est la fonction de transfert :

$$y_i = g\left(\sum_j w_{ij} x_j + m_i\right) \qquad (1.20)$$

Pour un problème de classification les sorties sont les différentes classes possibles. L'approche la plus utilisée est de réserver chaque sortie à une seule classe. Si la forme en entrée appartient à la classe **k**, toutes les sorties sont à 0, sauf la sortie $\mathbf{y_k}$ qui est à 1.

La principale limitation des perceptrons est qu'ils ne peuvent être appliqués qu'aux problèmes linéairement séparables. Pour les problèmes plus complexes, d'autres types de réseaux de neurones doivent être utilisés. Les plus courant sont les réseaux de neurones multi

couches **[Webb, 02]**. Pour ce type de réseaux, une ou plusieurs couches intermédiaires de neurones sont intercalées entre les neurones d'entrée et ceux de sortie, voir par exemple la **figure 1.10**.

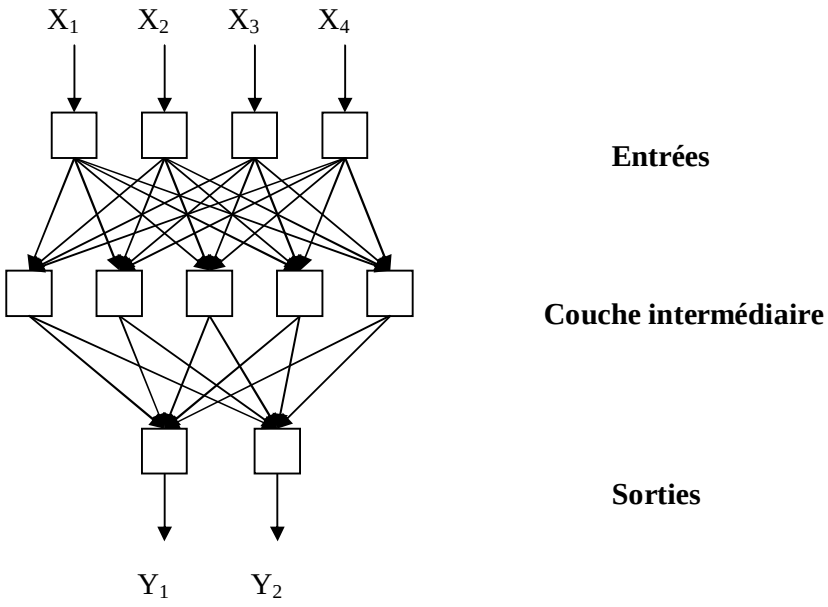

Figure 1-10 réseau multi couches

Ces nouvelles couches vont permettre au réseau de modéliser des fonctions plus complexes, et ainsi traiter les données non linéairement séparables. Généralement, plus il y a de couches et plus la fonction modélisée par le réseau est complexe. Un réseau de neurone étant complètement défini par ses paramètres ($\mathbf{w_{ij}}$, $\boldsymbol{\mu_i}$ et **g( )**) l'apprentissage va consister à trouver les valeurs de ces paramètres qui minimisent l'erreur de celui-ci (la déviation entre les sorties du réseau et celles attendues).

1.3.3.2 Algorithmes génétiques :

Les algorithmes génétiques sont le résultat de la modélisation sur machine des principes de la théorie de l'évolution des espèces de Darwin, ou la survie du meilleur. Bien que cette théorie soit fausse, parce qu'elle exclut la création par Dieu des espèces et prétend que les espèces actuelles sont le résultats de millions d'années d'évolution, cela n'empêche pas les algorithmes génétiques de connaître un véritable succès.

Le principe général des algorithmes génétiques est de faire évoluer une population de 'chromosomes' vers une nouvelle population en utilisant une sorte de 'sélection naturelle' combinée à des opérateurs bio – inspirés qui sont le croisement, la mutation et l'inversion **[Back et al, 97]**.

L'approche la plus couramment utilisée est de représenter chaque individu de la population par une chaîne générée à partir d'un alphabet fini, généralement (0, 1). Chaque individu représente une solution possible du problème à résoudre. Le succès de chaque individu est déterminé par l'évaluation de sa finesse. La fonction de finesse est spécifique au problème à résoudre, mais dans tous les cas elle devrait prendre un individu et retourner un nombre réel. Plus la finesse d'un individu est grande et plus il a de chances de se reproduire. La phase de reproduction inclut deux étapes, le croisement et la mutation.

Croisement : pour le croisement, deux chaînes parentes sont divisées par une ligne choisie aléatoirement sur leur longueur **[Back et al, 97]**. Les segments sont ensuite permutés pour obtenir deux nouvelles chaînes, comme le montre la **figure 1.11**.

Mutation : un point est choisit aléatoirement sur la longueur de la chaîne, et le bit correspondant est inversé **[Back et al, 97]**.

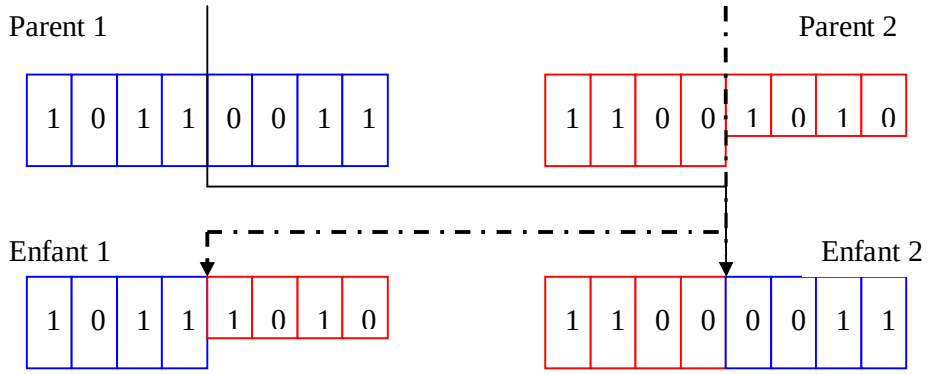

Figure 1-11 Croisement

Un algorithme génétique typique est comme suit **[Back et al, 97]**:

Finesse : fonction qui assigne un score à un individu.

Seuil : un seuil qui spécifie le critère d'arrêt.

p : nombre d'individus de la population.

r : fraction de la population qui sera remplacée par croisement à chaque itération.

m : taux de mutations.

- **Initialiser la population** : *P* ß générer *p* individus de façon aléatoire ;
- **Evaluation** : pour chaque h_i de *P* calculer *Finesse* (h_i) ;
- Tant que [*Finesse Max* (h_i) < *Seuil*] faire

  Créer une nouvelle population *Ps* :

- *Sélection* : choisir *(1-r)p* membres de *P* pour les ajouter à *Ps*. La probabilité $Pr(h_i)$ de choisir l'individu h_i de *P* est donnée par :

$$\Pr(h_i) = \frac{Finesse(h_i)}{\sum_{j=1}^{p} Finesse(h_j)} \qquad (1.21)$$

- *Croisement* : Choisir $\frac{r \cdot p}{2}$ paires d'individus de *P*, en utilisant la probabilité $Pr(h_i)$ donnée ci-dessus. Pour chaque paire (h_1, h_2) produire deux chaînes en appliquant l'opérateur de croisement. Ajouter les chaînes produites à *Ps* ;
- *Mutation* : choisir *m%* des membres de *Ps* avec une probabilité de choix uniforme. Pour chacun de ces individus, muter un bit choisit aléatoirement ;
- *Mise à jour* : *P* ß *Ps* ;
- *Evaluation* : pour chaque *h* de *P*, calculer *Finesse (h).*

- Renvoyer l'individu de *P* qui a la plus grande finesse.

Les algorithmes génétiques ont été utilisés dans un système d'apprentissage par requêtes **[Sakano, 05].** Ce type d'apprentissage est caractérisé par la génération de requêtes en direction de l'utilisateur humain lorsque le système n'arrive pas à classer correctement une forme. Le problème est que les méthodes d'extractions d'attributs utilisées génèrent des représentations optimisées pour la machine, mais non compréhensibles par l'homme. L'auteur a utilisé un algorithme génétique pour retrouver l'image à l'origine de la représentation.

L'algorithme général du système est comme suit :

- **Initialisation** : à partir des exemples d'entraînement, générer un prototype par classe en utilisant LVQ ;
- **Génération des requêtes** : le système génère un ensemble de vecteurs d'attributs proches des frontières entre les classes, ce sont ces exemples qui ont de grandes chances d'être mal classés ;
- **Translation des requêtes** : le système utilise pour chaque requête l'algorithme génétique afin de trouver l'image qui a le plus de chances de générer le vecteur d'attributs de la requête. Pour réduire l'espace de recherche, la population initiale est choisie à partir des exemples d'entraînement ;
- **Réponse aux requêtes** : un humain joue le rôle du superviseur. Le système lui présente l'image de la requête et c'est au superviseur de décider de la classe de celle-ci. Lorsque le superviseur n'arrive pas à reconnaître la forme, il la rejette et elle est éliminée de l'ensemble d'entraînement ;

- **Réentraînement** : les formes qui ont été classées par le superviseur sont ajoutées aux exemples d'entraînement, et LVQ est relancé pour calculer les nouveaux vecteurs de référence.

Ces quatre étapes représentent une itération du système. Sakano a montré qu'avec 10 itérations le système a réduit de 10% le nombre de formes mal classées en rajoutant 10% d'exemples générés à l'ensemble d'entraînement. Cette augmentation des performances n'aurait pas été possible en utilisant des formes d'entraînement choisies aléatoirement.

1.3.4 Méthodes structurelles :

Parmi toutes les méthodes structurelles disponibles, les arbres de décisions sont sans aucun doute les plus populaires. Les arbres de décisions sont des systèmes de prise de décision 'multi stages' dans lesquels les classes possibles sont rejetées séquentiellement jusqu'à ce qu'il ne reste qu'une seule classe possible. Pour se faire, l'espace d'attributs est divisé en plusieurs régions de façon séquentielle **[Theodoridis et Koutroumbas, 03] (**voir **figure 1.12**).

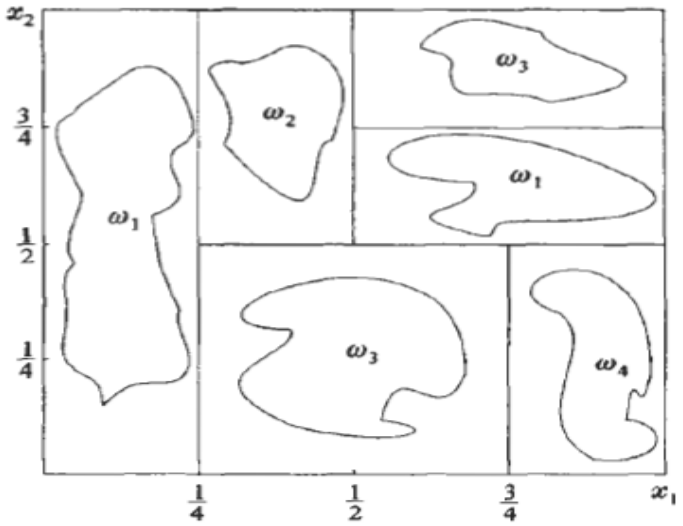

Figure 1-12 Division de l'espace des formes [Theodoridis et Koutroumbas, 03]

Lors de l'apprentissage, un arbre de décision est construit tel que chaque nœud contient une question de la forme « est-ce que l'attribut $\mathbf{x_i \leq \alpha}$? » où $\boldsymbol{\alpha}$ est un seuil déduit par l'algorithme d'apprentissage. Par exemple, l'arbre de décision qui correspond au découpage de la **figure 1.12** est donné à la **figure 1.13**.

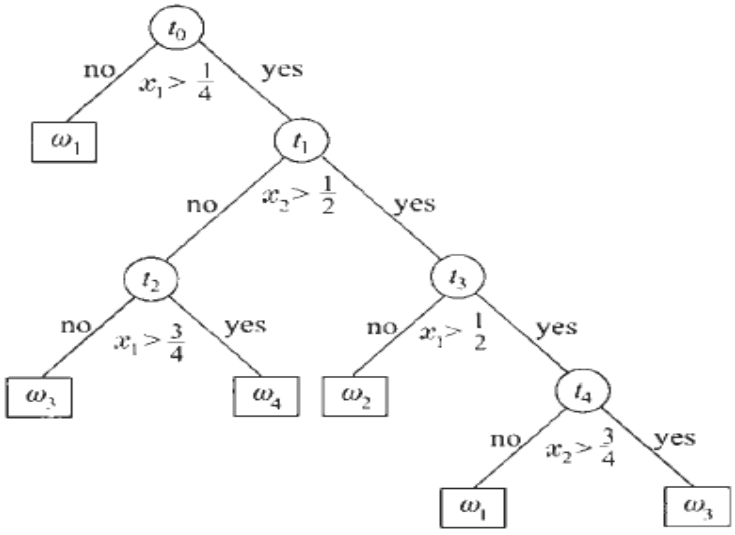

Figure 1-13 Arbre de décision [Theodoridis et Koutroumbas, 03]

Lors de la classification, la forme à reconnaître passe par les nœuds de l'arbre selon les réponses aux questions des nœuds jusqu'à atteindre une feuille, elle reçoit alors la classe de la feuille. L'avantage est que, dans la plupart des cas, la classification se fait en examinant seulement un sous ensemble des attributs de la forme, ceci rend les arbres de décision adaptés aux problèmes pour lesquels le nombre d'attributs est très grand, ou encore le coût d'acquisition des attributs est élevé.

1.4 Combinaison des classificateurs :

Pour résoudre les problèmes de reconnaissance, on doit parfois combiner plusieurs classificateurs. On le fait pour plusieurs raisons **[Jain et al, 00]** :

- Le concepteur du système peut avoir accès à plusieurs classificateurs, développés dans des contextes différents pour des représentations/descriptions totalement différentes d'un même problème.
- On peut avoir plusieurs ensembles d'entraînement. Ces ensembles ont été collectés/extraits à différents moments dans différents environnements, et peuvent être représentés par des paramètres différents.
- Des classificateurs différents, entraînés sur les mêmes données, peuvent présenter de grandes différences dans la qualité de la classification. Chaque classificateur peut avoir son propre sous espace de paramètres où il est le plus efficace.
- Certains classificateurs comme les réseaux de neurones montrent des comportements différents selon la phase d'initialisation, ceci est dû à la part aléatoire du processus d'initialisation. Au lieu de ne garder qu'un seul classificateur, on peut combiner les réseaux de neurones obtenus pour bénéficier des résultats de tous les entraînements.

Sommairement, on peut avoir des ensembles de paramètres différents, des ensembles d'entraînement différents, des méthodes de classification différentes ou des sessions d'entraînement différentes, tout ceci résulte en un ensemble de classificateurs dont les sorties peuvent être combinées, dans l'espoir d'améliorer la qualité globale de la classification.

Un schéma ou plan de combinaison classique consiste en un ensemble de classificateurs individuels et d'une fonction de combinaison qui combine les sorties de classificateurs individuels pour prendre une décision. Quand les classificateurs doivent-ils être invoqués et comment doivent-ils interagir entre- eux, c'est l'architecture du plan de combinaison qui le détermine. On a trois architectures principales **[Webb, 02]** :

- **L'architecture parallèle :** Tous les classificateurs individuels sont invoqués de façon indépendante, et leurs résultats sont combinés par une fonction de combinaison. Les sorties des classificateurs peuvent être pondérées avant la combinaison.
- **L'architecture en cascade :** Les classificateurs individuels sont invoqués de manière linéaire. Le nombre de classes possibles pour une forme donnée est réduit graduellement. Pour des raisons d'efficacité, les classificateurs imprécis mais peu gourmands en calcul sont invoqués en premier, suivis par des classificateurs plus précis mais aussi plus lents.
- **L'architecture hiérarchique :** Les classificateurs individuels sont combinés dans une structure, qui est similaire aux arbres de décision. Les nœuds de l'arbre peuvent être associés avec des classificateurs complexes qui nécessitent un grand nombre de caractéristiques. L'avantage de cette architecture est la grande efficacité et la flexibilité dans l'exploitation du pouvoir discriminant des attributs.

En combinant ces trois architectures basiques, on peut concevoir des systèmes de classification encore plus complexes.

1.5 Classification non supervisée :

Dans plusieurs applications de reconnaissance, il est extrêmement difficile ou coûteux, ou même impossible d'étiqueter de façon fiable chaque exemple d'entraînement avec sa classe correcte. La classification non supervisée fait référence aux situations où l'objectif est de construire des frontières de décision en se basant sur des données non étiquetées **[Theodoridis et Koutroumbas, 03]**. La classification non supervisée est aussi connue sous le nom de 'data clustering' qui est un nom générique à une variété de procédures conçues pour trouver les regroupements naturels, ou clusters, dans des données multi dimensionnelles, basés sur les similarités mesurées ou perçues entre les formes.

Le data clustering est un problème très difficile parce que les données peuvent révéler des clusters de différentes formes et tailles. Et pour rajouter à la difficulté du problème, le

nombre de clusters dépend généralement de la résolution (fine ou grossière) avec laquelle on désire observer les données.

L'analyse des clusters est une technique très importante et très utile. La rapidité et la fiabilité avec lesquelles un algorithme de clustering peut organiser une masse volumineuse d'informations constituent les principales raisons de son utilisation dans des applications aussi variées que le datamining **[Han et Kamber, 02]**, la recherche d'information **[Leuski, 01]**, la segmentation d'images **[Rezaee et al, 00],** le codage et la compression de signaux **[Popat et Picard, 03]** ainsi que pour l'apprentissage **[Meshoul et al, 05]**.

Le problème du clustering peut être formulé ainsi **[Jain et al, 00]** : étant donné **n** formes dans un espace à **d** dimensions, déterminer la partition des formes en **k** clusters, tel que les formes dans un cluster sont plus similaires les unes aux autres qu'aux formes des autres clusters. La valeur de **k** peut ne pas être spécifiée. Puisque la recherche exhaustive est généralement impossible, plusieurs heuristiques ont été utilisées pour réduire l'espace de recherche, mais sans garantir l'optimalité de la solution trouvée.

La grande majorité des algorithmes de clustering proposés sont basés sur les deux approches suivantes **[Jain et al, 00]** : l'approche par division, et l'approche par agglomération hiérarchique. Les algorithmes de partitionnement tentent d'obtenir la partition qui minimise la dispersion intra groupe (within cluster scatter) ou bien de maximiser la dispersion inter groupes (between cluster scatter). Les techniques hiérarchiques organisent les données dans une séquence imbriquée de groupes (clusters) qui peut être visualisée sous la forme d'un dendrogramme.

Avant d'utiliser un algorithme de clustering, toujours garder à l'esprit que : 1) un algorithme de clustering va trouver des clusters dans les données, même si ces clusters n'existent pas réellement. Donc, il faut toujours valider les clusters obtenus. 2) il n'y a pas d'algorithme de clustering optimal pour tous les domaines d'applications (sauf bien sur la recherche exhaustive), il faut alors essayer plusieurs algorithmes de clustering avant de faire son choix.

1.5.1 Approche par agglomération hiérarchique:

Soit $g(C_i, C_j)$ une fonction définie sur toutes les paires possibles de partitions de $\boldsymbol{X}$. cette fonction mesure la proximité entre $\boldsymbol{C_i}$ et $\boldsymbol{C_j}$. $\boldsymbol{t}$ dénote le niveau courant dans la hiérarchie.

Un algorithme général de partitionnement par agglomération peut être défini comme suit **[Theodoridis et Koutroumbas, 03]** :

- Initialisation :
  - Choisir $R_0 = \{C_i = \{x_i\}, i = 1, \mathbf{K}, n\}$ comme partitionnement initial ;
  - $\boldsymbol{t}$ =0
- Répéter

- $t = t + 1$
- parmi toutes les paires de partitions possibles (C_s, C_r) dans R_{t-1}, trouver la paire (C_i, C_j) tel que :

  $g(C_i, C_j) = \min_{r,s} g(C_s, C_r)$ si $g(C_i, C_j)$ est une fonction de dissimilarité

  $g(C_i, C_j) = \max_{r,s} g(C_s, C_r)$ si $g(C_i, C_j)$ est une fonction de similarité

- Définir $C_q = C_i \cup C_j$ et générer le nouveau partitionnement :

$$R_t = \{R_{t-1} - \{C_{i,} C_j\}\} \cup \{C_q\} \tag{1.22}$$

- Jusqu'à ce que toutes les entités soient dans le même cluster

1.5.2 Approche par division :

L'approche par division utilise la stratégie inverse de l'approche par agglomération. L'algorithme commence par une partition unique qui contient toutes les entités, et à chaque itération l'algorithme choisit la partition parmi toutes celles possibles qui optimise une certaine mesure de coût.

Soit $g(C_i, C_j)$une fonction de dissimilarité définie sur toutes les paires possibles de partitions de ***X***. $C_{t,j}$ dénote la *j*-eme partition de l'étape de partitionnement *t*. Si l'on divise en deux une partition $C_{t,j}$, les partitions obtenues sont notées $C_{t,i}^1, C_{t,j}^2$.

Un algorithme généralisé de partitionnement par division peut être défini ainsi **[Theodoridis et Koutroumbas, 03]** :

- Initialisation :
  - Choisir $R_0 = \{X\}$comme partitionnement initial ;
  - ***t*** =0
- Répéter
  - $t = t + 1$
  - pour i = 1 jusqu'à t faire
    - § parmi toutes les paires de partitions possibles obtenues lors de la division d'une partition $C_{t-1,j}$. trouver la paire $(C_{t-1,i}^1, C_{t-1,j}^2)$ qui donne la valeur maximale pour *g* ;

- A partir des *t* paires définies à l'étape précédente, choisir la paire $\left(C_{t-1,i}^{1}, C_{t-1,j}^{2}\right)$ qui maximise *g*.
- Le nouveau partitionnement est :

$$R_t = \left\{R_{t-1} - \left\{C_{t-1,j}\right\}\right\} \cup \left\{C_{t-1,j}^{1}, C_{t-1,j}^{2}\right\} \tag{1.23}$$

- Renommer les partitions de $\boldsymbol{R_t}$;

- Jusqu'à ce que chaque entité soit dans une partition a part.

1.6 Conclusion :

De manière générale, on peut dire que les systèmes de reconnaissance des formes sont des fonctions de classification qui associent à toute forme inconnue sa classe la plus probable. Malgré la difficulté de la tâche, les systèmes de reconnaissance des formes ont connus un grand succès dans des domaines variés.

Un système de reconnaissance des formes contient principalement quatre modules. Un module de préparation de données qui se charge de faciliter la tâche de reconnaissance en représentant les données en entrée sous une forme adaptée à la classification. Un module d'apprentissage qui va optimiser la fonction de classification pour le domaine d'application ciblé. Un module de classification qui va classer les formes inconnues et un module de post traitement qui va corriger les résultats du classificateur en utilisant des mécanismes spécifiques au domaine d'application.

Plusieurs approches ont été proposées pour la reconnaissance des formes. Les plus importantes sont. L'approche par prototypes, qui représente chaque classe par un ou plusieurs prototypes obtenus par apprentissage, la classification se fait en assignant la forme à reconnaître à la classe dont les prototypes sont les plus proches. L'approche statistique, qui représente chaque classe par une fonction qui renvoi pour chaque forme la probabilité qu'elle appartienne à la classe en question. Les approches bio inspirées qui tentent de reproduire des mécanismes naturels, les plus populaires sont les réseaux de neurones et les algorithmes génétiques. Sans oublier les approches structurelles (par exemple les arbres de décisions) qui représentent les mécanismes de classification sous la forme d'une structure (par exemple un arbre). Pour augmenter les performances du système de reconnaissance, on combine souvent plusieurs types de classificateurs afin de bénéficier des avantages de chaque type. La combinaison peut se faire en parallèle, en cascade ou bien de façon hiérarchique.

Lorsque les données du problème ne sont pas étiquetées, comme c'est le cas pour les applications de datamining, les approches de classification standard ne sont plus efficaces. On utilise alors des algorithmes dits de clustering, qui tentent de trouver la structure des données en analysant les données d'entraînement.

2 Les systèmes immunitaires :

2.1 Le système immunitaire naturel :

Le système immunitaire est responsable de la défense de l'organisme contre les maladies et autres infections. C'est un système complexe de molécules, cellules et tissus capable de reconnaître aussi bien les cellules anormales du soi que les microorganismes exogènes, ces derniers sont aussi appelés **antigènes [Roitt, 90]**.

Le système immunitaire est divisé en deux systèmes de défenses principaux: le système immunitaire inné, qui représente les défenses présentes dès la naissance de l'individu ; et le système immunitaire adaptatif, c'est l'ensemble des défenses apprises ou acquises au cours du temps **[Roitt, 90]**.

Les cellules qui forment le système immunitaire sont appelées leucocytes. La hiérarchie de cette famille est montrée dans la **figure 2.1**.

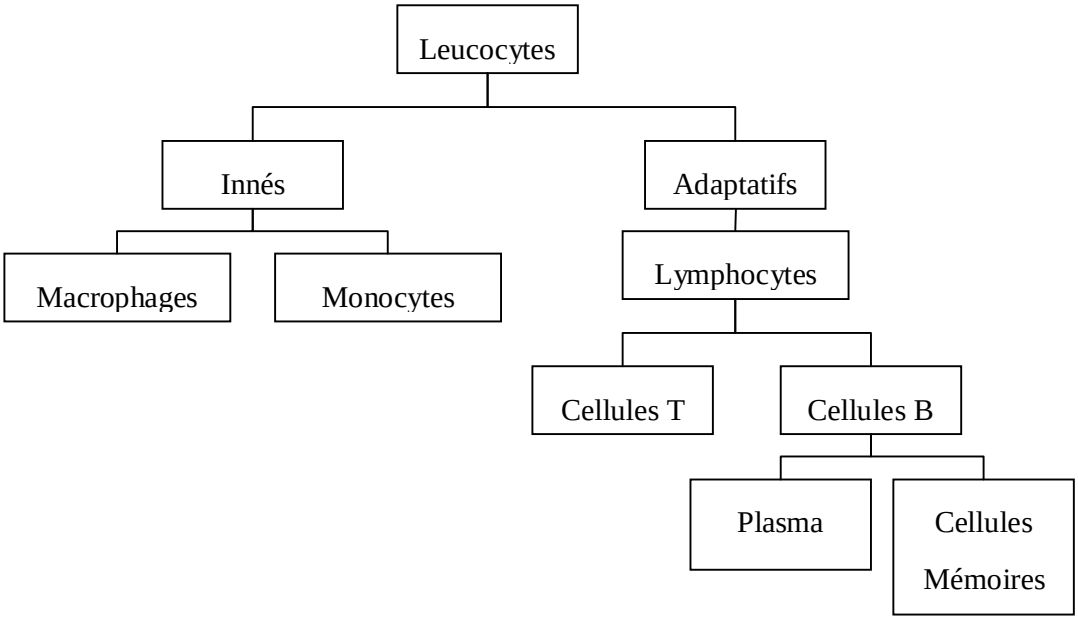

Figure 2-1 Hiérarchie des cellules immunitaires

La réponse immunitaire innée a rarement intéressée les informaticiens, à cause de ses capacités statiques, réagissant à l'infection sans pour autant apprendre à y réponde de manière plus efficace. Par contre, la réponse immunitaire adaptative est d'un grand intérêt dans le domaine informatique, principalement grâce à ses capacités d'apprentissage adaptatif.

2.1.1 Le système immunitaire inné :

Le système immunitaire inné forme la première ligne de défense contre les antigènes et inclut des éléments tels que la peau et les muqueuses. Il utilise différents mécanismes tel que la toux, l'éternuement, les larmes ainsi que la salive pour éliminer les agents nocifs. Cette couche du système immunitaire est capable de détruire une grande variété d'antigènes au premier contact. La réponse est systématique et c'est la même chez tous les individus.

Les cellules impliquées dans la réponse innée sont appelées phagocytes et incluent dans leurs rangs les monocytes et les macrophages. Ces cellules présentent à leurs surfaces des récepteurs qui sont programmés pour reconnaître une forme antigénique commune qui n'est produite que par les microbes et jamais par l'organisme hôte **[Roitt, 90]**.

Les phagocytes se lient aux agents infectieux pour les ingérer. Une fois digéré, l'antigène est découpé en plusieurs morceaux qui sont présentés par les macrophages comme un signal de présence de l'agent infectieux. Les cellules qui effectuent cette tâche de signalisation sont appelées cellules présentatrices d'antigènes. Ce signal peut être utilisé pour stimuler d'autres phagocytes et/ou la réponse immunitaire adaptative.

2.1.2 Le système immunitaire adaptatif :

Les microorganismes peuvent, grâce aux mutations, développer des stratégies qui rendent caduques nos défenses immunitaires innées. Le système immunitaire a donc besoin de développer des défenses spécifiques contre chacun de ces microorganismes, aussi nombreux soient-ils.

La réponse immunitaire adaptative est responsable de la protection de l'organisme contre les microorganismes infectieux non rencontrés auparavant. Ce système est capable d'apprendre et de se rappeler les formes moléculaires présentées par les antigènes. Les cellules impliquées dans la réponse immunitaire adaptative sont appelées lymphocytes, et peuvent être divisés en deux catégories : les cellules B et les cellules T. les cellules B se développent dans la moelle et les cellules T dans le Thymus.

L'immunité spécifique contient deux mécanismes principaux, l'un est responsable de la distinction entre le soi et le non soi, l'autre permet au système de mémoriser les antigènes rencontrés.

2.1.2.1 Distinction entre le soi et le non soi :

De tous les mécanismes qui constituent le système immunitaire naturel, celui-ci est le plus important. A tout moment le système doit être capable d'identifier ses propres molécules (le soi) des molécules étrangères. Les cellules T sont à la base de la distinction entre le soi et

le non soi **[Roitt, 90]**. Non seulement elles peuvent se lier (et détruire) les cellules du soi infectées. Mais en plus, elles sont indispensables à l'activation des cellules B et donc au lancement de la réponse immunitaire. En effet, même si une cellule B reconnaît un antigène, elle ne peut s'activer que si elle reçoit une confirmation des cellules T.

Lors de la maturation des cellules T dans le thymus, elles sont confrontées à un échantillon de molécules du soi. N'importe quelle cellule T qui s'active en présence de cet échantillon est immédiatement détruite. Ceci garantit que les cellules T matures ne vont jamais se lier avec des molécules du soi. Ceci veut dire aussi que si une cellule T se lie à une certaine molécule, cette dernière est forcément une molécule non soi.

2.1.2.2 Mémorisation des antigènes :

Le rôle principal des cellules B est de produire les anticorps **[Roitt, 90]**. Ces molécules, présentes en grandes quantités à la surface des cellules B, peuvent une fois libérées se lier à l'antigène. Les anticorps vont alors attirer les macrophages vers la zone d'infection, faciliter l'ingestion de l'antigène par le macrophage et peuvent même détruire l'antigène directement.

Chaque cellule B produit un seul type d'anticorps. Un anticorps est constitué de deux régions principales (voir **figure 2.2**) : la région variable (VH et VL), qui détermine le type d'antigène avec lequel cet anticorps peut se lier ; et la région constante (CH et CL), à la base de l'anticorps, qui permet entre autres aux macrophages d'ingérer plus facilement l'antigène.

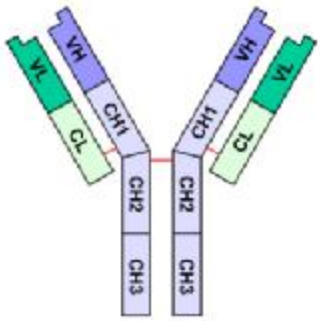

Figure 2-2 Structure d'un anticorps

Chaque anticorps est spécifique à un antigène particulier, et il ne peut se lier qu'avec les antigènes qui ressemblent beaucoup à l'antigène ciblé. La partie de l'anticorps qui se lie à l'antigène est appelée **paratope**, et la partie de l'antigène à laquelle se lie l'anticorps est appelée **épitope**. Il faut noter qu'un antigène contient généralement plusieurs épitopes, et

qu'un anticorps contient aussi des épitopes dans sa partie fixe. La force de la liaison entre l'antigène et l'anticorps est appelée **affinité**.

Lorsqu'un nouvel antigène pénètre dans le corps, la réponse immunitaire passe par les étapes suivantes (principes de la sélection clonale **[Roitt, 90]**, voir **figure 2.3)**:

- Au début, la concentration de l'antigène est tellement faible que seule l'immunité innée est activée. Comme l'antigène est nouveau aucune cellule B n'est assez spécifique pour se lier avec ;
- Au fur et à mesure que l'antigène se développe, sa concentration devient assez élevée pour activer les cellules B les moins spécifiques ;
- Une fois les cellules B activées, elles vont se multiplier pour produire un grand nombre de clones. Chaque clone est une cellule B identique à la cellule qui la produite. Le nombre de clones est proportionnel à l'affinité de la liaison cellule B – antigène ;
- Pour augmenter la spécificité des anticorps et l'efficacité de la réponse immunitaire, les clones entrent dans une phase d'hyper mutations, modifiant ainsi la structure de leurs récepteurs (anticorps). comme les mutations sont aléatoires, les cellules obtenues (dites matures) peuvent devenir plus spécifiques ou moins spécifiques ;
- Lorsque la concentration de l'antigène diminue (à cause de la réponse immunitaire), seules les cellules B les plus spécifiques continuent à être activées, les autres (les moins spécifiques) ne sont plus activées et finissent par mourir. Ceci a pour effet de rendre la population de cellules B de plus en plus spécifique à chaque génération ;
- Après maturation, les cellules B deviennent soit des cellules plasma, soit des cellules mémoire. Les cellules plasma sont de véritables usines à anticorps capable d'en produire en quantités impressionnantes. Les cellules mémoire quand à elles, vont survivre longtemps après la disparition de l'antigène, et peuvent une fois activée produire de grandes quantités d'anticorps en très peu de temps.

Lors de la réponse primaire (un nouvel antigène pénètre dans le corps), il faut plusieurs jours avant que des anticorps efficaces apparaissent dans le sang. Mais lors de la réponse secondaire (un antigène similaire pénètre dans le corps) et grâce aux cellules mémoire issues de la première infection, les anticorps correspondants sont produits beaucoup plus rapidement et en plus grandes quantités. On dit que le système est devenu immunisé contre l'antigène, ou encore qu'il l'a mémorisé (voir **figure 2.4**).

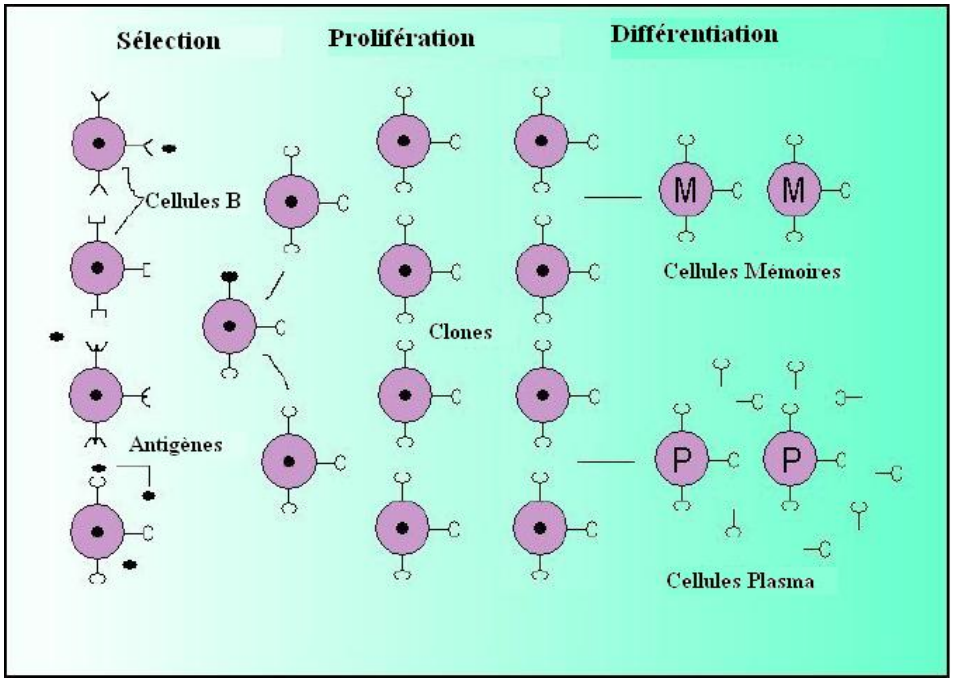

Figure 2-3 Sélection clonale [White, 04]

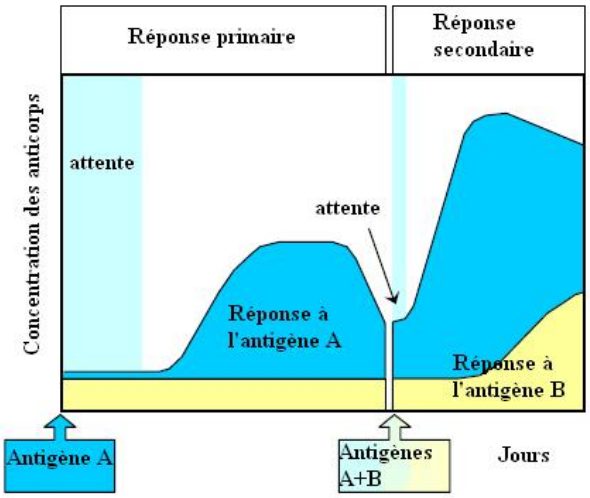

Figure 2-4 Réponses primaire et secondaire [de Castro et Von Zuben, 99]

Il faut préciser que l'acquisition de la mémoire et de l'immunité vis-à-vis d'un microorganisme ne protège en aucun cas des autres microorganismes non apparentés **[Roitt, 90]**. L'immunité acquise est donc spécifique et le système distingue précisément deux microorganismes infectieux différents. La base de ce phénomène réside dans la capacité des anticorps à distinguer entre des antigènes différents.

2.1.2.3 Les réseaux immunitaires :

La théorie des réseaux immunitaires, proposée originellement par Jerne **[Jerne, 74],** a offert un nouveau point de vue en ce qui concerne l'activité des lymphocytes, la production des anticorps, la tolérance, la distinction entre soi et non soi, la mémorisation ainsi que l'évolution du système immunitaire. Cette théorie suggère que le système immunitaire soit composé d'un réseau régulé de cellules et de molécules qui peuvent se reconnaître les unes les autres même sans la présence d'antigènes.

Le système immunitaire a été défini formellement comme un énorme réseau de paratopes qui reconnaissent des ensembles d'épitopes, et des épitopes qui sont reconnus par des ensembles de paratopes, donc chaque cellule peut aussi bien reconnaître qu'être reconnue. Les éléments importants du réseau ne sont pas seulement les molécules, mais aussi les interactions entre ces molécules.

Les cellules immunitaires peuvent répondre positivement ou négativement à un signal de reconnaissance (d'un antigène ou d'une autre cellule immunitaire). Une réponse positive induit l'activation de la cellule, sa prolifération et la sécrétion d'anticorps ; alors qu'une réponse négative conduit à la tolérance et à la suppression (voir **figure 2.5**).

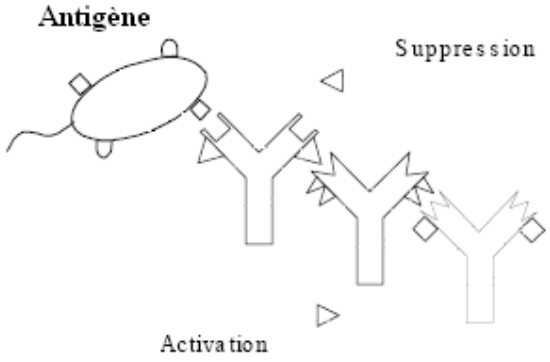

Figure 2-5 Activation/suppression d'un anticorps

La théorie des réseaux immunitaires peut être résumée comme suit [**deCastro et VonZuben, 01**] (**figure 2.6**). Lorsque le système immunitaire entre en contact avec un antigène (**Ag)**, son épitope est reconnu (à différents degrés de spécificité) par un ensemble de paratopes différents (**Pa**). L'ensemble d'épitopes (**Ib**) est appelé l'image interne de l'épitope (ou antigène) parce qu'il peut être reconnu par le même ensemble **Pa** qui a reconnu l'antigène. De plus, chaque épitope de l'ensemble **Ia** est reconnue par un ensemble de paratopes. Donc, la totalité de l'ensemble **Ia** est reconnue par un ensemble encore plus large de paratopes **Pc** associé à un ensemble d'épitopes **Ic**. Les flèches indiquent une stimulation

lorsque les épitopes sont reconnus par les paratopes des récepteurs des cellules, et une suppression lorsque des paratopes reconnaissent les épitopes des récepteurs des cellules.

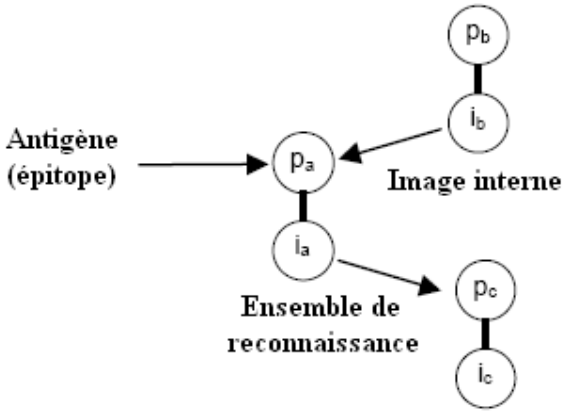

Figure 2-6 Principes des réseaux immunitaires

Ce qui rend cette théorie intéressante est qu'elle stipule que les propriétés des systèmes immunitaires, telles que la distinction entre soi et non soi, ou encore la mémorisation, émergent des interactions entre les cellules immunitaires (activation/suppression).

2.2 Les systèmes immunitaires artificiels :

2.2.1 Pourquoi utiliser les systèmes immunitaires artificiels pour la reconnaissance des formes:

Un système immunitaire artificiel qui réussirait à reproduire le comportement du système naturel aurait les avantages suivants:

- Le système adapte ses ressources selon la complexité de la forme à reconnaître :
  - **a.** Si la forme est connue, elle est traitée rapidement par la cellule mémoire correspondante ;
  - **b.** Moins la forme rencontrée ressemble aux formes précédemment rencontrées, plus la concentration de l'antigène correspondant sera élevée, et plus les ressources mobilisées (cellules activées) seront importantes, ainsi que le temps de traitement.
- Un mécanisme unique est utilisé à la fois pour la reconnaissance rapide des formes déjà rencontrées (classification) ainsi que la mémorisation des formes inconnues

(apprentissage). En plus, le système est en apprentissage continu, chaque forme inconnue enclenche le processus de mémorisation ;

- Le système est en constante adaptation avec son environnement : les formes qui sont rencontrées dans l'environnement sont mémorisées, et celles qui ne le sont pas (ou ne le sont plus) sont tout simplement oubliées. De cette manière, même si le système est placé dans un environnement avec une mémoire initiale très générale, avec le temps sa mémoire va devenir spécifique à son environnement de travail et donc, plus performante;
- Les singularités, même si elles sont mémorisées, peuvent être « oubliées » si elles ne sont pas rencontrées pendant longtemps ;
- La vaccination du système permet de lui faire apprendre rapidement de nouvelles formes, ou bien de restaurer une partie de sa mémoire ;
- Le système fait de la sélection d'attributs de façon automatique, un anticorps n'a pas besoin de se lier avec toute la surface de l'agent infectieux, mais seulement avec la partie qui le rend spécifique.

Cette liste n'est certainement pas exhaustive, et une recherche plus poussée dans les mécanismes de fonctionnement du système immunitaire naturel permettra certainement de trouver d'autres propriétés très intéressantes.

2.2.2 Sélection négative :

La sélection négative (ou détection négative) est une abstraction des mécanismes qui permettent aux systèmes immunitaires naturels de distinguer entre le soi et le non soi. Elle se concentre sur la génération de détecteurs de changements, ces détecteurs sont censés détecter qu'un élément d'un ensemble de chaînes (le soi) a changé.

L'algorithme général de sélection négative, introduit par **[Forrest et al, 94]** est comme suit :

- Au départ nous avons un ensemble de chaînes **S** qui représentent le soi (**figure 2.7**) ;

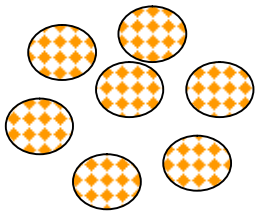

Figure 2-7 Chaînes soi

- On génère aléatoirement un ensemble de détecteurs $\mathbf{R_D}$ (**figure 2.8**) ;

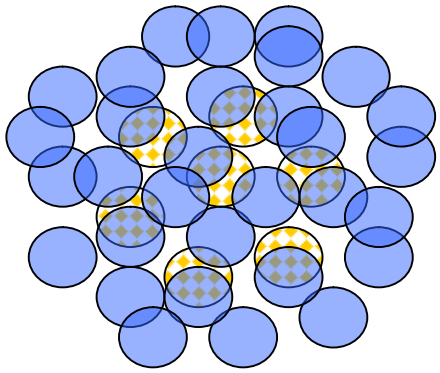

Figure 2-8 Chaînes soi et détecteurs

- Tous les détecteurs de $\mathbf{R_D}$ qui reconnaissent au moins une chaîne de **S** sont éliminés (**Figure 2.9**). La reconnaissance peut être définie, par exemple, en utilisant la règle des k bits contigus, qui dit qu'un détecteur **d** reconnaît une chaîne **s** si au moins **k** bits contigus de **d** sont égaux aux bits correspondants de **s** (voir **figure 2.10**) ;

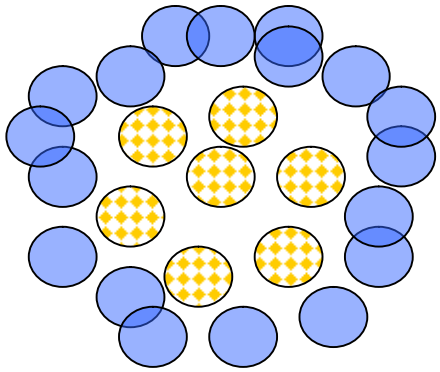

Figure 2-9 Tolérisation

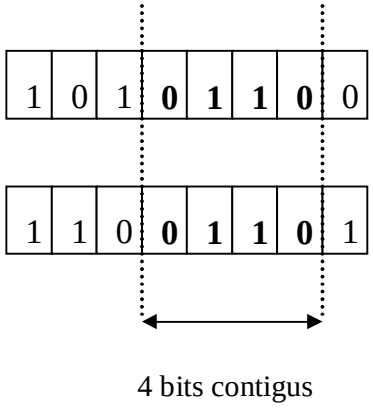

Figure 2-10 Règle des r-bits contigus

- Si une chaîne de **S** change de forme, elle a de grandes chances d'être reconnue par un des détecteurs de $\mathbf{R_D}$ (**figure 2.11**).

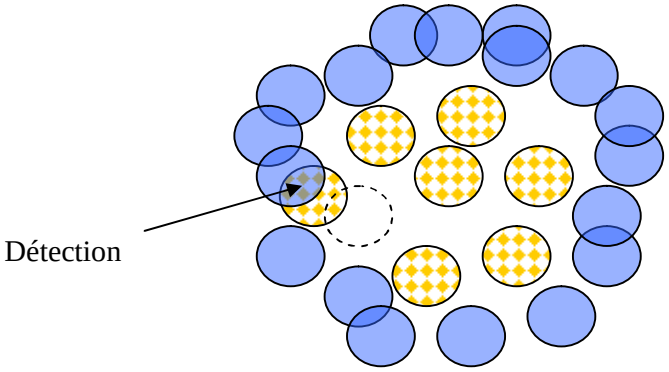

Figure 2-11 Détection de changement

La détection négative est inefficace dans le choix des détecteurs. Une partie de l'espace des formes peut être couverte par plusieurs détecteurs qui se recouvrent ; et d'autres parties peuvent ne pas être recouvertes du tout. Et le pire est que, pour certains ensembles de chaînes, des trous peuvent apparaître qui ne peuvent être couverts par n'importe laquelle des configurations possibles de détecteurs, sans que cela conduise à recouvrir des chaînes du soi (voir **figure 2.12**).

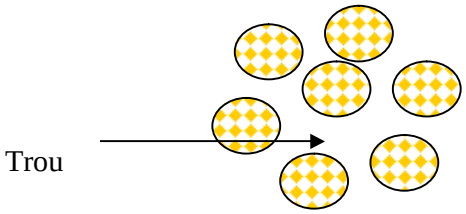

Figure 2-12 Trous dans les chaînes soi

Esponda et Forrest ont appliqué la sélection négative au problème de détection d'intrusions dans un réseau informatique **[Esponda et Forrest, 03]**. Le système, nommé LISYS, génère un ensemble de détecteurs qui, confrontés à un réseau sain, ne doivent détecter aucun paquet transmis dans le réseau. Mais, dès qu'il y a un trafic suspect dans les réseau, les détecteurs sont activés et l'utilisateur est prévenu.

Pour augmenter la diversité des détecteurs les auteurs ont introduit la notion de masques de permutations : à chaque détecteur est associé un masque qui indique pour chaque bit du détecteur le bit correspondant dans le paquet à analyser (**figure 2.13**). Les auteurs ont montré que les masques de permutations permettaient de réduire les trous dans la couverture.

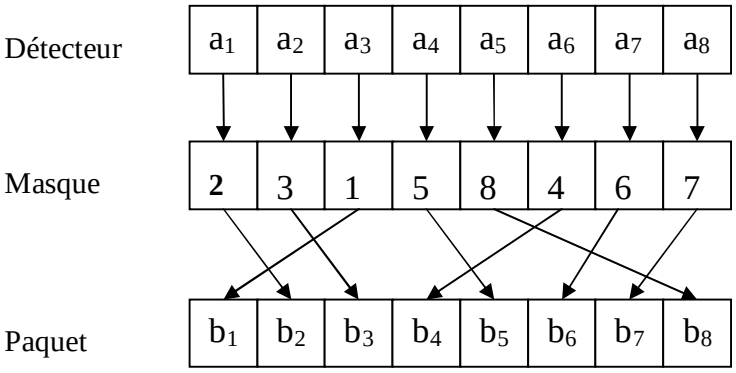

Figure 2-13 Masque de permutations

Les auteurs ont aussi introduit aussi la notion de 'r-chunks' (**figure 2.14**) , qui est une variante de la règle des k bits contigus. Pour les détecteurs utilisant la règle des r-chunks, on spécifie seulement **r** bits contigus du détecteur (appelés 'fenêtre'), les autres sont mis à l'état 'n'importe' (symbolisé par '*'). Un détecteur **d** se lie à une chaîne **x** si tous les bits de **d** appartenant à la fenêtre, sont égaux aux bits correspondant dans **x**. les auteurs ont montré que les r-chunks donnent de meilleurs résultats que la règle des k bits contigus pour l'ensemble de test utilisé.

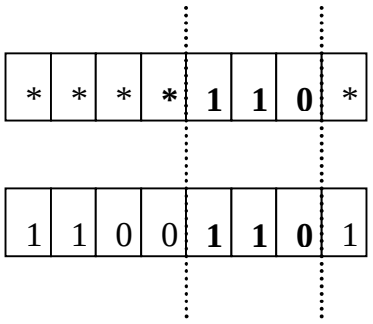

Figure 2-14 Règle des r-chunks

Dans **[Esponda et al, 2005]**, les auteurs ont appliqué la sélection négative aux bases de données 'négatives', qui permettent de stocker des données dans une forme 'négative' de façon à ce que la récupération des données sous leur forme d'origine est un problème NP – difficile. Les bases de données négatives sont très utiles dans le domaine de la sécurité informatique, par exemple pour stocker les mots de passes. Les requêtes de type « est-ce que la chaîne x se trouve dans la base ? » peuvent être satisfaites très rapidement, alors que des requêtes du genre « donnez moi toutes les chaînes qui commencent par '1' » sont très difficiles à résoudre.

Les auteurs ont utilisé les principes de la sélection négative, en stockant dans la base de données négative l'ensemble minimal de détecteurs qui ne détecte aucune chaîne de la base de données d'origine. Les auteurs ont proposé des algorithmes pour l'initialisation et la mise à jour de la base de données négative.

2.2.3 La théorie du danger :

La théorie du danger est une approche émergente pour la détection de changements basée sur une abstraction de Matzinger **[Matzinger, 02]**. La théorie du danger est une alternative à la sélection négative.

Dans le milieu des années 90, l'immunologie a effectuée plusieurs modifications à la théorie du soi/non soi. Principalement parce que cette théorie ne concordait plus avec les observations expérimentales, par exemple pourquoi le système naturel ne réagit-il pas à certaines bactéries dans nos intestins ou à l'air, qui sont tous les deux des agents non soi ? Matzinger a suggéré une autre approche : et si le système immunitaire, au lieu de réagir directement aux entités non soi, réagirait plutôt à des cellules du soi attaquées, ces cellules enverraient alors des signaux d'alarme ou de danger? Matzinger a caractérisé la théorie du danger comme un moyen de distinguer certaines cellules du soi de certaines cellules du non soi, ce qui pourrait expliquer pourquoi le système ne réagit pas aux entités non soi non dangereuses, et réagirait aux cellules soi dangereuses. Ceci est la base de la théorie du danger.

Il y a une grande différence entre la dichotomie soi/non soi et celle du dangereux/pas dangereux. Le concept du soi et non soi est relatif aux chaînes du soi, qui ne sont pas nécessairement l'ensemble complet du soi, ce dernier peut changer au cours du temps et peut aussi contenir un grand nombre d'attributs. Par contre, les concepts de dangereux pas dangereux sont reliés directement aux événements indésirables, qui sont ou qui seront dangereux, et sont en rapport uniquement avec les attributs concernés par ces événements.

Les signaux de danger peuvent être positifs, indiquant la présence d'un événement ou d'un état, par exemple : une grande activité du disque, la fréquence de changement des fichiers, ainsi que la présence du non soi. Les signaux peuvent aussi être négatifs, comme par exemple : l'absence de signaux ou d'états.

Selon la théorie du danger, lorsqu'une cellule est attaquée par un agent infectieux, elle émet un signal de danger qui se propage dans les alentours de la cellule (**figure 2.15**). Ce signal va définir une 'zone de danger' tout autour de la cellule attaquée. Les antigènes sont capturés par les macrophages (réponse innée) puis présentés aux lymphocytes. Les cellules B qui se lient avec les antigènes dans la zone de danger sont activées et commencent alors à se reproduire, les autres cellules B (qui ne se sont pas liées avec les antigènes ou bien qui sont en dehors de la zone de danger) ne sont pas stimulées.

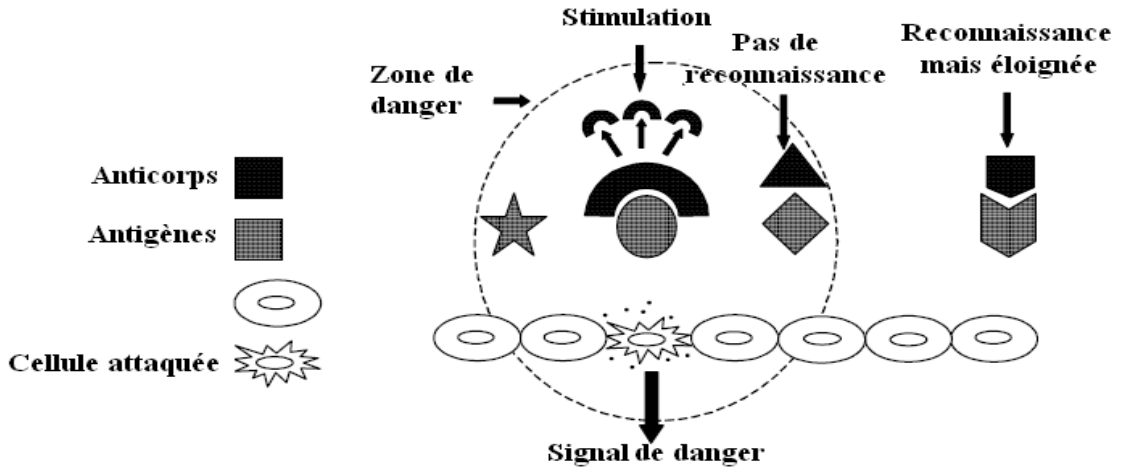

Figure 2-15 Principe de la théorie du danger

Aicklin et Cayzer **[Aicklin et Cayzer, 02]** ont montré l'applicabilité de la théorie du danger dans le domaine de la détection d'anomalies (virus informatiques, transactions frauduleuses, défaillances matérielles), dans le datamining et de façon générale dans tout domaine pour lequel on peut définir la nature du 'danger'.

Dans **[Aicklin et al, 03]**, les auteurs ont investigué les avantages de la théorie du danger appliqués à la détection d'intrusion : la théorie du danger permet de détecter les intrusions inconnues à partir des dommages infligés au système, elle réduit aussi le nombre de fausses alertes puisqu'elle n'est enclenchée que s'il y a réellement un danger. Elle peut aussi s'adapter aux changements et permet de réduire la propagation des intrusions en alertant les sites voisins de l'infection (zone de danger).

La théorie du danger a aussi été utilisée pour l'analyse du courrier électronique **[Secker et al, 03]**. Le système développé analyse le courrier entrant et accepte ou refuse des emails selon qu'ils soient intéressants pour l'utilisateur ou non. Bien que c'est principalement un problème de classification, la frontière entre les classes (emails intéressants/non intéressants) change selon l'humeur de l'utilisateur. La théorie du danger permet au système de s'adapter à ces changements.

Un inconvénient majeur de la théorie du danger est que le système doit attendre que le soi soit endommagé avant de pouvoir activer la protection, parce qu'il nécessite des exemples d'états dangereux. Ceci n'est pas le cas de la détection négative qui se contente des exemples d'états sains du système.

2.2.4 Sélection clonale artificielle et hyper mutations :

La sélection clonale artificielle est une abstraction des mécanismes de mémorisation des systèmes immunitaires. Les algorithmes développés sont généralement dédiés à l'optimisation ou à la recherche.

Un algorithme général de sélection clonale artificielle peut être décrit comme suit **[de Castro et Von Zuben, 02]** :

- **Initialisation :** générer aléatoirement la population initiale d'anticorps **Ab**, un anticorps est une abstraction de la cellule B et des anticorps qu'elle produit (**figure 2.16**);

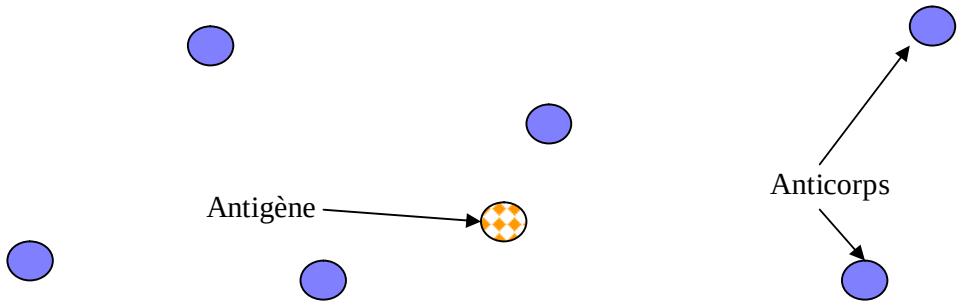

Figure 2-16 Initialisation de la population d'anticorps (cellules B)

- Répéter tant qu'une condition prédéfinie n'est pas vérifiée :
  - **Evaluation et sélection 1 :** sélectionner $\mathbf{s_1}$ anticorps de **Ab** qui ont les plus grandes affinités. Utiliser une certaine fonction $f(a)$ qui renvoie l'affinité de l'anticorps (**figure 2.17**);

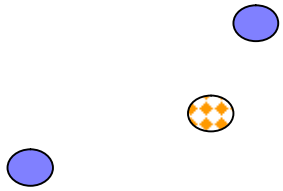

Figure 2-17 Cellules B activées

  - **Clonage :** cloner chaque anticorps sélectionné proportionnellement à son affinité, mettre les clones dans **C** ;
  - **Mutation :** chaque clone de **C** est muté avec un degré inversement proportionnel à son affinité, nous obtenons une population mature **C*** (**figure 2.16**);

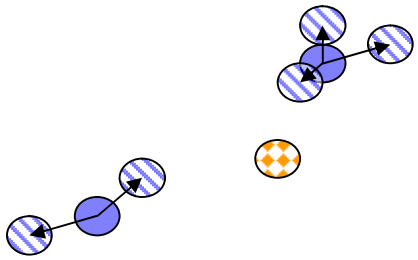

Figure 2-18 Maturation des cellules B

- **Evaluation et sélection 2 :** sélectionner $\mathbf{s_2}$ anticorps de **C*** qui ont les plus grandes affinités, nous obtenons **Ab'** (**figure 2.19**);

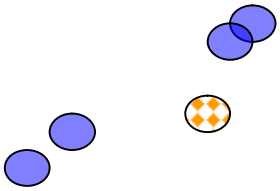

Figure 2-19 Nouvelle population d'anticorps

- **Diversité :** combiner **Ab'** avec **s3** anticorps générés aléatoirement, remplacer **Ab** par l'ensemble d'anticorps obtenus (**figure 2.20**);

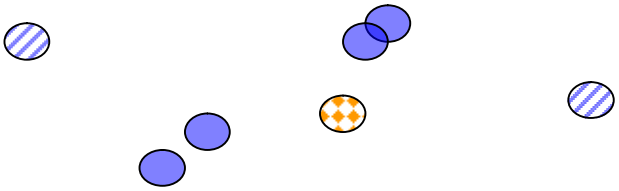

Figure 2-20 Diversification de la population d'anticorps

- **Mort :** les anticorps non retenus dans **Ab** sont éliminés.

De Castro et Von Zuben ont appliqué la sélection clonale à l'optimisation **[de Castro et Von Zuben, 02]**. Dans ce cas, chaque anticorps représente une solution possible au problème. La fonction affinité, quand à elle, renvoie la qualité de chaque solution (les meilleures solutions ont les plus grandes affinités).

Les auteurs ont aussi proposés CLONALG, une implémentation de la sélection clonale pour la reconnaissance des formes. Mais n'étant pas plus qu'une preuve de faisabilité, l'algorithme souffre de limitations majeures. La plus importante étant qu'il n'accepte pas plus d'un exemple d'entraînement par classe.

White et Garrett ont proposé CLONCLAS, qui est une amélioration de CLONALG **[White et Garrett, 03]**. Les auteurs ont utilisé la sélection clonale pour chercher pour chaque classe le prototype qui la représente le mieux. Ces prototypes sont ensuite utilisés dans un système de reconnaissance de chiffres imprimés. En d'autres termes, la sélection clonale est utilisée comme un algorithme d'apprentissage.

Les principes de la sélection clonale ont aussi été appliqués à la résolution de problèmes multi objectifs. Pour ce genre de problèmes, plusieurs objectifs doivent être optimisés ensemble. Les objectifs étant souvent en conflits, l'optimisation d'un objectif rend les autres objectifs non optimisés. Cuello a proposé un algorithme pour la résolution des problèmes multi objectifs en utilisant les principes de la sélection clonale. L'algorithme nommé MISA (pour Multiobjective Immune System Algorithm) est générique dans le sens où il peut être appliqué à n'importe quel problème d'optimisation quel que soit le nombre d'objectifs. Pour plus de détails sur l'algorithme et les résultats obtenus le lecteur est invité à consulter **[Coello et Cortez, 05]**.

Dans **[Kim et Bentley, 02]**, les auteurs ont proposé un algorithme de sélection clonale destiné à la détection d'intrusions. L'algorithme nommé DynamiCS, est un algorithme de classification qui a deux classes le soi et le non soi, et utilise les propriétés de la sélection clonale pour générer des cellules mémoire qui reconnaissent le non soi sans reconnaître le soi.

Bien que la sélection clonale ressemble beaucoup aux algorithmes génétiques, il reste des différences majeures entre les deux approches : l'utilisation de l'affinité pour déterminer le taux de mutation contraste avec les algorithmes génétiques où le taux de mutation dépend soi d'un paramètre, soi du nombre de générations. Similairement, la relation entre l'affinité d'une entité et le nombre de clones est inusuelle sinon unique à la sélection clonale.

2.2.5 Les modèles de réseaux immunitaires artificiels :

Dans **[deCastro et VonZuben, 01]**, les auteurs ont proposé aiNet, un algorithme qui combine la théorie des réseaux immunitaires et la sélection clonale. AiNet utilise la notion d'image interne pour représenter les regroupements de données dans un réseau. Par exemple, pour l'ensemble de données de la **figure 2.21**,une architecture hypothétique générée par aiNet est donnée à la figure **2.22**. Les nœuds représentent les anticorps, les lignes pleines sont des connections entre les anticorps, et les lignes en pointillé sont des connections qui seront éliminées pour révéler les regroupements. Comme le nombre de nœuds du réseau est plus petit que le nombre initial de données, aiNet peut être utilisé pour la compression.

Les cellules immunitaires sont en compétition pour se lier avec l'antigène, celles qui réussissent sont activées, alors que les autres sont éliminées. De plus, la liaison **Ab – Ab** (anticorps – anticorps) conduit à la suppression. Dans aiNet, la suppression se fait en éliminant les anticorps qui se lient à eux-mêmes.

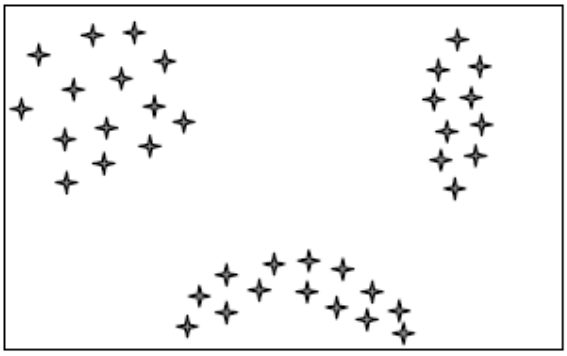

Figure 2-21 Exemple de groupements de données

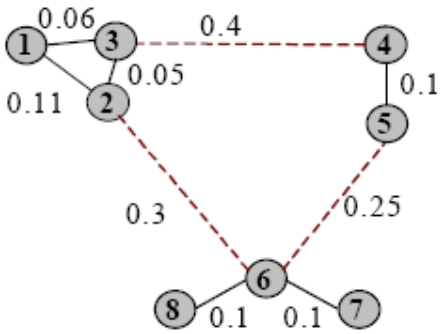

Figure 2-22 Réseau immunitaire généré par aiNet

L'algorithme de aiNet est comme suit :

Initialisation :

La population d'anticorps est générée aléatoirement ;

La population de cellules mémoire est initialement vide ;

La population d'antigènes contient les données d'entraînement.

Pour chaque antigène, faire :

Répéter un certain nombre d'itérations

Evaluation et sélection 1 :

Evaluer l'affinité de tous les anticorps pour l'antigène courant ;

Sélectionner **n** anticorps qui ont les plus grandes affinités.

Clonage :

Cloner les anticorps proportionnellement à leur affinité.

Mutation :

Muter les anticorps clonés avec un taux inversement proportionnel à leur affinité pour obtenir une population mature.

Evaluation et sélection 2 :

Evaluer l'affinité des anticorps matures pour l'antigène courant ;

Sélectionner *z* % anticorps qui ont les meilleures affinités pour obtenir les cellules mémoire.

Suppression clonale 1 :

Eliminer les cellules mémoire qui sont très similaires les unes aux autres.

Mémorisation :

Ajouter les cellules mémoire qui ont une affinité qui dépasse un certain seuil à la population de cellules mémoire.

Mort :

Les anticorps et cellules mémoire non mémorisés sont éliminés.

Suppression clonale 2 :

Eliminer les cellules mémoire (de la population de cellules mémoire) très similaires les unes aux autres.

Les auteurs utilisent ensuite des stratégies issues de la théorie des graphes pour détecter les regroupements dans le réseau obtenu, et ont appliqué aiNet pour des problèmes de clustering. L'approche réseau est particulièrement intéressante pour le développement d'outils de calculs parce qu'elle tient compte, potentiellement, de propriétés émergentes telles que l'apprentissage et la mémoire, la tolérance au soi, la gestion de la taille de la population de cellules ainsi que de la diversité.

Il y a beaucoup de similarité entre aiNet et les algorithmes de sélection clonale. En fait la sélection clonale artificielle est un cas particulier des réseaux immunitaires **[Garrett, 05]**. Dans aiNet, ce sont les étapes de suppression clonales qui distinguent cet algorithme de la sélection clonale puisqu'elles expriment des interactions explicites entre les anticorps (cellules mémoire) dans le réseau. Alors que dans les algorithmes de sélection clonale aucune interaction explicite ne se fait entre les anticorps.

De leurs coté, Knight et Timmis ont proposé MARITA, un algorithme pour l'apprentissage supervisé inspiré de la théorie des réseaux immunitaires et de la sélection clonale **[Knight et Timmis, 02]**. La nouveauté de leur approche est que le système est divisé en plusieurs couches, chaque couche étant responsable d'une fonction précise. La sortie de chaque couche est l'entrée de la couche suivante. Les couches du système sont : la couche d'anticorps, la couche des cellules B et la couche de cellules mémoire. Le système utilise un

mécanisme de contrôle de la population qui élimine de toutes les couches les cellules qui n'ont pas été stimulées pendant une longue période. MARITA a été appliqué au datamining pour lequel les antigènes sont les données non étiquetés, et les cellules mémoire représentent les structures décelées dans ces données (**figure 2.23**).

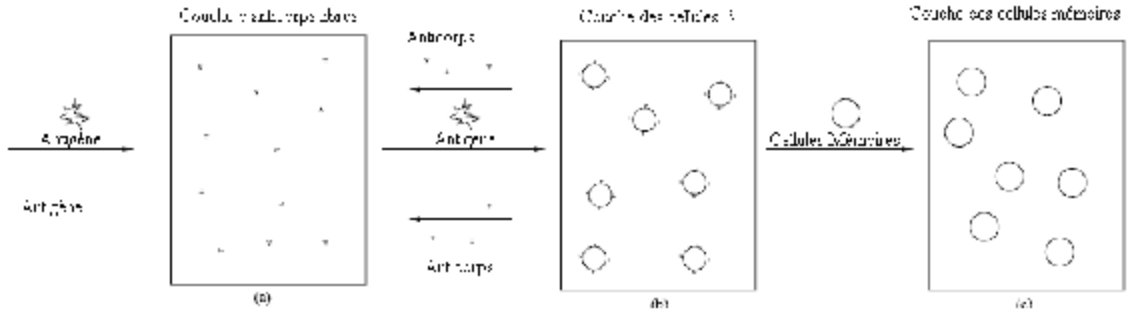

Figure 2-23 Couches des cellules immunitaires de MARITA

Nasraoui et al **[Nasraoui et al, 02]** ont appliqué les réseaux immunitaires artificiels à l'analyse des activités d'un serveur web. Le but du système est de détecter des utilisateurs 'types' à travers le parcours des utilisateurs dans le serveur. Ce système permet aux gestionnaires du système de mieux organiser les pages selon les types utilisateurs les plus fréquents. Le serveur pourra aussi prédire le type d'un nouvel l'utilisateur et adapter son interface pour ce dernier.

2.3 Conclusion :

Les systèmes immunitaires sont un exemple de systèmes naturels qui ont servi de source d'inspiration pour résoudre une grande variété de problèmes informatiques. Dans le cadre de la reconnaissance des formes, les systèmes immunitaires naturels présentent plusieurs propriétés très intéressantes comme la distinction entre le soi et le non soi, la détection de changement, la mémorisation, l'adaptabilité ainsi que la gestion des ressources.

Le système immunitaire naturel dispose de deux sortes de défenses : les défenses innées, à large spectre d'action mais d'efficacité moyenne, et les défenses acquises qui sont très spécifiques mais aussi beaucoup plus efficaces. Vu leur nature adaptative, ce sont les mécanismes de défense acquise qui ont le plus inspiré la recherche informatique.

Les systèmes immunitaires artificiels se divisent en trois grandes familles : la sélection négative, qui s'inspire des mécanismes naturels de distinction entre soi et non soi, pour résoudre des problèmes de surveillance et de détection de changement. La sélection clonale artificielle, qui s'inspire des mécanismes naturels de mémorisation pour résoudre des problèmes d'optimisation. Les réseaux immunitaires artificiels, qui s'inspirent de la théorie des réseaux immunitaires pour construire des systèmes qui permettent la distinction et la mémorisation, ces derniers ont surtout été utilisés pour l'analyse de données.

3 Contribution à l'amélioration de CLONCLAS:

3.1 CLONCLAS, l'original :

3.1.1 Introduction :

CLONCLAS (pour CLONal CLASsification) est l'adaptation de la sélection clonale pour la reconnaissance des formes. Le système est basé sur une approche de reconnaissance par prototypes et utilise la sélection clonale pour faire son apprentissage.

En entrée nous avons des images binaires de chiffres imprimés, un exemple est donné à la **figure 3.1**. Nous supposons donc qu'une étape de préparation de données a été effectuée, à l'issue de laquelle les chiffres ont été segmentés et mis sous forme binaire.

Figure 3-1 images binaires de chiffres

Pour chaque classe de chiffres (0-9) nous avons un certain nombre d'exemples d'entraînements. Le but de la phase d'apprentissage est de trouver pour chaque classe le modèle ou prototype, qui est l'image qui représente le mieux les exemples de la classe. Pour classer une forme inconnue, elle reçoit la classe du modèle qui lui ressemble le plus, en utilisant une certaine mesure de similarité.

Pour pouvoir utiliser les principes de la sélection clonale dans le système décrit, les auteurs de CLONCLAS ont défini les correspondances suivantes **[White et Garrett, 03]**:

- **Antigène :** représente la classe pour laquelle nous voulons calculer le modèle, chaque antigène étant traité indépendamment des autres. Concrètement l'antigène Ag_i est l'ensemble des exemples d'entraînement de la classe Cla_i. Chaque exemple est un vecteur de **len** bits, dont chaque bit correspond à un pixel de l'image du caractère (voir **figure 3.2**).

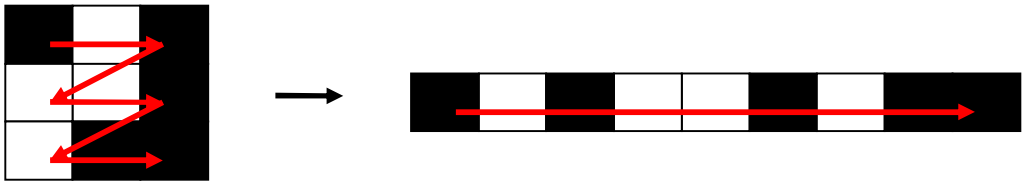

Figure 3-2 Image binaire à vecteur

- **Anticorps :** représente une solution possible pour le problème courant. Si le système est confronté à l'antigène Ag_i, chaque anticorps Ab_j représente un modèle possible pour la classe Cla_i. Un anticorps est un vecteur binaire de **len** bits, dont chaque bit est un pixel de l'image du modèle.
- **Affinité anticorps–antigène :** l'affinité entre un anticorps Ab_j et un antigène Ag_i indique le degré de similarité entre le modèle représenté par l'anticorps Ab_j et le meilleur modèle possible pour Ag_i. Plus l'affinité est grande et plus Ab_j a de chances d'être le meilleur modèle de Ag_i.
- **Cellule mémoire :** la cellule mémoire Abm_i représente le meilleur modèle trouvé pour la classe Cla_i.

3.1.1.1 Apprentissage :

La sélection clonale artificielle étant un algorithme d'optimisation, elle va servir à optimiser l'affinité des modèles (cellules mémoire) de chaque classe. L'algorithme CLONCLAS est le suivant **[White et Garrett, 03]** :

1) Générer aléatoirement la population initiale d'anticorps, **Ab**. Cette population est divisée entre : cellules mémoire **Abm** (1 par antigène), et population réservoir **Abr** ;
2) Créer un ensemble d'antigènes **Ag** à partir des exemples d'entraînement ;
3) Sélectionner un antigène Ag_i de la population **Ag** ;
4) pour **G** générations faire :
    - pour chaque membre de **Ab**, calculer son affinité avec l'antigène Ag_i en utilisant une certaine fonction de similarité ;
    - Sélectionner les **n** anticorps avec la meilleure affinité, cloner chaque anticorps proportionnellement à son affinité, placer les clones dans une nouvelle population C_i ;
    - Muter les éléments de C_i avec un degré inversement proportionnel à leurs affinités pour produire une population mature C_i^* ;
    - Calculer l'affinité de C_i^* avec Ag_i et choisir le meilleur anticorps comme candidat. Si son affinité est meilleure que celle de Abm_i, le candidat remplace Abm_i.
    - Remplacer les anticorps de **Abr** par les meilleurs de C_i^*.

- Remplacer **d** anticorps de **Abr** par des anticorps générés aléatoirement.

5) Retourner à 3 tant qu'il reste des antigènes dans **Ag**.

L'affinité entre un anticorps **Ab** et un antigène **Ag** est donnée par la formule :

$$\sum_{j=1}^{n}\sum_{i=1}^{len} Ab_i \oplus Ag_i^j \qquad (3.1)$$

Où Ab_i est le i^{ème} bit de **Ab**, Ag_i^j est le i^{ème} dit du j^{ème} exemple de **Ag**, **n** est le nombre d'exemples par classe et **len** est la longueur d'un exemple en bits. $\oplus$ est l'opérateur *ou exclusif*.

Le nombre de clones pour un anticorps est donné par la formule :

$$ceil((n \times l) \times m) \qquad (3.2)$$

Où $ceil(\ )$ est la fonction qui renvoie l'entier le plus grand ou égal au nombre réelle qu'elle reçoit. $n = \frac{affinité}{affinitéMax}$. *affinité* est l'affinité de l'anticorps, *affinitéMax* est la valeur maximale possible de l'affinité. **len** est la longueur de **Ab** en bits, et **m** est un facteur multiplicateur.

Le nombre de bits à muter pour chaque anticorps est donné par la formule :

$$round((1-n) \times len) \qquad (3.3)$$

Où $n = \frac{affinité}{affinitéMax}$ et **len** est la longueur de l'anticorps en bits.

3.1.1.2 Classification :

Figure 3-3 Exemple de cellules mémoire

A la fin de l'apprentissage nous disposons d'un ensemble de cellules mémoire (voir **figure 3.3**). Cet ensemble sera utilisé pour classer les formes inconnues de la façon suivante :

- En entrée nous avons les cellules mémoire obtenues par apprentissage (**Abm**) et une forme à classer *f* **;**

- Pour toute cellule mémoire Abm^j appartenant à **Abm**, calculer l'affinité Aff^j entre Abm^j et f en utilisant la formule suivante :

$$\sum_{i=1}^{len} f_i \oplus Abm_i^j \qquad (3.4)$$

Où f_i est le i^{ème} bit de f, Abm_i^j est le i^{ème} dit de Abm^j, **len** est la longueur d'un exemple en bits.

- Trouver la cellule mémoire Abm^k telle que Aff^k est la plus grande ;
- Si Aff^k dépasse un certain seuil ***s***, la forme f reçoit la classe **k**, sinon f est rejetée sans être classée.

Dans leur papier d'origine **[White et Garrett, 03]**, les auteurs ont utilisé un seuil ***s*** calculé à partir des données d'entraînement. Ce seuil permet de déterminer si une certaine forme peut être classée ou doit être rejetée. Cependant, les auteurs n'expliquent que vaguement comment ce seuil a été calculé.

3.1.2 Performances de CLONCLAS :

3.1.2.1 Description des tests effectués sur CLONCLAS :

CLONCLAS a subi cinq sortes de tests. Les deux premiers ne servaient qu'à comparer les performances obtenues par CLONCLAS avec celles obtenues par CLONALG. Ces deux tests ne contiennent qu'un seul exemple d'entraînement par classe, ils ne seront donc pas traités dans ce mémoire. Les trois tests restants ont servi à vérifier l'applicabilité de CLONCLAS à un problème de reconnaissance des formes plus réaliste.

Le test 3.1, (comme il a été nommé dans **[White et Garrett, 03]**) consiste à effectuer l'entraînement sur un ensemble de dix exemples par classe (voir **figure 3.4**). Les tests se font sur l'ensemble d'entraînement lui-même. Ce test permet de vérifier la validité de CLONCLAS comme algorithme d'entraînement pour un système de reconnaissance.

Le test 3.2 consiste à effectuer l'entraînement sur le même ensemble utilisé pour le test 3.1. Mais cette fois, le test se fait sur un ensemble différent de l'ensemble d'entraînement (voir **figure 3.5**). Ce test permet de vérifier le pouvoir de généralisation des modèles trouvés par CLONCLAS.

Figure 3-4 Exemples d'entraînement des tests 3.1 et 3.2

Figure 3-5 Exemples de vérification du test 3.2

Le test 4.1, consiste à effectuer l'entraînement sur un ensemble d'images non normalisées (voir **figure 3.6**). Le test se fait sur un autre ensemble d'images non normalisées (voir **figure 3.7**). Ce test permet de vérifier les performances de CLONCLAS face à des données non normalisées, causées par exemple par une méthode de segmentation peu efficace.

Figure 3-6 Exemples d'entraînement du test 4.1

Figure 3-7 Exemples de vérification du test 4.1

3.1.2.2 Résultats obtenus par CLONCLAS :

CLONCLAS a obtenu les résultats suivants **[White, 04]**:

- Pour le test 3.1, CLONCLAS atteint les 87.5% de classifications correctes en utilisant le seuil lors de la classification. Ces résultats sont obtenus après 50 générations. Cette limite de performance n'est pas dépassée même si le nombre de générations augmente. Lorsque le seuil est désactivé (y a pas de rejet) les performances augmentent pour atteindre les 90% de bonnes classifications (**figure 3.8, 3.9**).

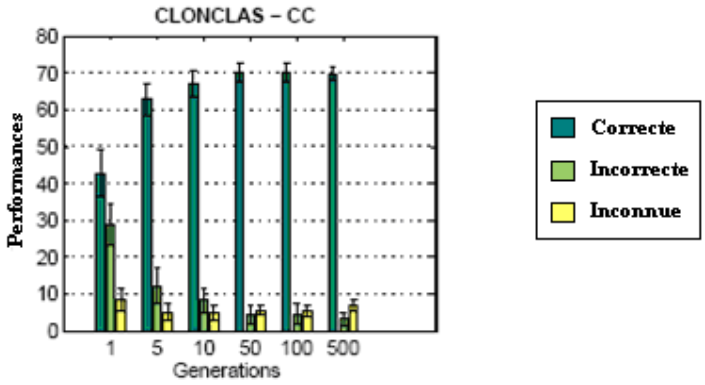

Figure 3-8 performances/génération (test 3.1)

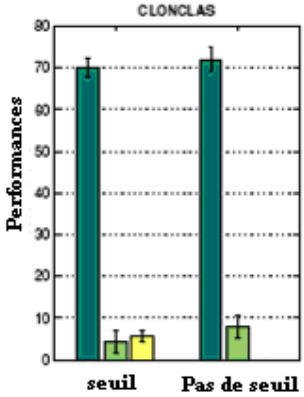

Figure 3-9 Influence du seuil sur les performances (test 3.1)

- Pour le test 3.2, CLONCLAS atteint les 76.3% de classifications correctes (en utilisant le seuil) en 50 générations. Lorsque le seuil est désactivé, CLONCLAS affiche 88% de bonnes classifications (**figure 3.10, 3.11**).

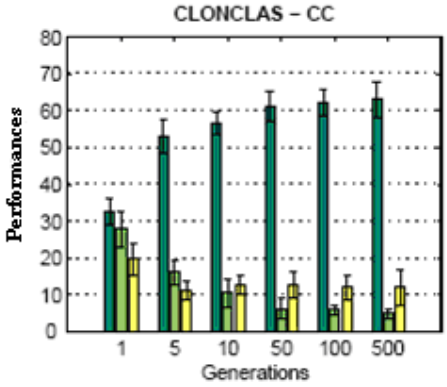

Figure 3-10 performances/génération (test 3.2)

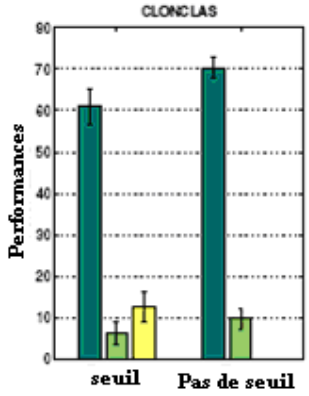

Figure 3-11 Influence du seuil sur les performances (test 3.2)

- Pour le test 4.1 (données non normalisées), les performances de CLONCLAS sont médiocres, ne dépassant pas les 28.8% (avec seuil) après 500 générations. Lorsque le seuil est désactivé, les performances de CLONCLAS atteignent les 40.8% de bonnes classifications (**figure 3.12, 3.13**).

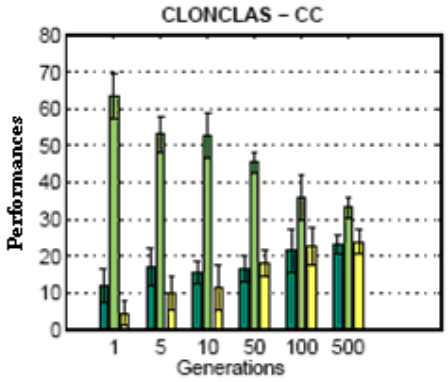

Figure 3-12 performances/génération (test 4.1)

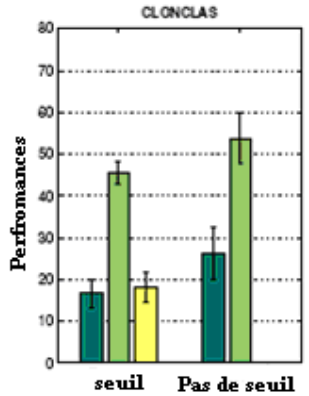

Figure 3-13 Influence du seuil sur les performances (test 4.1)

3.1.2.3 Complexité de CLONCLAS

Lors de l'exécution de CLONCLAS, une seule fonction semble influencer la vitesse d'exécution. Cette fonction est la fonction de calcul de l'affinité des anticorps. En effet, à chaque génération elle est appelée **N** fois pour calculer les affinités des anticorps sélectionnés, puis **Nc** fois pour calculer l'affinité des anticorps clonés (**Nc** étant le nombre de clones obtenus pour cette génération), ceci pour chacun des M antigènes. Donc pour **G** générations, la fonction affinité est appelée $G \times M \times (N + Nc)$.

La complexité de la fonction affinité elle-même dépend du nombre d'exemples d'entraînement qui est égal à $(NE \times M)$, où *NE* est le nombre d'exemples par classe. La complexité dépend aussi de la longueur *L* des anticorps. Donc la complexité de la fonction affinité est de l'ordre de

$$O(NE \times M \times L) \tag{3.5}$$

Au total la complexité de CLONCLAS est

$$O(G \times (N + Nc) \times NE \times M^2 \times L) \tag{3.6}$$

3.2 Réduction du nombre de clones :

3.2.1 Problématique :

Bien que les résultats de CLONCLAS soient très prometteurs. Dans **[Garrett, 05]** l'auteur a cependant noté un sérieux handicap, à savoir la formule de calcul du nombre de clones. Cette formule rend le nombre de clones proportionnel à l'affinité des anticorps sélectionnés mais sans poser de limites au nombre de clones générés. Plus la population d'anticorps se rapproche de l'optimum (local ou global) et plus l'affinité des anticorps sélectionnés augmente. Ceci peut sérieusement ralentir l'apprentissage du système puisque un grand nombre de clones signifie beaucoup d'évaluations de la fonction d'affinité.

La **figure 3.14** montre l'évolution du nombre de clones pour une exécution de CLONCLAS. Ce graphe montre clairement que le nombre de clones augmente au fur et à mesure que la population d'anticorps se rapproche de l'optimum recherché. Bien que la population d'anticorps n'atteint pas 600 individus, le fait que le nombre de clones augmente est un sérieux problème. Surtout si on veut l'appliquer à des problèmes complexes, pour lesquels la taille de la population clonée peut rapidement devenir énorme. L'effet direct de cette augmentation est un ralentissement du système lorsqu'il se rapproche de la solution recherchée.

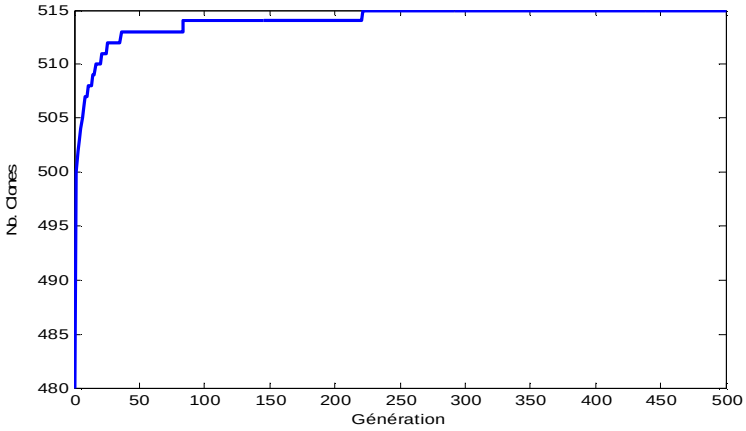

Figure 3-14 Nombre de clones / génération (CLONCLAS)

3.2.2 Une nouvelle formule pour le nombre de clones :

Pour réduire la complexité de CLONCLAS, la formule de calcul du nombre de clones doit impérativement être revue. La formule que nous proposons est comme suit :

$$round\left(b \times \frac{affinite_i^2}{\sum_j affinite_j^2} \right), \forall j \qquad (3.7)$$

Où $round(\ )$ est la fonction qui renvoie l'entier le plus proche du nombre réelle qu'elle reçoit. $affinite_i$ est l'affinité de l'anticorps cloné, β est la taille désirée de la population totale de clones.

La signification de cette formule est comme suit : la taille maximale de la population de clones **C** est bornée à β. Les anticorps sont en concurrence pour se partager cette population, les anticorps qui ont les plus grandes affinités se taillent les plus grandes parts de **C**.

D'un point de vue biologique, cette nouvelle formule est non seulement justifiée, mais elle est aussi plus proche de la réalité. Considérons l'exemple de la **figure 3.15**, un antigène est présent dans le sang en dix exemplaires. Trois types de cellules B sont dans son entourage. Pour chaque type, quatre cellules B sont disponibles. Comme les cellules B du premier type ont la plus grande affinité avec l'antigène elles se lient toutes les quatre. Pour les cellules B de type 2 et 3, ayant leur affinité moins importantes elles vont se partager les six antigènes restants. Les cellules B de type 2 et 3 ne vont lier que trois cellules chacune (**figure 3.16)**. Si maintenant chaque cellule liée va produire une dizaine de clones, nous aurons à la fin de cette génération 40 clones de type 1 et 30 clones pour chacune des types 2 et 3, soit un total de 100 clones.

Les cellules B sont en compétition pour se lier avec l'antigène, ce qui fait que les cellules B qui ont une affinité réduite n'auront pas la possibilité de produire autant de clones qu'elles le désire. L'ancienne formule ne tient absolument pas compte de cette compétition, alors que la nouvelle formule si.

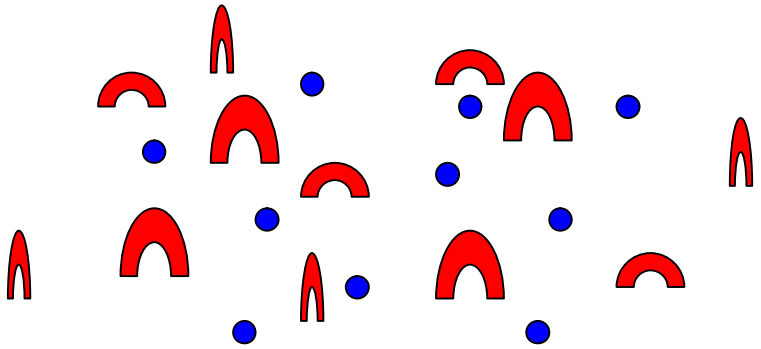

Figure 3-15 Antigènes et cellules B

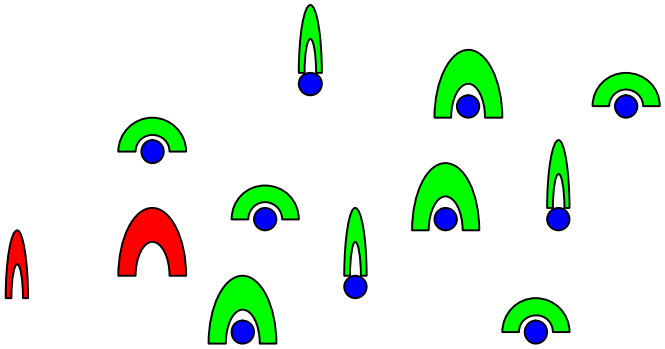

Figure 3-16 Cellules B liées aux antigènes

3.2.3 Performances obtenues :

- Pour le test 3.1, FCC atteint les 94% de classifications correctes sans utiliser de seuil, contre 90% pour CLONCLAS. Ceci représente une amélioration de 4%. Ces performances sont atteintes en 50 générations. Remarquez qu'à la 100ème génération les performances baissent à 92%. Une explication de cette baisse de performances est donnée plus loin (**figure 3.17, 3.18**).

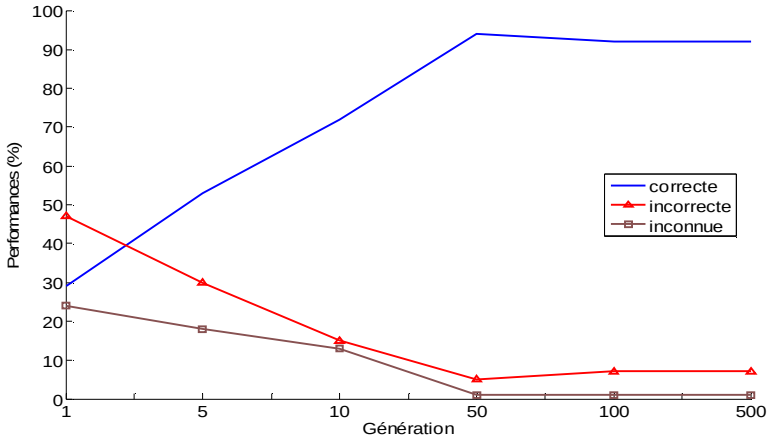

Figure 3-17 Performances/génération (test 3.1)

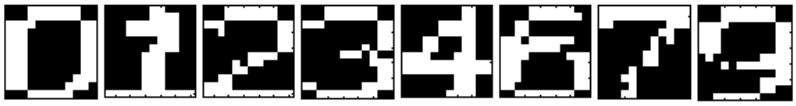

Figure 3-18 Cellules mémoire (test 3.1)

- Pour le test 3.2, FCC atteint ses meilleures performances, 94%, à la 50 génération. Ce qui représente 6% de plus que les performances de CLONCLAS. Mais comme pour le test précédant, les performances baissent à 92% à la 100ème. Ceci est probablement dû à un sur apprentissage (**figure 3.19, 3.20**).

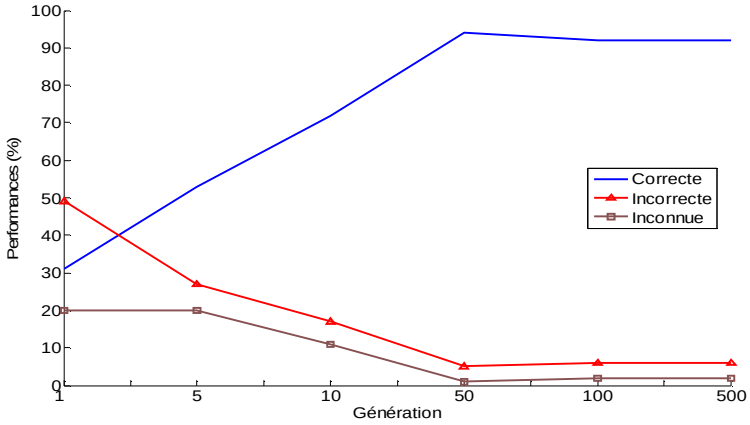

Figure 3-19 Performances/génération (test 3.2)

Figure 3-20 Cellules mémoire (test 3.2)

- Pour le test 4.1, FCC atteint les 43% de classifications correctes en seulement 50 générations. CLONCLAS quand à lui, ne dépasse pas les 41% en 500 générations. Les performances restent les mêmes pour les grandes générations (**figure 3.21, 3,22**).

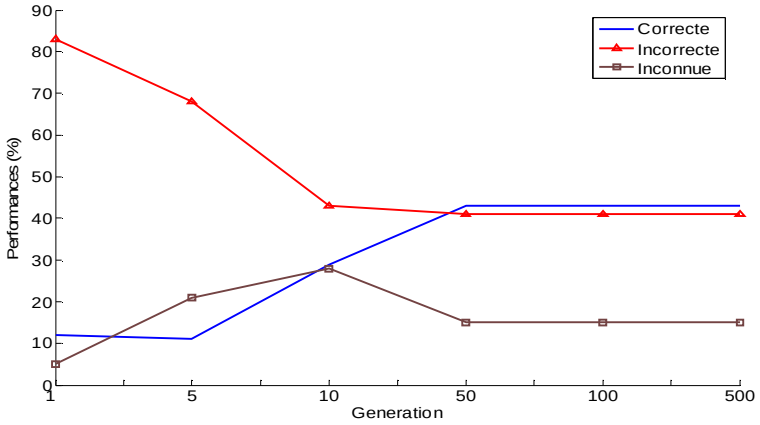

Figure 3-21 Performances/génération (test 4.1)

Figure 3-22 Cellules mémoire (test 4.1)

Remarque :

Pour les tests 3.1 et 3.2, bien que l'affinité des cellules mémoire trouvées augmente à la 100ème génération, les performances du système quand à elles baissent. Pour le test 3.2, comme les tests se font sur un ensemble différent de l'ensemble d'entraînement, la baisse de performances est probablement due à un sur apprentissage. Les cellules mémoire trouvées ressemblent 'trop' aux exemples d'entraînement et elles n'arrivent plus à distinguer les nouvelles formes. Mais pour le test 3.1, on ne peut parler de sur apprentissage, puisque les tests se font sur le même ensemble d'entraînement. La réponse est à chercher ailleurs.

Une explication possible est comme suit : la fonction affinité renvoie, pour une cellule mémoire, la somme des distances de Hamming par rapport à tous les exemples d'entraînement de la classe de la cellule. Comme la distance de Hamming est une mesure de dis similarité, maximiser l'affinité d'une cellule mémoire revient à minimiser la somme des distances entre les compléments bit à bit de la cellule mémoire et tous les exemples

d'entraînement de la classe. Donc, la sélection clonale cherche la cellule mémoire qui a son complément qui ressemble le plus à tous les exemples d'entraînements de la classe. Lors de la classification, chaque cellule mémoire va classer à elle tous les exemples pour lesquels elle a la plus grande affinité par rapport à toutes les autres cellules mémoire, ou en terme de complément, chaque complément classe à lui tous les exemples qui sont les plus proches de lui par rapport aux autres compléments. La baisse de performance pour le test 3.1 signifie que les exemples d'entraînement de ce test, sont éloignés les uns des autres, donc même si on trouve bien le complément qui soit le plus proche possible de tous les exemples de sa classe, ils sont tellement éloignés les uns des autres qu'il ne peut les classer correctement à lui.

3.3 Optimisation du calcul de l'affinité :

3.3.1 Problématique :

Comme nous l'avons déjà vu, la complexité de CLONCLAS dépend essentiellement du nombre d'évaluations de l'affinité entre les anticorps et l'antigène. Pour réduire la complexité du système et accélérer du même coup sa vitesse d'exécution, nous devons soit réduire le nombre d'évaluations effectuées par génération, soit accélérer la fonction de calcul de l'affinité.

Dans la section précédente nous avons proposé une nouvelle formule qui réduit la taille de la population de clones pour chaque génération, ce qui permet de réduire le nombre d'évaluations de l'affinité. Cependant, le système prend encore beaucoup de temps lors de la phase d'apprentissage, par rapport à la taille de l'ensemble d'entraînement (80 exemples). En sachant que pour les applications réelles, l'ensemble d'entraînement atteint facilement les 20000 exemples. Augmenter la vitesse de la phase d'apprentissage est une condition nécessaire pour pouvoir utiliser CLONCLAS dans des applications réelles.

3.3.2 Une nouvelle fonction d'affinité :

Considérons à nouveau la fonction d'affinité entre un anticorps *Ab* de longueur ***len*** bits, et un antigène *Ag* qui contient ***n*** exemples :

$$\sum_{j=1}^{n}\sum_{i=1}^{len} Ab_i \oplus Ag_i^j \qquad (3.8)$$

Où Ab_i est le i^{ème} bit de Ab, Ag_i^j est le i^{ème} dit du j^{ème} exemple de Ag.

Pour le i[ème] bit de Ab, son affinité est comme suit :

$$\sum_{j=1}^{n} Ab_i \oplus Ag_i^j \tag{3.9}$$

Si pour ce bit, nous avons n_i^1 exemples de Ag qui ont ce bit à 1, et n_i^0 exemples ont ce bit à 0 avec $n = n_i^1 + n_i^0$. Nous pouvons réécrire l'équation (2) comme suit :

$$n_i^1 \times (Ab_i \oplus 1) + n_i^0 \times (Ab_i \oplus 0) \tag{3.10}$$

Comme $Ab_i \oplus 1 = \neg Ab_i$, et $Ab_i \oplus 0 = Ab_i$, l'équation (3) s'écrit :

$$n_i^1 \times (\neg Ab_i) + n_i^0 \times (Ab_i) \tag{3.11}$$

La formule de l'affinité s'écrit alors :

$$\sum_{i=1}^{len} n_i^1 \times (\neg Ab_i) + n_i^0 \times (Ab_i) \tag{3.12}$$

Les valeurs n_i^1 et n_i^0 sont des constantes qui dépendent uniquement de l'ensemble d'entraînement utilisé. Elles peuvent donc être pré-calculées une fois par ensemble d'entraînement.

3.3.3 Performances obtenues :

La nouvelle fonction affinité réduit considérablement la complexité de l'apprentissage, puisqu'il ne dépend plus de la taille de l'ensemble d'entraînement. Donc utiliser un ensemble d'entraînement de 80 exemples prend le même temps que pour un ensemble de 20000 exemples ! Mis à part bien sûre la phase de pré calcul des constantes n_i^1 et n_i^0, mais comme elle se fait une seule fois, ça ne devrait pas poser de problèmes.

Un autre avantage de la nouvelle formule est la réduction de l'espace mémoire utilisé par CLONCLAS lors de son apprentissage. Avec l'ancienne formule, pendant toute la durée de l'apprentissage, tous les exemples d'entraînements doivent être chargés en mémoire, soit un coût de

$$n \times len \text{ bits} \tag{3.13}$$

Avec la nouvelle formule, nous n'avons besoin de stocker que les constantes, soit un coût de

$$2 \times ceil(\log_2(n)) \times len \text{ bits} \quad (3.14)$$

La **figure 3.23** montre la relation entre n et les coûts mémoires des deux formules, pour une longueur d'exemples ***len*** constante.

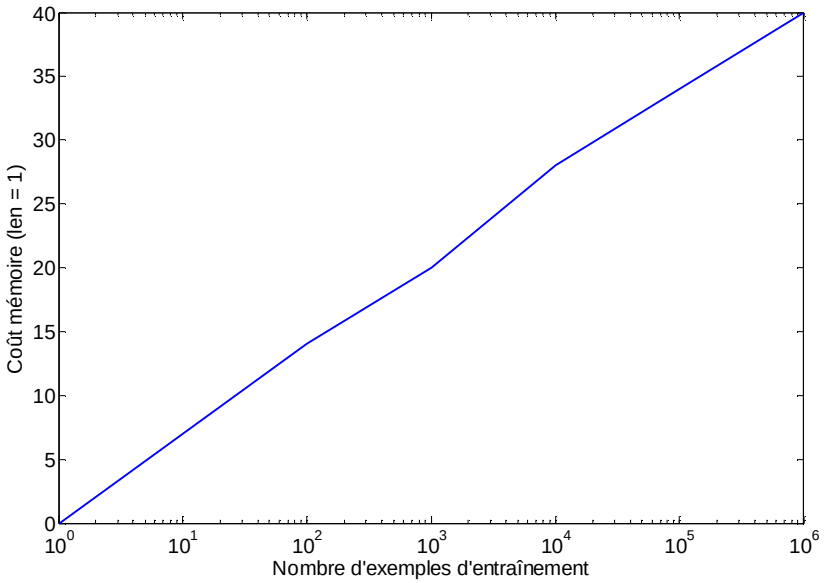

Figure 3-23 Coût mémoire/ taille de l'ensemble d'entraînement

Il faut cependant préciser, qu'avec les capacités de stockage des machines actuelles. Ce n'est que pour des ensembles d'entraînements vraiment grands que les différences de performances engendrés par la réduction des coûts mémoires se feront sentir.

3.3.4 Complexité de FCC

La complexité de FCC se calcule de manière analogue à celle de CLONCLAS. Il faut cependant tenir compte des modifications apportées aux formules de calcul du nombre de clones et de calcul de l'affinité des anticorps. La nouvelle formule du nombre de clones rend **Nc** (nombre de clones générés à chaque génération) constant et égal au paramètre β. La nouvelle fonction affinité rend la complexité de l'apprentissage complètement indépendante du nombre d'exemples d'entraînement. La complexité de FCC est donc

$$O\left(G \times (N + b) \times M^2 \times L\right) \quad (3.15)$$

3.4 Réduction du nombre d'attributs :

3.4.1 Problématique :

Dans notre système de reconnaissance, les attributs des formes sont les pixels. Ces attributs influencent la vitesse d'exécution du système, puisque la complexité de la fonction affinité dépend directement du nombre d'attributs utilisés. Mais en plus, les attributs influencent la qualité de la classification. Considérez l'exemple de la **figure 3.6**, il est évident qu'une partie des pixels des images (ceux du centre) peuvent être ignorés par les anticorps sans altérer les performances du système. En plus, le fait d'ignorer ces pixels peut même aider la cellule mémoire de la classe 0 à mieux reconnaître les chiffres '0'.

Le problème est de trouver l'ensemble minimal d'attributs qui améliore les performances du système, ou du moins ne les dégrade pas. Nous avons donc un problème à deux objectifs : 1) minimisation du nombre d'attributs sélectionnés, **2)** maximisation du taux de classifications correctes.

Lorsqu'un problème a plusieurs objectifs qui doivent tous être optimisés en même temps, on dit que c'est un problème multi objectifs. Dans la plupart des cas, les objectifs sont en conflits les uns avec les autres, c'est-à-dire que si on minimise un des objectifs, un autre objectif va augmenter. Généralement on ne peut pas trouver de solution optimale pour tous les objectifs en même temps. Pour ce type de problèmes on utilise la notion d'*optimalité Pareto*, proposée originellement par Francis Vsido EdgeWorth puis généralisée par Vilfredo Pareto **[Pareto, 96]**.

Soit $\Omega \subseteq R^n$ un ensemble de solutions possibles, et $\vec{f} : \Omega \to R^k$ qui renvoi pour chaque $\vec{x}_i \in \Omega$ la valeur $\vec{f}(\vec{x}_i) = [f_1(\vec{x}_i), f_2(\vec{x}_i), ..., f_k(\vec{x}_i)]$ tel que $f_j(\vec{x}_i)$ est la valeur de la j^{ème} fonction objective qui correspond à la solution $\vec{x}_i$.

On dit que $\vec{x}^*$ est *Pareto optimal* si pour chaque $\vec{x}_i \in \Omega$

$$f_i(\vec{x}) = f_i(\vec{x}^*) \quad \forall i \in \{1,2,...,k\} \tag{3.16}$$

ou bien il existe au moins un $i \in \{1,2,...,k\}$ tel que (pour un problème de minimisation)

$$f_i(\vec{x}) > f_i(\vec{x}^*) \tag{3.17}$$

L'ensemble $\boldsymbol{P}^*$ de toutes les solutions Pareto optimales est dit *ensemble Pareto optimal*. Le *front Pareto* $\boldsymbol{PF}^*$ est défini comme suit :

$$PF^* = \{\vec{y} = \vec{f}(\vec{x}) / \vec{x} \in P^*\} \tag{3.18}$$

On dit qu'un vecteur $\vec{u} = \{u_1, u_2, ..., u_k\}$ *domine* un vecteur $\vec{v} = \{v_1, v_2, ..., v_k\}$ si et seulement si :

$$\forall i \in \{1,2,...,k\} u_i \leq v_i \wedge \exists i \in \{1,2,...,k\} u_i < v_i \quad (3.19)$$

3.4.2 Description de MISA :

MISA (pour Multiobjective Immune System Algorithm) est un algorithme de sélection clonale dédié à l'optimisation multi objective **[Coello et Cortez, 05]**. C'est un algorithme générique qui peut être appliqué à tout problème multi objectifs à conditions de définir proprement les fonctions objectives.

Chaque anticorps représente une solution possible du problème à résoudre. Le but du système étant d'optimiser les fonctions objectives, il n'y a donc pas d'antigènes explicites. La fonction affinité, quand à elle, est calculée de manière à avantager les anticorps non dominés. Bien que la fonction affinité n'a pas été définie explicitement dans **[Coello et Cortez, 05]**, nous pouvons supposer qu'elle renvoi pour chaque anticorps le nombre d'anticorps qui le dominent, dans ce cas plus l'affinité est petite meilleure est la solution. Une affinité nulle signifie que l'anticorps n'est dominé par aucun autre anticorps, il fait donc partie du front Pareto.

Dans MISA il n'y a pas de cellules mémoire, mais plutôt une population secondaire. Cette population contient tous les anticorps non dominés trouvés par l'algorithme, ce qui représente une estimation du front Pareto du problème à résoudre. Pour maintenir une distribution uniforme des solutions trouvées, les auteurs ont utilisé un mécanisme de gestion des ressources basé sur une grille adaptative. Le principe est de diviser l'espace des solutions trouvées par une grille et de limiter le nombre maximal d'anticorps présents dans chaque case, plus de détails sont donnés dans **[Coello et Cortez, 05]**.

Pour des raisons de clarté, nous ne donnerons ici que la version de MISA pour les problèmes sans contraintes, pour la version plus générale consulter **[Coello et Cortez, 05]** :

- **Initialisation :**
  - La population initiale d'anticorps est générée aléatoirement ;
  - La population secondaire est vide.
- **Evaluation et sélection 1 :**
  - Calculer pour chaque individu de la population son affinité ;
  - Sélectionner tous les individus non dominés, si leur nombre est inférieur à 5% de la taille de la population d'anticorps compléter par les individus les moins dominés possibles.

- **Mémorisation :**
    - Copier les individus non dominés dans la population secondaire.
- **Clonage :**
    - Cloner les individus sélectionnés. Pour chaque individu le nombre de clones dépend du voisinage de cet individu, plus il a de voisins proches moins il a de clones (pour améliorer la distribution des solutions sur tout le front Pareto). En plus, si l'individu n'a pas été autorisé à entrer dans la population secondaire (soit parce qu'il y est déjà ou bien parce qu'il est dominé par les éléments de la population secondaire) il n'est pas cloné.
- **Mutation :**
    - Muter les individus clonés proportionnellement à leur affinité ;
    - Muter tous les individus les moins meilleurs en utilisant un opérateur de mutation non uniforme.
- **Evaluation et sélection 2 :**
    - Tronquer la population à sa taille d'origine en utilisant le même mécanisme de sélection utilisé dans 'Evaluation et sélection 1' ;
    - Bien que cela n'a pas été précisé dans l'article qui décrit MISA, la sélection se fait sur tous les anticorps y compris ceux de la population secondaire.
- **Mort :**
    - Les individus non retenus dans les populations primaires et secondaires sont éliminés.

MISA est un algorithme d'optimisation multi objectifs générique, nous pouvons donc l'utiliser pour résoudre notre problème à condition de définir les éléments suivants :

- Adapter les anticorps pour qu'ils reflètent des solutions à notre problème ;
- Spécifier les fonctions objectives à optimiser.

Un anticorps représente de façon générale, une solution possible au problème à résoudre. Dans notre cas un anticorps représente un ensemble d'attributs sélectionnés parmi tous les attributs de la forme. Nous allons reprendre la même représentation des anticorps utilisée dans CLONCLAS, à la différence que chaque bit de l'anticorps indique l'état de l'attribut correspondant de la forme, 1 = sélectionné et 0 = non sélectionné.

Pour les fonctions objectives, la première est évidemment le nombre d'attributs sélectionnés. La deuxième fonction objective va définir le type de sélection d'attributs opérée :

- Utiliser une approche enveloppante signifie que pour chaque anticorps, nous allons lancer FCC pour obtenir les meilleures cellules mémoire pour les attributs

sélectionnés, la valeur de la fonction objective étant les performances obtenues par le système en utilisant ces cellules mémoire ;

- Utiliser une approche filtrante consiste à calculer une certaine mesure qui va nous permettre d'estimer la qualité des attributs sélectionnés sans passer par FCC. Dans notre cas la mesure utilisée est le DB Index (voir chapitre 1). Il est à noter que cette mesure a déjà été utilisée pour un problème similaire mais avec un autre algorithme d'optimisation multi objective **[Marita et al, 2002]**.

3.4.3 Performances obtenues :

Après exécution de MISA nous obtenons un ensemble de solutions non dominées (le front Pareto), un exemple est donné à la **figure 3.24**. Chaque solution représente un compromis entre les deux objectifs à optimiser : le nombre d'attribut et les performances du système. Le choix de la solution à utiliser se fait soit manuellement par l'utilisateur qui choisit la solution la mieux adaptée à ses besoins, ou bien de façon automatisée en utilisant un critère de sélection. Nous avons opté pour un choix automatique en utilisant comme critère les performances obtenues par chaque solution du front Pareto pour un ensemble de formes différent de l'ensemble d'entraînement.

Vu la nature statistique de MISA, les résultats affichés dans cette section sont la moyenne de dix exécutions de l'algorithme de sélection d'attributs. Les **figures 3.25, 3.26** affichent les moyennes des performances obtenues (par chaque approche) pour les solutions choisies, selon le critère de sélection expliqué précédemment, pour les générations 1, 5, 10, 50, 100, 500. Les **figures 3.27, 3.28** montrent quand à elle le nombre d'attributs sélectionnés pour chaque approche.

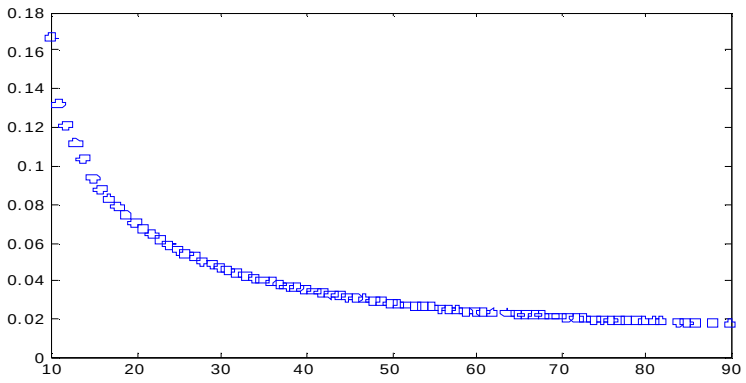

Figure 3-24 Exemple de front Pareto

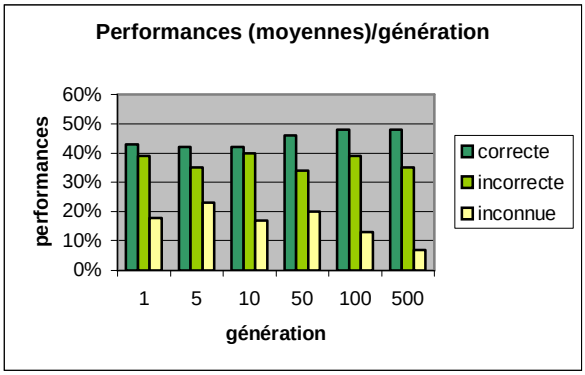

Figure 3-25 Performances/génération (approche enveloppante)

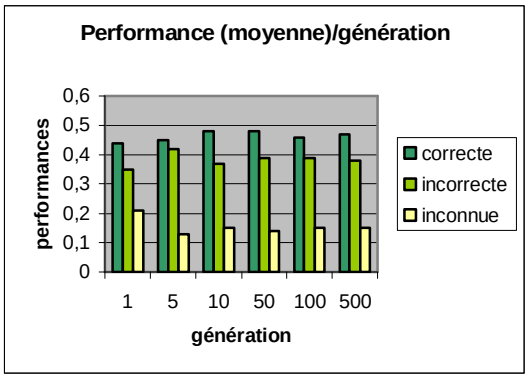

Figure 3-26 Performances/génération (approche filtrante)

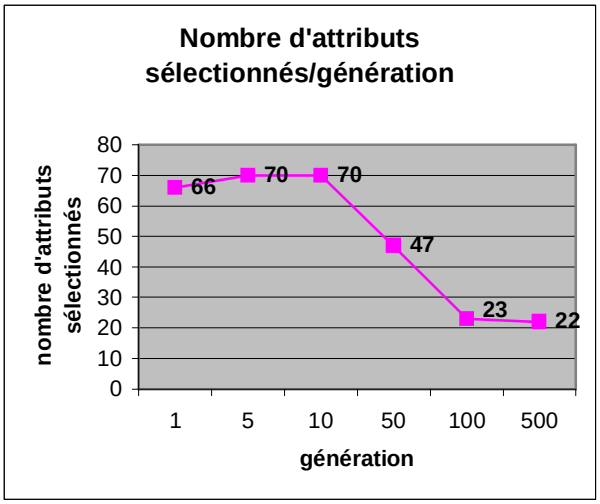

Figure 3-27 Nombre d'attributs sélectionnés/génération (approche enveloppante)

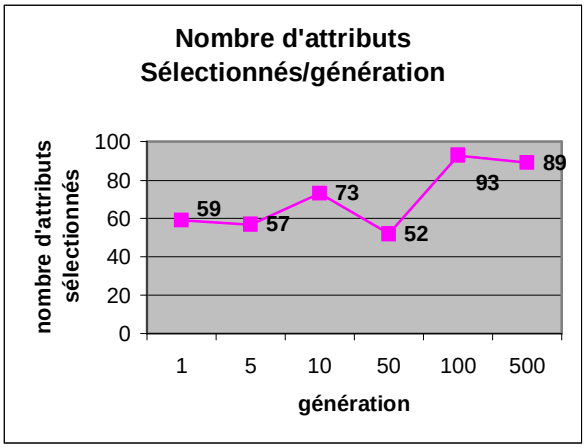

Figure 3-28 Nombre d'attributs sélectionnés/génération (approche filtrante)

La sélection d'attribut réduit effectivement le nombre d'attributs utilisés, qui passent de 144 attributs (pixels) à 89 attributs pour l'approche filtrante soit 40% de réduction, et à 22 attributs pour l'approche enveloppante soit 85% de réduction ! Cette réduction du nombre d'attributs ne dégrade en aucun cas les performances du système, qui au contraire semblent s'améliorer pour atteindre les 47% de bonnes classifications en utilisant les attributs trouvés par l'approche filtrante et 48% pour ceux trouvés par l'approche enveloppante. En sachant que FCC ne dépasse pas les 43% de classifications correctes. Cette amélioration confirme que la réduction des attributs utilisés a permis d'augmenter les capacités du système à distinguer entre les formes des différentes classes.

Comparons maintenant entre les deux approches utilisées. En ce qui concerne l'amélioration des performances du système de reconnaissance des formes, les deux approches se valent avec des performances de 47-48%. Pour ce qui est de la réduction du nombre d'attributs utilisés, l'approche enveloppante est la meilleure avec une réduction de prés de 85% contre 40% pour l'approche enveloppante. Enfin pour les temps d'exécution, c'est l'approche filtrante qui est la plus rapide puisqu'elle ne fait pas appel à l'algorithme d'apprentissage.

En résumé, si le but principal est l'amélioration des performances alors c'est la méthode filtrante qui doit être utilisée puisqu'elle est la plus rapide. Mais si la priorité est à la réduction du nombre d'attributs alors c'est la méthode enveloppante qui est recommandée malgré ses temps d'exécution élevés.

3.5 Conclusion :

CLONCLAS (pour CLONal CLASsification) est un algorithme de sélection clonale artificielle, dédié à l'apprentissage dans un système de reconnaissance de chiffres imprimés basé sur les prototypes. Pour permettre l'apprentissage. CLONCLAS utilise les propriétés de la sélection clonale pour optimiser l'affinité des cellules mémoire (qui représentent les prototypes) face aux antigènes (qui représentent les classes à apprendre). Les résultats obtenus pour les formes normalisées sont très prometteurs. Malgré cela, CLONCLAS souffre d'une complexité élevée.

FCC (pour Fast Clonal Classification) est un algorithme d'apprentissage qui traite les problèmes de complexité de CLONCLAS. Ce nouvel algorithme est une combinaison de plusieurs modifications de l'algorithme original CLONCLAS. Nous avons commencé par proposer une nouvelle formule pour le calcul du nombre de clones. Non seulement cette formule réduit le nombre de clones produits, mais en plus elle est plus proche du modèle réel des systèmes immunitaires.

Nous avons ensuite reformulé la fonction affinité, qui au prix d'une étape de pré–calcul au début de l'entraînement, permet de rendre la complexité de l'apprentissage indépendante de la taille de l'ensemble d'entraînement. Ceci facilite l'apprentissage sur de grands ensembles d'entraînement. En plus, la nouvelle formule d'affinité permet de réduire les besoins en place mémoire de l'algorithme d'apprentissage.

Enfin, pour réduire encore plus la complexité de l'apprentissage et améliorer les performances de la classification, nous avons ajouté une phase de sélection d'attributs. Cette phase utilise MISA, un algorithme de sélection clonale dédié à la résolution de problèmes multi–objectifs, pour minimiser le nombre d'attributs sélectionnés (objectif 1) tout en maximisant le taux de classification correctes du système (objectif 2). Nous avons implémenté deux approches pour la sélection d'attributs, l'une filtrante et l'autre enveloppante. Une comparaison entre les deux approches de sélection a montrée que l'approche filtrante est la plus rapide et l'approche enveloppante est celle qui réduit le plus le nombre d'attributs sélectionnés.

4 Clustering et classification clonale :

4.1 Introduction :

4.1.1 Problématique :

CLONCLAS est basé sur une approche par prototypes, chaque prototype étant une image binaire. Il est naturellement pénalisé par les limitations de tels systèmes : les modèles engendrés sont sensibles aux variations géométriques et photométriques. En plus, à cause de la fonction affinité utilisée, les modèles trouvés sont une sorte de moyenne des exemples d'entraînement de leurs classes respectives (voir **figure 4.1, 4.2**).

Figure 4-1 Exemples d'entraînement normalisés

Figure 4-2 Cellule mémoire obtenue

Cette fonction affinité montre ses limites pour le test 4.1. Comme les exemples d'entraînement ne sont pas normalisés, les cellules mémoire trouvées n'arrivent pas à capturer la forme générale des chiffres (voir **figure 4.3, 4.4**).

Figure 4-3 Exemples d'entraînement pour le chiffre '0' du test 4.1

Figure 4-4 Cellule mémoire du chiffre '0' du test 4.1

Il devient évident qu'avec de telles cellules mémoire, le système aura de faibles chances de reconnaître correctement les formes inconnues. Les résultats obtenus pour le test 4.1 confirment ce fait.

4.1.2 Solution proposée :

Plusieurs solutions sont envisageables. Par exemple, on pourrait modifier la représentation des exemples et utiliser des attributs, autres que les pixels, moins sensibles aux variations. On peut aussi modifier la fonction affinité pour tenir compte des variations. L'objectif de ce travail étant l'utilisation de méthodes bio inspirées, pourquoi ne pas chercher la solution dans les systèmes naturels eux-mêmes ?

Dans les systèmes immunitaires naturels, un antigène a plusieurs épitopes **[Roitt, 90]**. Donc, lors de la sélection clonale, le système immunitaire va produire autant de sortes de cellules mémoire que l'antigène a d'épitopes. Utiliser plusieurs cellules mémoire pour chaque antigène semble être une bonne voie. Même si les exemples d'entraînements sont trop variés, là où une cellule mémoire unique ne pouvait capturer la forme générale des exemples, plusieurs cellules mémoire vont pouvoir se partager la tâche.

Pour mettre en œuvre cette solution dans notre système, plusieurs alternatives se présentent à nous. Premièrement, nous pouvons diviser les formes d'entraînement en régions (voir **figure 4.5**). Chaque région représente un épitope de l'antigène. Cette solution nécessite cependant des modifications profondes dans le système puisque la représentation même des anticorps et des antigènes doit être modifiée.

Figure 4-5 Antigène divisé en régions

Une deuxième alternative serait de diviser l'ensemble d'entraînement de chaque classe en sous ensembles, tels que chaque sous ensemble contienne un certain type de variations des exemples (voir **figure 4.6**). Un antigène étant un ensemble d'exemples d'entraînement d'une certaine classe, partitionner cet ensemble revient à doter l'antigène d'autant d'épitopes qu'il y a de partitions. Cette solution présente l'avantage de ne pas modifier la représentation des anticorps, ni le système d'apprentissage. La seule différence est qu'il peut y avoir plusieurs antigènes pour une même classe, l'antigène signifiant maintenant **épitope**. Le partitionnement des antigènes se fait une seule fois et de manière off line pour chaque ensemble d'entraînement, puis le système continue normalement.

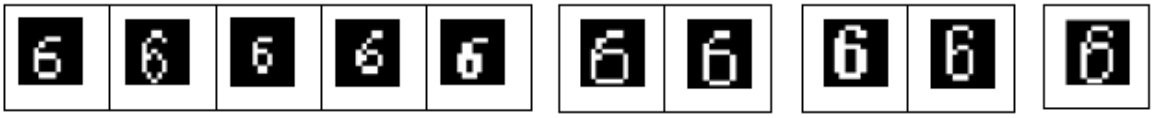

Figure 4-6 Sous classes du chiffre 6

4.2 Proposition 1 : partitionnement des classes par seuil manuel

4.2.1 Description de la proposition :

Pour chaque antigène (classe) nous allons appliquer l'algorithme de clustering par agglomération. Les partitions obtenues peuvent être organisées sous la forme d'un arbre (dendrogramme). Chaque feuille de l'arbre représente un exemple d'entraînement, la longueur d'une branche qui relie un nœud à une feuille est égale à la dissimilarité entre cette feuille et ce nœud. La mesure de dissimilarité utilisée est la même que celle utilisée lors de la classification. Un exemple de dendrogramme est donné à la **figure 4.7**.

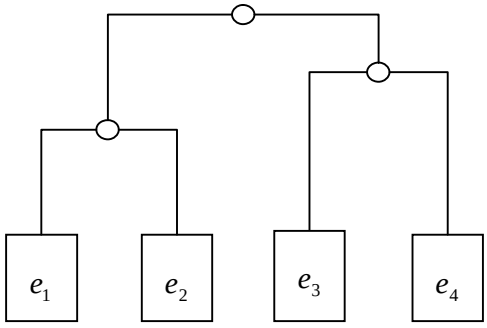

Figure 4-7 Exemple de dendrogramme

Pour obtenir les partitions nous devons appliquer un seuil, fournit par l'utilisateur, à l'arbre. Toutes les branches dont la longueur dépasse le seuil sont supprimées (**figure 4.8**). Les nœuds restants vont représenter les partitions de la classe.

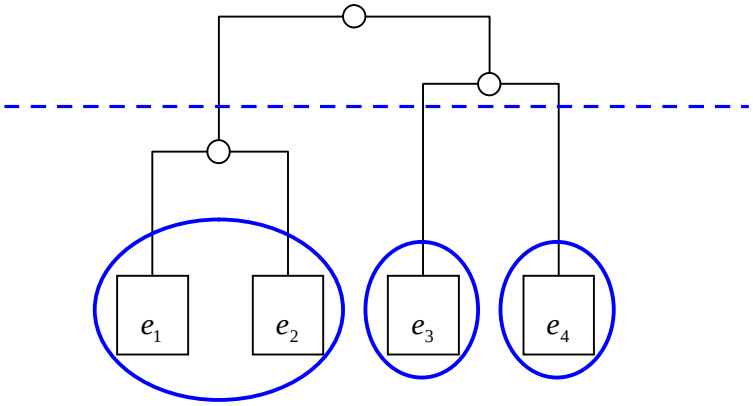

Figure 4-8 Partitionnement en utilisant un seuil

La procédure d'entraînement se déroule comme pour FCC, à la différence que le système va maintenant produire une cellule mémoire par partition. La procédure de classification est la même.

4.2.2 Résultats obtenus :

Les résultats obtenus sont comme suit :

- Pour le test 3.1, les performances du système atteignent les 99% en utilisant un seuil de partitionnement de 14%. Ce seuil génère 23 partitions pour toutes les classes, soit approximativement 3 cellules mémoire par classe (**figure 4.9, 4.10, 4.11**).

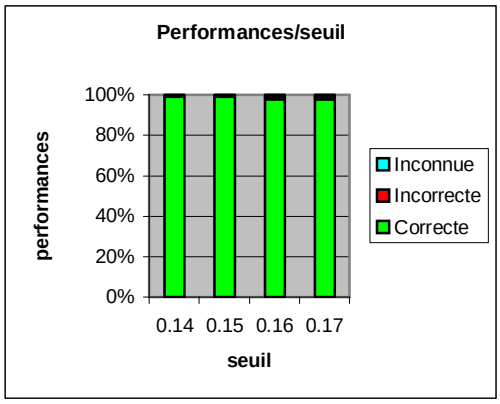

Figure 4-9 Performances/seuil (test 3.1)

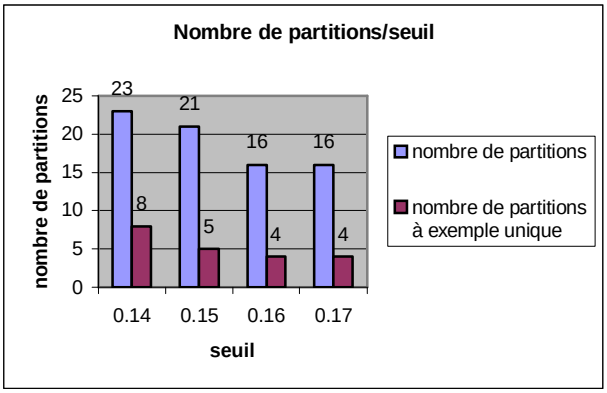

Figure 4-10 Nombre partitions/seuil (test 3.1)

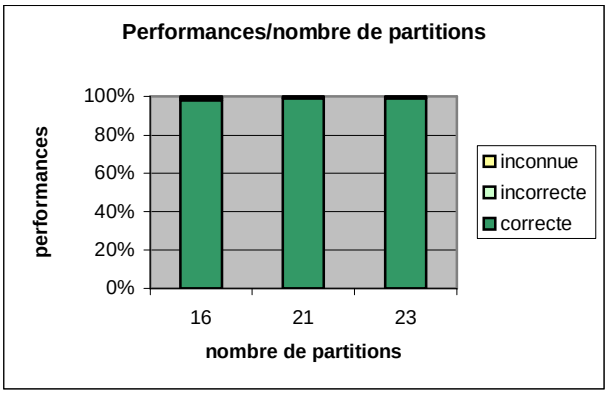

Figure 4-11 Performances/Nombre partitions (test 3.1)

- Pour le test 3.2, le système affiche, à notre grande surprise, des performances moins importantes que celles obtenues par FCC. D'ailleurs les meilleures performances sont obtenues par un seuil de 24% qui ne produit que 8 partitions, soit une cellule mémoire par classe ! ceci est un signe de sur apprentissage des exemples d'entraînement (**figure 4.12, 4.13, 4,14**).

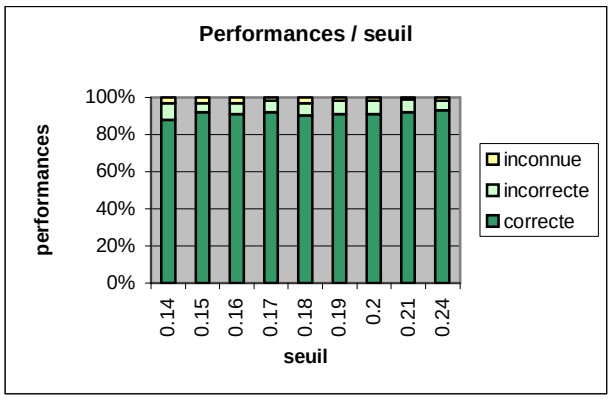

Figure 4-12 Performances/seuil (test 3.2)

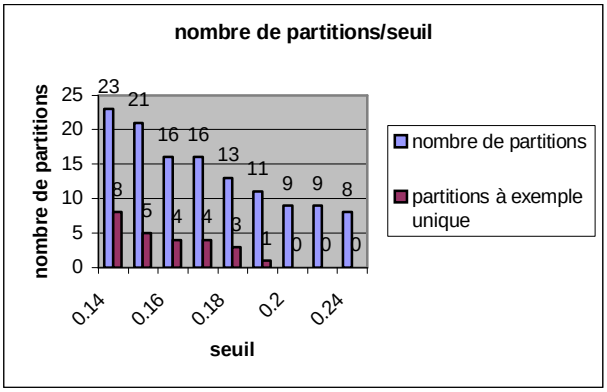

Figure 4-13 Nombre partitions/seuil (test 3.2)

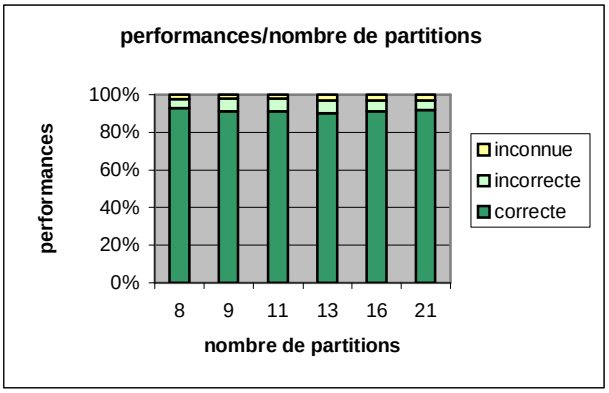

Figure 4-14 Performances/nombre partitions (test 3.2)

- Les meilleures performances du système sont obtenues pour le test 4.1. pour ce test le système atteint 60% de classifications correctes contre 43% pour FCC. Ces résultats sont obtenus avec un seuil de 8% qui génère 19 partitions, soit à peu prés 3 cellules mémoire par classe (**figure 4.15, 4.16, 4.17**).

Figure 4-15 Performances/seuil (test 4.1)

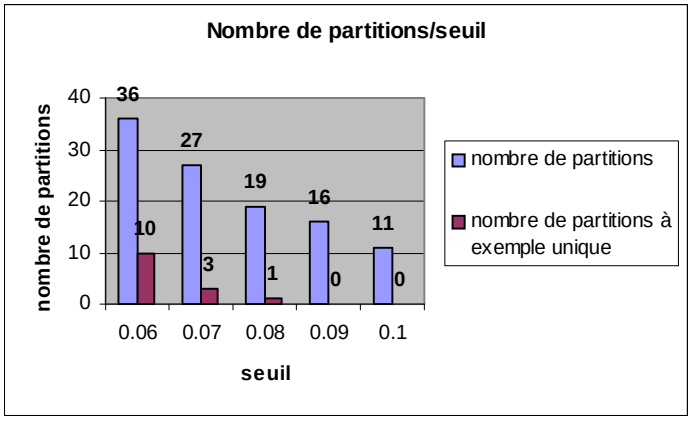

Figure 4-16 Nombre partitions/seuil (test 4.1)

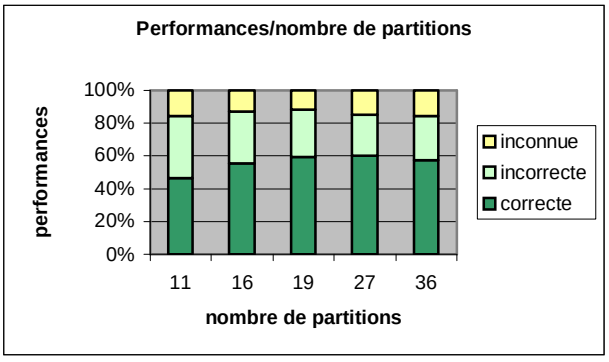

Figure 4-17 Performances/nombre partitions (test 4.1)

Comme prévu, c'est pour les exemples non normalisés (test 4.1), qui présentent le plus de variations intra classe que le système affiche les meilleures performances. En ce qui concerne les exemples normalisés (test 3.2) le partitionnement ne semble pas améliorer les performances du système, probablement à cause d'un sur apprentissage. Ce problème se posait déjà pour FCC lorsque le nombre de générations augmentait. Cependant, le principal inconvénient de cette approche est le choit manuel du seuil de partitionnement. L'utilisateur doit essayer plusieurs valeurs de seuil pour choisir le meilleur, ce qui signifie relancer l'entraînement à chaque fois, ce qui n'est pas très pratique surtout pour les grands ensembles d'entraînements.

La **figure 4.18** montre la cellule mémoire obtenue par FCC pour le chiffre '6'. Remarquez que cette cellule n'a pas réussi à capturer la forme générale des chiffres 6. La **figure 4.19** montre les cellules mémoire obtenues par C^3 pour les sous classes du chiffre '6'. Les cellules mémoire ont réussi à capturer les formes générales des principales variations du chiffre 6. Il est clair que les cellules mémoire de la **figure 4.19** ont plus de chances de classer correctement les formes que les cellules mémoire de la **figure 4.18**. Ce qui est d'ailleurs confirmé par les résultats obtenus par C^3.

Figure 4-18 Cellule mémoire du chiffre 6 (FCC)

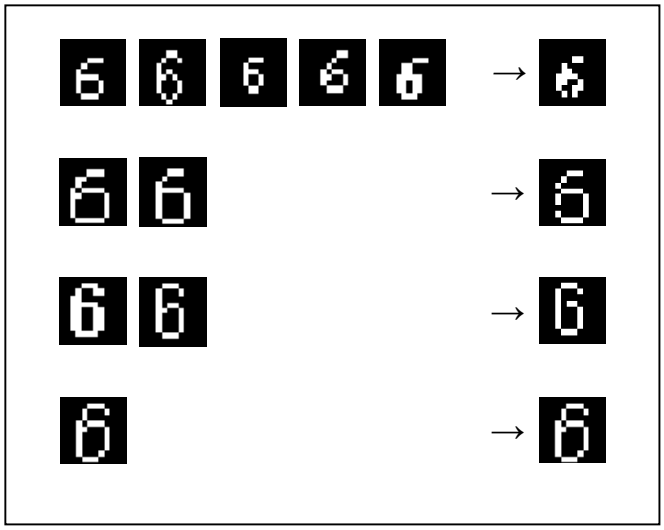

Figure 4-19 Cellules mémoire du chiffre 6 (C3)

4.2.3 Relation entre le seuil de partitionnement et le nombre de partitions :

Comme vous l'avez sans doute déjà remarqué, le nombre de partitions généré est inversement proportionnel au seuil de partitionnement, et ce quelque soit l'ensemble d'entraînement partitionné. L'explication de ce phénomène est simple : le seuil de partitionnement limite la distance maximale entre les feuilles d'une même partition. Graphiquement, comme on peut le voir sur la **figure 4.20**, plus le seuil est élevé, et moins il y a de nœuds séparés. Ces nœuds étant les représentants des cellules mémoire que l'on désire obtenir, il en résulte que plus le seuil est élevé, moins il y a de cellules mémoire (et donc de partitions) générées.

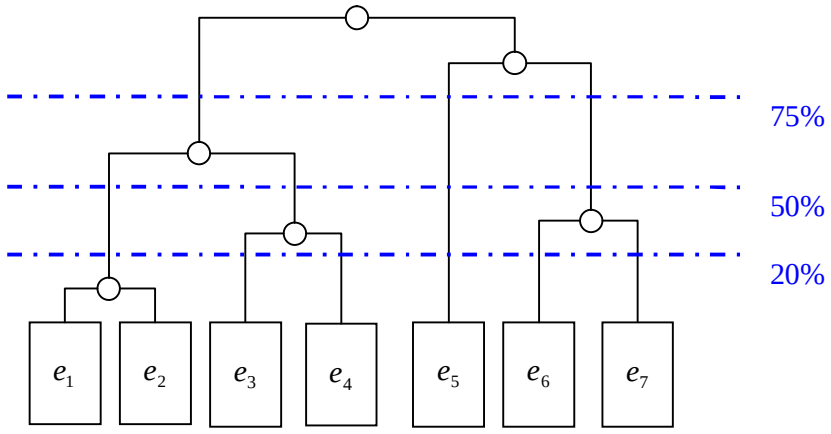

Figure 4-20 Relation entre seuil de partitionnement et nombre de partitions

4.3 Proposition 2 : partitionnement automatique

4.3.1 Description de la proposition :

Le fait de choisir manuellement le seuil de partitionnement est le principal inconvénient de la proposition précédente, rendant son automatisation difficile. Etant donné que l'on connaît déjà les classes des exemples à partitionner, n'est il pas possible de déduire la partition directement sans utiliser de seuil?

Observons de plus prés l'arbre généré par l'algorithme de clustering. Une des caractéristiques de cet arbre est que pour chaque nœud ses feuilles sont plus proches les unes des autres que n'importe quelle autre feuille de l'arbre. Par exemple, si nous avons deux classes de quatre exemples chacune, dans le cas idéal les exemples de même classe sont plus proches les uns des autres que n'importe quel autre exemple de l'autre classe. On devrait alors obtenir après application du clustering un arbre qui ressemble à celui de la **figure 4.21**.

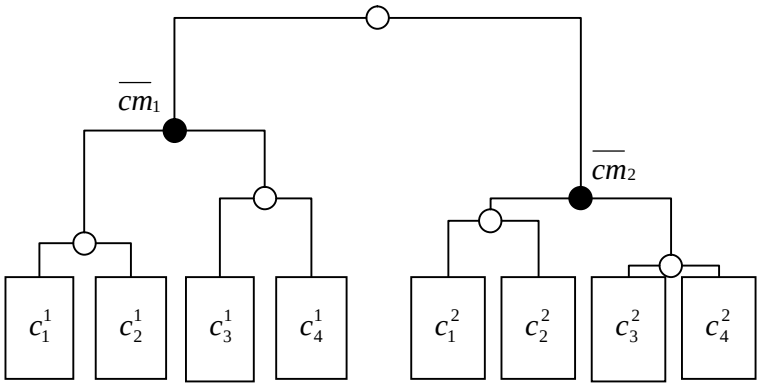

Figure 4-21 Dendrogramme idéal

Où c_i^j est le i^{ème} exemple de la classe $\mathbf{C_j}$.

Remarquez que **1)** pour chaque nœud, mis à part la racine, toutes ses feuilles sont de la même classe. **2)** chaque nœud $\bar{c}m_j$ est plus proche des exemples de la classe $\mathbf{C_j}$ que n'importe quel exemple de l'autre classe, ce nœud serait le représentant idéal de la classe $\mathbf{C_j}$.

Pour cet exemple, le partitionnement peut se faire juste au dessus des nœuds $\bar{c}m_j$, et ainsi obtenir deux partitions dont les exemples sont plus proches les uns des autres que n'importe quel exemple de l'autre classe.

Ce cas serait l'idéal, mais dans la pratique avoir une séparation si nette entre les différentes classes est chose rare. Pour les applications réelles nous sommes plutôt confrontés à des arbres de la forme présentée à la **figure 4.22**.

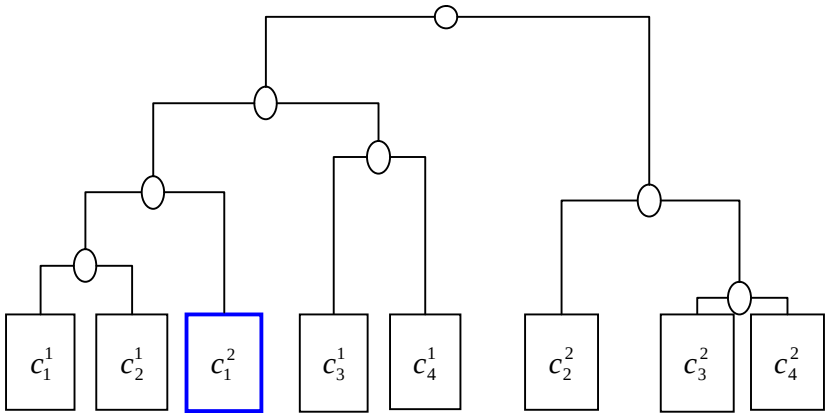

Figure 4-22 dendrogramme réaliste

Remarquez qu'un exemple de la classe $\mathbf{C_2}$ (c_1^2) c'est glissé entre les exemples de la classe $\mathbf{C_1}$. Cela veut dire que cet exemple représente une variante de la classe $\mathbf{C_2}$ qui est plus proche des exemples de la classe $\mathbf{C_1}$ que ceux de sa propre classe. Pour une application réelle

cet exemple aurait de grandes chances d'être un outkier. Mais comme nos exemples d'entraînement ont été vérifiés visuellement, cette possibilité est exclue.

Si cet exemple est utilisé lors de l'entraînement, la cellule mémoire engendrée aura un risque potentiel de générer de mauvaises classifications. Parce que si une forme inconnue de la classe **C_2** se présente, mais qu'elle ressemble beaucoup à l'exemple c_1^2, elle sera plus proche de la cellule mémoire $\bar{c}m_1$ que de $\bar{c}m_2$, et elle sera classée à tord comme une forme de la classe **C_1**.

Partant de cette constatation, et sachant que chaque nœud de l'arbre est un représentant possible de ses feuilles. Une cellule mémoire doit être un nœud qui a toutes ses feuilles de la même classe. Donc, pour l'exemple précédent les meilleures cellules mémoire sont celles de la **figure 4.23**.

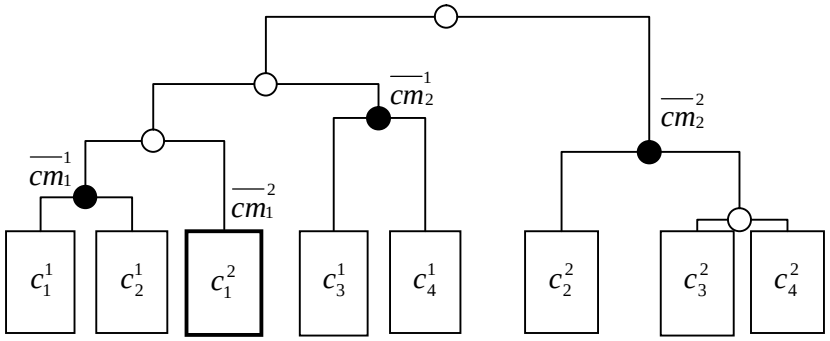

Figure 4-23 Cellules mémoire du dendrogramme réaliste

Où $\bar{c}m_j^i$ est la i^{ème} cellule mémoire de la classe ***Cj***. La partition qui permet de produire ces cellules mémoire est :

$$\{c_1^1, c_2^1\}, \{c_1^2\}, \{c_3^1, c_4^1\}, \{c_2^2, c_3^2, c_4^2\}$$

L'avantage est que nous avons partitionné les exemples d'entraînement sans utiliser de seuil, donc sans aucune intervention de l'utilisateur.

L'algorithme de cette proposition est le suivant :

- Utiliser le clustering pour construire le dendrogramme des exemples de toutes les classes ;
- Examiner chaque nœud en partant de la racine :
    - Si toutes ces feuilles (y compris celles de ses nœuds fils) appartiennent à la même classe, rassembler toute ces feuilles dans une même partition ;
    - Sinon, si ce nœud n'a que des feuilles (qui ne sont pas de la même classe), mettre chaque feuille dans une partition à part ;
    - Sinon (ce nœud a des nœuds fils), examiner ses nœuds fils.

4.3.2 Performances obtenues :

Les résultats obtenus en utilisant cette proposition sont résumés dans les figures **4.24** et **4.25**.

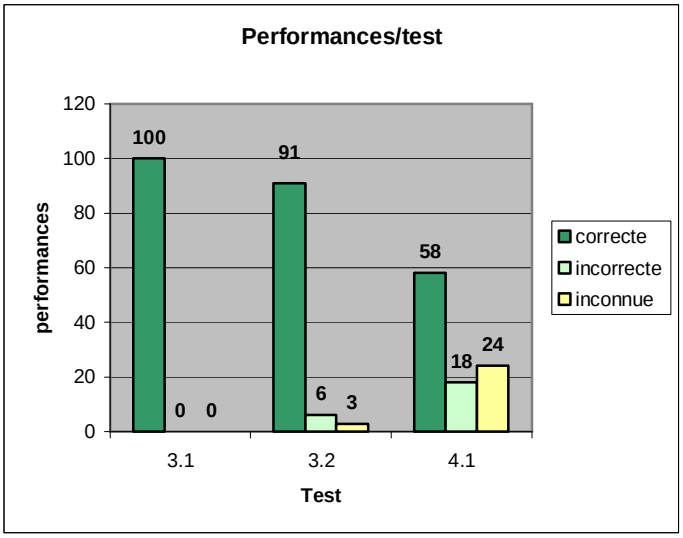

Figure 4-24 Performances/Test

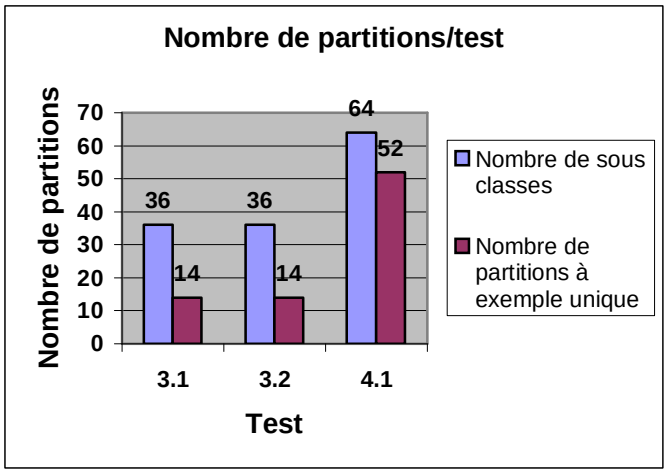

Figure 4-25 Nombre de partitions/Test

Pour le test 3.1, le système atteint tout simplement les 100% de bonnes classifications en 50 générations. Cela veut dire que : **1)** nos suppositions sur les liens entre les nœuds et les cellules mémoire sont fondées, ce qui explique les 100% de performances, parce que comme le test ce fait sur le même ensemble d'entraînement, les cellules mémoire trouvées sont les plus performantes. **2)** cela représente une preuve que FCC converge bien vers les meilleures cellules mémoire pour la partition utilisée. **3)** le nombre de sous classes obtenues est 36 ce qui montre que, comme nous l'avons supposé, il n'y a pas de séparation nette entre les exemples des différentes classes, sinon on n'aurait obtenu que huit sous classes.

Pour le test 3.2 en revanche, les résultats sont toujours peu convaincants. Le système affiche 91% de classifications correctes en 100 générations et 36 sous classes, contre 94% pour FCC en 50 générations. Une explication possible est le sur apprentissage.

Pour le test 4.1, les résultats sont de 58% de bonnes classifications en 50 générations. Bien que cela soit meilleur que les maigres 43% de FCC, le nombre de sous classes produites est trop important, 64 sous classes pour un total de 80 exemples. Une analyse plus poussée des résultats a montré que 52 sous classes ne contiennent qu'un seul exemple, soit presque 80% des classes produites. Non seulement cela n'est pas très pratique pour les ensembles d'entraînement de grandes tailles, mais en plus cela rend l'algorithme d'entraînement complètement inutile, puisque lorsque la classe n'a qu'un seul exemple, la cellule mémoire correspondante est le complément bit à bit de cet exemple.

4.4 Proposition 3 : combinaison des propositions 1 et 2

4.4.1 Description de la proposition :

L'approche précédente bien qu'elle permet d'automatiser le partitionnement, souffre néanmoins de deux limitations majeures : **1)** les partitions obtenues sont trop optimisées pour les exemples d'entraînement, ce qui conduit à un sur apprentissage. Ceci est confirmé par les 100% de performances pour le test 3.1 (parce les tests se font sur les même exemples de l'entraînement) mais non satisfaisants pour le test 3.2. **2)** le nombre de sous classes générées est trop important.

Revenons à l'exemple de la **figure 4.22**, nous avons vu que le fait d'utiliser directement FCC sur les exemples d'entraînement présentait un risque de mauvaises classifications à cause de l'exemple c_1^2. Dans l'approche précédente nous avons opté pour une élimination totale de ce risque, du moins pour les exemples d'entraînement. Les cellules mémoire obtenues (**figure 4.23**) ont toutes leurs feuilles de la même classe.

Nous allons tenter de corriger les limitations de l'approche précédente en réduisant un peu les contraintes : au lieu d'avoir toutes les cellules mémoire avec des feuilles de même classe, nous allons en avoir un maximum. Mais certaines aurons toujours des exemples étrangers dans leurs feuilles. Cela devrait réduire les problèmes liés au sur apprentissage, ainsi que le nombre de partitions trouvées.

L'algorithme général de cette approche est comme suit :

- Utiliser le clustering pour construire le dendrogramme des exemples de toutes les classes ;
- Trouver tous les nœuds qui ont leurs feuilles de la même classe ;
- Subdiviser juste au dessus du nœud le plus haut dans l'arbre (celui qui est le plus éloigné de ses feuilles).

Par exemple, pour notre exemple de la **figure 4.23**, nous obtenons le résultat montré à la **figure 4.26**.

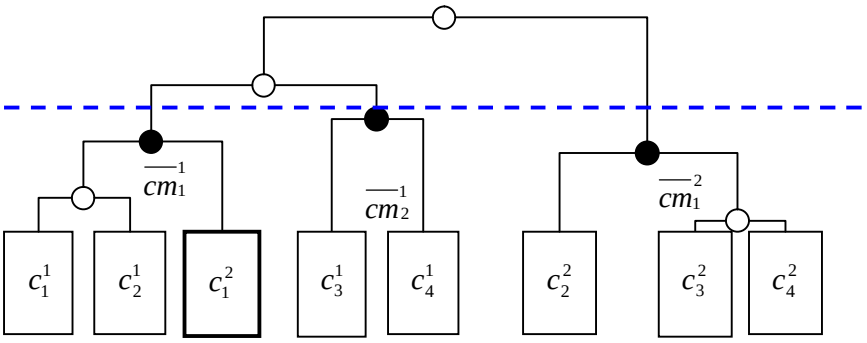

Figure 4-26 Cellules mémoire obtenues par la proposition 3

Remarquez que l'on obtient moins de partitions par rapport à la proposition précédente. Bien que c_1^2 présente un risque pour les exemples d'entraînement, ce n'est pas forcément le cas pour les exemples de test.

4.4.2 Performances obtenues :

Les résultats obtenus par cette proposition sont résumés dans les figures **4.27** et **4.28**. On remarque que les performances du système sont semblables à celles de la proposition précédente sauf pour le test 3.1 dont les performances ont un peu baissé à cause des cellules mémoire à risque produites lors de l'apprentissage. En fait, la baisse des performances pour le test 3.1 peut être un signe de réduction des effets du sur apprentissage.

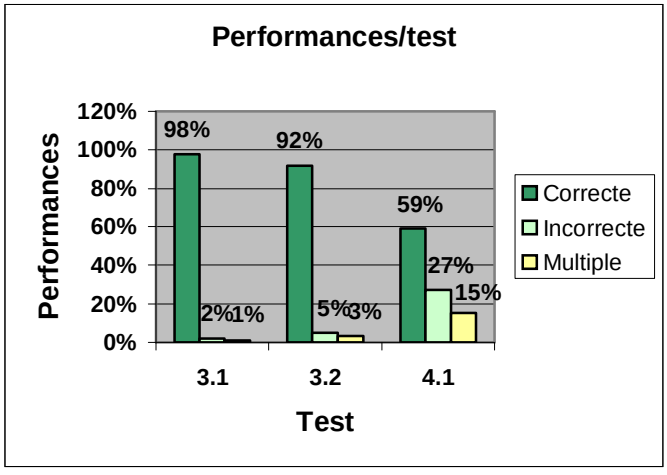

Figure 4-27 Performances/test

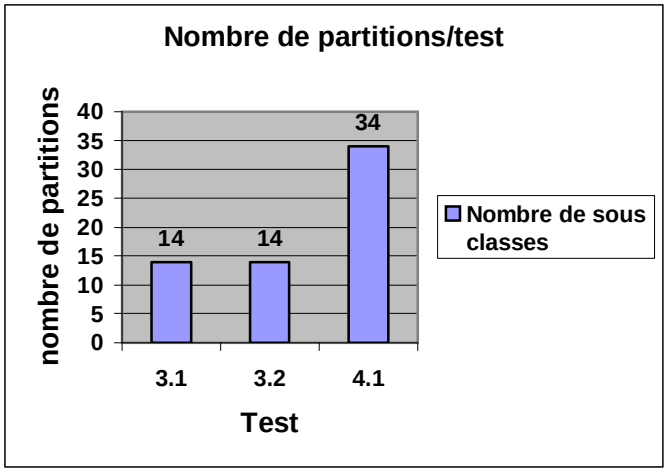

Figure 4-28 Nombre de partitions/test

En ce qui concerne le nombre de partitions générées, les résultats de cette proposition sont encourageants, puisque l'on passe de 64 partitions pour le test 4.1 à 34 partitions, soit à peu prés 4 cellules mémoire par classe. Pour les tests 3.1 et 3.2 on passe de 36 partitions à 14 soit 2 cellules mémoire par classe.

Bien que les performances n'atteignent pas celle de la 1ère proposition (seuil manuel), il n'empêche que cette proposition permet d'avoir des performances bien meilleures que celles obtenues par FCC tout seul. En plus, il n'y a aucun seuil manuel à choisir.

4.5 Proposition 4 : amélioration des performances du seuil manuel

4.5.1 Description de la proposition :

Comme nous l'avons vu, l'approche de partitionnement qui utilise un seuil manuel est celle qui donne les meilleurs résultats. Que se soit pour les performances ou bien pour le nombre de partitions trouvées. En utilisant maintenant les risques de classification des cellules mémoire, n'est il pas possible, tout en utilisant un seuil manuel, d'améliorer d'avantage les performances du système ?

Lorsque l'on applique un seuil à un dendrogramme nous allons obtenir un ensemble de partitions. Dans le cas idéal, chaque partition va contenir des exemples de la même classe. Mais comme nous l'avons vu, ceci est rarement vérifié. Certaines classes vont contenir des exemples de plusieurs classes, ce qui rend les cellules mémoire de ces partitions prédisposées à générer de mauvaises classifications.

Pour réduire ces risques, après avoir partitionné les exemples en utilisant le seuil fourni par l'utilisateur, chaque partition est subdivisée à nouveau afin de séparer les exemples de classes différentes. Pour l'exemple de la **figure 4.23**, et en utilisant le même seuil que celui de la figure **4.26** nous obtenons les partitions suivantes :

$$\left\{c_1^1, c_2^1, c_1^2, c_3^1, c_4^1\right\}, \left\{c_2^2, c_3^2, c_4^2\right\}$$

Comme la première partition contient des exemples de classes différentes elle est subdivisée de nouveau pour obtenir au final les partitions suivantes :

$$\left\{c_1^1, c_2^1, c_3^1, c_4^1\right\}, \left\{c_1^2\right\}, \left\{c_2^2, c_3^2, c_4^2\right\}$$

L'analyse de l'arbre a permis d'isoler l'exemple c_1^2 tout en gardant les exemples de la classe $\mathbf{C_1}$ dans la même partition.

4.5.2 Performances obtenues :

Les résultats obtenus par cette proposition sont les suivants :

- Pour le test 3.1, les performances sont optimales avec 100% de bonnes classifications pour quatorze sous classes produites. Ceci représente une amélioration par rapport à la **proposition 1** non seulement en ce qui concerne les performances mais aussi pour le nombre de sous classes générées (**figure 4.29, 4.30, 4.31**).

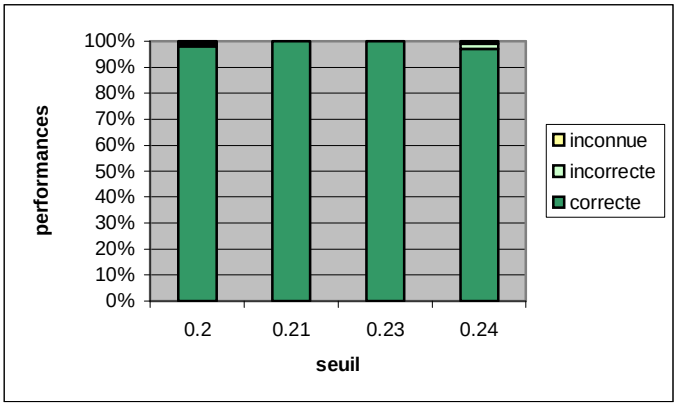

Figure 4-29 Performances/seuil (test 3.1)

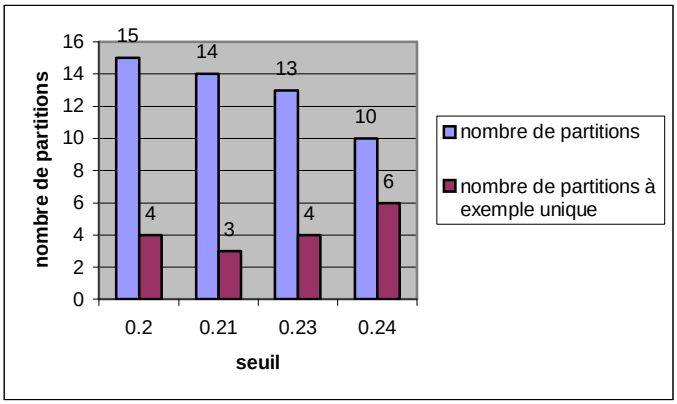

Figure 4-30 Nombre de partitions/seuil (test 3.1)

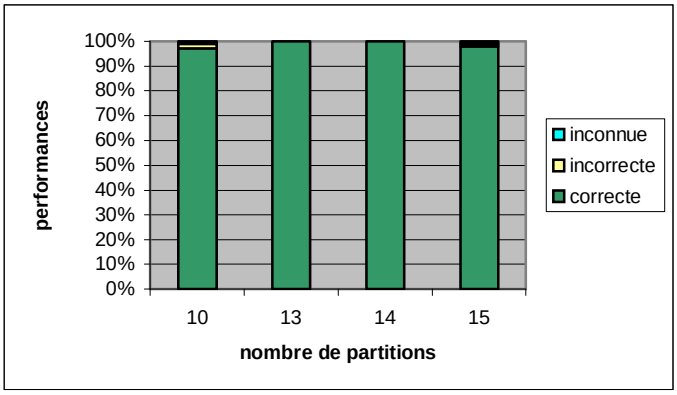

Figure 4-31 Performances/nombre de partitions (test 3.1)

- Pour le test 3.2, les performances sont moins importantes que celles de la **proposition 1** ce qui laisse à penser que le partitionnement des exemples d'entraînement n'est vraiment pas adapté aux images normalisées (**figure 4.32, 4.33, 4.34**).

Figure 4-32 Performances/seuil (test 3.2)

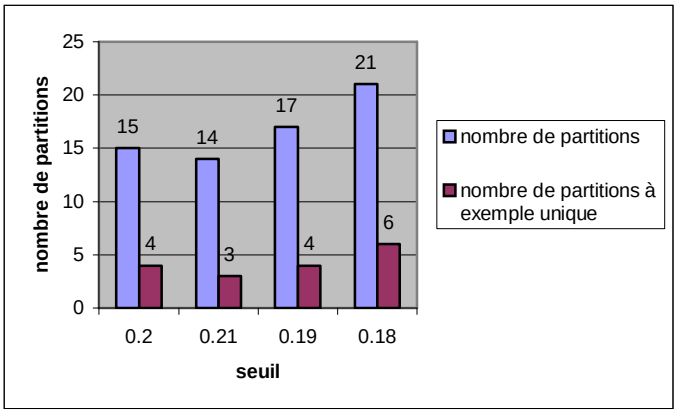

Figure 4-33 Nombre de partitions/seuil (test 3.2)

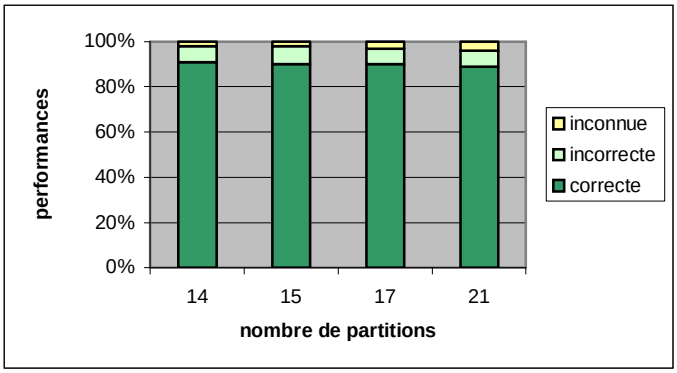

Figure 4-34 Performances/nombre de partitions (test 3.2)

- Pour le test 4.1, les performances du système atteignent les 65% de classifications correctes ce qui est 5% mieux que les performances de la **proposition 1**. Par contre, le nombre de partitions générées reste élevé avec prés de six cellules mémoire par classe (**figure 4.35, 4.36, 4.37**).

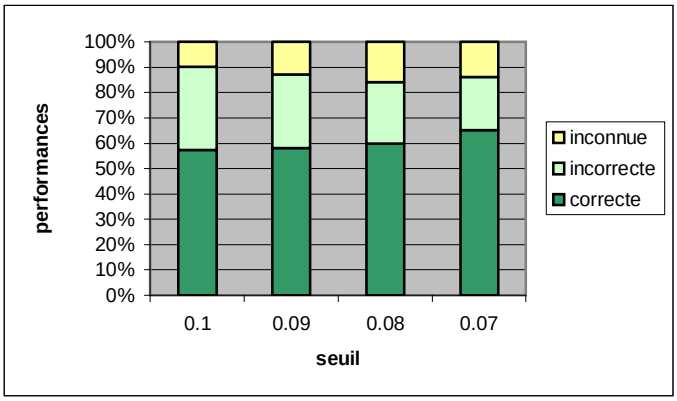

Figure 4-35 Performances/seuil (test 4.1)

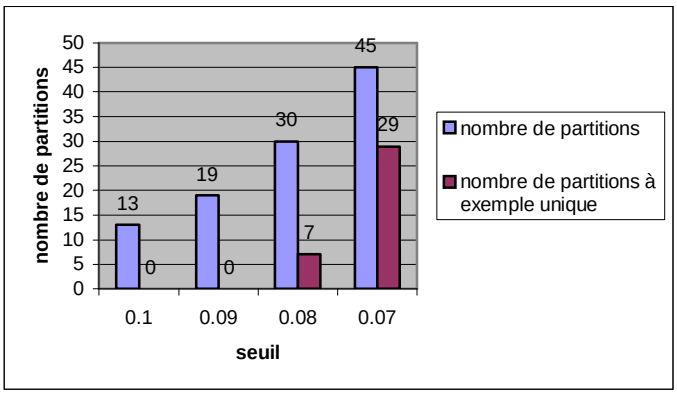

Figure 4-36 Nombre de partitions/seuil (test 4.1)

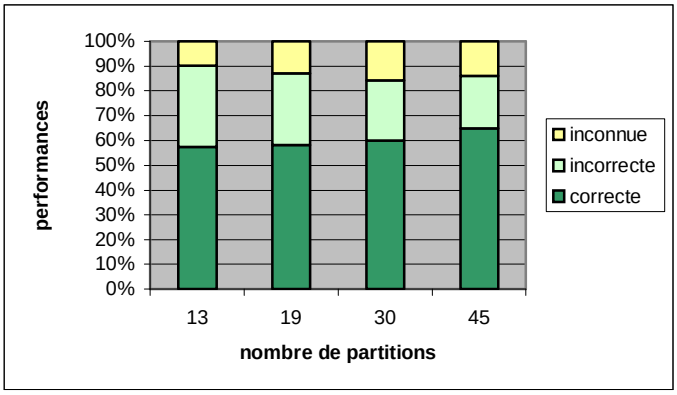

Figure 4-37 Performances/nombre de partitions (test 4.1)

4.6 Traitement des cas multiples

Le principe de fonctionnement du classificateur est assez simple : on compare la forme à reconnaître à toutes les cellules mémoire disponibles et on lui affecte la classe de la cellule mémoire qui lui ressemble le plus. Lorsque la forme à reconnaître a la même valeur de similarité pour deux cellules mémoire (ou plus) de classes différentes, le classificateur ne peut rien dire et la forme est rejetée sans être classée.

Maintenant que nous avons plusieurs cellules mémoire par classe, n'est il pas possible d'utiliser ces informations supplémentaires lors du processus de classification ?

La nouvelle fonction de classification est comme suit :

- Comparer la forme inconnue à toutes les cellules mémoire disponibles ;
    - Si une seule cellule mémoire ressemble le plus à la forme, alors elle reçoit la classe de cette cellule mémoire ;
    - Sinon (plusieurs cellules mémoire ont la même similarité avec la forme), dans ce cas il y a un conflit. Pour chacune des classes en conflit, calculer la nouvelle similarité comme étant la moyenne des similarités des cellules mémoire de cette classe. Utiliser cette nouvelle valeur pour classer la forme

4.6.1 Performances obtenues :

Nous avons appliqué ce nouveau classificateur en utilisant chacune des propositions précédentes, les résultats obtenus sont les suivants :

- Pour la **proposition 1**, les performances du système ont été améliorées passant de 60% à 65% pour le test 4.1. ceci représente une amélioration importante en sachant que FCC tout seul ne dépasse pas les 43% (**figure 4.38**);

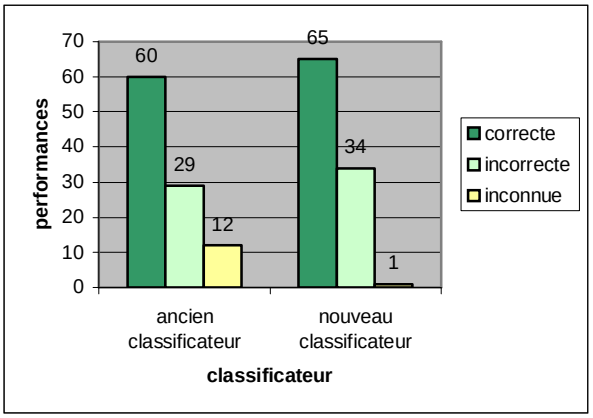

Figure 4-38 Performances/classificateur (proposition 1, test 4.1)

- Pour la **proposition 2**, les performances pour le test 4.1 passent de 58% à 66% de bonnes classifications, ainsi que pour le test 3.2 dont les performances passent de 91% à 94% (**figure 4.39**);

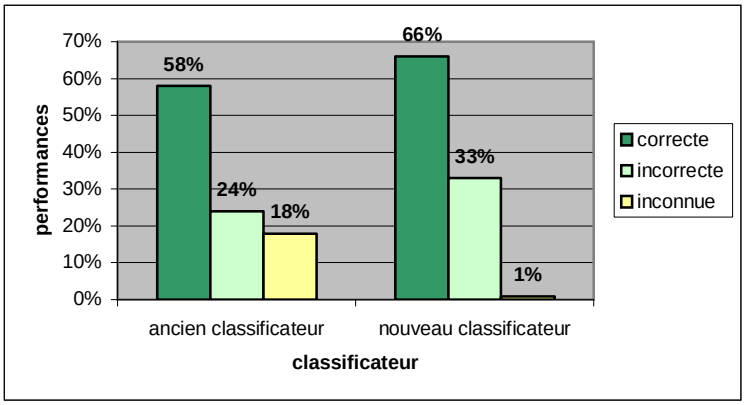

Figure 4-39 Performances/classificateur (proposition 2, test 4.1)

- Pour la **proposition 3**, les performances passent de 59% à 63% pour le test 4.1, et de 92% à 94% pour le test 3.2 (**figure 4.40**);

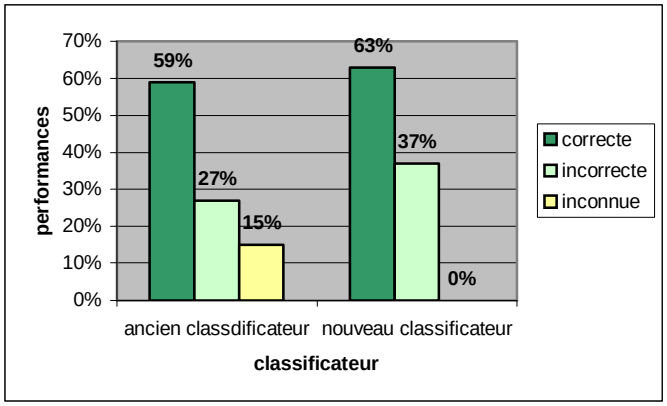

Figure 4-40 Performances/classificateur (proposition 3, test 4.1)

- Pour la **proposition 4** les performances du système passent de 65% à 72% en 50 générations, ce sont les meilleures performances jamais obtenues sur ce test. Pour le test 3.2 le taux de classifications correctes passe de 91% à 92% (**figure 4.41**).

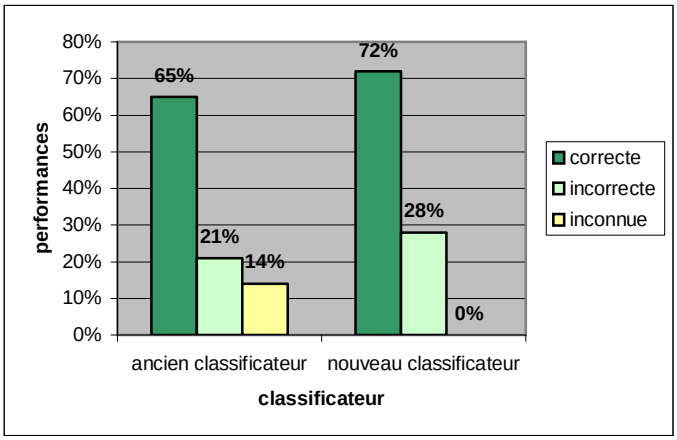

Figure 4-41 Performances/classificateur (proposition 4, test 4.1)

4.7 Etude Comparative et discussion :

Dans ce chapitre, quatre propositions ont été développées afin d'améliorer les performances de classification du système de reconnaissance des formes dont le cœur est l'algorithme d'apprentissage inspiré des systèmes immunitaires naturels FCC. Ces propositions sont comme suit:

- Appliquer un algorithme de clustering avec un seuil de partitionnement choisi manuellement.
- Déduire automatiquement les partitions en analysant le dendrogramme généré par l'algorithme de clustering, et se passer ainsi du seuil manuel.
- Relâcher les contraintes de la proposition précédente en acceptant des cellules mémoire à risque, ceci afin de réduire le nombre de partitions générées.
- Améliorer les résultats de la première proposition (seuil manuel) en analysant le dendrogramme.

Le classificateur a aussi été amélioré pour exploiter les nouvelles informations disponibles grâce à la multitude de cellules mémoire pour chaque classe.

Les figures **4.42** et **4.43** résument les résultats obtenus pour chacune des propositions développées avec le nouveau classificateur.

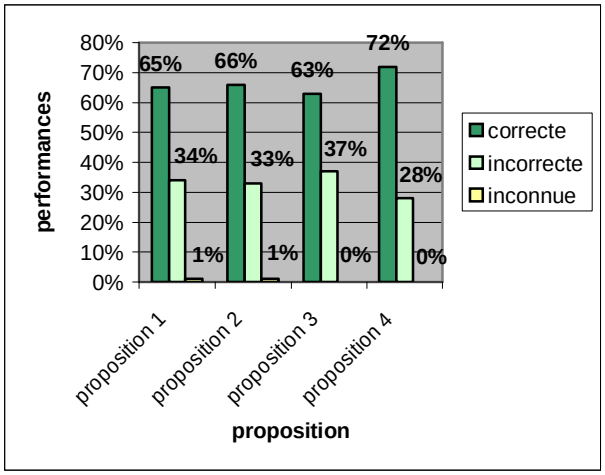

Figure 4-42 Comparatif entre les performances de toutes les propositions

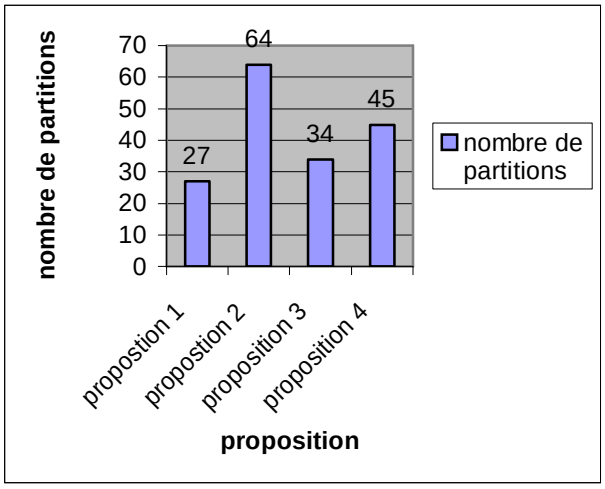

Figure 4-43 Comparatif entre les nombres de partitions de toutes les propositions

De façon générale toutes les propositions améliorent les performances du système pour les images non normalisées, par rapport à FCC. Avec l'ancien classificateur la performance la moins bonne (58%) est bien meilleure que celle de FCC sans clustering (43%). Cela montre en partie l'importance de la phase de prétraitement pour l'amélioration des performances globales d'un système de reconnaissance des formes. Avec le nouveau classificateur, les performances les moins bonnes sont celles de la **proposition 3** avec 63% de bonnes classifications. La **proposition 4** l'emporte avec 72% de bonnes classifications, soit 29% de plus que FCC sans clustering.

En ce qui concerne le nombre de partitions générées, nous avons vu que ce nombre n'a pas intérêt à être très grand. La proposition qui génère le moins de partitions pour les exemples non normalisés est sans conteste la **proposition 1**, avec l'inconvénient de devoir

choisir manuellement le seuil de partitionnement le plus performant ce qui implique de devoir relancer l'apprentissage pour chaque seuil.

Bien que la **proposition 4** donne les meilleures performances pour les exemples non normalisés, elle produit cependant trop de partitions. Cette proposition n'est pas recommandée pour les larges ensembles d'entraînement.

La proposition qui semble donner le meilleur compromis entre les performances et le nombre de partitions, pour les exemples non normalisés, semble être la **proposition 3**. Cette proposition produit un nombre acceptable de partitions pour des performances presque égales à celles obtenues par un seuil manuellement choisi, avec l'avantage d'effectuer le partitionnement sans nécessiter aucune intervention de l'utilisateur.

Pour les exemples normalisés, par contre, aucune proposition ne semble améliorer les performances de FCC. Au contraire, le fait de partitionner les exemples d'entraînement normalisés semble conduire le système à un sur apprentissage. Cette hypothèse est confirmée par l'augmentation des performances sur l'ensemble d'entraînement (test 3.1) jusqu'à 100%, et la baisse de performances pour les exemples test (test 3.2).

4.8 Problème de la fonction d'affinité :

Comme nous l'avons déjà signalé, la fonction d'affinité utilisée dans CLONCLAS et FCC permet au système de trouver pour chaque classe la cellule mémoire qui représente le mieux les exemples de cette classe. Nous avons vu que cette cellule mémoire est une sorte de moyenne des exemples de la classe, et contient la forme générale de ces exemples.

N'est il pas possible de trouver les cellules mémoire directement sans passer par CLONCLAS ou FCC ? En d'autres termes, ne peut-on pas trouver de formule pour calculer les valeurs des pixels des cellules mémoire à partir des exemples d'entraînement ?

Soit **Abm** la cellule mémoire trouvée pour l'antigène **Ag**. Théoriquement elle a l'affinité maximale pour cet antigène, et comme l'affinité de **Abm** pour un pixel i est :

$$\left(Abm_i \oplus 1\right)\times n_i^1 + \left(Abm_i \oplus 0\right)\times n_i^0 \qquad (4.1)$$

Donc, si $n_i^1 > n_i^0$ $\mathbf{Abm_i}$ qui donne la plus grande affinité ne peut être que 0.

De même, si $n_i^1 < n_i^0$ $\mathbf{Abm_i}$ qui donne la plus grande affinité ne peut être que 1.

Et pour finir, si $n_i^1 = n_i^0$ $\mathbf{Abm_i}$ donne la plus grande affinité qu'il soit à 1 ou à 0.

On peut alors trouver par calcul direct la cellule mémoire qui optimise la fonction d'affinité directement à partir des exemples d'entraînement sans avoir à passer par CLONCLAS ou FCC !

L'algorithme de calcul direct des cellules mémoire est le suivant :

- Pour chaque antigène **Ag^j**, calculer n_i^1 et n_i^0 ;
- Calculer la valeur de chaque bit Abm_i^j de la cellule mémoire **Abm^j** correspond à **Ag^j**, de la manière suivante :
  - Si $n_i^1 > n_i^0$, Abm_i^j a pour valeur **0** ;
  - Si $n_i^1 < n_i^0$, Abm_i^j a pour valeur **1** ;
  - Si $n_i^1 = n_i^0$, la valeur de Abm_i^j est choisie aléatoirement dans l'ensemble **{0, 1}** (puisque CLONCLAS et FCC sont stochastiques).

Pour valider nos suppositions, nous avons repris les mêmes tests effectués avec CLONCLAS en utilisant le calcul direct des cellules mémoire. Le tableau suivant indique le taux de classifications correctes en utilisant les cellules mémoire obtenues grâce à CLONCLAS et celles par calcul direct (**figure 4.44**).

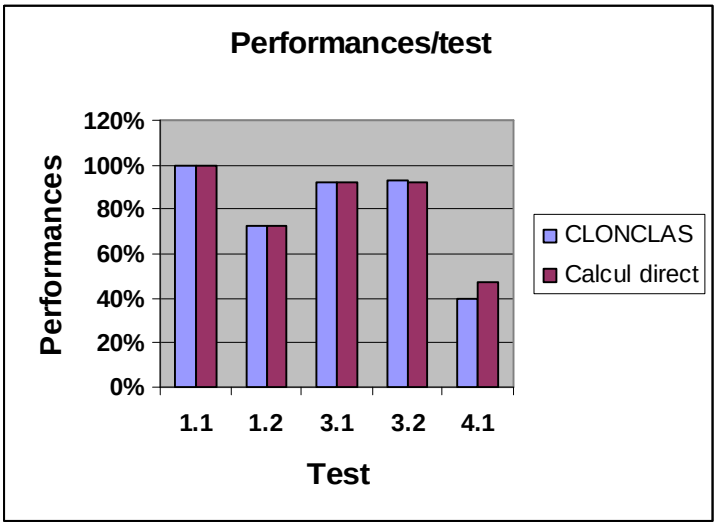

Figure 4-44 Comparatif entre les performances de CLONCLAS et le calcul direct

Vous remarquerez que les résultats sont équivalents ce qui confirme la validité de l'algorithme de calcul des cellules mémoire. Bien que ce résultat signifie que la fonction affinité utilisée actuellement dans CLONCLAS et FCC doit impérativement être modifiée. Cela représente quand même une preuve de la convergence des deux algorithmes vers les cellules mémoire optimales pour la fonction affinité utilisée. En sachant que l'ensemble de recherche contient 2^{144} formes possibles. FCC arrive à trouver les meilleures cellules mémoire

en exploitant seulement ***50 générations x (200 clones + 5 anticorps sélectionnés)*** formes, soit moins de 2^{14} formes.

4.9 Conclusion :

Bien que FCC donne de bonnes performances lorsque les formes sont normalisées, ces performances sont très en dessous de la moyenne lorsque les formes ne sont pas normalisées. Une raison possible à ces faibles performances est le faible nombre de cellules mémoire utilisées pour capturer les caractéristiques des classes. Une seule cellule mémoire par classe a de faibles chances de capturer la forme générale des exemples de la classe, surtout lorsqu'il y a de grandes variations entre ces exemples (ce qui est le cas pour les formes non normalisées).

La solution développée **[Deneche et al 05, Meshoul et al 05]** s'inspire des systèmes naturels, qui produisent pour chaque antigène rencontré plusieurs sortes de cellules mémoire afin de capturer le plus de caractéristiques possibles. Quatre propositions ont été développées, qui ont toutes pour principe d'appliquer un algorithme de clustering sur les exemples d'entraînement pour découvrir la structure interne de chaque classe. La 1ère proposition applique l'algorithme de clustering en utilisant un seuil fournit par l'utilisateur. La 2ème proposition tente de déterminer le partitionnement en analysant les données d'entraînement. La 3ème proposition combine entre les deux propositions précédentes afin de limiter les inconvénients de chaque méthode. La 4ème méthode utilise l'analyse des risques utilisée par la proposition 2 pour maximiser les performances de la proposition 1. Un nouveau classificateur a aussi été proposé qui profite des nouvelles informations disponibles pour résoudre les ambiguïtés de classification.

La proposition 4 est celle qui donne les meilleures performances, mais avec un nombre élevé de partitions et un seuil défini manuellement. La proposition qui semble donner le meilleur compromis entre performances et nombre de partitions est la proposition 3. En plus cette proposition utilise un seuil déduit de manière automatique. De façon générale, toutes les propositions améliorent les performances du système par rapport à FCC. Ceci montre clairement que pour obtenir un système de reconnaissances de formes aux performances optimales, tous les modules du système doivent être optimisés pour le problème en question, d'où la complexité rencontrée lors du développement de tels systèmes.

Conclusion générale :

Dans le cadre de ce magistère nous nous sommes intéressés à l'application des mécanismes inhérents aux systèmes immunitaires naturels pour résoudre des problèmes de reconnaissance des formes. La reconnaissance des formes étant un vaste domaine, nous avons délibérément limité nos recherches à la phase d'apprentissage pour un système de reconnaissance de chiffres imprimés basé sur une approche par prototypes. Comme l'apprentissage des systèmes de reconnaissance ressemble beaucoup à la mémorisation des antigènes dans les systèmes immunitaires naturels, c'est donc naturellement que notre choix c'est porté sur la sélection clonale artificielle.

Nos travaux ont démarré là où c'est arrêté CLONCLAS, qui est une adaptation de la sélection clonale artificielle pour l'apprentissage. CLONCLAS souffrant d'une complexité élevée, nous avons commencé par nous attaquer à ce problème, qui doit impérativement être résolu si on veut faire l'apprentissage sur des bases d'entraînement de grandes tailles. En proposant une nouvelle formule pour le calcul du nombre de clones et en optimisant la fonction affinité, non seulement nous avons réduit la complexité de l'algorithme et les coûts mémoire, mais plus important encore, la complexité de l'apprentissage est devenue indépendante du nombre d'exemples d'entraînement ! Dorénavant, entraîner le système sur une base de milliers d'exemples prend le même temps que pour une base d'une centaine d'exemples !! L'algorithme résultant a été nommé ***FCC***.

En ce qui concerne les performances du système, CLONCLAS trouve des difficultés à classer les exemples non normalisés. 43% de bonnes classifications après 500 générations d'entraînement. Pour améliorer ces performances, nous avons proposé une extension de ***FCC*** nommée C^3 **[Deneche et al 05, Meshoul et al 05]**. Cette extension permet après analyse des exemples d'entraînement d'augmenter le nombre de cellules mémoire produites par le système. Quatre propositions ont ainsi été développées : la première utilise un seuil fournit par l'utilisateur et permet au système d'atteindre les 60% de performances en 50 générations, en générant 28 cellules mémoire. La deuxième se passe du seuil et déduit de manière automatique les cellules mémoire potentielles, les performances atteignent les 58% en 50 générations mais avec 64 cellules mémoire sur un total de 80 exemples. La troisième proposition est un compromis entre les deux propositions précédentes, elle combine le faible nombre de cellules mémoire générées par la **proposition 1** et l'automatisation de la **proposition 2**. Elle atteint les 59% de performances avec 34 cellules mémoire. Enfin, la quatrième proposition essaye d'améliorer les résultats obtenus par la **proposition 1** en utilisant l'analyse des exemples d'entraînement de la **proposition 2**. Les performances sont de 65% de bonnes classifications mais avec 45 cellules mémoire produites. Le classificateur a lui aussi été amélioré en utilisant les nouvelles informations disponibles grâce à la multiplication des cellules mémoire. Les résultats ont montré que ce nouveau classificateur améliorait effectivement les performances des quatre méthodes développées, avec 72% de

classifications correctes pour la **proposition 4**. en sachant que la meilleure proposition devrait non seulement augmenter les performances du système mais aussi réduire le nombre de cellules mémoire générées, la **proposition 3** semble être le bon compromis avec 63% de performances pour le nouveau classificateur, et 35 cellules mémoire.

Il est clair que le principal axe des recherches futures va concerner l'étude de fonctions affinités alternatives. La nouvelle fonction affinité devra non seulement tenir compte des variations intra-classes mais aussi des variations inter classes. En d'autres termes, chaque cellule mémoire produite devra ressembler le plus possible aux exemples de la classe qu'elle représente, et ressembler le moins possible aux exemples des autres classes. Un autre axe de recherche tout aussi important est l'implémentation des autres propriétés intéressantes des systèmes immunitaires naturels, le plus important est assurément le mécanisme d'adaptabilité des systèmes immunitaires face à un environnement externe en constante mutation. Ce mécanisme devrait permettre d'avoir des systèmes de reconnaissance qui apprennent à chaque nouvelle forme rencontrée et pas seulement au début.

Bibliographie:

[Aicklin et al, 03] U. Ackelin, "Danger Theory: The link between AIS and IDS?", *2nd ICARIS*, 2003.

[Back et al, 97] T. Back, D.B. Fogel, Z. Michalewicz, "Handbook of Evolutionary Computation", *IOP Publishing Ltd and Oxford University Press*, 1997.

[Barber, 03] D. Barber, "Learning form Data", http://anc.ed.ac.uk/~dbarber/lfd1/lfd1.html, 2003.

[Chung et al, 03] K.W. Cheung, J.T. Kwok, M.H. Law, K.C. Tsui,"Mining customer product ratings for personalized marketing", *Decision Support Systems*, vol. 35, issue 2, pp. 231-243, 2003

[Coello et Cortez, 05] C.A.C. Coello, N.C. Cortez, "Solving Multiobjective Optimization Problems Using an Artificial Immune System", *Genetic Progrqmming and Evolvable Machines*, vol. 6, pp. 163-190, 2005.

[Davies et Bouldin, 79] D. L. Davies, W. Bouldin. "A cluster separation measure". *IEEE PAMI*, vol. 1, pp. 224-227, 1979.

[Devijver et Kittler, 82] P. A. Devijver, J. Kittler, « Pattern Recognition : A statistical Approach ». *London : Prentice-Hall*, 1982

[de Castro et Von Zuben, 99] L. N. de Castro, F. J. Von Zuben, "Artificial immune systems: Part I—basic theory and applications". *Technical Report DCA-RT 01/99*, School of Computing and Electrical Engineering, State University of Campinas, Brazil.

[de Castro et Von Zuben, 02] L. N. de Castro, F. J. and Von Zuben, "Learning and optimization using the clonal selection principle". *IEEE Transactions on Evolutionary Computation*, vol. 6, issue 3, pp. 239–251.

[Deneche et al, 05] A. Deneche, S. Meshoul, M. Batouche, "Une approche hybride pour la reconnaissance de formes en utilisant un système immunitaire artificiel", *Journées d'étude en Informatique graphique JIG'05*, 2005.

[Elnagar et Al Hadj, 05] A. Elnagar, R. Al-Hajj, "Segmentation of Handwritten Touching Digits: A Multiagents Approach", *In Computer Aided Intelligent Recognition Techniques and Applications*, John Wiley & Sons, 2005.

[Esponda et Forrest, 03] F. Esponda, S. Forrest, “The crossover closure and partial match detector”. *In ICARIS-2003*, pp. 249–260, 2003.

[Forrest et al., 94] S. Forrest, A. S. Perelson, L. Allen, R. Cherikuri, “Self-nonself discrimination on a computer”. *In Proceedings of IEEE Symposium Research in Security and Privacy*, pp. 132-143, 1994.

[Garrett, 05] S.M. Garrett, “How do we Evaluate Artificial Immune Systems?”, *Evolutionary Computation*, vol. 13, Issue 2, pp.145-178, 2005.

[Gupta et al, 95] S.K. Gupta, W.C. Regli, D.S. Nau, "Manufacturing Feature Instances: Which Ones to Recognize?", *Symposium on Solid Modeling and Applications*, pp. 141-152, 1995.

[Hippert et al, 01] H.S. Hippert, C.E. Pedreira, R.C. Souza, "Neural networks for short-term load forecasting: a review and evaluation", *IEEE Transactions on Power Systems*, vol. 16, Part 1, pp. 44-55, 2001.

[Hall et Holmes, 03] M.A. Hall, G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining", *IEEE Trans. On Knowledge and Data Engineering*, vol. 15, no. 3, 2003.

[Han et Kamber, 02] J. Han, M. Kamber, "Data Mining : Concepts and Techniques", *SIGMOD Record*, 2002.

[Hse et Newton, 04] H. Hse, A. R. Newton, “Sketched Symbol Recognition using Zernike Moments”, *Proceedings of ICPR’2004*, 2004.

[Jain et al, 00] A.K. Jain, R.P.W. Duin, J. Mao, "Statistical Pattern Recognition: A Review", *IEEE Trans. Pattern Analysis and Machine Learning*, vol. 22, no. 1, 2000.

[Jerne, 74] N. Jerne, “Towards a network theory of the immune system”. *Annals of Immunology*, vol. 125, pp. 373–389.

[Kaastra et Boyd, 96] I. Kaastra, M. Boyd, "Designing a neural network for forecasting financial and economic time series", *Neurocomputing*. Vol. 10, no. 3, pp. 215-236. 1996.

[Kim et Bentley, 02], “A model of Gene Library Evolution in the Dynamic Clonal Selection Algorithm”, *ICARIS*, pp.175-182, 2002

[Leuski, 01] A. Leuski. "Evaluating document clustering for interactive information retrieval", *Proceedings of CIKM'01*, pp. 41-48, ACM Press, 2001.

[Liu et Nagakawa, 01] C.L. Liu, M. Nakagawa ,"Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition", *Pattern Recognition*, vol. 34, part. 3, pp. 601-616, 2001.

[Matzinger, 02] P. Matzinger, "The Danger Model: a renewed sense of self", *Science*, 296(5566), pp. 301-305, 2002.

[Meshoul et al, 05] S. Meshoul, A. Deneche, M. Batouche, "Combining an Artificial Immune system with a Clustering Method for Effective Pattern Recognition", *International Conference on Machine Intelligence*, pp. 782-787, 2005.

[Moghaddam et al, 00] B. Moghaddam, T. Jebara, A. Pentland, « Bayesian Face Recognition », *Pattern Recognition*, vol. 33, no. 11, pp. 1771-1782, 2000.

[Morita et al, 02] M. Morita, R. Sabourin, F. Bortolozzi, C. Y. Suen, "*Unsupervised Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten Recognition*", *Proceedings of the 16th IAPR*, pp. 568-571, 2002.

[Munich et Perona, 02] M.E. Munich, P. Perona, "Visual Input for Pen-Based Computers", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, issue 3, 2002.

[Nasraoui et al, 02] O. Nasaroui, F. Gonzalez, D. Dasgupta, "The Fuzzy Artificial Immune System: Motivations, Basic Concepts, and Application to Clustering and Web Profiling", *International Joint Conference on Fuzzy Systems*, 2002.

[Pareto, 96] V. Pareto, "Cours D'économie politique", volume I et II, F. Rouge, Lausanne, 1896.

[Popat et Picard, 03] K. Popat, R.W. Picard, "Novel cluster-based probability model for texture synthesis, classification, and compression", *Proceedings of SPIE*, 2003.

[Raudys et Jain, 91] S.J. Raudys, A.K. Jain, "Small Sample Siwe Effects in Statistical Pattern Recognition: Recommendations for Practitioners", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252-264, 1991.

[Rezaee et al, 00] M.R. Rezaee, P.M.J. van der Zwet, B.P.F. Lelieveldt, R.J. van der Geest, J.H.C. Reiber, "A Multiresolution Image Segmentation Technique Based on Pyramidal Segmentation and Fuzzy Clustering", *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 9, part. 7, pp. 1238-1248, 2000.

[Roitt, 90] I. Roitt, "Immunologie", *Editions Pradel*, 1990.

[Sakano, 05] H. Sakano, “Query Learning for Character Recognition Methods using Genetic Algorithms”, *IEICE Trans. Inf. & Syst.*, vol. E88-D, no. 10, 2005.

[Secker et al, 03] A Secker, AA Freitas, J Timmis, “A Danger Theory Inspired Approach to Web Mining”, *LNCS*, 2003.

[Skurichina et al, 02] M. Skurichina, L.I. Kuncheva, R.P.W. Duin, "Bagging and Boosting for the Nearest Mean Classifier: Effects of Sample Size on Diversity and Accuracy", *LNCS*, issue 2, pp. 62-71, 2002.

[Theodoridis et Koutroumbas, 03] S. Theodoridis, K. Koutroumbas, “Pattern Recognition, Second Edition”, *Academic Press, Elsevier*, 2003.

[Trier et al, 96] O.D. Trier, A.K. Jain, T. Taxt, "Feature Extraction Methods for Character Recognition – A Survey", *Pattern Recognition*, vol. 29, no. 4, pp. 641-662, 1996.

[Tzanakou, 00] E.M. Tzanakou, “Supervised and Unsupervised Pattern Recognition: Feature Extraction and Computational Intelligence”, *CRC Press*, 2000.

[Watanabe, 85] S. Watanabe, “Pattern recognition: Human and Mechanical”, New York : Wiley, 1985

[Webb, 02] A.R. Webb, “Statistical Pattern Recognition, Second Edition”, John Wiley and Sons, 2002.

[White, 04] J. White, “Artificial Clonal Selection for Pattern Recognition”, PhD thesis, University of Wales, Aberystwyth, December 2004.

[White et Garrett, 03] J. A. White, S. M. Garrett, “Improved pattern recognition with artificial clonal selection”. *In Proceedings of ICARIS*, 2003.

[Woznica et Menal, 03] P. Woznica, M. Mennal, "Reconnaissance des formes", 2003.

[Yang et al, 02] M.H. Yang, D.J. Kriegman, N. Ahuja, "Detecting Faces in Images: A Survey", *IEEE Transactions On Pattern Analysis And Machine Intelligence PAMI*, vol.24; part 1, pp. 34-58, 2002.