Chapitre 5

Problèmes d'apprentissage

Dans ce chapitre, nous examinons différentes approches pour acquérir de la connaissance à partir de données incomplètes et mal définies. Trois thèmes dominent, en premier lieu, celui de la classification, c'est un exemple type de problème mal spécifié, en effet, à partir d'un ensemble d'objets, qui, suivant la perception humaine, sont «plus ou moins» répartis en sous-ensembles aux frontières «plus ou moins bien nettes», on souhaite élaborer des méthodes pour arriver à une répartition satisfaisante en classes. A un classement en groupes prédéfinis, s'ajoute le problème plus ambitieux de l'énonciation de règles à partir d'exemples ayant des attributs vagues et de l'extraction de prédicats flous. Le deuxième thème est celui du connexionisme, c'est-à-dire des différents modèles de réseaux de neurones artificiels chargés d'apprendre à reconnaître une forme à partir d'un ensemble d'exemples sur lequel l'apprentissage a été effectué, ou bien à décider de la détermination de classes à partir de la donnée de prototypes et de reconnaître des objets voisins. Le troisième thème est celui des algorithmes génétiques ou plus généralement des stratégies d'évolution artificielle, heuristiques destinées aux problèmes d'optimisation, qui, comme on le verra, permettent d'apprendre un comportement simplement à partir de contraintes. Après avoir exposer les principales méthodes, nous envisageons comment tous ces sujets sont naturellement liés au flou.

5.1. Classement et classification

Etant donnés un ensemble $X = \{x_1, x_2, ... x_n\}$ de n objets dans un espace métrique, et un nombre m de classes fixé à l'avance, le but du problème de classification ou coalescence est de séparer X en une partition $C = \{c_1, c_2, ... c_m\}$ suivant les «nuages de points» observés. Ce nuage se présente bien sûr (et ce n'est bien visible qu'en dimension deux) avec des zones de plus ou moins fortes densités. Par ailleurs, le problème en lui même n'est pas vraiment bien défini, et des éléments sont manifestement toujours à cheval sur plusieurs classes, c'est pourquoi l'usage de

classes floues semble bien indiqué, cependant diverses approches ne donneront pas necéssairement les mêmes résultats pour la raison que le but n'est pas véritablement spécifié.
Nous réservons le terme de «classement» pour l'opération consistant à ranger des objets dans des classes prédéfinies, et celui de «classification» à l'opération consistant à créer des classes, leur nombre n'étant même pas fixé initialement.

L'ALGORITHME DES K PLUS PROCHES VOISINS

Un premier algorithme très simple consiste à débuter avec un point (ou m points) et à classer chaque nouveau point dans la classe la plus repésentée chez ses k (de l'ordre de 3 à 5) plus proches voisins (il ne s'agit donc pas de trouver les classes, ce qui est un vrai problème d'apprentissage, mais de classer de nouveaux candidats). Cette méthode est très simple, mais lente et son inconvénient est que les résultats sont sensiblement différents suivant les valeurs de k adoptées.

ALGORITHME FLOU DES K PLUS PROCHES VOISINS

Si les objets x_1, x_2, ... x_n sont déjà classés et si x est un nouvel exemple, on calcule les distances $d_j = d(x, x_j)$ en retenant l'ensemble V des indices des k exemples les plus proches, puis, λ étant plus grand que 1, on calcule une fonction d'appartenance du nouvel individu x relative à chaque classe C_i par :

$$\mu_i(x) = \frac{\sum_{j \in V} \mu_{ij} . d_j^{\frac{2}{1-\lambda}}}{\sum_{j \in V} d_j^{\frac{2}{1-\lambda}}}$$

Chaque voisin intervient donc pour chaque classe avec un coefficient décroissant suivant sa distance à x. La décroissance étant réglée par le coefficient λ.

ALGORITHME DE BEREAU [Béreau, Dubuisson 91]

Il s'agit d'un raffinement de la méthode précédente. Cet algorithme prend les points à classer dans l'ordre où ils se présentent et forment des classes au fur et à mesure en s'inspirant de la méthode des k plus proches voisins, ceux-ci étant tempérés par une fonction décroissante de leur distance au point courant. L'avantage de cette méthode est que les points peuvent être ajoutés en temps réel et que le nombre de classes m n'est pas fixé à l'avance mais dépend bien sûr d'un certain nombre de paramètres.
On fixe un seuil d'appartenance s (par exemple 0.5), un paramètre a (de l'ordre de 0.45) et un nombre k (de l'ordre de 3) pour l'examen des plus proches voisins.
Au cours de l'algorithme, si une classe C_i est définie, pour un nouvel objet x (non encore placé), on calcule un degré d'appartenance par :
$\mu_i(x) = \max\{\mu_i(y).\exp(-ad^2(x, y) / d^2_m) / y$ parmi les k plus proches voisins de x$\}$ où d_m est la distance moyenne entre éléments de la classe C_i et a est un réel fixé entre 0 et 1. S'il est supérieur au seuil s, x est retenu dans la classe courante et d_m est alors mis à jour, x peut donc être placé dans une des classes déjà formées, ou bien rejeté (placé dans un ensemble R).

Pour chaque $x \in R$ on calcule donc les appartenances de x à tous les C_i pour i compris entre 1 et la valeur provisoire de m, s'il existe un x tel que $\mu_i(x) > s$, alors il est placé dans la classe C_i, sinon il est rejeté.
L'ensemble de rejet est alors analysé de la même façon.
Le premier calcul consiste en celui du centre de gravité G du nuage et on lui donne la valeur d'appartenance $\mu_1(G) = 1$, et on pose m = 1. On classe ensuite les points dans l'ordre décroissant de leur distance à G, on ne retient que ceux qui ont une $\mu_1(x) = \max\{\mu_1(y).\exp(-ad^2(x, y) / d^2_m) / y$ parmi les k plus proches voisins de x} supérieure à s.
Si le nombre de points restants dans R dépasse un seuil fixé à l'avance et que le nombre de classes m n'est pas trop important (une limite est également fixée), on initialise une nouvelle classe avec R comme la première.
La méthode s'achève avec une éventuelle fusion de classes C_i et C_j c'est à dire si $\forall x$ $|\mu_i(x) - \mu_j(x)| <$ seuil.

UN ALGORITHME DE CLASSIFICATION FLOUE EN DEUX CLASSES

Si $X = \{x_1, x_2, ... x_n\}$ sont n objets à classer dans un espace muni de la distance d et si l'on dispose de la donnée des distances mutuelles, on peut en déduire une relation floue de «proximité» définie par $r_0 (x, y) = 1 - d(x, y) / \max (d)$. On pose R_0 la matrice symétrique n*n de cette relation. Initialement, on part de la partition U_0 formée par les n singletons $\{x_i\}$.
En prenant les deux objets les plus proches x_k et x_l, on construit une seconde partition U_1 formée par les n - 2 autres singletons $\{x_i\}$ et la paire $\{x_k, x_l\}$. On construit alors la matrice carrée R_1 (n - 1)*(n - 1) identique à R_0 sauf pour $r(\{x_i\}, \{x_k, x_l\}) = \min(r(x_i, x_k), r(x_i, x_l))$ et $r(\{x_k, x_l\}, \{x_i\}) = \max(r(x_k, x_i), r(x_l, x_i))$. Cette matrice n'est donc plus symétrique.
A chaque étape, à partir de la matrice R_{k-1} et de la distance d_{k-1}, on redéfinit une distance d_k entre les éléments de la partition U_k de fonctions d'appartenance μ et μ' par : $d_k(\mu, \mu') = \Sigma_{x \in X} |\mu^\lambda(x) - \mu'^\lambda(x)|$ avec un $\lambda > 0$. La relation floue r est modifiée $r(\mu, \mu') = 1 - d_k(\mu, \mu') / \max(d_k)$.
En prenant le couple de parties floues les plus proches au sens de d_{k-1}, on construit une partition U_k avec $\mu_C(x) = \max\{r(x, y) / y \ C\}$.
En itérant n - 2 fois ce procédé, on obtient deux ensembles flous [Dubuisson, Oppencham 80].

MÉTHODE DES CENTRES MOBILES

C'est la méthode classique en analyse des données, on se donne également un ensemble de points de R^n et le nombre m, choisi a priori, de classes voulues.
On choisit aléatoirement des centres $G_1, G_2, ... G_m$, puis on place dans la classe i, tous les points plus proches de G_i que des G_j pour $j \neq i$.
On recalcule les centre G_i obtenus pour cette partition.
On recherche à nouveau la partition autour de ces nouveaux centres et ainsi de suite jusqu'à stabilisation relativement à un certain seuil, des G_i. Cette méthode très simple dans son principe a inspiré les algorithmes itératifs de Bezdek.

CLASSIFICATION AUTOMATIQUE

Le théorème de König-Huygens [Henry, Labordère 77] exprime la relation ci-dessous pour n points P_i répartis en m classes C_j (G_i est le centre de gravité de la classes C_i).

$$\sum_{i=1}^{n} \overrightarrow{GP_i}^2 = \sum_{j=1}^{m} \mathrm{card}(C_j)\, \overrightarrow{GG_j}^2 + \sum_{j=1}^{m} \sum_{P_i \in C_j} \overrightarrow{G_jP_i}^2$$

Comme cette somme est fixe (G est le centre de gravité du système), on voit que le but de la classification étant d'avoir des classes les plus séparées possibles, donc de maximiser la première somme nommée d_{inter}, et donc de minimiser la seconde nommée d_{intra}. Cette somme étant la même quelle que soit la partition, on voit que d_{intra} a des chances de diminuer si deux classes sont fondues en une. Le problème est alors de trouver une heuristique séparant en classes, mais sans arriver à une seule classe.

Si on souhaite cette fois, avec plus d'ambition, que le nombre de classes soit lui-même trouvé automatiquement, une méthode itérative est donnée ci-dessous.

On donne un ensemble de points de R^n et la matrice de leurs distances deux à deux.

En procédant par itérations avec un incrément d_0, on regroupe en classes tous les éléments p, q tels que $d(p, q) < d$ ($d = d_0$ initialement puis d sera incrémenté de d_0 à chaque étape). Le nombre de classes diminue donc à chaque fois.

Si, à chaque étape, on cherche les centres de gravités G_i de ces classes, on pose «l'inertie» de la partition : $I = \sum d^2(G_i, G)$ en notant G est le centre de tout le nuage.

On peut alors chercher l'étape (donc le nombre de classes) minimisant l'inertie I, car si I est minimum, celà entraîne que chaque dérivée partielle par rapport à une des distances est nulle : $I' = \sum d_i d'_i = 0$ signifie que chaque centre est stabilisé par rapport au centre G de tout le nuage.

EXTENSION À DES CLASSES FLOUES - MÉTHODE DE BEZDEK

La donnée d'une partition floue en m classes, équivaut à la donnée d'une matrice m*n (avec $m \leq n$) d'élément générique $\mu_{ik} = \mu_{Ci}(x_k)$ unistochastique c'est à dire vérifiant $\sum_{1 \leq i \leq m} \mu_{ik} = 1$ (la somme de chaque colonne est 1). Cette matrice a une et une seule valeur 1 par colonne (les autres étant 0) si et seulement si la partition est exacte. Le problème de trouver une telle partition réalisant un «bon équilibre» correspond à l'optimisation d'une heuristique.

Si maintenant, on tolère des valeurs autres que 0 ou 1, en notant β_i le «prototype» de la classe ou centre de gravité provisoire, la fonction :

$$J(B, C, X) = \sum_{(1 \leq i \leq m)} \sum_{(1 \leq k \leq n)} d^2(x_k, \beta_i)$$

n'est plus suffisante pour apprécier les nuances.

Toutes les méthodes de classification d'un ensemble X de n points d'un espace métrique consistent à tenter de minimiser une certaine fonction (analogue au d_{intra} du paragraphe précédent) définie à partir de X et des partitions C formées par un nombre m de classes. [Ruspini 69, 70] a étudié quelle genre de fonction des valeurs $\mu_{ij} = \mu_{Ci}(x_k)$ il convient de minimiser pour obtenir de bons résultats.

D'autres idées ont été avancées, ainsi l'optimisation de J peut être faite par algorithmes génétiques, un chromosome étant une liste $B = (\beta_1, \beta_2, ..., \beta_m)$ de

prototypes [Schulte 94], ceux-ci peuvent être employés pour obtenir des prototypes initiaux [Nascimento, Moura-Pirès 96].
Cependant nous exposerons ici la méthode du «fuzzy c-means» que [Bezdek 74, 81] a introduit pour minimiser la fonction :

$$J_\lambda(B, C, X) = \sum_{1\le i\le m} \sum_{1\le k\le n} (\mu_{ik})^\lambda d^2(x_k, \beta_i)$$

avec un coefficient $\lambda \ge 1$ (on prend généralement 2 dans les applications), la contrainte de partition floue étant toujours $\sum_{1\le i\le m} \mu_{ik} = 1$ pour tout élément x_k, et $B = (\beta_1, \beta_2, ..., \beta_m)$, β_i désignant le «prototype» de la classe C_i (ce sont tout simplement les barycentres suivant les pondérations μ^λ).
On peut partir initialement de centres de gravités aléatoires ou fixés suivant les concentrations du nuage ou encore régulièrement disposés dans une fenêtre englobant X.
La méthode consiste à chaque étape à minimiser J grâce à la partition courante.

Justification et rappel sur la méthode des multiplicateurs α de Lagrange :
Si $H(u_1, u_2, ... u_m)$ est une fonction différentiable de plusieurs variables dont on cherche les extremums relatifs alors que les variables sont liées par une relation $\Phi(u_1, u_2, ... u_m) = 0$, on construit la fonction auxiliaire (Lagrangien de H) $L(u_1, u_2, ... u_m) = H(u_1, u_2, ... u_m) - \alpha \Phi(u_1, u_2, ... u_m)$ qui est donc égale à H indépendamment de α. Un point singulier de H est donc en même temps point singulier de L et donc admet toutes ses dérivées partielles nulles.
Dans le cas de la fonction J, il est possible de raisonner pour chaque point x_k séparément, et donc de considérer la fonction :

$$H_k(\mu_{1k}, \mu_{2k}, ... \mu_{mk}) = \sum_{1 \le i \le m} (\mu_{ik})^\lambda d^2(x_k, \beta_i) - \alpha[(\sum_{1 \le i \le m} \mu_{ik}) - 1]$$

dont les dérivées partielles sont faciles à calculer (on note $d_{ik} = d(\beta_i, x_k)$) :

$$\frac{\partial H_k}{\partial \alpha} = (\sum_{i=1}^{m} \mu_{ik}) - 1 = 0 \quad \text{et} \quad \frac{\partial H_k}{\partial \mu_{ik}} = \lambda \mu_{ik}^{\lambda - 1} d_{ik}^2 - \alpha = 0 \quad \text{d'où on déduit} \quad \mu_{ik} = \left(\frac{\alpha}{\lambda d_{ik}^2}\right)^{\frac{1}{\lambda - 1}}$$

$$\text{Et comme} \sum_{i=1}^{m} \mu_{ik} = \left(\frac{\alpha}{\lambda}\right)^{\frac{1}{\lambda - 1}} \sum_{i=1}^{m} \left(\frac{1}{d_{ik}^2}\right)^{\frac{1}{\lambda - 1}} = 1, \text{ on déduit } \mu_{ik} = \frac{1}{\sum_{j=1}^{m} \left(\frac{d_{ik}}{d_{jk}}\right)^{\frac{2}{\lambda - 1}}}$$

On vérifie que chaque μ_{ik} est compris entre 0 et 1, sauf si le dénominateur est nul si $d_{ik} = 0$ auquel cas, il est normal de prendre $\mu_{ik} = 1$.
La distance d peut être calculée par $||x - y||^2 = (x - y)^t A(x - y)$ où A est une matrice symétrique définie et positive n*n. En reprenant la fonction J, pour C et X fixés, et en considérant la dérivée suivant une direction quelconque w à partir du point $B = (\beta_1, \beta_2, ..., \beta_m)$, c'est la dérivée par rapport à h :
$H'(B, w) = \sum_{1 \le k \le n} (\mu_{ik})^\lambda (x_k - \beta_i - hw)^t A (x_k - \beta_i - hw)]'_h$ pour h = 0.
la nullité de cette dérivée donne $-2\sum_{1 \le k \le n} (\mu_{ik})^\lambda (x_k - \beta_i)^t A\, w = 0$ pour tout w, et donc on a $\sum_{1\le k\le n} (\mu_{ik})^\lambda (x_k - \beta_i) = 0$ d'où en développant :

$$\beta_i = (\Sigma_{1\leq k\leq n} (\mu_{ik})^{\lambda} x_k) / (\Sigma_{1\leq k\leq n} (\mu_{ik})^{\lambda}).$$

L'algorithme consiste donc en résumé, à modifier à chaque étape les μ_{ik} en :

$$\mu_{ik} = \frac{1}{d_{ik}^{\left(\frac{2}{\lambda - 1}\right)} * \sum_{j=1}^{m} d_{jk}^{\left(\frac{2}{1-\lambda}\right)}}$$, en supposant au moins deux classes distinctes.

Si le dénominateur est nul, on posera $\mu_{ik} = 1$. Ces calculs sont suivis d'une normalisation pour obtenir $\Sigma_{1\leq i\leq m} \mu_{ik} = 1$ pour chaque point x_k. Puis à mettre à jour les prototypes, ceux-ci devenant $\beta_i = (\Sigma_{1 \leq k \leq n} (\mu_{ik})^{\lambda} x_k)/(\Sigma_{1 \leq k \leq n} (\mu_{ik})^{\lambda})$.
L'arrêt peut être décidé lorsque chaque variation de μ_{ik} est inférieure à un seuil donné comme par exemple 1%.

REMARQUES

Pourquoi un tel plutôt λ que 1?, car cela accentue les faibles niveaux d'appartenance, donc contribue à mieux séparer les classes. Lorsqu'il est proche de 1, les degré d'appartenance sont proches de 0 ou de 1, on a donc un renforcement des classes floues vers des classes nettes, en revanche, lorsqu'il est plus grand (de l'ordre de 2, 3, 5 ...) la convergence est plus rapide, mais les classes restent d'autant plus floue que λ est grand. L'expérience montre que si les prototypes sont initialement trop rapprochés, des classes ont tendance à fusionner, en tout cas, le résultat dépend largement de ces positions initiales. Néanmoins, cette démarche de classification donne de bons résultats, car elle est appliquée dans la pratique à des ensembles d'objets sur lesquels on a déjà une certaine idée a priori de leur classement, c'est pourquoi, les prototypes initiaux peuvent grossièrement être placés dès le départ.

La distance d est quelconque, en particulier celle de Hamming $d(x, y) = \Sigma|x_i - y_i|$, qui est la plus simple, mais elle peut être calculée avec la norme $||Z||^2 = Z^t A Z$ où ^{t} est l'opérateur de transposition, A est la matrice identité pour la distance euclidienne ou bien l'inverse de la matrice de covariance des exemples pour la distance de Mahalanobis, en ce cas on a $d^2(x_k, \beta_i) = (x - \beta_i)^t C_i^{-1} (x - \beta_i)$ et C_i est la matrice $\Sigma_{1\leq k\leq n} (\mu_{ik})^{\lambda} (x_k - \beta_i) (x_k - \beta_i)^t / (\Sigma_{1\leq k\leq n} \mu_{ik}^{\lambda})$ [Gustafson, Kessel 79].
Si A est une matrice symétrique, comme on sait que $Z^t A Z = 0$ définit une quadrique, [Krishnapuram, Frigui, Nasraoui 95] ont étendu le fuzzy c-means a la reconnaissance de classes qui ont la forme de coniques planes en mettant à jour les matrices A correspondants aux contours à chaque étape. La classification est a l'heure actuelle très employée en traitement d'images, afin de reconnaître des zones similaires (contours, coins, régions homogènes ...) et en reconnaissance des formes en général.
La classification «possibiliste» au sens de Krishnapuram est la méthode du c-means dans laquelle on retire la contrainte $\Sigma_{1\leq i\leq m} \mu_{ik} = 1$, pour chaque exemple k.

DU CLASSEMENT À LA CLASSIFICATION

Deux idées simples peuvent être exploitées pour acompagner la méthode précedente, tout d'abord l'initialisation des prototypes peut être arbitraire suivant un positionnement régulier dans l'espace contenant les exemples. Il est toutefois préférable de les placer approximativement sur les nuages de points que l'on peut déceler. Pour ce faire, on regarde les distributions des projections des points sur chacun des axes, prenons par exemple, en dimension 2, n points de [-1, 1].
En choisissant un nombre de tranches de l'ordre de la dizaine, nous pouvons obtenir les histogrammes relatifs aux coordonnées des exemples sur les deux axes. Il est aisé de repérer les modes de ces histogrammes, (p tranches modales pour x et q tranches modales pour y) puis par croisement, de fixer pq prototypes initiaux.
Une stratégie simple consiste alors à projeter les points à classer sur d'autres axes comme y = ax pour a = 1, -1, 2, -1/2, -2, 1/2, ... pour vérifier que ces prototypes initiaux correspondent à des tranches modales jusqu'à ce qu'il n'y ait plus d'élimination de faux prototypes.
L'inconvénient réside dans l'arbitraire sur la fixation du nombre de tranches des histogrammes, heureusement, une éventuelle fusion de classes voisines peut y pallier dans la suite du traitement.
A l'issue de cette méthode, le nombre m de prototypes est provisoirement fixé, elle s'accompagne ensuite des itérations de celle de Bezdek, en examinant à chaque fois si deux prototypes ne sont pas suffisamment voisins.
Si, pour $i < j$, $|\beta_i - \beta_j| < \partial$ (fixé au regard de [-1, 1] par exemple à 0.1) alors pour tout exemple x_k, μ_{ik} est remplacé par $(\mu_{ik} + \mu_{jk}) / 2$ et le prototype β_j est supprimé, donc le nombre m des prototypes est décrémenté.

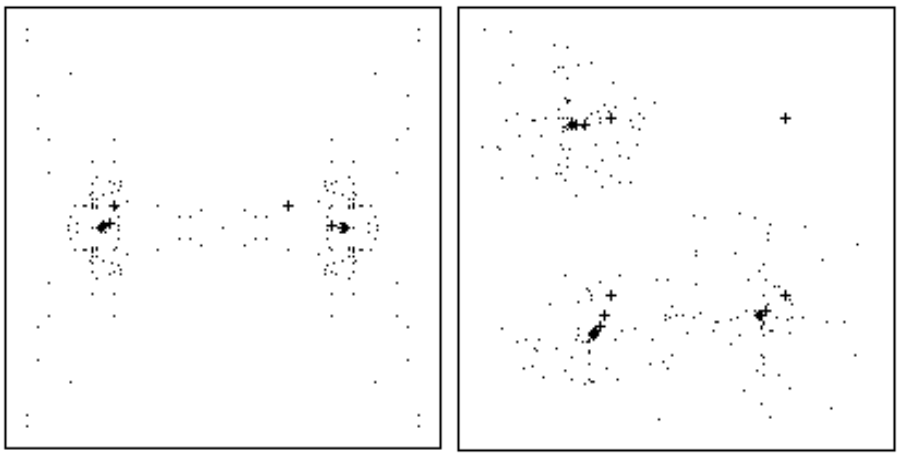

Figure 5.1 Exemple classique d'un papillon symétrique à gauche. Les deux classes sont immédiatement détectées grâce à la position des axes. Quelques itérations suffisent à déplacer les centres vers leur position définitive au seuil fixé. A droite, 300 points de $[-1, 1]^2$ sont plaçés en trois classes, grossièrement en L, quatre prototypes sont initialisés par croisement, une seule projection supplémentaire suffit à éliminer la classe parasite. 5 itérations suffisent pour assurer la convergence des prototypes.

Dans ces exemples, λ est fixé à 3 et on montre que la trajectoire des prototypes (marqués par des croix) converge bien vers les «centres» (gros points) des nuages.

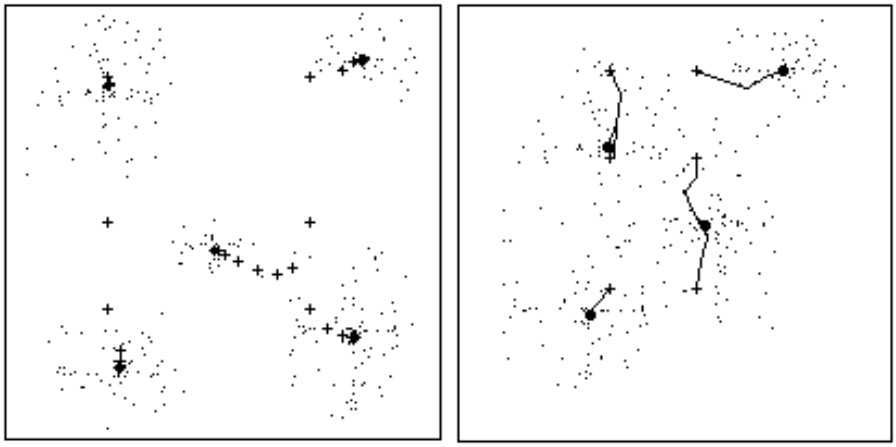

Figure 5.2 Cinq classes relativement séparées sont données à gauche. Six sont détectées par croisement des modes sur les deux axes, une est éliminée (la croix au centre à gauche), celle de centre droite, se déplaçant vers le petit nuage central. A droite, les points sont beaucoup plus diffus et le prétraitement a été anihilé de façon à observer la fusion de classes. De 6 classes fixées initialement (croix), il ne subsiste que 4 classes à la fin. Les trajectoires suivies par les prototypes laissant leur trace.

APPLICATION À LA RECONNAISSANCE DE CONTOURS

La méthode de Bezdek a été utilisée à divers propos, notamment la segmentation d'images (pour la reconnaissance de poissons) afin de reconnaître des contours [Shahin, Demko, Coisne 95] où on définit le contour (flou) ∂C d'une classe floue C par :

$$\mu_{\partial C}(x) = \min (\mu_C(x), \max\{\min(\mu_{V(x)}(y), 1 - \mu_C(y)) / y \}$$

où V(x) est un voisinage déterminé de x (par exemple x et les 8 pixels autour de lui). Etre dans le contour de C signifie être dans C et avoir un voisin y qui ne l'est pas. On définit également la frontière (floue) entre deux classes C et C' par :

$$\mu_{F(C, C')}(x) = \min (\mu_{\partial C}(x), \max\{\min (\mu_{V(x)}(y), \mu_{\partial C'}(y)) / y \}$$

(x est dans la frontière de C s'il est dans son contour et a un voisin y dans le contour de C'). Le meilleur taux de reconnaissance d'images a été obtenu avec les t-normes et t-conormes de Zadeh (les formules ci-dessus) comparées avec les t-normes de Lukasiewicz, probabiliste et de Weber au lieu de min et max.

5.2. Les arbres de décision flous

La recherche de règles à partir d'un ensemble d'exemples est un problème connu et d'ailleurs résolu depuis longtemps par l'algorithme simple ID3 [Quinlan 79, 86]. L'introduction du flou est alors tout à fait naturelle dans la mesure où un problème réel comportera des individus dont les attributs qualitatifs sont incertains (couleur, profession ...) ou numériques (taille, prix ...). Dans ce dernier cas, il se peut qu'une partition pour l'attribut soit donnée a priori (petit si inférieur à a, grand si supérieur à b ...) et que cette partition soit floue, auquel cas le problème est d'examiner l'ensemble des exemples donnés avec leurs attributs afin d'en extraire des règles floues (si X est de taille grande et de catégorie moyenne alors le prix de X est assez élevé). Mais le problème se complique encore si les exemples ne sont donnés qu'avec leurs attributs numériques, comment trouver automatiquement une bonne partition de chaque univers d'attribut avec des prédicats flous de façon à obtenir des règles floues simples et parlantes ? On voit qu'il s'agit d'un but typiquement mal défini, mais diverses méthodes vont y parvenir avec un certain succès.

UN PREMIER ALGORITHME D'EXTRACTION DE RÈGLES FLOUES À PARTIR D'UNE BASE DE RÈGLES, MÉTHODE DE WANG

Cette méthode inspirée de [Michalski 83] peut s'utiliser dès que l'on dispose d'un ensemble d'exemples de la forme (x_1, x_2, ... x_n, y), on sait donc de quelle conclusion «y» on souhaite parler, et que l'univers de chaque prémisse x_i est déjà partitionné par une famille de prédicats A_{i1}, A_{i2}, ...

En ce cas on construit une règle ($A_{1,k1}$, $A_{2,k2}$, ..., $A_{n,kn}$, C_k) à partir de chaque exemple (x_1, x_2, ... x_n, y), en choisissant simplement pour prédicat $A_{i,ki}$, celui qui est le plus vérifié par la valeur x_i, et pour prédicat C_k celui qui est le plus vérifié par y.

Le degré de cette règle est défini tout simplement comme le produit des fonctions d'appartenance aux dits prédicats apparaissant dans la règle pour l'exemple considéré (x_1, x_2, ... x_n, y).

Dans le cas de l'établissement de règles de Sugeno, une autre solution consiste à prendre comme degré la moyenne de ces conclusions exactes, pondéeée par les degrés d'appartenance aux dits prédicats, pris à une puissance supérieure à 1 [Nozaki 97]. Une puissance de l'ordre de 10 accentue fortement ce degré.

On supprime ensuite les règles inutiles, une règle R = ($A_{1,k1}$, $A_{2,k2}$, ..., $A_{n,kn}$, C_k) est dite inutile s'il existe une autre règle R' = ($A'_{1,k1}$, $A'_{2,k2}$, ..., $A'_{n,kn}$, C'_k) telle que $C'_k \subset C_k$ et pour chaque prémisse, on a $A_{i,ki} \subset A'_{i,ki}$.

Le conflit (règles ayant les mêmes prémisses) peut être réglé en prenant celle qui possède le plus grand degré [Wang 97].

L'étape suivante consiste à amplifier chaque règle ($A_{1,k1}$, $A_{2,k2}$, ..., $A_{n,kn}$, C_k) en remplaçant $A_{i,ki}$ par ($A_{i,p}$ ou $A_{i,q}$ ou $A_{i,r}$ ...) disjonction de tous les prédicats relatifs à x_i qui ne sont présents dans aucune autre règle, puis par ¬($A_{i,s}$ ou $A_{i,t}$ ...) si cette expression équivalente est plus simple.

Naturellement cette méthode ne donnera de bons résultats que si elle est raffinée par une optimisation de la définition des prédicats, ce qui est plus difficile à réaliser.

LA RECHERCHE DE RÈGLES PAR ITÉRATION DE DICHOTOMIES, (ALGORITHME ID3)

Le principe de cet algorithme [Quinlan 79], est de créer un arbre de classification à partir d'un ensemble d'exemples dont chacun est une liste de descripteurs. Prenons l'exemple de 8 appartements caractérisés par :

Exemple	Ensoleillement	Taille	Balcon	Prix
A1	non	studio	oui	cher
A2	oui	studio	non	moyen
A3	oui	2 pièces	oui	moyen
A4	non	appartement	oui	moyen
A5	oui	appartement	oui	moyen
A6	oui	studio	oui	cher
A7	oui	appartement	non	moyen
A8	non	studio	non	moyen

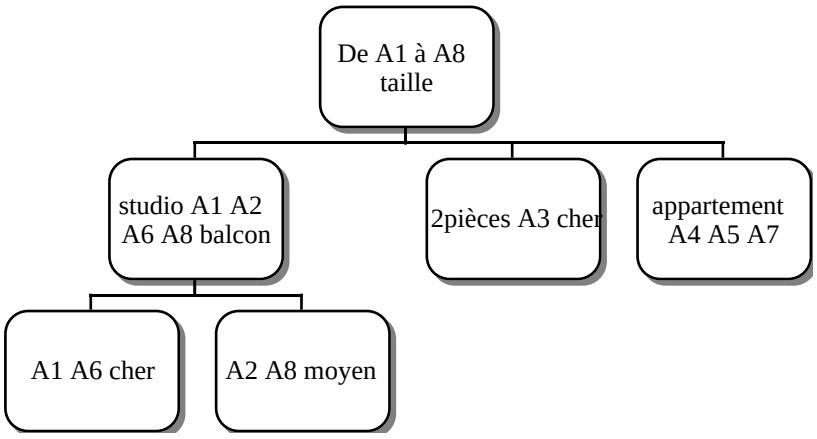

Figure 5.3 Exemple d'arbre résultant d'un choix d'attributs.

On crée un arbre de décision en choisissant l'un des attributs (par exemple la taille) avec autant de fils qu'il y a de valeurs (ici studio, 2pièces, appart) et on recommence pour chaque noeud tant que la partie des exemples concernés par ce noeud n'est pas uniforme du point de vue d'un attribut fixé à l'avance, (par exemple on veut des règles déterminant des tranches de prix). L'inconvénient de cet algorithme est que la forme de l'arbre dépend des choix successifs et reste limité à des problèmes de classification assez simples.
En fait il faut calculer la séquence d'attributs commune à tous les chemins partant de la racine, permettant d'obtenir l'arbre minimal. Pour des données symboliques, l'algorithme est facile à implémenter et donne de bons résultats. Généralement, on décide à chaque noeud de l'arbre, de sélectionner le prochain attribut comme celui qui apporte le plus d'information (entropie minimale) $I(a) = - \log p(a)$, donc l'attribut «a» qui a la probabilité minimale.

LE CONCEPT D'ENTROPIE

Exemple, pour coder 32 cartes, 5 questions suffisent pour choisir des questions discriminantes, n questions suffisent afin de déterminer un objet particulier parmi 2^n, on dira que la quantité d'information pour identifier un objet dans un ensemble de m éléments est $I(E_m) = \log_2 (m)$. Dans toute la suite, les logarithmes choisis sont dans une base supérieure à 1 quelconque.
Pour une variable aléatoire discrète, elle est définie par

$$I(X) = -\sum_X pr(X = x)\log(pr(X = x))$$

L'entropie conditionnelle de X connaissant Y est :

$$I(X / Y) = -\sum_j pr(y_j) \sum_i pr(x_i / y_j) \log(pr(x_i / y_j))$$

Si p est une mesure de probabilité, l'entropie de Shannon (1948) pour une partie finie A est :

$$\mathbf{H(A) = - \sum_{x \in A} p(x).\log_2 p(x)}$$

C'est donc une fonction dans R+, sur un univers continu avec une densité de probabilité f, ce sera alors :

$$H(A) = - \int_{x \in A} f(x).\log_2 f(x).dx$$

Cette notion a été élargie à un ensemble flou F (entropie de Higashi-Klir) avec :

$$I(F) = \int_{0 \leq \alpha \leq 1} \log_2(\text{card } F_\alpha).d\alpha$$

On a définit également une «mesure de confusion» pour une masse m par :

$C(m) = - \sum_{A \in F} m(A).\log_2 cr(A)$ où F est l'ensemble des parties focales ainsi qu'une «mesure de dissonance» $E(m) = -\sum_{A \in F} m(A).\log_2 pl(A)$ où cr et pl sont définis comme au chapitre 2.2.

L'entropie est en outre élargie à la détermination d'une classe parmi deux classes d'effectifs n et m avec :

$H = -n / (n+m) \log(n / (n+m)) - m / (n+m)\log(m / (n+m))$

En particulier si des classes $C_1, \ldots C_m$ sont déterminées dans un ensemble et qu'un attribut A peut prendre les valeurs $v_1, \ldots, v_p$, on définiera l'entropie conditionnelle par rapport à une valeur par $d(v_i) = - \sum_{1 \leq j \leq m} p(c_j / v_i) \log p(c_j / v_i)$ qui est une mesure du désordre local lorsque l'attribut vaut v_i.
Une mesure du désordre global est alors l'entropie moyenne par rapport à l'attribut A à savoir $H (C / A) = -\sum_{1 \leq i \leq p} p(v_i) \sum_{1 \leq j \leq m} p(c_j / v_i) \log p(c_j / v_i)$ sa minimisation peut rendre le service de guider un apprentissage. Enfin on peut définir un :

$$\text{gain } (A) = H - H(C / A).$$

Dans le cas où chaque classe est déterminée par une valeur précise de l'attribut, on a H (C / A) qui est nulle, c'est la valeur minimale pour le «désordre» et il est maximal lorsque toutes les décisions sont équiprobables (c'est d'ailleurs 1 si on prend le logarithme en base p). On a donc une notion adéquate que l'on peut chercher à minimiser grâce à un algorithme de recherche opérationnelle.
Au cas où les classes sont floues, l'entropie d'une partition $C = \{C_i / 1 \leq i \leq m\}$ en classes floues est $H(C) = - \sum p(C_i) \log p(C_i)$ avec $p(A) = \int_{x \in U} \mu_A(x).dp(x)$.
D'autres définitions d'entropie ayant la même propriété de minimalité pour une probabilité 1, les autres nulles et de maximalité pour équidistribution, peuvent être données, ainsi $1 - \sup\{p_i\}$, ou bien $\sum p_i(1 - p_i)$ (entropie quadratique), ou encore $\sum\sum_{i < j} (p_i p_j)^{1/2}$ (entropie de Bhattacharyya).

LA «RELAXATION» Basée sur l'entropie, cette méthode consiste à laisser évoluer un système formé d'un certain nombre d'exemples x classés en m classes en minimisant

une «certaine énergie» qui peut être l'entropie de Shannon $H(x, C) = -\sum \mu_i(x) \log \mu_i(x)$ pour chaque x, cette entropie, qui est une mesure du désordre, étant nulle lorsque x est vraiment dans une classe et maximale si x est équidistribué dans toutes les classes de la partition $C = \{C_1, C_2, ..., C_m\}$.
La mesure globale du désordre, si aucun μ_{ik} n'est nul, est l'entropie de la partition :

$$H(C, X) = -\sum_{1 \le k \le n} \sum_{1 \le i \le m} \mu_{ik} \log(\mu_{ik}) / n \text{ comprise entre 0 et } \log(m).$$

Elle est nulle dans le cas d'une seule classe, et maximale dans le cas d'une équidistribution (on peut rendre ce maximum égal à 1 si on utilise le logarithme en base m).

ENTROPIE STAR Si $C = (c_1, ... , c_m)$ est une partition floue de E fini, card(E) = n, et card $(c_i) = \pi_i = \sum \mu_{ci}(x)$, on a donc une probabilité d'être dans la classe c_i qui s'exprime par $p_i = \pi_i / n$, et on définit [Ramdani 94] $I^* = -\sum p_i \log(p_i)$.
On montre $I^*(CC') = I^*(C).I^*(C')$, si C et C' sont deux partitions floues, pour les m.m' classes de CC'.
Si un attribut A admet les valeurs $v_1, ... , v_p$, on pose comme pour l'entropie conditionnelle :
$I^*(C / A) = -\sum_{1 \le i \le p} p(v_i) \sum_{1 \le j \le m} p(c_j / v_i) \log p(c_j / v_i)$ qui va mesurer le degré d'incertitude d'une prise de décision (dire dans quelle classe on est), connaissant les valeurs imprécises v_i. (Si A exact, alors I* = I).
Si I^*_i est l'entropie de la décision sachant que A est v_i, alors I*(C/A) est la moyenne de ces entropies $p_1 I^*_1 + p_2 I^*_2 + ... + p_p I^*_p$.
On a de plus les résultats :

I* nulle (minimale) ⇔ Chaque I^*_i nulle ⇔ Chaque v_i est entièrement déterminé

I* maximale ⇔ Indépendance stochastique entre les classes et les valeurs de A

Cas particulier de partition :
Si S est une relation de similarité dans E ensemble de n éléments, muni d'une distribution de probabilité p,

$$H(p / S) = -\sum_{1 \le i \le n} p_i \log \sum_{1 \le j \le n} S(x_i, x_j)$$

On a toujours $0 = H(P / S^0) \le H(P / S) \le H(P) = H(P / S^1)$ où les relations extrêmes sont les relations triviales $S^0(x, y) = 1$ (une seule classe) et :
$S^1(x, y) = [\text{si } (x = y) \text{ alors 1 sinon 0}]$ (n classes).

EXTRACTION DE RÈGLES FLOUES

Une classification à l'aide de l'algorithme ID3, lorsque les données sont imprécises et floues n'a que des performances réduites. Trouver des règles floues à partir d'un ensemble d'exemples (où les attributs sont donnés avec des degrés), pose d'abord le problème du partitionnement du domaine de chaque attribut numérique, et donc de la définition de prédicats flous sur chaque domaine. Trouver la meilleure coupure entre deux prédicats «petit» et «grand», trouver des fonctions d'appartenance qui les définissent, faire abstraction des cas pathologiques (on souhaite toujours des ensembles flous convexes) et pour finir adapter l'algorithme ci-dessus pour obtenir des règles floues, ne sont pas des problèmes simples.

En ce qui concerne la détermination des prédicats flous relatifs à l'univers d'un attribut, le problème se pose aussi bien dans le cas discret où des attributs Aj ont des valeurs exactes $V_{j1}, V_{j2}, ... V_{jk}$ que dans le cas continu où il faut trouver une ou

plusieurs coupures pour éventuellement déterminer les valeurs floues (petit, moyen, grand) sur lesquelles on pourrait fonder des classes. Une première méthode consiste à construire l'histogramme des valeurs prises par l'attribut A. Cet histogramme peut lui-même être flou dans la mesure où une valeur d'attribut pour un exemple donné serait pondéré par un coefficient. Les deux histogrammes aux bornes de l'univers de A sont extrapolés.
[Marsala 94] indique une méthode d'inférence automatique de partitions floues fondée sur la morphologie mathématique.
Coupure : la meilleure coupure peut être trouvée en minimisant l'entropie-star.
Arrêt : Une méthode simple à chaque noeud de l'arbre, consiste à arrêter la construction du sous-arbre en ce noeud si les exemples qui y sont représentés le sont à un seuil près tel que 90% (degré cumulé par «min» des degrés avec lequel cet exemple est arrivé en ce noeud depuis la racine) ou bien s'ils sont peu nombreux [Ishigashi 96].

EXEMPLE

Après un prétraitement consistant à séparer des chiffres manuscrits et en isoler de 1 à 6 éléments parmi 12 types, [Chi, Yan 96] obtiennent grâce à un arbre de 828 noeuds, un certain nombre de règles. Les types reconnus sont V (segment vertical), H (horizontal), P (montant), N (descendant), de type C, D, A, U, S, Z, O et «courbe» pour le reste. Une matrice fixée empiriquement indique que par exemple un segment N peut être à 1 un N, mais à 0.25 une ligne de type H et à 0.1 une des courbes S ou Z. Les règles sont du type «si le plus grand segment est de type O et le type du deuxième est C et l'ordonnée normalisée du centre de l'image est supérieure à 0.586 alors c'est un 6».

5.3. Réseaux de neurones et neuro-flou

LE PERCEPTRON

Le perceptron est une machine parallèle de conception déjà ancienne [MacCulloch, Pitts 43], [Rosenblatt 58], réalisant un modèle de réseau de neurones assez simplifié, dans lequel une première couche de cellules (rétine) recevant des informations binaires (0 ou 1) transmettent cette information à des cellules de décision (éventuellement une seule reconnaissant ou non par 0 ou 1 l'image sur la rétine) par l'intermédiaire d'une couche cachée (par la suite de plusieurs autres couches). Chaque cellule reçoit la somme des informations de la couche précédente, pondérée par des poids attachés à chacun des arcs, et modifie son état suivant que cette somme pondérée dépasse ou non un certain seuil. C'est le prototype du système auto-organisateur grâce à différentes stratégies de modification des poids en vue de minimiser l'erreur entre la sortie du réseau et la sortie attendue pour un exemple, ces réseaux dits «à couches» sont appliqués à l'apprentissage «supervisé» (on donne un lot d'exemples constitués d'entrées de données et des sorties correspondantes que l'on souhaite obtenir, base sur laquelle le réseau s'entraîne, puis on le teste sur une base dite de généralisation beaucoup plus vaste).

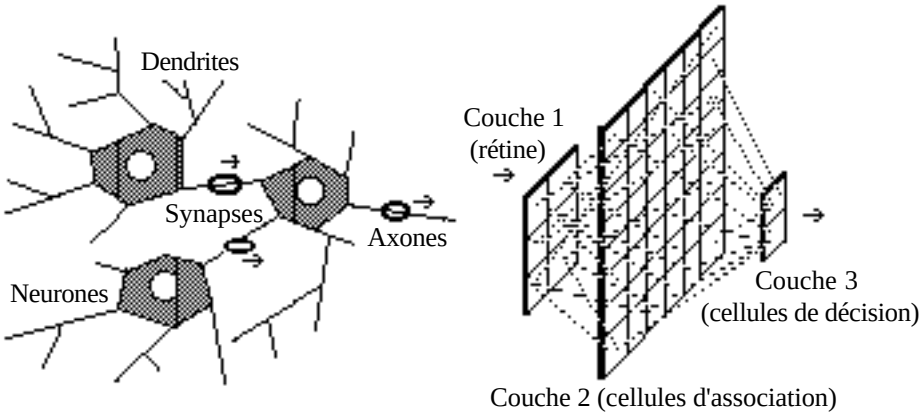

Figure 5.4 Neurones et liaisons synaptiques

Si maintenant on définit un état de la rétine par une fonction d'appartenance μ dans [0, 1] (par exemple des gris au lieu du noir et blanc), on pourra définir la propriété à reconnaître par un prédicat flou P sur les ensemble flous de la rétine, et le réseau constituant une fonction F, celle-ci ne sera plus booléenne, mais à valeurs dans l'intervalle [0, 1]. On pourra toujours dire que la propriété est vérifiée à $F(e_1, ... , e_n)$ près.

Exemple : si la rétine comporte 4 cases ternaires (noir ou gris ou blanc d'où 81 entrées possibles), on peut définir un prédicat P_1 «foncé» en donnant une fonction d'appartenance suivant le nombre de points (0 pour un blanc, 1/8 pour un gris, et 1/4 pour un noir). Par ailleurs un prédicat P_2 «pauvre en symétrie» par 5 valeurs 1 - nbsym/4, alors la reconnaissance des figures foncées et pauvres en symétrie au seuil de 0,6 se fera par $\min(p_1, p_2) \geq 0,6$.

Le perceptron à une couche caché est capable de reconnaître des classes linéairement séparables.

LES RÉSEAUX À COUCHES

Les réseaux de neurones sont donc des schématisations de cerveaux, dont on postule qu'ils sont constitués d'un très grand nombre d'unités simples en interaction. L'un des modèles les plus séduisants et les plus utilisés pour la reconnaissance des formes, est celui du réseau à couches (la plupart du temps une ou deux couches cachées). Dans chacune des couches, chaque neurone est relié à ceux de la couche précédente dont il reçoit les informations, et à chaque neurone de la couche suivante à qui il transmet des informations, mais il n'est pas relié aux autres neurones de sa propre couche. On considère que chaque neurone reçoit par ses «dendrites» une certaine activation électrique dont une somme pondérée constitue l'entrée e (un certain poids $w_{i,j}$ sera affectée à la liaison entre les neurones i et j). Par suite, le neurone passe à un certain «état» s = f(e) qui sera la mesure de sa sortie portée par son «axone» (f peut être modelisée par une simple fonction de seuil ou par une fonction du type arctangente ou tangente hyperbolique). L'axone transmet ensuite sa valeur par l'intermédiaire de «synapses» aux dendrites d'autres neurones. (Les poids mesurent en fait l'efficacité des synapses) L'apprentissage du réseau, consiste en une

modification de ces poids au fur et à mesure des expériences, c'est à dire de la confrontation entre le vecteur sortant de la dernière couche et celui qui est attendu en fonction d'un vecteur d'entrée.

EXEMPLE : on voudrait qu'en traçant sur une grille carrée des dessins formés par une seule ligne, on ait en sortie deux neurones dont le premier réponde 1 si la courbe est fermée et 0 sinon, le second donnant 1 s'il y a un point double et 0 sinon.

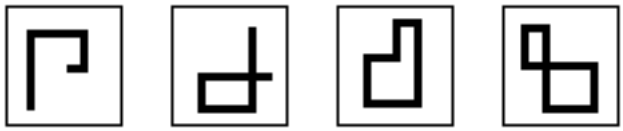

Figure 5.5 Dessins codés successivement par 00, 01, 10 et 11.

Les réseaux à couches sont surtout utilisés en reconnaissance de formes (écriture manuscrite ...)

ALGORITHME DE RÉTROPROPAGATION [Rumelhart 86], [Le Cun 87]

Les différentes règles (Hebb, Widrow-Hoff) de modification des poids utilisées cherchent à exprimer l'erreur entre la sortie réelle et la sortie désirée puis à calculer la contribution à cette erreur de chaque poids comme dérivée partielle. Pour Widrow, si l'entrée est $(x_1, x_2, ..., x_m)$ d'un réseau à deux couches, et l'erreur globale $E = \sum_{1\leq i\leq p}(y_i - s_i)^2$ alors, après calcul de la dérivée partielle de E par rapport à w_{ij} , la règle est de modifier chaque poids w_{ij} en $w_{ij} - \eta(s_j - y_j)x_i$.

Dans un réseau multicouche, chaque neurone n, à l'intérieur, ayant $e = \sum w_{i,n}s_i$ pour entrée, où i parcourt les neurones de la couche précédente, aura une sortie $s = f(e)$. Lorsque l'on propose le vecteur $X= (x_1, x_2, ..., x_m)$, à la première couche, la dernière restitue le vecteur $S = (s_1,..., s_p)$ alors qu'on attend $Y = (y_1,..., y_p)$ comme réponse.
Le but de cet algorithme est d'exprimer l'erreur quadratique $E = \sum_{1\leq i\leq p}(y_i - s_i)^2$ et de chercher à la minimiser en modifiant chaque poids w, suivant l'influence qu'il a sur E : $w(t + dt) - w(t) = -\eta.\partial E/\partial w$ où η est un coefficient positif, le «pas» du gradient. Le choix de ce pas est délicat (il est en général pris assez grand entre 0.01 et 0.1 au début de l'apprentissage pour être ensuite diminué.
On cherche donc à exprimer ce gradient qui est le vecteur formé par tous les $\partial E/\partial w$
Plaçons nous entre le neurone i d'une couche et un neurone n fixé :

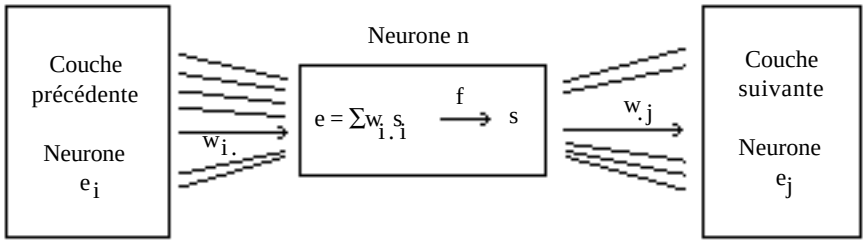

Figure 5.6 Schéma d'un neurone formel.

Pour la commodité de la lecture, on notera i l'indice parcourant la couche précédente, et j celui de la couche suivante, si d_n est la dérivée partielle de l'erreur E par rapport à e, on a :

$$\frac{\partial E}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.\frac{\partial e}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.\frac{\partial \sum_j w_{j,n}.s_j}{\partial w_{i,n}} = \frac{\partial E}{\partial e}.s_i = d_n.s_i$$

Car tous les termes de la sommation pour $j \neq i$ ne dépendent pas de $w_{i,n}$.

Si le neurone n'est pas en sortie, alors on calcule :

$$d_n = \frac{\partial E}{\partial e} = \sum_{j \text{ suivant}} \frac{\partial E}{\partial e_j}.\frac{\partial e_j}{\partial e} = \sum d_j \frac{\partial e_j}{\partial s}.\frac{\partial s}{\partial e} = \sum d_j .w_{n,j}.f'(e)$$

Et si n est en sortie :

$$d_n = \frac{\partial E}{\partial e} = \frac{\partial \sum_{j=1}^{p} (Y_j - S_j)^2}{\partial e} = 2(S_n - Y_n).f'(e)$$

La règle d'apprentissage est donc à chaque présentation d'exemple, (X, Y), de mesurer la sortie S, l'erreur E, et de modifier chaque poids (de i à j) $w_{i,j}$ en le remplaçant par $w_{i,j} - \eta d_j s_i$ avec $d_n = (\Sigma d_j w_{n,j})f'(e_n)$ pour les indices j en aval de n, sauf si n est en sortie auquel cas $d_n = 2(S_n - Y_n)f'(e_n)$. Il y a rétro-propagation de l'erreur commise dans la mesure où les modifications vont devoir être effectuées de la dernière couche vers la première.

Ce processus est répété sur plusieurs exemples (X, Y) jusqu'à ce qu'il y ait convergence suivant un seuil fixé. Les poids initiaux sont en principe distribués aléatoirement autour de 0,5. Expérimentalement les meilleurs résultats, de manière assez générale, ont été obtenus pour des réseaux où le nombre de poids est grossièrement de l'ordre du nombre d'exemples, ainsi que pour la stratégie (dite stochastique) de rétropropagation après chaque présentation d'exemple. La stratégie par lots consiste au contraire à n'opérer de rétropropagation qu'après la présentation de tous les exemples, en cumulant les erreurs.

RÉSULTATS RÉCENTS

Le théorème de Cybenko (89) montre que l'ensemble des fonctions réalisées par un réseau à une seule couche cachée dont la fonction d'activation est continue et vérifie $f(-\infty) = 0$ et $f(+\infty) = 1$, est dense dans l'ensemble des fonctions continues à support compact (borné suffit) [Hornik 88, 89].

Cet ensemble est fermé dans l'ensemble de toutes les fonctions définies sur R^m dans le cas où l'activation est gaussienne (Poggio T. Girossi F.).

Le théorème de Palmer (91) montre enfin que les réseaux à une seule couche cachée suffisent pour les problèmes linéairement séparables, et deux couches cachées suffisent pour approcher n'importe quelle fonction.

RÉSEAU RBF «RADIAL BASIS FUNCTION NETWORK»

Ce sont des réseaux [Broomhead D.S. Lowe D. 88], [Poggio 90] et [Platt 91] à trois couches dont la première possède p neurones correspondant à la dimension des vecteurs d'entrées.

La couche cachée possède un nombre de neurones qui peut être le nombre n d'exemples proposés pour l'apprentissage ou un peu moins, (ce nombre peut même

être variable si on crée un neurone supplémentaire chaque fois que l'erreur dépasse un certain seuil) mais sa particularité est que les fonctions de transfert qui leur sont associées sont des fonctions gaussiennes avec un centre et un écart-type propre à chacun de ces neurones et qui sont modifiés en fonction des performances. La troisième couche constitue la sortie (un ou plusieurs neurones) telle que les poids associés constituent dans le cas d'une sortie, en quelque sorte les conclusions des «règles floues». Une méthode pour mettre au point ces fonctions gaussiennes est d'utiliser la rétropropagation, mais ce n'est pas la seule possible : les moyenne et variance constituent deux poids arrivant à chaque neurone, et qui vont être modifiés à chaque rétropropagation. Le but de l'apprentissage est que chaque neurone de la couche cachée devienne un spécialiste reconnaissant un exemple, il le laissera passer et non les autres.

Une autre méthode inspirée des moyennes floues a été proposée par [Mustawi 92] pour le problème suivant de classement. Supposons que l'on dispose de n exemples X_1, X_2, ..., X_n tous dans R^p, et que ces points-exemples soient classés parmi m classes déjà définies. Le réseau va être entrainé sur ces exemples par un algorithme dépendant des paramètres η et c, afin de généraliser par la suite.

0) Le réseau est formé de 3 couches dont la première possède p neurones (une entrée X est acceptée à raison de ses p composantes), la couche cachée possède n neurones et la couche de sortie m neurones correspondant aux m classes prédéfinies. A chaque exemple X_i, est associé un neurone de fonction matricielle de transfert :

$f(X) = \exp[- (1/2) (X - C_i)^t Q_i (X - C_i)] = \exp[- (1/2)\Sigma_{1 \leq k \leq p} (x_k - c_{ik})^2 / \sigma_{i,k}^2]$

où C_i est fixé initialement en X_i, et Q_i est la matrice diagonale des inverses des variances $\sigma_{i,k}^2$. $\sigma_{i,k}$ est l'écart-type de la gaussienne correspondant au i-ième neurone de la couche cachée pour la k-ième composante (il peut être initialement fixé à 1). On peut remarquer que f(x) étant l'exponentielle d'une somme est donc le produit (t-norme probabiliste) des degrés d'appartenance des différentes composantes du vecteur d'entrée X aux différents prédicats gaussiens.

1) Du premier au dernier de ces neurones cachés i :

2) On examine tour à tour tous les neurones j correspondant à des exemples x_j de la même classe. On envisage alors le regroupement des deux neurones en calculant le barycentre c_k entre c_i et c_j.

On calcule la distance r_k entre c_k et l'exemple de la même classe qui est le plus éloigné (rayon de la classe), ainsi que la distance d_k entre c_k et le barycentre de la classe distincte la plus proche.

Si $d_k > \eta r_k$ avec la constante de Mustawi η située entre 1 et 3, alors le regroupement des deux neurones est définitif, sinon on recommence en 2) avec un autre neurone j jusqu'à une acceptation ou bien jusqu'à ce que tous les neurones aient été examinés.

3) Revenir en 1) jusqu'à ce qu'il n'y ait plus de regroupement possible.

4) Le nombre et les centre c_i des neurones cachés étant déterminés, on prend alors $\sigma_i = cr_i$ avec une constante qui peut être expérimentalement fixée à la valeur c = 1.2.

Cette architecture et cet algorithme sont à l'heure actuelle employés en reconnaissance des formes en concurrence avec les réseaux multicouches à rétropropagation. Leur intérêt, pour le contrôle flou, est de réaliser une interpolation avec un petit nombre de neurones cachés [Cho, Wang 96].

RÉSEAU DE HOPFIELD [Hopfield 82], [Kurbel 94]

Il s'agit d'une structure adéquate à la reconnaissance de m classes par n neurones complètement connectés (il n'y a pas de couches). Soit un ensemble de n «neurones» d'état $x_i = \pm 1$, tous connectés entre eux par des «poids» $w_{i,j} = w_{j,i}$ avec $w_{i,i} = 0$. A chaque instant, l'entrée du neurone x_i, est la somme pondérée des états de tous les autres neurones qui lui sont connectés, c'est à dire $\Sigma_{1\leq j\leq n} w_{i,j} x_j$.
La règle d'évolution du réseau est donnée par la modification simultanée de tous les neurones suivant laquelle x_i est remplacé par signe de ($\Sigma_{1\leq j\leq n} w_{i,j} x_j$).
Une modification asynchronr des neurones est étudiée dans [Abdi 94].
Hopfield démontre la décroissante au cours de l'évolution de l'énergie du réseau définie par la fonction $H = (-1/2)\Sigma_{1\leq i,j\leq n} x_i x_j$ et que pour un état initial, le réseau converge vers un état «attracteur».
Soit maintenant un ensemble X formés par m vecteurs de $\{-1, 1\}^n$ appelés prototypes, le problème est de constituer un réseau susceptible de reconnaître des vecteurs voisins (bruités) de ces prototypes. On peut alors montrer que chacun de ces prototypes possède un «bassin attracteur» dans lequel des vecteurs bruités (correspondant à un état du réseau) vont évoluer avec une convergence plus ou moins rapide grâce à la règle ci-dessus vers un des exemples de X. Appelons S la matrice n*m formée par les vecteurs exemples (et S^t sa matrice transposée), son élément générique $s_{i,j}$ désigne donc la i-ième composante de l'exemple j.
On définit $W = SS^t - I_n$ la matrice des poids, et on voudrait obtenir une stabilité s'exprimant par un «point fixe» S, c'est à dire WS = S. W possède une diagonale nulle $w_{i,\,i} = 0$ et soit $w_{i,\,j}$ l'élément i, j de SS^t c'est à dire $\Sigma_{1 \leq k \leq m} s_{ik}s_{jk}$
On remarque que pour un rang k entre 1 et m,
$\Sigma_{1 \leq j \leq n} w_{ij}s_{ik} = \Sigma_{1 \leq j \leq n}(\Sigma_{1 \leq p \leq m} s_{ip}s_{jp})s_{jk} = \Sigma_{1 \leq j \leq n} \Sigma_{1 \leq p \leq m} s_{ip}(s_{jp}s_{jk}) = \Sigma_{1 \leq p \leq m} s_{ip} \Sigma_{1 \leq j \leq n} (s_{jp}s_{jk})$
Dans le cas où les exemples sont deux à deux orthogonaux c'est à dire si $SS^t = I$, alors cette dernière expression est $s_{i,k}$ ce qui signifie que le réseau est stable (chaque état de neurone est égal à son entrée). En fait une hypothèse plus large sur le fait d'avoir le même signe suffit.
Une autre façon d'initialiser les poids est de prendre (théorème de Penrose) la matrice de poids $W = SS^+ + Z(I_n - SS^+)$ où Z est quelconque n*n et S^+ est la pseudo-inverse de S, car alors $WS = SS^+S + ZS - ZSS^+S = S$ Empiriquement de bons résultats sont obtenus pour la proportion $0.03\ n \leq m \leq 0.14\ n$, au delà, le réseau a tendance à oublier.

LES RÉSEAUX RÉCURRENTS

Il s'agit d'un réseau [Elman 90] très simple avec une couche d'entrée, une couche cachée et une couche de sortie, utilisant l'algorithme de rétropropagation. Cependant il existe une autre couche dite de «contexte» qui est une simple recopie de la couche cachée, de telle sorte qu'à chaque propagation, la couche cachée recoit les entrées

accompagnée du «souvenir» de l'état de la couche cachée à l'étape antérieure. Le nouvel état de la couche cachée est alors recopié dans la couche de contexte et la rétropropagation est opérée de façon classique. Une autre architecture a été réalisée où la couche d'entrée est cette fois accompagnée du souvenir de la couche de sortie relative à l'étape antérieure. Ces réseaux ont été utilisés pour le traitement des séries temporelles, mais aussi en informatique linguistique.

RÉSEAU DE KOHONEN

Le but de cette architecture [Kohonen 89] est un apprentissage cette fois non supervisé, où les entrées possibles sont classées. Il n'y a que deux couches, l'entrée $(x_1, \ldots, x_n)$ comportant autant de neurones que la dimension n des exemples, et la sortie comportant p neurones en compétition. On définit le voisinage V_j de la sortie j, par exemple si p = 16, chaque neurone peut avoir son voisinage décrit par ses 4 voisins, puis 3, puis 2. Chaque poids w_{ij} est modifié par la règle suivante :

A chaque sortie j (avec $1 \leq j \leq p$), on calcule la distance $\sum_{1 \leq i \leq n} (x_i - w_{ij})^2$ et si j_0 est la sortie pour laquelle cette distance est minimale (le neurone «vainqueur»), on met les poids à jour pour tous les i, mais seulement pour les j dans le voisinage de j_0 par la règle $w_{ij} \leftarrow w_{ij} + \eta(x_i - w_{ij})$. Ainsi donc le neurone j_0 se spécialise pour la reconnaissance d'une certaine classe de données en entrée, mais aussi ses neurones voisins, dans une moindre mesure.

Puis on fait décroître la taille du voisinage, et enfin η qui mesure le taux d'apprentissage, doit décroître aussi de 0.8 initialement avec un décrement de 0.005 par exemple. La convergence doit se faire en une dizaine de cycles vers un réseau reconnaissant p classes.

LE RÉSEAU NEURO-FLOU

Plusieurs structures [Sanchez 90], [Berenji 92], [Yamakawa, Uchino, Miki, Kusagani 92] ont été proposées, nous présentons la plus claire de [Glorennec 91]. L'esprit de ce réseau est d'intégrer a priori, une connaissance même imparfaite, mais permettant un apprentissage plus rapide qu'avec des poids initiaux aléatoires. Un réseau neuro-flou est donc un réseau de 4 couches (3, 11, 25, 1) conçu de la façon suivante pour deux entrées exactes x, y il fournira une sortie u :

(Si n prédicats et p entrées, on aura un réseau $p + 1, np + 1, n^p, 1$).

La couche 1 possède les deux entrées x, y et un troisième neurone toujours à 1.

La couche 2 est constituée par les 10 états $\mu_{Ai}(x)$, $\mu_{Bi}(y)$ dans l'hypothèse où on a choisi 5 prédicats NB, NS, ZE, PS, PB. Ceux-ci sont gaussiens donc entièrement déterminés par des couples (a_i, b_i) de réels fixés initialement par exemple à (a, a), (a, a/2), (a, 0), (a, -a/2), (a, -a).

Cela constitue une famille de prédicats se coupant à la hauteur $\exp(-a^2/16)$ soit 0.5 si $a = 4 \ln^{1/2}(2)$.

a_i est le poids initial depuis x, et b_i celui depuis y vers chaque neurone de la seconde couche. Le second terme du couple est le poids initial du neurone fixe 1 vers chacun de ceux de la seconde couche. Si la fonction de transfert est donnée par $f(t) = \exp(-t^2)$ on obtient bien les prédicats gaussiens en sortie c'est à dire les états.

Ainsi l'état du premier neurone de la seconde couche est $f(ax + b) = \exp(-(ax + b)^2) = \mu_A(x)$ pour un certain prédicat gaussien A.

Les poids vers la 3° couche sont tous fixés à 1.
Un 11° neurone est fixé à 1 et les poids le joignant à la couche aval sont fixés à -1.
La couche 3 est constituée par 25 règles a priori (si 2 entrées et 5 prédicats).
Chaque entrée sera donc $\mu_A(x) + \mu_B(y) - 1$.

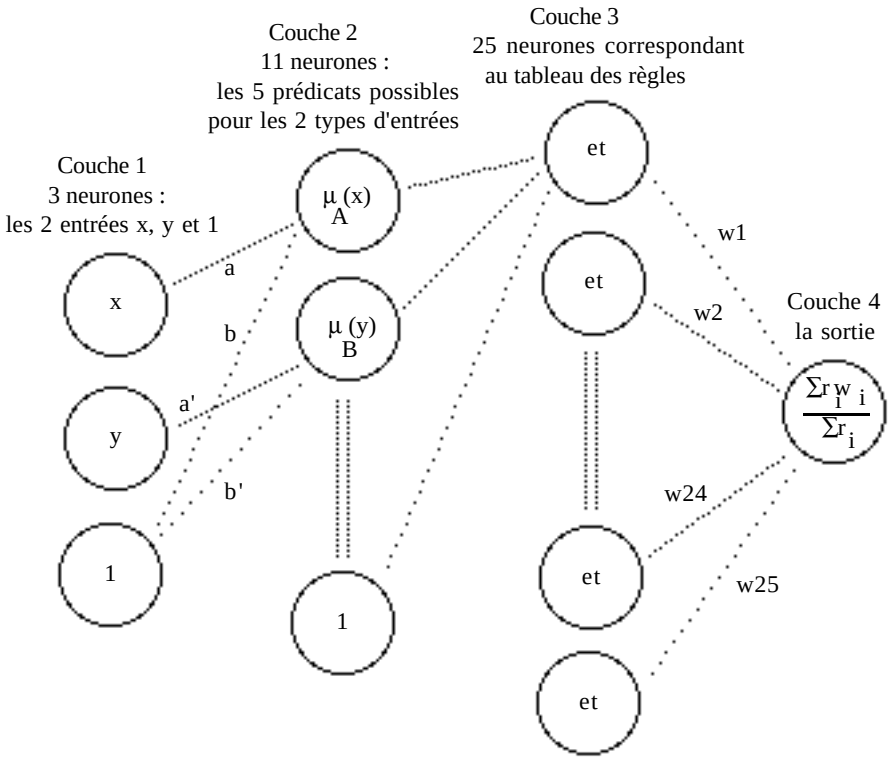

Figure 5.9 Schéma général du réseau neuro-flou.

La fonction de transfert de la troisième couche est $g(t) = 1 / (1 + \exp(2 - 4t))$ elle approche ainsi $\max(0, t)$. Par ce stratagème, la sortie de chaque neurone réalise donc approximativement une version lissée (c'est à dire dérivable) du «et» de Lukasiewicz.
Les 25 derniers poids $w_1,...,w_{25}$ sont mis initialement de façon intuitive comme conclusions numériques des 25 règles. Si r_i est le niveau de satisfaction de la i-ième règle l'entrée du dernier neurone est $\sum r_i * w_i$.
La sortie sera alors calculée comme la moyenne $(\sum r_i * w_i) / (\sum r_i)$.

La différence avec l'algorithme de rétropropagation est que les fonctions de transfert sont distinctes à chaque couche et que la dernière est variable, son calcul est bien spécifique de la méthode de Sugeno.
Cette architecture a été appliquée par Berenji au pendule inversé avec 4 entrée (x, x', θ, θ'). Elle est testée dans bien d'autres domaines, notamment au domaine bancaire.

Tous les poids peuvent changer au cours de l'apprentissage y compris ceux du «et», cependant il est prudent de n'appliquer la rétropropagation que par étape, après un tel apprentissage séparé (couche par couche) [Brunet 96] a pu observer une déformation de la famille des 5 prédicats NB, NS, ZE, PS, PB vers une stabilisation dans laquelle NS tend à se confondre avec NB, PS avec PB et où on a une perte de la symétrie aussi bien en ce qui concerne les valeurs des poids en sortie que pour ZE légèrement déplacé vers la gauche.

REMARQUE , La sigmoïde de pente p en (m, 1/2) est peut être donnée par la fonction $f(t) = 1 / [1 + \exp(-4p(t - m))]$.

LA MÉTHODE DU GRADIENT STOCHASTIQUE

Un réseau (2, 10, 25, 1) est constitué de la même façon, mais avec la véritable t-norme choisie et des prédicats triangulaires, seuls les poids de la dernière couche seront modifiés grâce à :

$\Delta w(t) = -\varepsilon[y(t) - d(t)]r_i / \sum r_i + \eta\Delta w(t-1)$ où $r_i / \sum r_i$ est la force relative de la règle y, la sortie réelle et d la sortie désirée. ε et η sont le «gain» et le «moment». (La méthode du gradient de Cauchy consiste à suivre la ligne de plus grande pente en chaque point pour optimiser une fonction différentiable).
L'intérêt de la méthode par rapport à la rétropropagation classique est la diminution du calcul et aussi que pour deux entrées, si la famille de prédicats est une partition floue, alors de 1 à 4 règles sont modifiées à chaque fois seulement.

5.4. Algorithmes génétiques et stratégies d'évolution

Les algorithmes génétiques [Bagley 67], [Goldberg 82, 89] sont inspirés par les lois de l'évolution en biologie où par croisements, la population d'une espèce se renouvelle et où des mutations accidentelles de très faibles probabilité peuvent modifier dans un sens ou dans l'autre les aptitudes d'un individu et par suite favoriser ou non la possibilité qu'il aura de se reproduire. Un individu muté dans le sens d'une plus grande performance (suivant certains critères, ne serait-ce que la longévité) aura alors plus de chance de se reproduire et donc de transmettre à sa descendance son nouveau code.
C'est la fameuse théorie darwinienne de la sélection naturelle [Monod 70], [Dessalles 96]. Cette théorie a récemment inspiré une méthode stochastique appliquée à la résolution des problèmes non seulement mal aisés à résoudre mais à définir. C'est avec un certain succès que des problèmes mal posés se voient offrir des solutions souvent inatendues par une simulation d'évolution :
Pour un problème quelconque d'optimisation, les solutions vont d'abord être codées sous forme de chaînes de caractères appelées «chromosomes», les caractères étant les «gènes». L'algorithme consiste alors à reproduire par étapes une «population» P de chromosomes.
Au départ on prendra donc des chromosomes plus ou moins arbitraires, générés aléatoirement ou réalisant déjà un semblant de solution intuitive, ou bien de réelles solutions mais non optimales.

De génération en génération, la population doit s'améliorer suivant une fonction d'évaluation dictée par le problème, on peut alors classer la population suivant cette fonction pour permettre aux meilleurs chromosomes de se reproduire.
L'illustration est assez simple à faire sur le problème de la minimisation d'une fonction numérique définie sur l'intervalle [a, b]. Toutes les méthodes de descentes du gradient (adaptées aux fonctions différentiables et convexes), ou bien les méthodes stochastiques telle que celle du recuit simulé consistent à itérer l'évaluation de la fonction sur une suite de points «voisins» en partant d'un point initial arbitraire. Si la fonction n'est pas convexe et présente plusieurs minima, on ne peut garantir la convergence vers le minimum global même si une certaine remontée est tolérée suivant une probabilité à définir comme c'est le cas dans le recuit simulé.

Nous signalons à présent deux méthodes d'optimisation présentant un certain rapport avec les algorithmes génétiques qui suivent.

MÉTHODES STOCHASTIQUES D'OPTIMISATION, LE RECUIT SIMULÉ

Cette méthode [Kirkpartrick 83], [Aarts Korst 89], [Reeves 93] consiste à obtenir par itérations successives le minimum absolu d'une fonction, elle est inspiré d'une technique de refroidissement consistant à accepter dans certains cas une remontée de la fonction (pour ne pas descendre trop vite vers un minimum local). En thermodynamique la probabilité d'avoir une augmentation d'énergie ΔE (la fonction que l'on veut minimiser suivant la loi de Boltzmann) est $e^{-\Delta E/\theta}$. On accepte un état voisin s_1 augmentant la fonction, dans la mesure où la probabilité ci-dessus est décroissante suivant ΔE. Le paramètre de contrôle (la température) va décroître, ce qui fait que pour un même ΔE, la probabilité d'accepter une remontée diminue suivant le refroidissement. Plus précisément :
a) Choisir un état initial s_0
b) Faire le «palier» consistant à répéter n fois
Chercher un voisin s_1 de s_0, on calcule $\Delta = f(s_1) - f(s_0)$
Si $\Delta < 0$ alors $s_0 \leftarrow s_1$
Si $\Delta > 0$ alors on accepte la même affectation de s_0
avec une probabilité $e^{-\Delta/\theta}$
c) On abaisse la température θ et on réitère ces paliers jusqu'à un test d'arrêt défini par le problème.
La difficulté de cette méthode, qui par ailleurs donne de bons résultats, réside dans la détermination des paramètres. On peut montrer la convergence sous certaines conditions reliant n et la loi de décroissance de la température.
La température initiale peut être déterminée par une probabilité, par exemple de 1/2, d'accepter une hausse au cours du premier palier, si m est la valeur moyenne de ces hausses, alors $\theta_0 = m / \ln(2)$. Le nombre d'itérations sur chaque palier peut être de l'ordre de la centaine, la loi de décroissance de la température est généralement prise comme une suite géométrique telle que $\theta_{n+1} = 0{,}9\ \theta_n$

MÉTHODE TABOU

On part de la même façon d'une «solution» initiale quelconque x. A chaque point courant x du domaine de définition du problème, on cherche grâce à une discrétisation du problème, le meilleur voisin non tabou de x (ce sera le point suivant). La liste des «tabous» est simplement constituée d'états antérieurs sur lesquels on ne souhaite pas revenir. Cette liste doit avoir une taille fixe et chaque état n'y reste donc qu'un nombre fini d'itérations. On la gère en file : chaque mise-à-jour y place un nouveau venu et en retire le plus ancien [Glover 86].

Les algorithmes génétiques et plus généralement toutes les stratégies d'évolution partent d'une population de points (10 sur la figure ci-dessous), il est clair que moyennant un temps de calcul multiplié, les chances d'obtenir plusieurs minima et le minimum global en sont augmentés. Mais ces stratégies ne se contentent pas de poursuivre des itérations séparées dans plusieurs régions de «l'espace de recherche» en recréant des «niches écologiques» si l'on poursuit l'image de l'évolution naturelle des espèces, elles innovent dans l'exploration de cet espace de recherche en se croisant et en brouillant les éventuelles solutions par des «opérateurs génétiques».

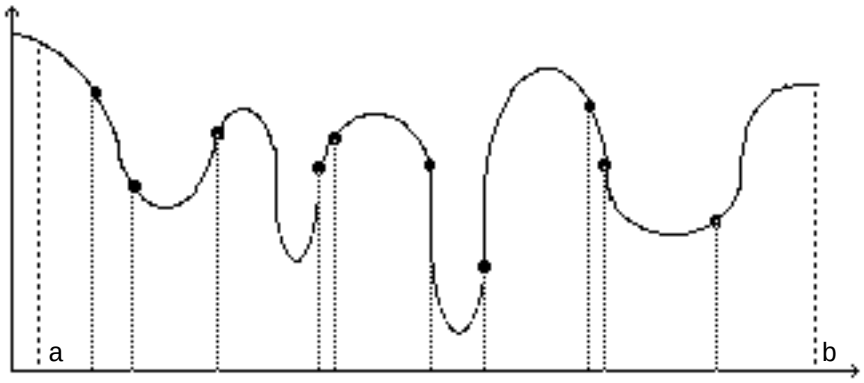

Figure 5.10 Population aléatoire dans un intervalle

LES STRATÉGIES D'ÉVOLUTION EN VUE D'UNE OPTIMISATION

Les différentes stratégies se distinguent sur les 7 points suivant :

a) Définition du problème : la fonction d'évaluation.
Naturellement les algorithmes génétiques seront utilisés pour l'optimisation de fonctions non triviales (non continues, bruitées et multimodales), sans propriétés de dérivabilité connues et définies sur des domaines plus complexes qu'un intervalle. Elles sont utilisées là où les méthodes analytiques ont échoué, mais aussi sur les problèmes symboliques pour lesquels les méthodes de dénombrement sont trop coûteuses. En fait pour des problèmes d'optimisation multicritère, nous verrons plus loin qu'il est très difficile de donner une fonction d'évaluation, le seul fait de bien poser le problème, n'est pas trivial.

b) Codage des solutions

Les algorithmes génétiques ont débuté avec un codage binaire des éventuelles solutions de façon à ce qu'il ait une grande possibilité d'exploration de l'espace de recherche avec simplement de petites transformations comme la mutation consistant à modifier un 1 en 0 ou le contraire au sein de la chaîne chromosomique. Si les codages les plus adaptés aux stratégies que l'on a en tête ne sont pas ceux qui permettent la plus grande capacité d'exploration, le manque de signification est l'inconvénient du codage binaire. En fait, comme cela a été pratiqué par la suite sous le nom d'algorithmes évolutifs, le codage peut être quelconque ainsi un nombre peut être codé par sa suite décimale et rester «lisible» et une solution d'un problème symbolique peut rester une suite finie (ou non) de symboles. Le «génotype» ou «génome» est l'ensemble des gènes, la question est d'établir le lien avec le «phénotype» ou ensemble des caractéristiques de morphologie et de comportement.

c) Choix de la population initiale

On peut partir d'un ensemble de solutions approchées déjà connues, mais d'une manière générale il est souvent avantageux de partir d'une population aléatoire (sa taille peut aller de quelques dizaines à quelques centaines). Dans l'algorithme «standard» qui est une reproduction assez fidèle de l'évolution, la population est toujours de grande taille, mais avec beaucoup de recopies d'individus identiques. [Cerf 94] montre que pour ces algorithmes, la vitesse de convergence est de l'ordre de la taille de la population.

d) Choix des opérateurs

Les transitions entre générations se font par «opérateurs génétiques» c'est à dire des «fonctions aléatoires» de $P \rightarrow P$ ou $P^2 \rightarrow P^2$ si P est la population.

L'opérateur le plus simple est la mutation, un gène est remplacé par un autre aléatoirement choisi, par exemple au chromosome AB**C**BBAC → AB**E**BBAC. Des mutations bien particulières peuvent être envisagées suivant le problème, ainsi la transposition : deux gènes aléatoirement choisis sont permutés AB**C**EB**A**CD → AB**A**EB**C**CD, ou l'inversion : une coupure est choisie et les deux parties sont permutées ABCB**DEA** → **DEA**ABCB, l'inversion partielle : une partie du chromosome est inversée ABCD**EFD**CD → ABCD**DFE**CD etc.

Le cross-over : deux parents engendrent deux enfants en échangeant une partie choisie aléatoirementAA**BED**ABC, DA**C B C** DAE → AA**C B C** ABC, DA**BED**DAE

C'est l'opérateur le plus puissant car intuitivement cela correspond à deux solutions comportant chacune une bonne partie du code, aussi en regroupant ces parties peut-on améliorer la solution globale.

La reproduction : si les valeurs des individus sont $v_1, v_2, ..., v_n$ et que $p_i = v_i / (\sum v_k)$, en tirant au hasard dans la population un numéro entre 1 et n avec cette distribution $(p_1, p_2, ..., p_n)$ le chromosome choisi est tout simplement recopié à la génération suivante.

D'autres opérateurs peuvent être imaginés notamment des opérateurs respectant la cohérence dans la représentation du problème en chromosomes (exemple : respecter un aspect injectif ou bijectif...).

Les opérateurs non uniformes [Michalewicz 92], consistent à effectuer par exemple des mutations de plus en plus faible, c'est à dire si l'ensemble des gènes est

métrisable, à opérer dans des voisinages de plus en plus fins lors du remplacement d'un gène par un autre.
Parmi toutes les idées, on peut citer celle du «poissson de corail» [Koza 94] consistant à déclarer «étalon» les 10% individus les plus performants et à réserver le croisement entre un étalon et un autre individu afin de favoriser le brassage et l'exploration de la population.

e) Mode d'application des opérateurs
Dans l'algorithme génétique désormais appelé «standard» les opérateurs sont appliqués suivant une certaine probabilité choisie au départ et suivant le principe de roue de loterie biaisée, on donne l'avantage aux meilleurs individus (stratégie «ranking»), le cardinal de la descendance d'un individu est a (en moyenne) pour le meilleur, et b pour le moins bon. La différence a - b $\in$ [1, 2] est appelée la pression de sélection, et il a été observé une uniformisation de la population d'autant plus rapide que cette pression est grande.
Afin de garder aussi longtemps que possible tous les minimums d'une fonction, il est possible de faire une partition de la population en «niches» [Horn 93] et d'effectuer périodiquement des «migrations» entre les sous-population [Franz 72].
Un autre point de vue est de changer la fonction d'évaluation [Goldberg 87] :
Si sh est une fonction de partage (sharing function) sur [0, 1] alors on peut voir comme une «accentuation» de la négation avec $sh(x) = 1 - x^{\alpha}$ où $\alpha < 1$ pour avoir $sh(x) < x$ ou $\alpha > 1$ pour le contraire $sh(x) > x$. Soit enfin $sh(x) = 0$ si $x > 1$.
Si n est la taille de la population et σ un paramètre déterminant la taille des niches, au cas où il est possible de définir une distance d dans l'espace de tous les chromosomes, alors [Horn 93] modifie la fonction d'évaluation. Si f_0 est l'évaluation à la génération courante où les individus sont $c_1, c_2, \ldots, c_n$ alors la nouvelle évaluation f_1 pour la génération suivante est :
$f_1(c_i) = f_0(c_i) / \sum_{1 \leq k \leq n} sh[d(c_i, c_k) / \sigma]$. Dans le but de maximiser cette évaluation, on peut voir que plus un chromosome possède d'autres chromosomes voisins, meilleure est son évaluation, en cherchant à pénaliser les zones denses, l'exploration est ainsi favorisée. [Goldberg, Richardson 87] ont montré qu'une stabilisation est observée quand les valeurs des maximums f(i) sont en proportion avec la taille des ensembles flous $m_i = \sum_{1 \leq k \leq n} sh[d(c_i, c_k) / \sigma]$ des niches.

f) Renouvellement des générations
A chaque étape ces opérateurs sont tirés et appliqués au hasard, on peut alors prendre les chromosomes et leurs images et éliminer les plus mauvais (ce qui fait par exemple deux suppressions pour quatre individus dans le cas du cross-over) ou alors trier toute la population et sa descendance et ne conserver que les meilleurs en assurant à P un cardinal constant.
La stratégie standard consiste à appliquer (suivant de très faibles probabilités) des mutations (plutôt, d'après ce qui précède, aux meilleurs individus issus du cross-over) et de garder une taille fixe pour la population, en supprimant les plus mauvais.
Les stratégies dites élitistes (évolution lamarckienne) consistent à trier immédiatement entre parents et enfants à chaque application d'un opérateur génétique, pour ne garder que les meilleurs. Mais on trouve également les stratégies étudiées depuis 1964 en Allemagne, suivant laquelle les opérateurs sont appliqués à

chaque génération à chacun des μ individus et produisent λ enfants (le rapport λ / μ étant de l'ordre de 7).
Le choix est alors la statégie $(\mu + \lambda)$ où μ parents produisent λ enfants, et le tout est trié à l'issue de la génération ou bien la stratégie (μ, λ) où les μ meilleurs de la descendance sont retenus, en ce cas, la meilleure performance n'est pas obligatoirement monotone comme pour les algorithmes standard [Bäck, Kursawe 94].

g) Arrêt de l'évolution
Il peut être décidé si la solution connue est obtenue à un seuil de précision fixé, ou bien au bout d'un nombre de générations limité ou encore après un nombre donné d'évaluations de la fonction. L'intérêt de la méthode est son applicabilité aux architectures parallèles.

EXEMPLE
Pour la fonction de Shaffer $f(x, y) = [1 - \sin^2\rho] / [1 + 0{,}001\rho^2]$ qui possède le maximum 1 pour (0, 0), (ρ est la distance à l'origine de (x, y)) il ne faut pas moins de 1500 évaluations avec un algorithme standard où $p_C = 0.6$ et $p_m = 0.001$, pour y accéder, compte tenu du grand nombre d'oscillations sur $[-100, 100]^2$.

REMARQUES Si $G = \{g_1, \ldots, g_{ng}\}$ est l'ensemble des gènes sur lequel on définit une distance $d(g_i, g_j) = \partial_{ij}$ ou par $|i - j| / (ng - 1)$ au cas où on dispose d'un ordre canonique sur G. Si G_n est l'ensemble des chromosomes, donc de topologie discrète, on peut cependant définir une distance dans [0, 1] par :
$d(C, C') = (\sum_{1 \le i \le n} d(g_i, g'_i))) / n$, on aura par exemple :
Pour une mutation (ou une recopie d'un gène à droite) $d(C, mut(C)) \le 1 / n$
Pour une transposition (un échange de deux gènes) $d(C, transpo(C)) \le 2 / n$
Pour un cross-over où k gènes sont échangés, si C'_1 and C'_2 sont les fils de C_1 et C_2 alors : $d(C_1, C'_1) \le k / n$ mais surtout : $\max [d(C_1, C'_1), d(C_1, C'_2), d(C_2, C'_1), d(C_2, C'_2)] \le d(C_1, C_2)$ se qui s'exprime par le fait que la distance entre les parents détermine deux voisinages dont l'intersection contient les enfants.

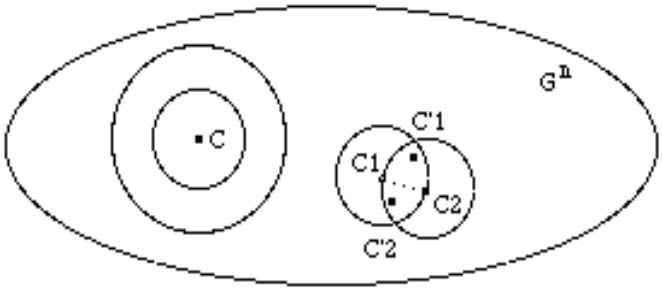

Figure 5.11 Dans l'espace métrique des chromosomes, visualisation de l'exploration par mutation, transposition et cross-over.

P ne doit pas être trop petit, ni la fonction d'évaluation trop restrictive sinon on risque de voir les solutions converger vers un optimum local.
La fréquence d'application de chacun des opérateurs doit être surveillé afin de d'utiliser les «bonnes» transitions entre générations.

Il a été proposé de mesurer la diversité de la population, si m = card(P), par l'indice $\partial(P) = (2 / m(m - 1)) \sum\sum_{i < j} d(C_i, C_j)$ (le cross-over ne la change pas) afin de trouver des opérateurs qui ne baissent pas trop cette diversité.

EXEMPLE, dans le problème classique du voyageur de commerce, une solution est une permutation de toutes les villes, ABCDEF par exemple s'il y en a 6. La fonction d'évaluation à minimiser est alors la somme des distances, mais les transitions devront être adaptées de façon à ce que les «solutions» restent toujours bijectives sur l'ensemble des villes. Les algorithmes génétiques sont appliqués à toutes sortes d'autres problèmes d'optimisation depuis la recherche du dessin d'un graphe minimisant le nombre d'arètes se coupant jusqu'au problème de l'ordonnancement d'atelier où il faut minimiser la charge totale (temps utilisé par chaque machine où différentes pièces occupent différentes machines pour des durées différentes et suivant des contraintes de préséances) [Ghedjati 94].

LE THÉORÈME FONDAMENTAL

Pour un alphabet de k symboles (plus le symbole * utilisé pour signifier un gène quelconque) et des chromosomes de longueur l, il y a $(k + 1)^l$ schémas. Un schéma est une famille de chromosomes décrite par un mot de l'alphabet avec *, ainsi pour le mot 0*1*, il y a une unification possible avec les chromosomes 0010, 0011, 0110, 0111 sur l'alphabet {0, 1}.
Chaque chromosome choisi appartient à 2^l schémas.
Pour un schéma H, on note o(H) le nombre de gènes définis, ainsi o(*10*1**) = 3, et d(H) la longueur entre les deux gènes définis extrêmes ainsi d(*10***1*) = 5 et d(**1*) = 0.
Soit f la fonction d'évaluation, (ici à maximiser) et f_m sa moyenne sur la population entière à chaque génération. Pour un schéma H, on notera f(H) la moyenne de f sur la famille représentée par H dans la population.
Si à la génération t on note m(H, t) le nombre de chromosome ayant le schéma H le théorème est alors :
$m(H, t + 1) \geq m(H, t) f(H) [1 - p_c d(H) / (l - 1) - o(H) p_m] / f_m$ où p_m est la probabilité d'une mutation, p_c celle du croisement. Il signifie que les schémas de courte longueur, d'ordre bas et de bonne valeur (les «building blocks») vont se reproduire exponentiellement.
On a pour une reproduction $m(H, t + 1) = m(H, t) f(H) / f_m$, en effet si C est un chromosome présentant le schéma H, la probabilité qu'il se reproduise est le quotient $f(C) / \sum f_i$ et $f(H) / f_m = n\sum\{f_i / C \in H\} / card(H).\sum f_i$ d'où la relation :

$$m(H, t) f(H) / f_m = n \sum\{f_i / C \in H\} / \sum fi = n \, proba(H)$$

En notant $f(H) = (1 + c) f_m$, la relation s'écrit $m(H, t + 1) = (1 + c)m(H, t)$ soit plus précisément $m(H, t) = (1 + c)^t$ ce qui veut dire que si $c > 0$ le bon schéma croît exponentiellement et que si $c < 0$ le mauvais schéma décroît exponentiellement.
Pour l'action d'un croisement simple (échange des parties droite et gauche d'une coupure), la probabilité p_s de survivance du schéma H de longueur d(H) est que la coupure survienne hors de la zone définie :

$$p_s = 1 - d(H) / (l(H) - 1) \geq 1 - p_c d(H) / (l(H) - 1)$$

Pour l'action d'une mutation, la probabilité de survivance de chaque gène est $1 - p_m$ et donc la probabilité de survivance du schéma H est $(1 - p_m)^{o(H)}$ approchée par la valeur $1 - o(H)p_m$ dans la mesure où $p_m << 1$, d'où l'inégalité annoncée.

NOUVELLES IDÉES MISES EN OEUVRE DANS LES STRATÉGIES D'ÉVOLUTION

Le problème essentiel des stratégies d'évoluton est d'arriver à un compromis entre l'exploration de l'espace de recherche et l'exploitation des régions déjà explorées. C'est pourquoi, beaucoup d'idées implémentées, cherchent à éviter une trop grande homogénéité de la population. La population peut être de taille variable, les opérateurs peuvent être notés pour être triés à chaque génération, ces opérateurs peuvent évoluer au cours (mutation non uniforme), on peut créer des sous-populations mises en compétition et surtout créer un partage de la population en «niches» correspondant à des minima locaux.
Si le problème du codage est délicat, on avance souvent que l'exploration de l'espace est favorisée par un codage très raffiné comme un codage binaire, le problème de la définition de la performance est encore plus difficile. En effet, dans bien des questions pratiques, l'adéquation d'un individu à un objectif n'est pas clairement formalisable, le plus souvent, il s'agit d'une agrégation de plusieurs critères hétérogènes (performances physiques, prix, complexité du codage) dont les importances relatives sont sujettes à discussion.
En vue d'optimiser plusieurs critères, il est possible de ne conserver au sein de la population que des individus incomparables entre eux au sens de Pareto, c'est à dire qu'il existe au moins deux critères pour lesquels deux à deux les individus sont classées de façon inverse [Fonseca, Fleming 93], [Gacôgne 97].
La programmation génétique [Koza 92, 94] constitue un intéressant domaine d'expérimentation où chaque chromosome est un arbre (un fonction structurée en Lisp) dans le but de trouver des expressions de fonctions correspondant à un ensemble de couples (x, y) ou une courbe séparatrice de points par exemple.

Plusieurs travaux ont été réalisés en vue de déterminer une partie ou tout un contrôleur flou (les prédicats, les règles, certains paramètres numériques ...) [Geyer-Shulz 95], [Yuan Zhuang 96]. On pourra citer l'approche dite de Pittsburgh où la population est formée d'éventuelles solutions au problème, et celle de Michigan dans laquelle ce sont les règles qui sont individuellement notés pour leur contribution [Valenzuela, Rendon 91], [Cordon, Herrera 95]. C'est la première approche qui est présentée dans les exemples suivants.

APPLICATION AU CHOIX DES RÈGLES DE CONTRÔLEURS FLOUS POUR LE SUIVI D'UNE LIGNE

Un robot doit suivre une trajectoire en mesurant à chaque fois sa distance (entre 0 et une distance maximale de reconnaissance dm = 25) à ±45° de sa direction. Les deux entrées sont donc cette distance x (négative à gauche, positive à droite) et la variation dx de cette distance entre deux prises de données (cette variation est qualifiée par les mêmes prédicats dans $[-a_m, a_m]$ avec un angle maximal de braquage $a_m = 30°$. La sortie u suivant l'algorithme de Mamdani est un angle toujours qualifié par les mêmes prédicats mais dans [-30°, 30°]. Pour 5 prédicats, on veut trouver 25 termes linguistiques codant une table de règles identifiant ce contrôleurr flou pour donner des conclusions aux règles. On utilisera un prédicat NUL (fonction d'appartenance nulle) pour les règles supprimées puisque NUL est neutre vis à vis du max.
Une «solution» sera donc ici une suite de 25 symboles, on peut prendre les diagonales successives du tableau des règles (numérotées sur la figure). Nous devons faire intervenir le nombre de virages effectués par le robot afin de minimiser

les oscillations, mais si nous voulons obtenir un tracé le long de la courbe, ce nombre doit peser moins que la somme des distances.
En s'arrêtant sur l'heuristique attachant 3 fois plus d'importance à la somme des distances, on prend la définition :
Valeur (C) = $\sum|x| + (\sum|a_m * u|)/3$ - vides(C), (u est la sortie du contrôleur et a_m l'angle maximal de braquage) le dernier terme (nombre de cases vides) servant à classer les solutions suivant leur nombre de règles.
Il est assez fascinant de remarquer qu'un mécanisme d'évolution aveugle, uniquement guidé par une telle fonction de performance, peut orienter la population vers des solutions notablement différentes : très proches de la trajectoire mais oscillantes ou bien plus rapides mais s'en éloignant.
Le long d'une sinusoïde amortie, l'expérience montre que si cette valeur est calculée sur 30 pas (le premier virage de forte courbure) ou bien sur 150 pas (l'ensemble de la fenêtre) les populations ne convergent pas véritablement vers les mêmes tables et l'apprentissage sur un virage très serré ne donne pas nécessairement de bons résultats pour la suite.

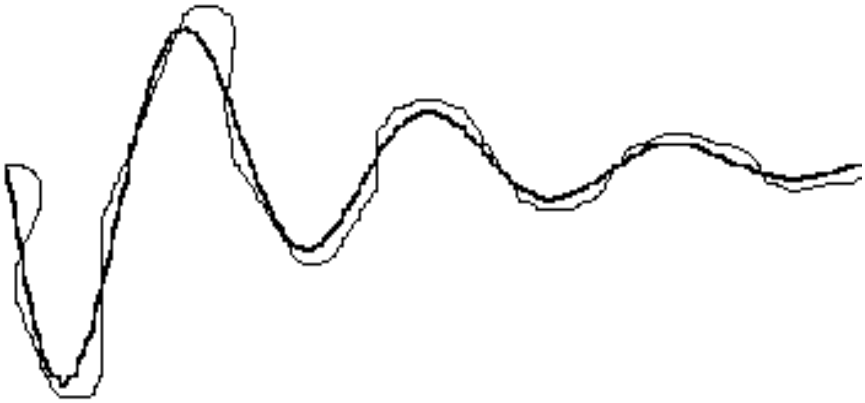

Figure 5.12 Sept règles obtenues codées par la chaîne (pb nul nul nul nul pb nul nul nul nul nul ze ze ze nul nul nul nul nul nb nul nul nul nul nb), la valeur est 2288.

La symétrie est imposée de sorte que seuls les 13 premiers symboles constituent réellement le chromosome, les meilleurs résultats comportent toujours PB dans les cases numérotées ci-dessous suivant un ordre cantorien, 1 et 6, et ZE au centre. Il semble au cours de l'apprentissage, que l'apparition d'un ZE (voire d'un NS ou NB dans d'autre sessions) en case 10 soit responsable d'une nette amélioration. En 7 et 8 plusieurs individus conservent PS ou PB et la différence se joue sur les cases 11 et 12 supportant ZE, NS ou NUL pour la première et NB, NS, ZE pour la seconde. Le prototype (on notera les effets conjugués des prédicats se disposant autour du centre ainsi que les couples de cases 6 et 10 et 16 et 20 dans les deux tables) est :

PB					25 NB
PS	10 ZE	14 PS	18 NB		
ZE	6 PB		13 ZE		20 NB
NS			8 PB	12 NS	16 ZE
NB	1 PB				
dx / x	NB	NS	ZE	PS	PB

APPLICATION À UN PROBLÈME DE POURSUITE

Ce problème est extrêmement intéressant à plus d'un titre, il s'agit pour le chat d'attraper la souris, et pour la souris, de lui échapper, s'il est assez facile d'imaginer des règles floues pour le chat, elles n'auront de valeur que si elles réussissent en toutes circonstances, mais s'il s'agit toujours de la même souris, ces règles seront trop particulières et n'auront pas d'effet face à une autre cible.
On voit là la difficulté qu'il y a à définir la fonction à optimiser pour le chat. D'autre part, quelles sont les paramètres sur lesquels le chat doit se fonder pour modifier sa trajectoire ?
Même à vitesse constante, ce qui est déjà une grande simplification du problème, la réponse n'est pas facile. Enfin, et surtout, il paraît très difficile de trouver des règles intuitives permettant à la souris d'échapper au chat.

Nous posons que le chat va m fois plus vite que la souris, mais que celle-ci peut virer m fois mieux. L'apprentissage a été effectué avec m = 2, un pas ds = 8 et un angle maximal a_m = 60°. Le chat avance donc d'un pas (m.ds) et la souris peut tourner jusqu'à (m.a_m) par rapport à sa direction.

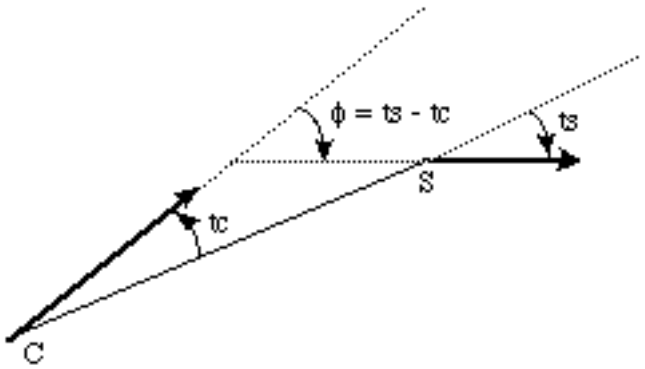

Figure 5.13 Les deux entrées tc et ts des contrôleurs flous.

Une première question est le choix des arguments à retenir pour exercer un contrôle, nous pouvons faire intervenir la distance CS, les angles tc, ts, ϕ = ts - tc et leurs variations. Des expériences préalables ont montré que les résultats pouvaient être bons en ne considérant que tc et ts.

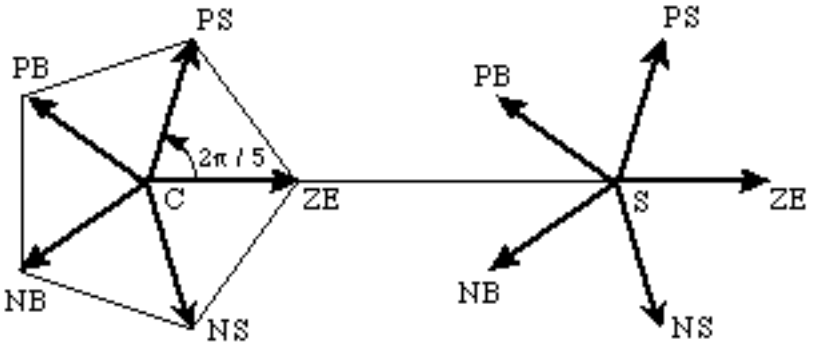

Figure 5.14 Les modes des cinq prédicats qualifiant les angles tc et ts.

Mais le problème de trouver les règles pour tc et ts n'est pas simple, car en prévoyant toutes les situations, ces deux angles appartiennent à]-π, π] où π serait à la fois PB et NB. Nous nous ramenons donc aux cinq prédicats habituels par $t \rightarrow 5t / (4\pi)$ de telle façon que PB sera entièrement réalisé pour les angles de $4\pi / 5$ à π, et NB entre -π et $-4\pi/5$. Les prédicats correspondent aux rayons de la figure 5.14.

VALEUR DU CHAT Pour évaluer un ensemble de règles de poursuite d'une cible mobile, nous avons sommé les 5 nombres de pas nécessaires pour atteindre une souris au comportement aléatoire lors de départs respectifs à une distance de 6.ds avec des valeurs initiales respectives ts = 0, $-\pi/4$, $\pi/2$, $-3\pi/4$ et π, en majorant chaque poursuite à 15 pas. Nous ajoutons à ce total le nombre de règles et cherchons donc à minimiser cette valeur du chat. L'inconvénient majeur de cette définition est qu'il ne s'agit pas à proprement parler d'une fonction puisque deux évaluations du même individu ne donneront pas la même valeur et les différents chats de la population ne sont que grossièrement comparés lors de l'évolution.
Parmi les règles intuitives du chat, on pourait avoir par exemple :
(tc est PB) et (ts est ZE) alors (dtc est NB) ou encore la règle (tc est PB) et (ts est NS) alors (dtc est PB), mais le risque d'agrégation nulle entre NB et PB montre que des cases voisines du tableau des règles ne doivent pas porter des valeurs linguistiques opposées. Une des meilleures solutions trouvées par algorithmes génétiques.

PB					
PS		14 PS	18 ZE		
ZE		9 PB		17 NS	
NS			8 ZE	12 NS	
NB					
ts / tc	NB	NS	ZE	PS	PB

La vérification de ces résultats en faisant un cheminement aléatoire de la souris est excellente.

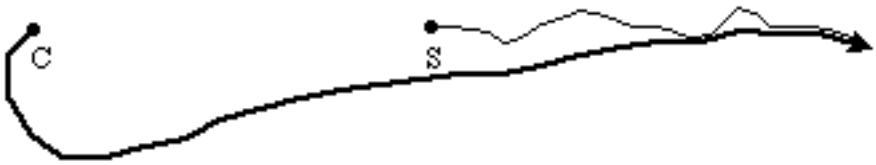

Figure 5.15 Départs tc = π, ts = 0.

VALEUR DE LA SOURIS Il est difficile d'évaluer un comportement sur des situations initiales identiques, il faudrait pouvoir obtenir une moyenne sur un nombre très important d'épeuves, ce qui rend l'évaluation très longue, aussi avons-nous arrêté un compromis. Pour évaluer le comportement d'une souris, nous cherchons (avec les mêmes entrées tc et ts) à maximiser le total des pas effectués lors de 6 orientations initiales : la poursuite (tc = ts = 0), l'attaque frontale (tc = 0, ts = π), les départs opposé (tc = π, ts = 0), tête-bêche (tc = π/2, ts = -π/2) enfin (tc = π/2, ts = π) et (tc = 0, ts = -π/2).

Lors de cet apprentissage on voit se dessiner petit à petit une stratégie globale consistant à tourner au plus court et faire une boucle lorsque le chat est juste derrière puis à se maintenir derrière le chat dans son «angle mort». Ces boucles sont provoquées par des règles PB en position 10, 11 et 12 de la table.
En suivant cette stratégie, la souris retrouve un résultat élémentaire : en tournant toujours dans le même sens mais en augmentant sa courbure, la souris arrive quasiment à décrire un cercle, le chat décrivant «devant» elle un cercle de rayon m fois plus grand, les deux mobiles ayant même vitesse angulaire arrivent à des mouvements quasi-périodiques.

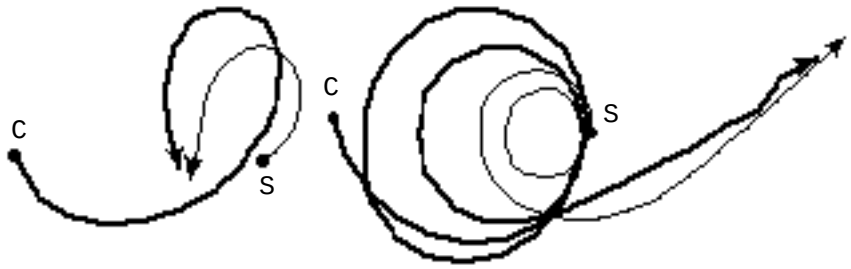

Figure 5.16 Deux exemples de départs tc = -π/2, ts = 0.

L'observation plus attentive des blocs performants comme PS NB PS en positions 7, 8, 9 ou seulement NB en 8 comme la suite l'a montré, permet d'énoncer la règle :
Si le chat va dans la direction de la souris (tc est ZE) et que celle-ci a une direction légèrement négative (ts est NS), alors elle doit la rendre plus négative encore (NB en 8), ce qui signifie virer brusquement devant le chat.

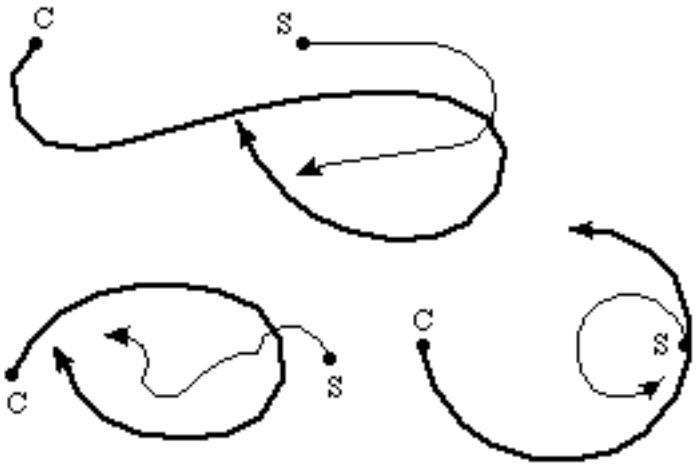

Figure 5.17 L'observation des trajectoires en cours d'apprentissage révèle aussi cette stratégie que l'on pourrait décrire par : obliger la chat à tourner, et celui-ci ne pouvant réduire sa vitesse, est contraint de dépasser la souris, à ce moment, celle-ci oblique pour passer derrière lui, ou plutôt se maintient à environ 135° de la direction du chat en l'obligeant à poursuivre son virage.

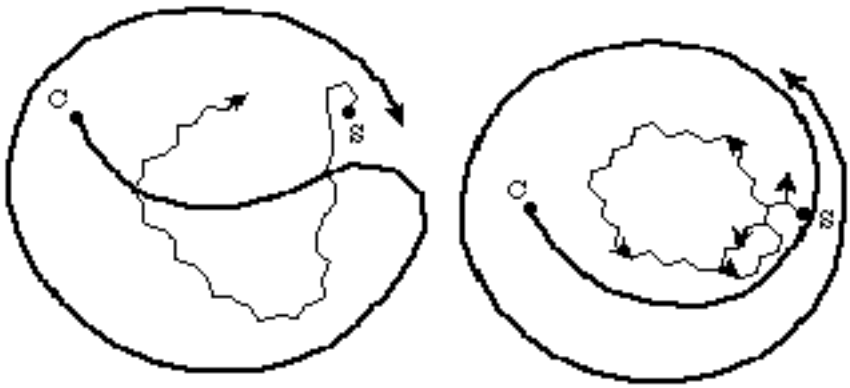

Figure 5.18 Grâce à NB en 8 et NS en 4 qui, moins indispensable, mais présent dans toutes les meilleures tables, on arrive à une trajectoire en ligne brisée qui oblige le chat à tourner en le contraignant de surcroît à décrire un large cercle. Le chat ne sait, en effet, si la souris va se maintenir en gros à 135° de sa direction, ce qu'elle fait une fois sur deux, ou bien passer derrière lui. Ici, NB est remplacé par NS en 8.

Les règles de la souris ne sont pas véritablement toujours efficaces, notamment au cas où tc = ts = 0 (la case centrale) la souris devrait obliquer, mais ceci contrarie les règles 8 et 18 qui, elles, sont indispensables. Les règles 4 et 22 le sont un peu moins, quant aux règles 10 et 16, elles peuvent être omises ou déplacées en 11 et 15. Le prototype est donc réduit à deux règles (8 et 18) avec éventuellement 2 ou 4 autres règles :

PB			22 PS		
PS	10 PB		18 PB		
ZE					
NS			8 NB		16 NB
NB			4 NS		
ts / tc	NB	NS	ZE	PS	PB

La robustesse de ces règles est néanmoins surprenante, si on modifie l'angle a_m (par exemple 30°) et surtout le rapport m, en ne conservant que les deux règles 8 et 18, on peut observer que la souris arrive véritablement à se placer derrière le chat pour m = 1 et à lui faire faire de larges boucles pour m = 1.2.

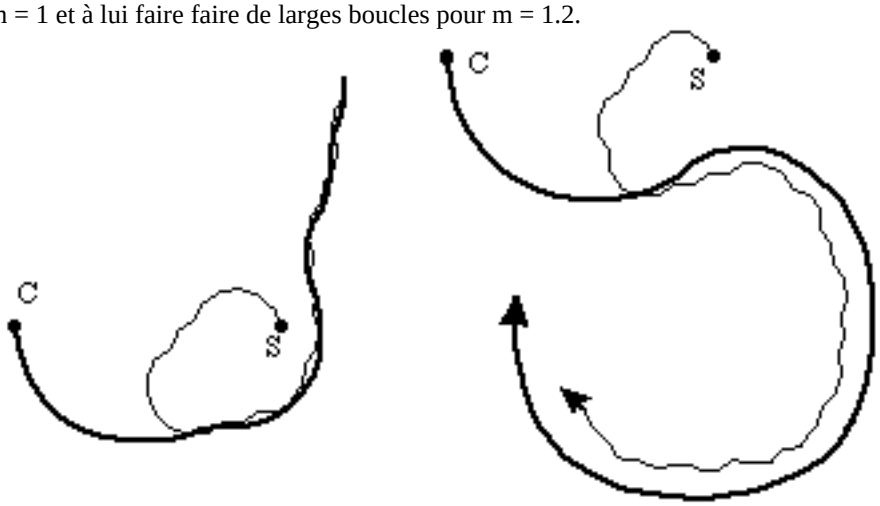

Figure 5.19 Succès ou échec du chat dans des conditions très voisines.

APPLICATION À LA RECHERCHE DES PRÉDICATS ET DES RÈGLES, EXEMPLE DU PARCOURS D'OBSTACLES (2 ENTRÉES, 2 SORTIES)

Dans [Glorennec 92], on part d'une famille de prédicats sur [-1, 1] se coupant à 0,5 et définis par des sigmoïdes f(x) = 1 / (1 + exp(-ax+b)), pour deux entrées et 5 prédicats on peut n'avoir que 10 valeurs continues à régler. Une solution sera une suite de 10 réels qui peut être mutée par modifications d'un nombre aléatoire d'éléments ou par addition d'un «bruit» gaussien.
Dans l'application ci-dessous, nous avons toujours 5 prédicats qui sont les troncatures dans [0, 1] des triangles de hauteur h et de demi-base r.
Les éventuelles solutions sont codées de manière structurées comme (r h R_1 R_2 R_3 ...) où les règles R_i sont maintenant en nombre variable et de la forme (p (∂dir ∂pas) d a). Il y a deux entrées d et a qui sont la position polaire du plus proche obstacle vu dans le secteur limité par la distance d_m et les angles $\pm a_m$, et deux sorties ∂dir (variation de direction appréciée dans $[-a_m, a_m]$) et ∂pas ∈ [-acc, acc] la variation du pas tel que celui-ci soit toujours entre une vitesse minimale avec ds et une vitesse maximale p_m.

La priorité d'une règle signifie que les règles particulières de priorité 0 s'appliquent pour conclure à un couple de conclusions, et si elles ne peuvent s'appliquer, des règles plus générales (de priorité 1) s'appliquent alors.
Les opérateurs génétiques utilisés pour cette expérience sont les bruits gaussiens apportés à r et h, les mutations dans une règle, la création d'une nouvelle règle aléatoire, la suppression d'une règle, les changements de priorité et le cross-over.
La fonction à minimiser est (nb de pa.(p_m - $\sum$ pas) + nb de symboles + tenue de route
Exemple de solution obtenue par algorithme évolutif :

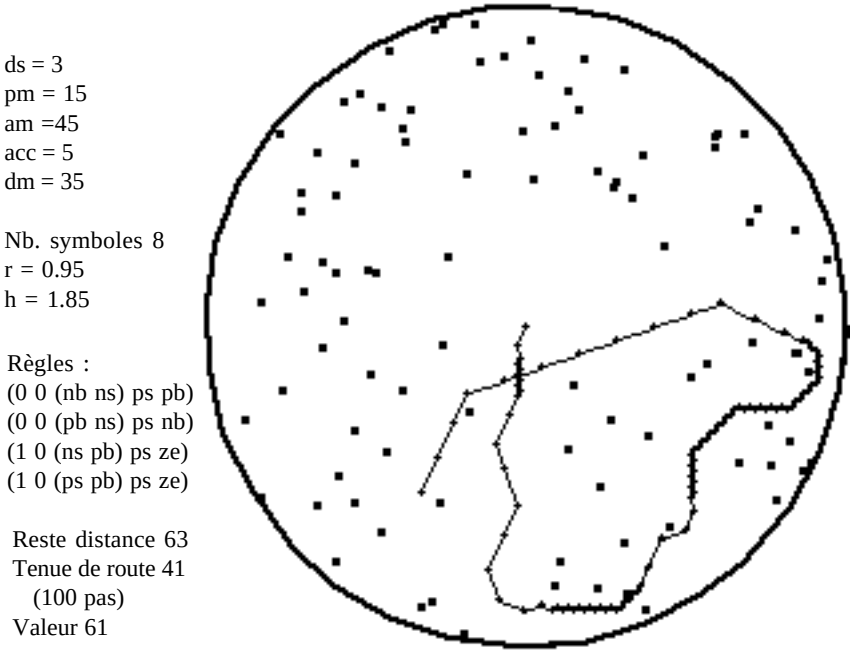

Figure 5.20 Exemple du parcours d'obstacles.

APPLICATION À LA RECHERCHE COMPLÈTE D'UN CONTRÔLEUR FLOU POUR UN ROBOT DEVANT FRANCHIR DES PORTES DE SLALOM

On souhaite maintenant la mise au point de contrôleurs flous utilisant des prédicats relativement variables mais définis par un petit nombre de paramètres. Il est assez difficile de considérer le problème dans son entière généralité. La plupart du temps de fortes restrictions permettent d'obtenir des résultats acceptables mais la recherche ne s'appliquant que sur un aspect du problème (les règles ou une famille de prédicats bien définie). Nous avons cherché ici à réaliser un compromis entre une famille générale et un petit nombre de paramètres. Le quadruplet (nb, contr, trap, fuz) défini ci-dessous permet d'obtenir des familles très diverses recoupant un grand nombre de familles de prédicats utilisées couramment dans les applications

concrètes. Nous définissons ces familles à partir de fonctions d'appartenance sur l'intervalle [-1, 1] grâce à :
Le nombre $p \in \{1, 2, 3, 4\}$ permet de déduire 2p + 1 prédicats.
Le second $r \in [0, 1]$ nommé «contraction» des prédicats, il mesure la manière dont ils sont resserés autour de 0.
Le coefficient suivant $h \in [0, 1]$ est un indice («trap») de chevauchement, et $c \in]0, 1]$ («fuz») est une mesure du flou grâce à la «pente» des trapèzes.
Soit m_k le «sommet» d'une fonction triangulaire pour $0 \le k \le p$ (avec $m_{-1} = m_1$) definie par :

$$m_k = r\left(\frac{k}{p}\right)^2 + (1 - r)\frac{k}{p} \quad \text{et :}$$

$$\mu_k(x) = \min\left(1, \max\left(0, h + 1 - \frac{1}{c}\left|\frac{m_k - x}{m_k - m_{k-1}}\right|\right)\right)$$

Chaque fonction d'appartenance est la troncature d'une «fonction triangulaire» dont le maximum est $1 + h + r.m_k$ pour m_k, (ainsi $m_0 = 0$) et qui est linéaire entre ce sommet et le point $(m_{k-1}, 0)$, symétrique autour de m_k et bornée par [0, 1]. On nomme ZE l'ensemble flou de fonction d'appartenance μ_0 et on impose une symétrie telle que $\mu_{-k}(x) = \mu_k(-x)$. On peut alors appeler les prédicats PS et NS pour k = 1 ou -1 dont les fonctions sont respectivement μ_1 and μ_{-1}, PM and NM pour μ_2 and μ_{-2} et PB, NB pour ±3.
Les figures suivantes montrent quelques possibilités de familles sur [0, 1].

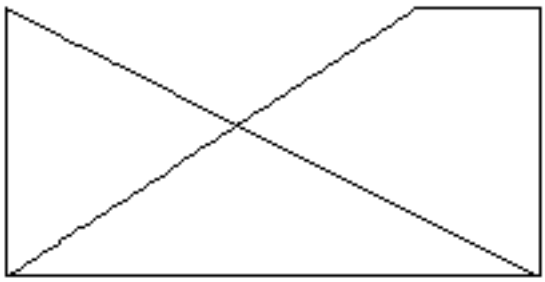

Figure 5.21 Famille de predicats sur [0, 1] avec nb = 1, contr = 0.3, trap = 0 et fuz = 1.

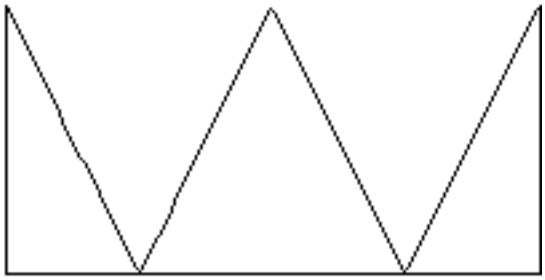

Figure 5.22 nb = 2, contr = 0, trap = 0 et fuz = 0.5

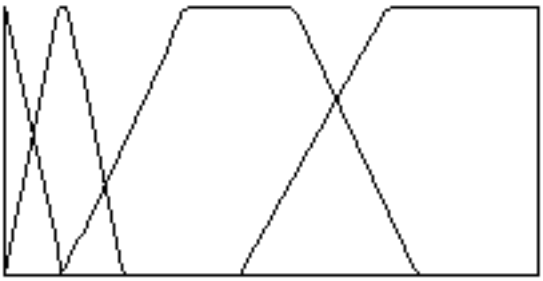

Figure 5.23 nb = 3, contr = 1, trap = 0 et fuz = 1.

CODAGE D'UN CHROMOSOME Les chromosomes n'ont pas une longeur fixe, leur structure est (nb contr trap fuz ouv R_1 R_2 R_3 ...) où nb, contr, trap, fuz sont les coefficients décrits plus haut, «ouv» un angle en degrés, et R_i les règles, toutes de la forme (pr f h_1 ... h_{nh} c_1 c_2 ... c_{nc}).
Les hypothèses h_i sont des symboles $\in$ {any, nb, ns, ze, ps, pb} et les conclusions c_j sont des entiers [-100, 100], pr est la priorité de la règle, et f est sa force. Cette force est simplement le niveau d'utilisation de la règle au cours d'une épreuve, elle n'est présente que pour l'utilisation de deux opérateurs, la mutation sur la règle la plus utilisée et la suppression de la règle la moins utilisée.

OPÉRATEURS GÉNÉTIQUES Ce sont, outre les deux derniers mentionés, le bruit sur les paramètres numériques, la création d'une nouvelle règle ou d'un nouvel individu aléatoire (migration) et la suppression d'une règle. Ces opérateurs sont notés suivant leur capacité cumulée à améliorer les 4 critères définis plus bas.
A chaque génération, la population est classée et les opérateurs le sont également, chaque opérateur est ensuite appliqué à l'individu de même rang de sorte que les meilleurs opérateurs s'essaient sur les meilleurs individus. D'autre part il n'y a pas reproduction, mais les individus les moins bons au sens de Pareto pour les 4 critères sont éliminés? C'est pourquoi la population peut être réduite à quelques individus incomparables deux à deux.

EPREUVE RÉALISÉE Il s'agit de faire suivre un parcours matérialisé par des portes, le robot M devant effectuer un slalom en mesurant à chaque étape, les angles qui séparent sa direction des visées des deux côtés P_1, P_2 de la prochaine porte, en mesurant de plus la distance d qui le sépare du milieu de cette porte, il va considérer les deux entrées constituées par a_1+a_2 et d.
Plus précisément, nous allons chercher des contrôleurs flous de [-1, 1]*[0, 1] vers le carré $[-1, 1]^2$. Une règle ayant deux prémisses pour a_1+a_2 apprécié dans [-ouv, ouv], et d apprécié dans [0, dm] avec dm = 40 et deux conclusions dans [-100, 100] donnant les pourcentages de variations de direction (le maximum est am = 60°) et de vitesse (maximum acc = 10 pixels) à effectuer.
Le pas variable restant dans l'intervalle [4, 35].

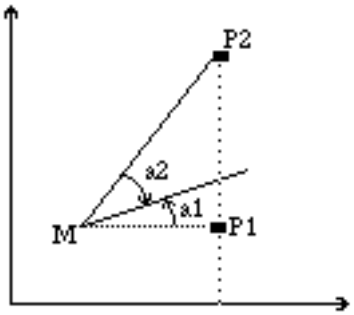

Figure 5.24 Le mobile M mesure les deux angles et sa distance au centre de la prochaine porte à passer, le but qui lui est assigné par les règles serait $a_1+a_2 = 0$.

CRITÈRES À OPTIMISER

Nous allons orienter notre recherche avec les quatre critères suivant à minimiser :
1 - Le nombre «np» de portes restantes à l'issue de 35 pas du robot.
2 - Le second critère est un malus attribué aux portes non franchies, c'est la moyenne «val» des distances mesurées au passage de chaque porte si celles-ci sont passées à l'extérieur.
3 - Un autre critère est la «tenue de route» [Gacôgne 94]. C'est le nombre en pourcentage :

$$tr = \left| \frac{|da|}{am} + \frac{dp}{2acc} - \frac{1}{2} \right| \in [0, 1]$$

Dans lequel da et dp sont les changements de direction et de pas réalisés à chaque prise de données, «am» et «acc» étant les maxima de ces changements. Plus tr est proche de 0, meilleure nous estimons la trajectoire (accélération en ligne droite, freinage dans les virages), lorsqu'il est proche 50, nous l'estimons mauvaise, et plus tr est proche de 100, plus la trajectoire est à l'opposé d'une bonne conduite.
4 - Nous tenons enfin compte du nombre «ns» de symboles significatifs dans une liste de règles, afin de favoriser les petits nombres de règles.

EXPÉRIENCE D'ÉVOLUTION

Afin d'explorer continuellement l'espace des solutions, nous avons choisi une taille minimale de 5 individus. Ainsi avec 2 ou 3 critères, il est fréquent que l'élimination des solutions comparables les plus mauvaises amène à 3 solutions retenues à l'issue d'une génération. En regard de ces 4 critères mentionnés, la taille de la population oscille entre une et deux dizaines, c'est pourquoi, nous avons choisi une taille maximale de 25.
Nous avons fait 20 évolutions, chaque expérience étant limitée à 5000 évaluations (5000 parcours du slalom). Ces expériences ont été faite pour moitié en triant la population suivant l'ordre lexicographique sur (np val tr ns), et pour moitié sur (val np tr ns) attendu que les deux premiers critères sont manifestement les plus importants.

A l'issue de ces évolution nous observons la variété des solutions trouvées et faisons concourir ensemble les meilleurs individus.

CONCLUSIONS DE L'EXPÉRIENCE

L'observation des opérateurs lors des différentes sessions montre l'importance de l'opérateur de «migration» qui permet, surtout en début d'évolution de ne pas converger trop rapidement vers une solution locale. Il est néanmoins évident que ce sont des évolutions séparées suivies de confrontations qui permettent d'obtenir de bons résultats. Ce fait a été remarqué par de nombreux chercheurs qui ont cherché à simuler des «niches écologiques» [Goldberg, Richardson 87]. L'action du «cross-over» doit être freiné en début de parcours par un grand nombre d'autres opérateurs, et on remarque en effet que le croisement n'opère bien qu'en fin de parcours. On peut remarquer que si on regroupe les opérateurs les plus perturbants (le plus liés à une exploration aléatoire uniforme), ceux-ci sont largement classés en tête.
En ce qui concerne les individus trouvés, on peut remarquer que presque tous réalisent des contrôleurs flous basés sur des familles de 5 prédicats (un seul est à 3 prédicats, 4 le sont des avec 7 prédicats) se chevauchant fortement (contr = 0.43 et trap = 0.45 en moyenne) et avec des trapèzes peu spécifiques (fuz = 0.77 en moyenne) c'est à dire assez larges. Ces contrôleurs ont peu de règles (généralement deux et leurs symétriques la plupart du temps ayant le même niveau de priorité) et l'univers d'appréciation des angles est donné par «ouv» de 25° à 94° (moyenne 60°).

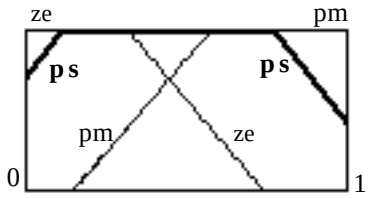

Figure 5.25 Prédicats ZE, PS, PM se chevauchant fortement, obtenus sur [0, 1] par algorithme génétique.

Un des meilleurs résultats (3 portes ratées de peu). Les prédicats NM, NS, ZE, PS, PM signifient respectivement «negative medium» et «small», zero, «positive small» et «medium». Les constantes sont ds = 4, pm = 35, am = 60, acc = 10 et dm = 40. Le chromosome trouvé a les caractéristiques nb = 2, contr = 0.21, trap = 0.76, fuz = 0.95. Les angles sont appréciés dans [-86°, 86°] et les règles toutes de même priorité sont :

$a_1 + a_2$ ze & dist ps $\rightarrow$ da = -53 & dp = 75
$a_1 + a_2$ ze & dist ps $\rightarrow$ da = 53 & dp = 75
$a_1 + a_2$ pm & dist pm $\rightarrow$ da = -86 & dp = -22
$a_1 + a_2$ nm & dist pm $\rightarrow$ da = 86 & dp = -22

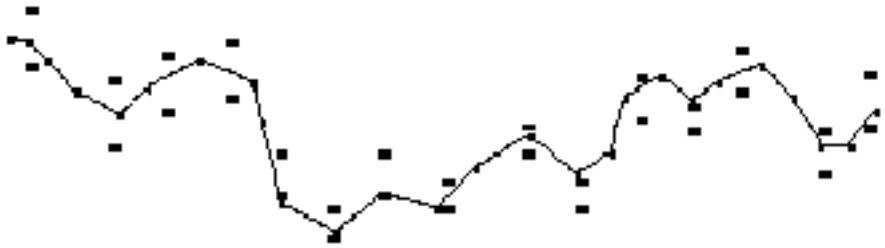

Figure 5.26 Robot traversant des portes de slalom (toutes les portes sont atteintes, mais malus de 5 pour celle qui est ratée et tenue de route 17).