Méthodologie de conception de contrôleurs intelligents par l'approche génétique. Application à un bioprocédé

Ouabib Guenounou

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Systèmes Automatiques*

Présentée et soutenue par *Ouahib GUENOUNOU*
Le *22 avril 2009*

Titre : *Méthodologie de conception de contrôleurs intelligents par l'approche génétique- application à un bioprocédé*

JURY
Gilles ROUX, Professeur à l'UPS de Toulouse, Président
Monique POLIT, Professeur à l'université de Perpignan, Rapporteur
Saïd DJENNOUN, Professeur à l'université de Tizi Ouzou, Rapporteur
Boubekeur MENDIL, Professeur à L'université de Béjaïa, Examinateur

---

Ecole doctorale : *Ecole Doctoral Système*
Unité de recherche : *Nom de l'Unité de recherche*
Directeur(s) de Thèse : *Prof. Ali BELMEHDI et Prof. Boutaib DAHHOU*
Rapporteurs : *Prof. Monique POLIT et Prof. Saïd DJENNOUN*

Avant propos

Les travaux présentés dans le mémoire ont été effectués dans le cadre d'une cotutelle entre l'université Paul Sabatier Toulouse III et l'université A.Mira de Bejaia. Ils ont été réalisés au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS et au Laboratoire de technologie Industrielle et de l'information de l'université A. mira de Bejaia. A ce titre je remercie le service de l'Agence Universitaire de Francophonie (AUF) qui par son financement m'a aidé à effectuer mes travaux dans de bonnes conditions.

Je remercie Messieurs Malik GHALLAB et Raja CHATILA, respectivement ancien et actuel Directeurs du LAAS CNRS et Madame Louise TRAVE-MASSUYES, responsable du groupe DISCO.

Je remercie également Monsieur Ali BELMEHDI, Directeur du LTII.

Je tiens à remercier vivement Monsieur Gilles Roux, Professeur à l'UPS de Toulouse pour avoir accepté d'examiner ce travail et pour m'avoir fait l'honneur de présider la commission d'examen.

Je tiens également à remercier très sincèrement Madame Monique Polit, Professeur à l'université de Perpigan et Monsieur Saïd Djennoune, Professeur de l'université Mouloud Mameri de Tizi ouzou, pour l'honneur qu'ils m'ont fait d'examiner cette étude et d'en être les rapporteurs. Je remercie également Monsieur Boubekeur Mendil, Professeur de l'Université A. Mira de Béjaia d'avoir accepté d'examiner cette thèse.

J'exprime ma profonde reconnaissance à mon directeur de thèse côté français, Monsieur Boutaib Dahhou, Professeur à l'UPS de Toulouse, et je le remercie pour son soutien scientifique et pour la confiance qu'il m'a accordée tout au long de mes travaux.

Je ne saurais exprimer ma sympathie et ma reconnaissance envers mon directeur de thèse côté Algérien, Monsieur Ali Belmehdi, Professeur de l'Univeristé A. Mira de Béjaia pour le soutien qu'il a pu m'apporter, pour ses orientations, sa disponibilité et ses conseils. Qu'il soit assuré de ma respectueuse reconnaissance.

Finalement, je remercie sincèrement tous mes amis en particulier Nassim pour le soutien qu'il m'a apporté durant les séjours effectués à Toulouse.

Table des matières

Table des figures

Liste des tableaux

Introduction

Historique

Les outils intelligents sont de plus en plus utilisés dans la conception, la modélisation et la commande de systèmes complexes tels que les robots, les procédés biologiques, les véhicules routiers · · · . On entend par outils intelligents les techniques du soft computing à savoir : les réseaux de neurones, la logique floue et les algorithmes génétiques.
Les réseaux de neurones sont apparus dans les années cinquante mais n'ont reçu cependant un intérêt considérable qu'à partir des années 80 avec l'apparition de l'algorithme de rétropropagation (Rumelhart et McClelland, 1986). Leur capacité d'apprentissage et d'approximation de fonctions leur procure un intérêt considérable de la part des chercheurs. Il suffit de voir les nombreuses applications industrielles qui en découlent à partir des années 90 et de consulter l'abondante littérature sur le sujet pour s'en convaincre.
La logique floue introduite par Zadeh (1965) dans les années soixante constitue un outil très puisant pour la représentation des termes et des connaissances vagues. Elle est issue de la capacité de l'homme à décider et à agir d'une manière intelligente malgré l'imprécis et l'incertitude des connaissances disponibles. Son utilisation dans le domaine du contrôle (contrôle flou) a été l'une des premières applications de cette théorie dans l'industrie avec les travaux de Mamdani et Assilian (1975). Depuis, les applications de la logique floue se sont multipliées pour toucher des domaines très divers.
L'utilisation de la commande floue est particulièrement intéressante lorsqu'on ne dispose pas de modèle mathématique précis, voir inexistant, du système à commander ou lorsque ce dernier présente de fortes non linéarités. Contrairement aux approches classiques de l'automatique, qui se basent en grande partie sur un modèle mathématique, la commande par logique floue, repose sur une collection de règles linguistiques de la forme " Si · · · Alors " qui traduisent la stratégie de contrôle d'un opérateur humain.

Les algorithmes génétiques sont des méthodes stochastiques basées sur une analogie avec des systèmes biologiques. Ils reposent sur un codage de variables organisées sous forme de structures chromosomiques et prennent modèle sur les principes de l'évolution naturelle de Darwin pour déterminer une solution optimale au problème considéré. Ils ont été introduits par Holland (Holland, 1975) pour des problèmes d'optimisation complexe. Contrairement aux méthodes d'optimisation classique, ces algorithmes sont caractérisés par une grande robustesse et possèdent la capacité d'éviter les minimums locaux pour effectuer une recherche globale. De plus, ces algorithmes n'obéissent pas aux hypothèses de dérivabilité qui contraignent pas mal de méthodes classiques destinées à traiter des problèmes réels.

Au cours de ces dernières années, la combinaison de ces techniques a attiré l'attention de beaucoup de chercheurs. Plusieurs hybridations ont été alors proposées dont les plus rencontrées sont: Algorithme Génétique /Contrôleur flou (AG/CF) et Réseau de neurones/Contrôleur flou (RN/CF).

La première combinaison (AG/CF) vise à la conception des CF par algorithmes génétiques. Karr et Gentry (1993) utilisent un algorithme génétique standard (codage binaire, opérateurs de croisement et de mutation simples) pour l'optimisation des paramètres des fonctions d'appartenance. Le nombre de termes linguistiques associé à chaque variable du contrôleur est différent, mais il reste fixe durant tout le processus d'optimisation. Plusieurs méthodes ont été proposées par la suite, dont certaines diffèrent par le type de codages et/ou des fonctions d'appartenance utilisées (Liska et Melsheimer, 1994; Herrera *et al.*, 1995). Récemment, les algorithmes hiérarchisés ou AGH sont aussi impliqués. Dans (Acosta et Todorovich, 2003), on utilise un AGH avec un seul niveau de gènes de contrôle pour chercher le nombre minimal de fonctions d'appartenance à associer à chacune des variables du contrôleur flou.

Thrift (1991) est le premier à décrire une méthode d'optimisation des règles floues par algorithme génétique standard, en utilisant trois bits pour coder chaque règle. A la même époque, Karr propose une méthode permettant de faire intervenir dans le contrôleur flou à la fois des règles spécifiées par un expert humain et des règles optimisées par un algorithme génétique (Karr, 1991). Quelques améliorations ont été apportées par la suite (Herrera *et al.*, 1998; Chen et Wong, 2002; Belarbi *et al.*, 2005).

Homaifar et MCCormick (1993) proposent une méthode d'optimisation qui prend en compte simultanément les fonctions d'appartenance des variables d'entrées et les règles floues. Les différents paramètres sont codés sur le même chromosome en base 6. Dans (Lee et Takagi, 1993), les auteurs partagent la même idée de Hamaifar et MCCormik (co-

dage de l'ensemble des paramètres dans un chromosome unique) en utilisant un codage binaire. Dans l'article de Zhou et Lai (2000), on adopte un codage mixte des paramètres. Ainsi les chaînes réelles et les chaînes entières sont utilisées respectivement pour coder les fonctions d'appartenance et les règles floues.
La combinaison (RN/CF), ou tout simplement contrôleur neuro-flou est une combinaison de la logique floue et des réseaux de neurones qui tire profit des deux approches. Plusieurs architectures neuro-floues ont été proposées dans la littérature suivant le type de règles floues qu'elles intègrent (Mamdani ou Sugeno). La puissance de ces structures réside dans la possibilité d'incorporer une base de connaissances, de traiter les données imprécises et vagues par logique floue et en même temps d'introduire l'apprentissage via le réseau de neurones. Pour la plupart des architectures proposées, les procédures d'apprentissage appliquées sont soit supervisées et se basent sur les techniques d'optimisation classique (descente du gradient, les moindres carrés), soit non supervisées et on utilise les algorithmes de classification (K-means).

Problématique

L'optimisation indépendante, des paramètres des fonctions d'appartenance ou des règles floues par algorithme génétique est juste mais reste incomplète car les deux parties (fonctions d'appartenance et règles floues) ne peuvent être dissociées. Par conséquent, pour obtenir de meilleurs résultats, il faudrait prévoir un codage qui prend en charge simultanément les fonctions d'appartenance et les règles floues. En raison d'une part d'un espace de recherche très grand (fonctions d'appartenance + règles floues) et d'autre part de l'utilisation d'une procédure d'initialisation des chromosomes basée souvent sur un processus aléatoire, les méthodes proposées nécessitent généralement un nombre important de générations. De plus l'optimisation aveugle des algorithmes génétiques, dont le seul objectif est l'amélioration d'un critère numérique, conduit souvent à la présence de règles floues incohérentes (brouillées) en fin du processus d'optimisation. L'amélioration de l'étape d'initialisation des chromosomes ainsi que l'utilisation d'une procédure de sélection qui défavorise l'apparition de règles incohérentes peut être une solution fiable pour contourner ces inconvénients. C'est dans ce cadre que se situent nos motivations et intérêts pour la conception de contrôleurs flous par algorithmes génétiques.

Présentation des chapitres

Ce mémoire est organisé en quatre chapitres comme suit:

Le premier chapitre est consacré aux algorithmes génétiques. Il est constitué de trois parties principales. La première présente une description détaillée des algorithmes génétiques simples dans laquelle nous rappelons les définitions relatives à leur fonctionnement. Dans la deuxième partie nous introduirons les algorithmes génétiques hiérarchisés, version améliorée des algorithmes génétiques simples. La troisième partie traite de l'optimisation multi-objectifs par algorithmes génétiques. Nous décrivons quelques algorithmes de résolution tout en accordant plus d'importance à l'algorithme NSGA-II (Nondominated Sorting Genetic Algorithm-II) qui est utilisé dans ce mémoire.

Le deuxième chapitre est divisé en trois parties. La première traite les réseaux de neurones et leur modalité d'utilisation pour les problèmes de modélisation et de commande. La deuxième partie est consacrée aux systèmes flous. Nous rappelons d'abord les notions de bases sur lesquelles reposent ses systèmes puis nous décrivons leur principe de fonctionnement et leurs différentes composantes. Enfin, dans la troisième partie nous aborderons l'approche neuro-floue ainsi qu'une classification des ses différentes structures.

Le troisième chapitre constitue le noyau de ce mémoire. Nous commençons par une description de la première méthode de synthèse des contrôleurs flous de Mamdani par algorithmes génétiques simples. Nous introduisons par la suite le NSGA-II avec un codage hiérarchisé des paramètres. Enfin nous proposons un algorithme hybride pour la synthèse des contrôleurs de Sugeno.

Le quatrième chapitre concerne une application des méthodes de synthèse des contrôleurs intelligents pour la conduite d'un procédé de fermentation alcoolique.

Chapitre 1

Algorithmes génétiques

1.1 Introduction

Les ingénieurs se heurtent quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des secteurs très divers, comme le traitement d'images, la conception de systèmes de commande et de diagnostic, etc . . .. Le problème à résoudre peut souvent être considéré comme un problème d'optimisation dans lequel on définit une ou plusieurs fonctions objectif, ou fonctions de coût, que l'on cherche à minimiser (ou maximiser) par apport à l'ensemble des paramètres concernés. La résolution d'un tel problème a conduit les chercheurs à proposer des méthodes de résolution de plus en plus performantes, parmi lesquelles on peut citer les métaheuristiques. Ces dernières qui comprennent notamment la méthode du recuit simulé (Kirkpatrick *et al.*, 1983), la méthode de recherche Tabou (Glover, 1989), les algorithmes de colonies de fourmis, les algorithmes génétiques etc . . ., présentent des caractéristiques communes, qui sont:

1. Raisonnement par analogie avec les systèmes réels (physique, biologie, éthologie, · · ·).
2. Stochastiques

et partagent aussi les mêmes inconvénients :

1. Difficulté de réglage des paramètres de la méthode
2. Temps de calcul élevé.

Dans ce chapitre, nous allons nous intéresser uniquement aux algorithmes génétiques qui ont connu ces quinze dernières années un développement considérable grâce à l'augmentation vertigineuse de la puissance des calculateurs et notamment suite à l'apparition des architectures massivement parallèles qui exploitent leur parallélisme intrinsèque.
Le chapitre est organisé en trois parties. La première est consacrée aux algorithmes génétiques comme étant une méthode d'optimisation mono-objectif. Après une présentation générale du principe de fonctionnement d'un algorithme génétique, nous ferons une description détaillée de ses différentes composantes. La deuxième partie est consacrée aux algorithmes génétiques hiérarchisés (AGH), une version améliorée des AGS. Ils diffèrent de ces derniers dans la structure de leurs chromosomes qui présente un niveau hiérarchique et se compose de deux types de gènes: les gènes de contrôle et les gènes de paramètres. Enfin la dernière partie traite de l'optimisation multi-objectifs par algorithmes génétiques. Nous commençons d'abord par un bref rappel sur les spécificités d'un problème à plusieurs objectifs, puis nous évoquons le concept de dominance et la notion d'optimalité au sens de Pareto qui caractérise cette classe de problèmes. Enfin, nous détaillons quelques algorithmes génétiques, dédiés à l'optimisation multi-objectifs tout en accordant plus d'importance à un algorithme élitiste de référence (NSGA-II) qui sera utilisé dans la suite de ce travail.

1.2 Algorithmes génétiques

Les algorithmes génétiques (AG) développés par J. Holland (Holland, 1992) présentent des qualités intéressantes pour la résolution de problèmes d'optimisation complexes. Leurs fondements théoriques furent exposés par Goldberg (Goldberg, 1994). Ils tentent de simuler le processus d'évolution des espèces dans leur milieu naturel: soit une transposition artificielle de concepts basiques de la génétique et des lois de survie énoncés par Darwin. Rappelons que la génétique représente un individu par un code, c'est-à-dire un ensemble de données (appelées chromosomes), identifiant complètement l'individu. La reproduction est, dans ce domaine, un mixage aléatoire de chromosomes de deux individus, donnant naissance à des individus enfants ayant une empreinte génétique nouvelle, héritée des parents. La mutation génétique est caractérisée dans le code génétique de l'enfant par l'apparition d'un chromosome nouveau, inexistant chez les individus parents. Ce phénomène génétique d'apparition de " mutants" est rare mais permet d'expliquer les changements dans la morphologie des espèces, toujours dans le sens d'une meilleure

adaptation au milieu naturel. La disparition de certaines espèces est expliquée par " les lois de survie " selon lesquelles seuls les individus les mieux adaptés auront une longévité suffisante pour générer une descendance. Les individus peu adaptés auront une tendance à disparaître. C'est une sélection naturelle qui conduit de génération en génération à une population composée d'individus de plus en plus adaptés.
Un algorithme génétique est construit de manière tout à fait analogue. Dans l'ensemble des solutions d'un problème d'optimisation, une population de taille N est constituée de N solutions (les individus de la population) convenablement marquées par un codage qui les identifie complètement. Une procédure d'évaluation est nécessaire à la détermination de la force de chaque individu de la population. Viennent ensuite une phase de sélection (en sélectionnant les individus au prorata de leur force) et une phase de recombinaison (opérateurs artificiels de croisement et de mutation) qui génèrent une nouvelle population d'individus, qui ont de bonnes chances d'être plus forts que ceux de la génération précédente. De génération en génération, la force des individus de la population augmente et après un certain nombre d'itérations, la population est entièrement constituée d'individus tous forts, soit de solutions quasi-optimales du problème posé.

En, le fonctionnement d'un AG est alors basé sur les phases suivantes :

1. **Initialisation** : une population initiale de taille N chromosomes est tirée aléatoirement.
2. **Evaluation** : chaque chromosome est décodé puis évalué.
3. **Reproduction:** création d'une nouvelle population de N chromosomes par l'utilisation d'une méthode de sélection appropriée.
4. **Opérateurs génétiques:** croisement et mutation de certains chromosomes au sein de la nouvelle population.
5. **Retour** à la phase 2 tant que la condition d'arrêt du problème n'est pas satisfaite.

1.2.1 Codage et population initiale

Premièrement, il faut représenter les différents états possibles de la variable dont on cherche la valeur optimale sous forme utilisable pour un AG: c'est le codage. Cela permet d'établir une connexion entre la valeur de la variable et les individus de la population, de manière à imiter la transcription génotype-phénotype qui existe dans le monde vivant. Il

existe principalement trois types de codage : le codage binaire, le codage réel et le codage en base n.

1.2.1.1 Codage binaire

Ce codage a été le premier à être utilisé dans le domaine des AG. Il présente plusieurs avantages : alphabet minimum $\{0,1\}$, facilité de mise en point d'opérateurs génétiques et existence de fondements théoriques (théorie sur les schémas). Néanmoins ce type de codage présente quelques inconvénients :

1. Les performances de l'algorithme sont dégradées devant les problèmes d'optimisation de grande dimension à haute précision numérique. Pour de tels problèmes, les AG basés sur les chaînes binaires ont de faibles performances comme le montre michalewicz (Michalewicz, 1992).
2. La distance de Hamming entre deux nombres voisins(nombre de bits différents) peut être assez grande dans le codage binaire : l'entier 7 correspond à la chaîne 0111 et la chaîne 1000 correspond à l'entier 8. Or la distance de hamming entre ces deux chaînes est de 4, ce qui crée bien souvent une convergence, et non pas l'obtention de la valeur optimale.

1.2.1.2 Codage réel

Il a le mérite d'être simple. Chaque chromosome est en fait un vecteur dont les composantes sont les paramètres du processus d'optimisation. Par exemple, si on recherche l'optimum d'une fonction de n variables $f(x_1, x_2, \ldots, x_{n-1}, x_n)$, on peut utiliser tout simplement un chromosome ch contenant les n variables: Avec ce type de codage, la pro-

ch:

x_1	x_2	$\cdots$	x_{n-1}	x_n

cédure d'évaluation des chromosomes est plus rapide vu l'absence de l'étape de transcodage (du binaire vers le réel). Les résultats donnés par Michalewicz (Michalewicz, 1992) montrent que la représentation réelle aboutit souvent à une meilleure précision et un gain important en termes de temps d'exécution.

1.2.1.3 Codage en base n

Dans ce type de codage, les gènes constituant un chromosome sont des chiffres exprimés dans une base de numération n, ce qui permet de représenter n valeurs discrètes.
L'AG démarre avec une population composée de N individus dans le codage retenu. Le choix des individus conditionne fortement la rapidité de l'algorithme. Si la position de l'optimum dans l'espace de recherche est totalement inconnue, il est intéressant que la population soit répartie sur tout l'espace de recherche. Si par contre des informations à priori sur le problème sont disponibles, il paraît évident de générer les individus dans un espace particulier afin d'accélérer la convergence. Disposant d'une population initiale souvent non homogène, la diversité de la population doit être entretenue aux cours des générations afin d'explorer le plus largement possible l'espace de recherche. C'est le rôle des opérateurs de croisement et de mutation.

1.2.2 Opérateur de croisement

Le croisement est le principal opérateur agissant sur la population des parents. Il permet de créer de nouveaux individus par l'échange d'information entre les chromosomes par leur biais de leur combinaison. La population courante est divisée en deux sous populations de même taille $(N/2)$ et chaque couple formé par un membre provenant de chaque sous population participe à un croisement avec une probabilité (p_c) souvent supérieure à $0{,}5$. Si le croisement a eu lieu entre deux chromosomes parents (ch_1 et ch_2), constitués de l gènes, on tire aléatoirement une position de chacun des parents. On échange ensuite les deux sous chaînes terminales de chacun des chromosomes, ce qui produit deux enfants ch_1' et ch_2' comme indiqué sur la figure 1.1.

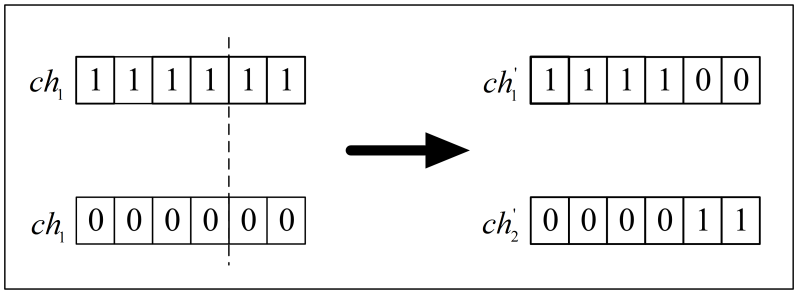

FIG. 1.1 – *Croisement standard en un seul point*

Dans notre exemple, Figure 1.1, un croisement localisé à la quatrième position a eu lieu entre les chromosomes ch_1 et ch_2: il s'agit bien d'un croisement en un seul point. Ainsi on peut étendre ce principe de combinaison en choisissant non pas un seul point, mais 2, 3, etc...(croisement en multipoints) (Man *et al.*, 1996). Sur la figure 1.2 nous représentons un croisement en deux points, que nous utiliserons dans la suite de ce mémoire.

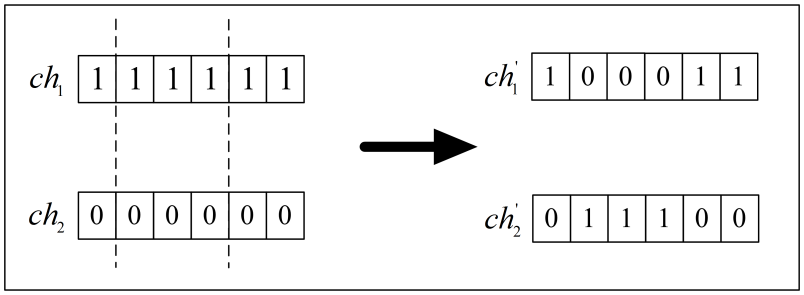

FIG. 1.2 – *Croisement standard en deux points*

Ce type de croisement est très efficace et peut s'étendre à n'importe quel type de chaînes (réelles, en base n, etc...). Néanmoins certains auteurs (Wu et Lin, 1999) préfèrent utiliser dans le cas des chaînes réelles, un croisement de type barycentre: deux gènes $ch_1(i)$ et $ch_2(i)$ sont sélectionnés dans chacun des parents à la même position i. Ils définissent deux nouveaux gènes $ch_1^{'}(i)$ et $ch_2^{'}(i)$ par combinaison linéaire :

$$\begin{cases} ch_1^{'}(i) = \alpha \times ch_1(i) + (1-\alpha) \times ch_2(i) \\ ch_2^{'}(i) = (1-\alpha) \times ch_1(i) + \alpha \times ch_2(i) \end{cases} \tag{1.1}$$

où α est un paramètre de pondération aléatoire qui prend généralement ses valeurs dans l'intervalle $[-0.5, 1.5]$ (ceci permet de générer des points entre ou à l'extérieur des deux gènes considérés).

Quoi qu'il en soit, il se peut que l'effet de l'opérateur de croisement soit insuffisant pour assurer une meilleure exploration de l'espace de recherche. Ainsi dans le cas du codage binaire, certaines chaînes peuvent totalement disparaître de la population. Par exemple, si aucun chromosome de la population initiale ne contient de 1 en première position et que ce 1 fasse partie de la chaîne optimale à trouver, aucun croisement ne peut faire apparaître cet élément. Ce dernier ne peut s'introduire dans la population que si l'on introduit un autre opérateur et c'est, entre autre, pour remédier à ce problème que l'opérateur de mutation est utilisé.

1.2.3 Opérateur de mutation

Le rôle de cet opérateur est de modifier aléatoirement la valeur d'un gène dans un chromosome. Dans le cas du codage binaire, chaque bit $a_i \in \{0,1\}$ est remplacé par son complémentaire $\overline{a_i} = 1 - a_i$. Dans l'exemple de la figure 1.3, une mutation a eu lieu sur le troisième gène du chromosome ch et elle a transformé ce gène de 1 en 0.

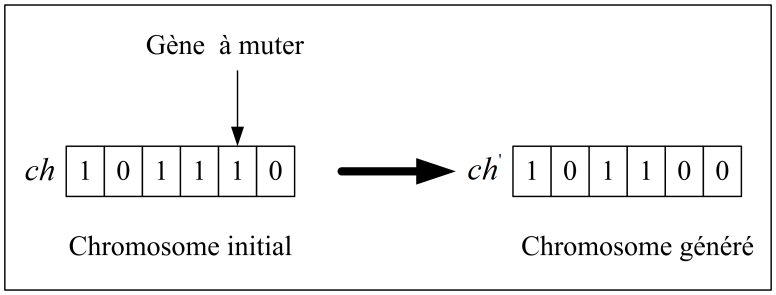

FIG. 1.3 – *Principe de l'opérateur de mutation*

Pour les chaines codées en base n, la mutation consiste à remplacer le gène initial par un chiffre en base n tiré au sort.

Dans le cas d'un codage réel, on utilise principalement deux opérateurs de mutation: la mutation *uniforme* et la mutation *non uniforme* (Michalewicz, 1992). En supposant fixée la probabilité de mutation p_m, un tirage au sort pour chaque gène x_k d'un chromosome ch permet de décider si ce gène doit être ou non modifié. Nous supposons que le gène prend ses valeurs dans un intervalle $[x_k^{min},x_k^{max}]$.

Pour la mutation uniforme, qui est une simple extension de la mutation binaire, on remplace le gène x_k sélectionné par une valeur quelconque $x_k^{'}$ tirée aléatoirement dans l'intervalle $[x_k^{min},x_k^{max}]$.

Pour la mutation non uniforme, le calcul de la nouvelle valeur d'un gène est un peu plus complexe. Le gène x_k subit des modifications importantes durant les premières générations, puis graduellement décroissantes au fur et à mesure que l'on progresse dans le processus d'optimisation. Pour une génération t, on tire au sort une valeur binaire qui décidera si le changement doit être positif ou négatif. La nouvelle valeur $x_k^{'}$ du gène x_k est donnée par :

$$x_k^{'} = \begin{cases} x_k + \triangle(t,x_k^{max} - x_k) & \text{si } rand = 0 \\ x_k - \triangle(t,x_k - x_k^{min}) & \text{si } rand = 1 \end{cases} \tag{1.2}$$

où $\triangle(t,y)$ est une fonction qui définit l'écart entre la nouvelle valeur et la valeur initiale

à la génération t et rand est nombre alétoire qui prend les valeurs 0 ou 1.
Dans (Cordon *et al.*, 2001), les auteurs proposent d'utiliser une fonction $\triangle(t,y)$ correspondante à une décroissance exponentielle de l'écart à travers les générations. Cette fonction est définie par :

$$\triangle(t,y) = y \times (1 - r^{(1-t/T)^{\beta}}) \tag{1.3}$$

où T est l'indice de génération pour laquelle l'amplitude de la mutation s'annule, β est un paramètre de l'opérateur de mutation (souvent $\beta = 5$), r est un nombre produit aléatoirement dans l'intervalle $[0,1]$ et t le numéro de la génération.
Si par contre, l'intervalle de variations du gène x_k n'est pas connu, une mutation gaussienne est souvent utilisée. Le principe de base de ce type de mutation est d'ajouter un bruit gaussien centré $N(0,\sigma)$ au gène que l'on désire faire muter:

$$x_k^{'} = x_k + N(0,\sigma) \tag{1.4}$$

où σ représente la variance.
La mutation est traditionnellement considérée comme un opérateur intervenant à la marge, dans la mesure où sa probabilité est en général assez faible (de l'ordre de 1%). Mais elle confère aux algorithmes génétiques une propriété très importante : l'ergodicité (tous les points de l'espace de recherche peuvent être atteints). Cet opérateur est donc d'une grande importance et il est loin d'être marginal, d'ailleurs un algorithme génétique peut converger sans l'opérateur de croisement et certaines applications fonctionnent de cette manière (Hwang, 1999).

1.2.4 Fonction d'évaluation

Un algorithme génétique nécessite généralement la définition d'une fonction rendant compte de la pertinente des solutions potentielles, à partir des grandeurs à optimiser. Nous la nommerons fonction d'adaptation f (ou fitness function en terminologie anglo-saxonne). Elle est souvent exprimée par la composition de deux fonctions g et h:

$$f = g \circ h \tag{1.5}$$

avec:
$\circ$: loi de composition de fonctions
g: fonction de transformation, pouvant être linéaire, exponentielle, logarithmique, etc ....
h: fonction objectif ou de coût, elle dépend de l'objectif recherché.

1.2.5 Opérateur de sélection

La sélection crée une population intermédiaire constituée de copies des individus de la population courante. En règle générale, le nombre de copies d'un individu est lié directement à la fitness relative de l'individu au sein de la population. Il existe plusieurs méthodes heuristiques qui représentent la reproduction, la méthode la plus connue et la plus utilisée est la sélection par roulette biaisée (roulette wheel selection) de Goldberg (Goldberg, 1994). Selon cette méthode, chaque chromosome est copié dans la nouvelle population proportionnellement à sa fitness. On effectue en quelque sorte, autant de tirages avec remise que d'éléments existant dans la population. Ainsi pour un chromosome particulier ch_i de fitness $f(ch_i)$, la probabilité de sa sélection dans la nouvelle population de taille N est :

$$p(ch_i) = \frac{f(ch_i)}{\sum_{j=1}^{N} f(ch_j)} \tag{1.6}$$

Plus la performance d'un individu est élevée par rapport à celle des autres, plus il a une chance d'être reproduit dans la population. Les individus ayant une grande fitness relative ont donc plus de chance d'être sélectionnés. On parle alors de sélection proportionnelle. Le nombre de copies espérées pour chaque individu ch_i qui va résulter de la sélection est alors égal à :

$$n_i = N \times p(ch_i) = \frac{f(ch_i)}{\frac{1}{N}\sum_{j=1}^{N} f(ch_j)} = \frac{f(ch_i)}{\overline{f}} \tag{1.7}$$

L'inconvénient majeur de ce type de reproduction vient du fait qu'il peut favoriser la dominance d'un individu qui n'est pas forcément le meilleur. Cette méthode peut aussi engendrer une perte de diversité par la dominance d'un super-individu. Pour palier cet inconvénient, on préfère souvent des méthodes qui n'autorisent en aucun cas l'apparition de super-individu. Par exemple, la sélection par tournoi (tournement selection) ou d'autres méthodes faisant intervenir un changement d'échelle (Scalling) et/ou des notions de voisinage entre chromosomes (Sharing).

1.2.5.1 Sélection par tournoi

On tire deux individus aléatoirement dans la population et on reproduit le meilleur des deux dans la nouvelle population. On répète la procédure jusqu'à ce que la nouvelle population soit complète.

1.2.5.2 Sharing

Le sharing consiste à ajuster la fitness des individus pour éviter qu'ils se concentrent dans une niche principale (optimum globale). La technique de partage de la fitness (fitness sharing), introduite par Goldberg et Richardson (Goldberg et Richardson, 1987), réduit la fitness de chaque individu d'un facteur correspondant environ au taux d'agrégation de la population autour de son voisinage :

$$f'(ch_i) = \frac{f(ch_i)}{m_i} \tag{1.8}$$

où le compteur de niche m_i se calcule de la manière suivante:

$$m_i = \sum_{j=1}^{N} sh(d_{ij}) \tag{1.9}$$

Où N désigne la taille de la population et sh mesure la similarité entre deux individus i et j en fonction de la distance d_{ij} et le rayon de niche σ_{shar} :

$$sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\sigma_{shar}})^{\alpha} & \text{si } d_{ij} < \sigma_{shar} \\ 0 & \text{sinon} \end{cases} \tag{1.10}$$

La figure 1.4 montre deux exemples de répartition de populations dans le cas d'une fonction multimodale: le premier sans sharing et le deuxième avec sharing.

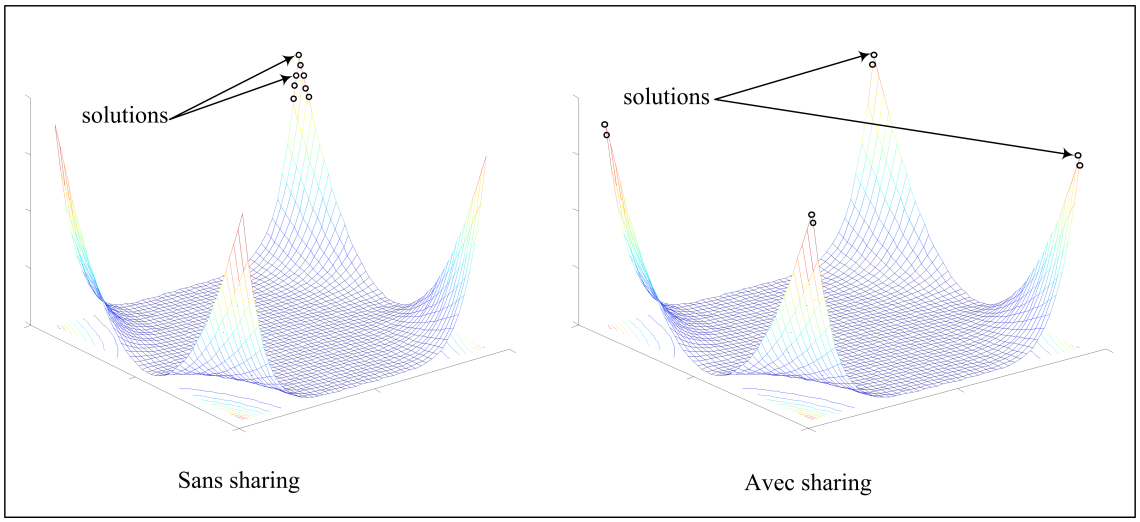

FIG. 1.4 – *Effet de sharing sur la répartition des solutions*

Si le principe de sharing a été utilisé initialement dans l'espace de paramètres (le critère de distance étant fonction des paramètres), il est tout a fait possible de l'adapter dans l'espace des objectifs. C'est d'ailleurs cette idée qui est exploitée dans les algorithmes génétiques multi-objectifs.

1.2.5.3 Elitisme

La stratégie élitiste consiste à conserver le meilleur individu à chaque génération. Ainsi l'élitisme empêche l'individu le plus performant de disparaître au cours de la sélection ou que ses bonnes combinaisons soient affectées par les opérateurs de croisement et de mutation. Après chaque évaluation de la performance des individus à une génération t donnée, le meilleur individu de la génération précédente $(t-1)$ est réintroduit dans la population si aucun des individus de la génération t n'est meilleur que lui. Par cette approche, la performance du meilleur individu de la population courante est monotone de génération en génération. Il apparaît que l'élitisme améliore considérablement les performances de l'algorithme génétique pour certaines classes de problèmes, mais peut les dégrader pour d'autres classes, en augmentant le taux de convergences prématurées.

1.3 Algorithmes Génétiques Hiérarchisés

En dépit des avantages des algorithmes génétiques standards (AGS) pour la résolution des problèmes d'optimisation difficiles où d'autres techniques ont échoué, ils présentent quand même des limites, quand on s'intéresse à l'optimisation de la topologie d'un système plutôt qu'à l'optimisation seule de ses paramètres, car le chromosome et la structure du phénotype dans l'AGS sont fixés ou prédéfinis, ce qui conduit à l'utilisation des AGH qui surmontent ces contraintes (Tang *et al.*, 1995; Xiang *et al.*, 2006). Les AGH diffèrent des AGS dans la structure des chromosomes. Ils présentent une structure hiérarchique et se composent de deux types de gènes : les gènes de contrôle et les gènes de paramètres. Les gènes de contrôle ont un rôle d'activation ou de désactivation des blocs, tandis que les gènes de paramètres définissent la valeur des coefficients dans chaque bloc.

1.3.1 Formulation du chromosome hiérarchisé

La formulation hiérarchique du chromosome se base sur le principe naturel de l'ADN : les gènes paramétriques (analogie aux gènes structuraux en ADN) et les gènes de contrôle sous forme de bits (analogie aux ordres de normalisation en ADN). Pour généraliser cette architecture, plusieurs niveaux de gènes de contrôle sont présentés d'une manière hiérarchique comme illustrée par la figure 1.5.

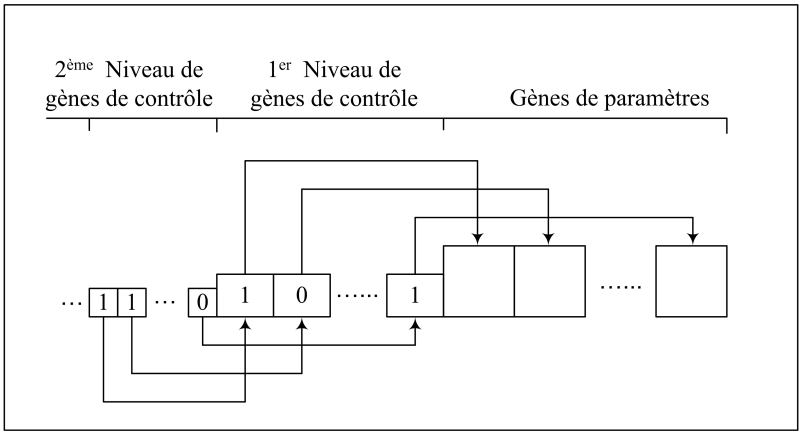

FIG. 1.5 – *Structure hiérarchique d'un chromosome*

Dans ce cas-là, l'activation des gènes paramétriques est régie par la valeur du premier niveau du gène de contrôle, qui est régi à son tour par la valeur du deuxième niveau

du gène de contrôle, et ainsi de suite. Pour indiquer l'activation du gène de contrôle, un nombre entier "1" est assigné pour chaque gène de contrôle actif et "0" pour chaque gène de contrôle inactif. Quand "1" est signalé, les gènes de paramètres associés à ce gène actif, sont activés dans la structure inférieure. Pour mieux comprendre le concept, prenons les exemples de la figure 1.6: Les chromosomes ch_1 et ch_2, sont constitués chacun de 6 gènes de contrôle et 6 gènes de paramètres. Le chromosome ch_1 correspond à une séquence de paramètres $S(ch_1) = (3,2,7,8)$ d'une longueur qui diffère de celle du chromosome ch_2 $(S(ch_2) = (2,5))$. Ceci signifie que le phénotype peut exister avec des longueurs différentes dans une structure fixe. Ainsi un AGH peut explorer toutes les longueurs possibles, y compris les paramètres de sort à satisfaire les exigences et les objectifs tracés. Enfin il est possible d'augmenter le niveau hiérarchique dans le chromosome en ajoutant un ou plusieurs niveaux de gènes de contrôle. Ceci est illustré par la figure 1.6-b où une structure à deux niveaux de gènes de contrôle est présentée.

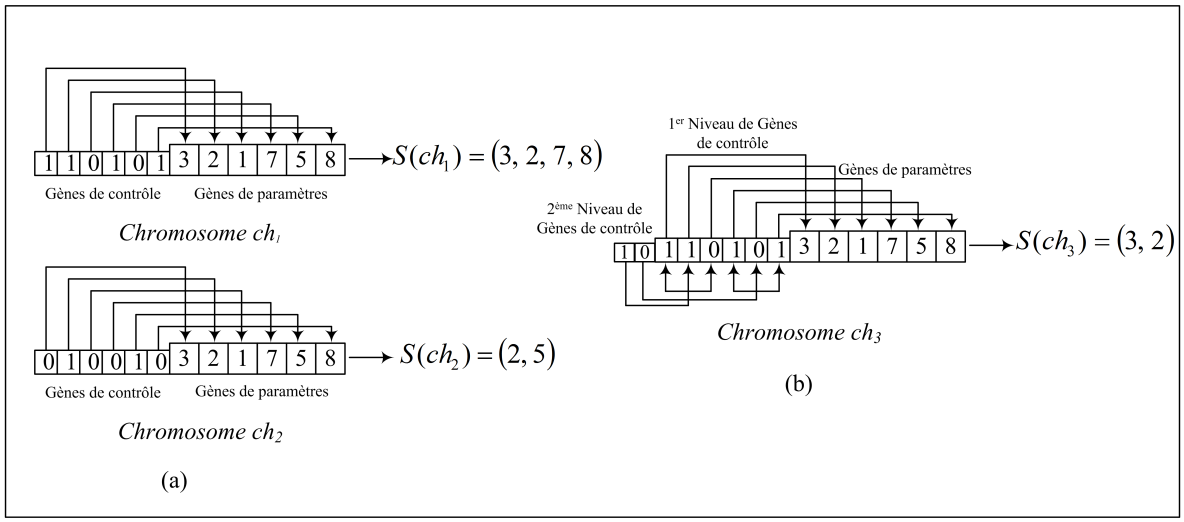

FIG. 1.6 – *Principe de décodage des chromosomes des AGH*

La structure d'un chromosome contient à la fois des gènes actifs et inactifs. Il faut noter que les gènes inactifs existent toujours dans le chromosome et peuvent être transmis ensuite aux nouvelles générations. Cette architecture hiérarchique implique que le chromosome contient plus d'informations que celle de la structure conventionnelle des AG. Par conséquent, la puissance artificielle est largement améliorée.

1.3.2 Opérateurs génétiques

Les opérateurs génétiques de base peuvent être utilisés sans aucune contrainte. La mutation et le croisement peuvent être appliqués indépendamment à chaque niveau de gènes de contrôle. Cependant, les opérateurs génétiques qui affectent les niveaux hauts des gènes de contrôle et produisent des changements des gènes actifs mènent à des changements multiples dans les niveaux inférieurs. C'est la raison pour laquelle, un AGH est capable d'obtenir le meilleur ensemble de paramètres d'un système avec une topologie ou structure minimisée.

1.4 Optimisation multi-objectifs

Dans la section précédente, nous avons considéré uniquement le cas où le problème à traiter possédait un objectif unique à optimiser. En pratique ces problèmes sont rares, il s'agit souvent de satisfaire plusieurs critères simultanément. L'optimisation multi-objectifs s'intéresse à ce type de problème que l'on peut définir de la manière suivante :

$$\begin{cases} min(F(X) = (f_1(X), f_2(X) ... f_n(X))) \\ X \in C \end{cases} \tag{1.11}$$

où n est le nombre de fonctions objectifs, $X = [x_1, x_2, ... x_m]$ est le vecteur représentant les variables de décision. C représente l'ensemble des solutions réalisables associé à des contraintes d'égalité et d'inégalité et $F(X) = (f_1(X), f_2(X) ... f_n(X))$ est le vecteur d'objectifs.

Dans un problème d'optimisation multi-objectifs, il y a plus qu'une fonction d'objectif ($n \geq 2$), chaque fonction peut avoir un optimum différent. Le but d'un problème multiobjectifs est donc de trouver de "bons compromis" plutôt qu'une seule solution (à moins qu'une solution soit optimale pour toutes les fonctions objectifs, ce qui est rarement le cas). Lorsqu'il y a plusieurs objectifs, la notion d'optimum change, elle est remplacée par les notions de dominance et d'optimalité de Pareto.

Définition 1 (la dominance): une solution A domine une solution B pour un problème de minimisation (resp. maximisation) si et seulement si:

$$\begin{aligned} &\forall i \in \{1, 2, \ldots, n\} : \quad f_i(A) \leq f_i(B) \, (resp. \, f_i(A) \geq f_i(B)) \\ &et \, \exists j \in \{1, 2, ..., n\} : \quad f_j(A) < f_j(B) \, (resp. \, f_j(A) > f_j(B)) \end{aligned} \tag{1.12}$$

On dit que B est dominée par A ou entre les deux solutions, A est la solution non dominée.

Définition 2 (Pareto optimum): un vecteur $X^* \in C$ est un optimum de Pareto s'il n'existe aucune solution X de C qui domine X^*.

Au lieu d'une solution unique, l'optimisation multi-objectifs donne lieu à un ensemble de solutions optimales. Toute solution de cet ensemble est optimale dans le sens qu'il est impossible d'améliorer les performances, sur un critère de cette solution, sans que cela entraîne une dégradation des performances, sur au moins un autre critère. Ces solutions optimales forment l'ensemble de solutions Pareto optimales, elles sont aussi connues sous le nom de *solutions efficaces*, *non inférieures* et *non dominées*. La représentation de ces solutions non dominées dans l'espace d'objectif est appelée front de Pareto. La figure 1.7 montre un exemple de front de Pareto pour un problème de minimisation à deux objectifs. L'ensemble de points blancs représentent le front de Pareto.

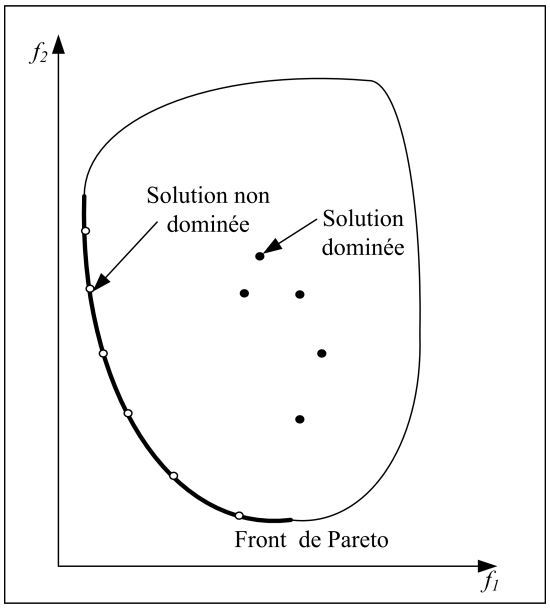

FIG. 1.7 – *Front de Pareto*

Les algorithmes génétiques, avec un bon réglage de leurs paramètres, constituent une approche intéressante pour la résolution des problèmes d'optimisation multi-objectifs. De plus, ce domaine est très dynamique et ne cesse de se développer. Il a été proposé plusieurs méthodes pour le traitement des problèmes multiobjectifs. Ces méthodes peuvent être classées principalement en deux catégories: La première catégorie enveloppe les méthodes "agrégatives", qui transforment le problème en un problème uni-objectif utilisant

une fonction objectif équivalente. La deuxième famille comporte les méthodes dites "non agrégatives", dans ces méthodes il n'y a pas fusion d'objectifs pour se ramener à un problème d'optimisation mono-objectif.

1.4.1 Méthode agrégative

C'est l'une des premières méthodes utilisée pour résoudre les problèmes d'optimisation multiobjectifs (MO). Elle consiste à transformer le problème MO en un problème monoobjectif en combinant les composantes f_i du vecteur objectif du problème en une seule fonction scalaire f. Il existe dans la pratique, différentes façons de construire la fonction f . La plus classique et la plus utilisée se ramène à une simple somme pondérée des objectifs f_i (agrégation additive):

$$f = \sum_{i=1}^{n} \lambda_i \times f_i \tag{1.13}$$

où les paramètres λ_i sont les poids de pondération. La figure 8 illustre l'interprétation géométrique de la méthode de pondération dans le cas d'un problème à deux objectifs. Fixer un vecteur de poids revient à trouver un hyper plan dans l'espace objectif (une ligne pour un problème bi-objectif). La solution Pareto optimale est le point où l'hyper-plan possède une tangente commune avec l'espace réalisable. La méthode de pondération est relativement simple à utiliser mais il est assez difficile de fixer a priori les valeurs des poids associés à chaque objectif. Il est par ailleurs impératif de normaliser les objectifs lorsque ceux-ci sont non-commensurables, ce qui est souvent le cas. Ce problème de mise à l'échelle s'avère complexe car les valeurs de normalisation ne sont pas faciles à déterminer. De plus, comme le montre la figure 1.8-b, cette méthode n'est pas adaptée aux espaces non convexes qui empêchent l'algorithme de converger vers l'optimum global. Le champ applicatif de cette méthode se trouve alors fortement réduit.
Grâce au principe de pondération, il est possible avec un algorithme génétique standard d'obtenir le front de Pareto d'un problème multi-objectifs. Il est nécessaire pour cela d'exécuter l'algorithme plusieurs fois successives en faisant varier les facteurs de pondération (voir figure 1.9).

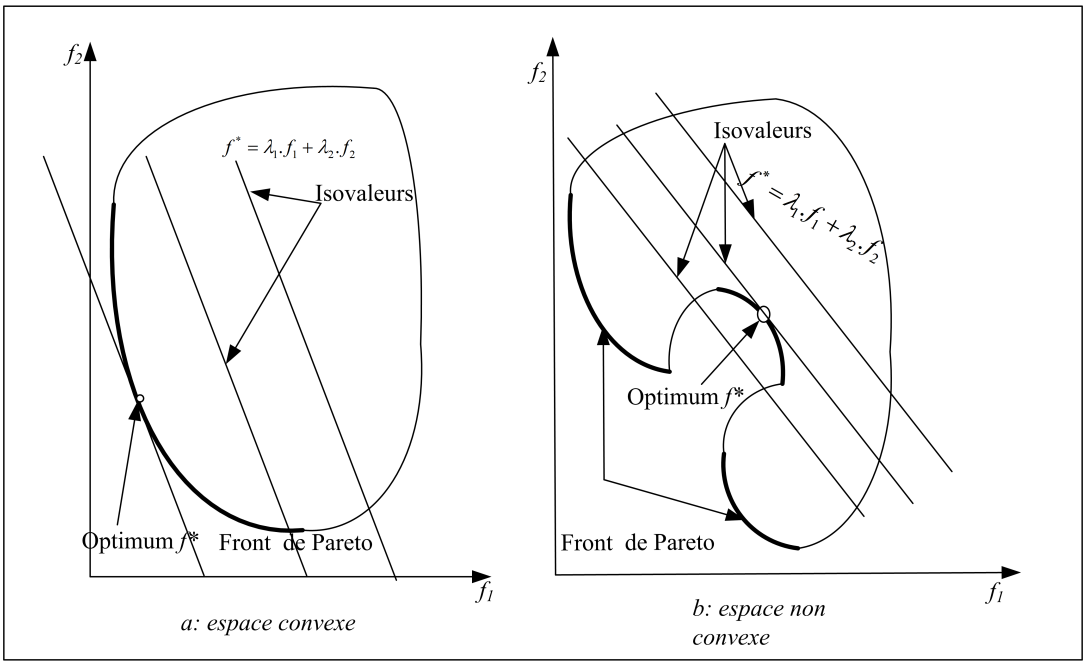

FIG. 1.8 – *Interprétation géométrique de la méthode d'agrégation*

Toutefois, une variation uniforme des poids ne garantit pas la détermination des solutions Pareto optimales uniformément réparties sur le front.

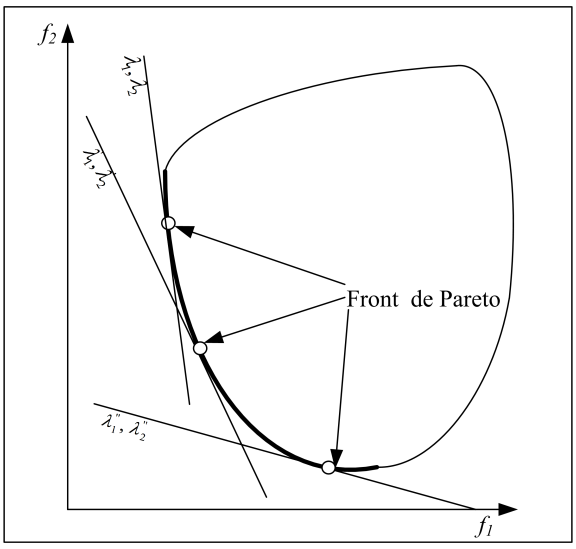

FIG. 1.9 – *Illustration de la variation des coefficients de pondération pour la recherche du front de Pareto*

1.4.2 Méthode non agrégative

Comme nous l'avons signalé précédemment, la méthode agrégative peut être utilisée de façon séquentielle pour obtenir le front de Pareto pour un problème d'optimisation multi-objectifs. Toutefois, cette approche n'est généralement pas satisfaisante car le nombre d'exécutions successives nécessaires pour déterminer les différents compromis conduit alors à un nombre d'évaluations de critères prohibitif. Ainsi, pour surmonter cette difficulté, on préfère utiliser des méthodes permettant d'une part de trouver l'ensemble de solutions Pareto optimales en une seule exécution et d'autre part de s'affranchir des problèmes de mise à l'échelle des objectifs. Parmi ces méthodes on peut citer :

- Vector Evaluated Genetic Algorithm (VEGA)
- Multiple Objective Genetic algorithm (MOGA)
- Niched Pareto genetic algorithm (NPGA)
- Nondominated sorting genetic algorithm (NSGA) et sa deuxième version (NSGA-II)

1.4.2.1 Vector evaluated genetic algorithm (VEGA)

Le VEGA (Vector evaluated genetic algorithm) proposé par Schaffer (1985), a été la première méthode non agrégative utilisant les algorithmes génétiques pour résoudre un problème d'optimisation multiobjectif. Cet algorithme considère une population de N individus. A chaque génération, la population est divisée en un nombre de sous populations égal au nombre d'objectifs. Chaque sous population i est sélectionnée en considérant un seul objectif f_i. Ensuite, ces sous populations sont regroupées afin d'obtenir une nouvelle population de N individus et les opérateurs de croisement et de mutation sont appliqués.
L'avantage de cet algorithme est qu'il est facile à implémenter et à combiner avec n'importe quel mode de sélection (tournoi, roulette, rang), mais son inconvénient majeur est qu'il a tendance à générer des solutions qui excellent dans un seul objectif, sans tenir compte des autres objectifs (points extrêmes du front). Toutes les solutions de performance moyenne (ne possédant aucun objectif fort) et qui peuvent être de bons compromis, risquent de disparaître avec ce type de sélection.

1.4.2.2 Multiple Objectives Genetic Algorithm (MOGA)

Cet algorithme, proposé par Fonseca et Fleming (1993), utilise la notion de dominance pour ranger les individus de la population. Il diffère de l'algorithme génétique standard uniquement dans la manière dont la fitness est assignée pour chaque solution. Pour démarrer l'algorithme, les relations de domination sont d'abord calculées pour chaque solution. Puis, pour une solution i, un rang égal à un plus le nombre de solutions n_i qui dominent la solution i est attribué. Une fitness est ensuite attribuée à chaque solution en fonction de son rang, les individus avec les rangs les plus faibles ayant les meilleures fitness. Afin de maintenir la diversité entre les solutions non dominées, les auteurs utilisent une fonction de partage (Sharing). La méthode permet d'obtenir des solutions de bonne qualité et s'implante facilement. Toutefois, les performances sont très dépendantes de la valeur du paramètre σ_{shar} utilisé dans le sharing.

1.4.2.3 NSGA (Nondominated Sorting Genetic Algorithm)

Dans l'algorithme NSGA proposé par Srinivas et Deb (1993), le calcul de fitness s'effectue en divisant d'abord la population en plusieurs fronts en fonction du degré de dominance au sens de Pareto de chaque individu. Les individus non dominés de la population courante constituent le premier front de Pareto. On attribue alors à tous les individus de ce front la même valeur de fitness factice. Cette valeur est supposée donner une chance égale de reproduction à tous ces individus. Mais pour maintenir la diversité de la population, il est nécessaire d'appliquer une fonction de partage sur cette valeur. En suite, ce premier groupe d'individus est temporairement supprimé de la population. On recommence cette procédure jusqu'à l'identification des solutions du deuxième front. La valeur factice de fitness attribuée à ce second groupe est inférieure à la plus petite fitness, après application de la fonction de partage sur le premier front. Ce mécanisme est répété jusqu'à ce que l'on ait traité tous les individus. L'algorithme se déroule ensuite comme un algorithme génétique standard. Grâce à sa procédure d'assignement de fitness basée à la fois sur la notion de dominance et la fonction de partage, le NSGA semble le plus approprié à maintenir la diversité de la population et à répartir plus efficacement les solutions sur le front de Pareto. Néanmoins, cet algorithme présente quelques insuffisances en raison de sa complexité de calcul et de sa sensibilité au choix de la valeur σ_{shar}.

1.4.2.4 Niched Pareto Genetic Algorithm (NPGA)

Cette méthode proposée par Horn et Napfliotis (1994) utilise une sélection par tournoi en se basant sur la notion de dominance de Pareto. Le NPGA exécute les mêmes étapes que l'AG standard, la seule chose qui diffère étant la méthode de sélection. A chaque tournoi, deux individus candidats A et B sont pris au hasard dans la population courante. Au lieu de limiter la comparaison aux deux individus (comme c'est le cas pour l'AG standard), une sous population (ou ensemble de comparaison) de taille t_{dom} est également choisie au hasard. Les deux candidats sélectionnés sont comparés à chaque individu du sous-groupe. Si l'un des candidats est dominé par l'ensemble de comparaison et le second ne l'est pas, ce dernier est alors positionné dans la population suivante. Dans les autres cas, une fonction de partage est appliquée pour choisir le candidat gagnant. Le paramètre t_{dom} permet de contrôler la pression de sélection ou de dominance.
L'algorithme NPGA est considéré comme étant l'algorithme le plus rapide parmi les approches précédentes car à chaque génération la comparaison n'est appliquée que sur une portion de la population. Le principal inconvénient de cet algorithme est qu'il nécessite, en plus de spécifier le paramètre de sharing σ_{shar}, un autre paramètre supplémentaire qui est la taille du tournoi t_{dom}.

1.4.2.5 NSGA-II

Toutes les méthodes que nous venons de présenter ne conservent pas leurs solutions Pareto-optimales trouvées au cours des générations. Elles sont dites non élitistes. Pour résoudre cette difficulté, de nouvelles techniques ont été appliquées. Nous avons choisi de présenter uniquement le NSGA-II car il a été utilisé dans nos travaux. Nous ne ferons que citer les deux autres méthodes les plus connues à savoir Strength Pareto Evolutionary Algorithm (SPEA) et Pareto Envelope-based Selection Algorithm (PESA) présentées plus amplement dans les références (Zitzler et Thiele, 1999) et (Corne *et al.*, 2000).
En proposant le NSGA II, Deb (Deb *et al.*, 2002) a tenté de résoudre toutes les critiques faites sur NSGA: non élitiste, complexité de calcul et utilisation de sharing qui implique le réglage d'un ou plusieurs paramètres. Dans cet algorithme, à chaque génération t une population de parents (P_t) de taille N et une population d'enfants (Q_t) de même taille sont assemblées pour former une population (R_t) de taille $2N$, comme indiqué sur la figure 1.10. Cet assemblage permet d'assurer l'élitisme. La population (R_t) est ensuite

répartie en plusieurs fronts $(F_1, F_2, \ldots)$ par une procédure de tri, plus rapide que celle proposée dans la première version de NSGA. Une nouvelle population parent (P_{t+1}) est formée en ajoutant les fronts au complet (premier front F_1, second front F_2, etc...) tant que ce ceux-ci ne dépassent pas N. Si le nombre d'individus présents dans (P_{t+1}) est inférieur à N, une procédure de crowding est appliquée sur le premier front suivant F_i non inclus dans (P_{t+1}). Le but de cet opérateur est d'insérer les $(N - P_{t+1})$ meilleurs individus de F_i qui manquent dans la population (P_{t+1}). Une fois que les individus de la population (P_{t+1}) sont identifiés, une nouvelle population enfant (Q_{t+1}) est créée par sélection, croisement et mutation. La sélection par tournoi est utilisée mais le critère de sélection est maintenant basé sur l'opérateur de comparaison $(<_n)$ défini ci-dessous. Le processus se répète d'une génération à une autre jusqu'à satisfaction d'un critère d'arrêt.

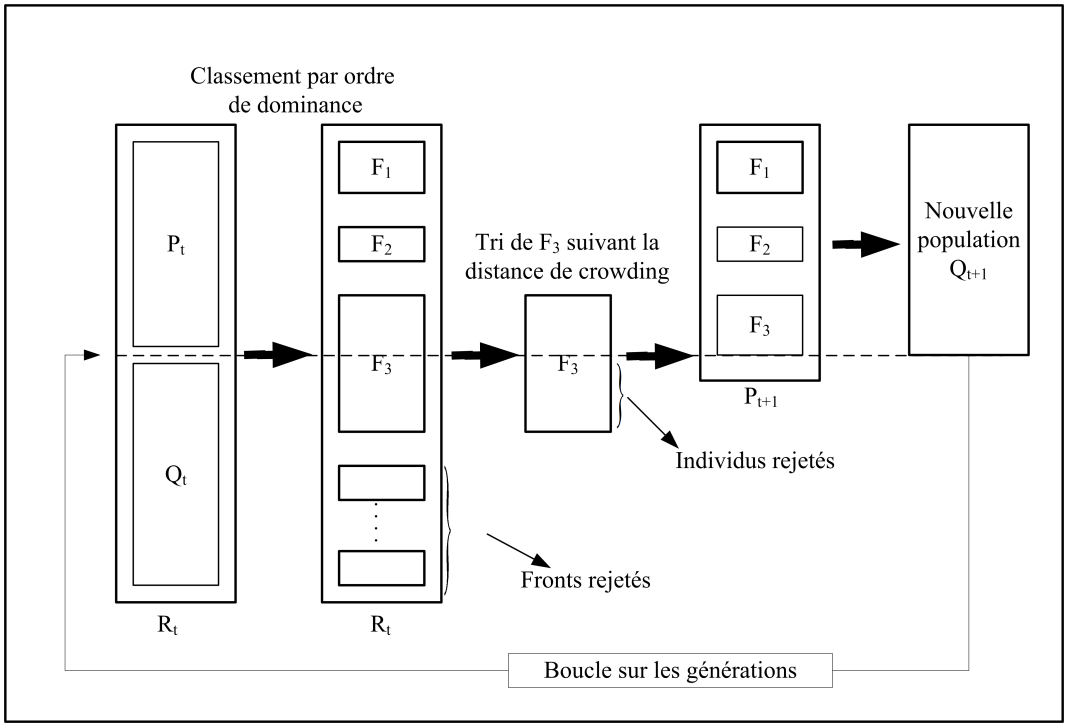

FIG. 1.10 – *Schéma de l'évolution de l'algorithme NSGA-II*

Procédure de tri rapide (Fast sorting non dominated)

La répartition de la population en plusieurs fronts s'effectue de la manière suivante :

1. Pour chaque solution p de R_t on calcule deux paramètres:
   - n_p (compteur de dominance), il représente le nombre de solutions qui dominent la solution p

– S_i ensemble de solutions dominées par p.

2. $i = 1$ initialisation du compteur de front

3. Identification des solutions non dominées $n_p = 0$, ces solutions forment le front F_i.

4. Pour chaque solution p de F_i on parcourt l'ensemble S_p et on retranche 1 au n_p de chaque solution.

5. $i = i + 1$ incrémente le compteur de front

6. On recommence les étapes à partir de 3 jusqu'à ce que tous les points soient traités.

Cet algorithme est d'une complexité de $O(k.N^2)$, alors que celui utilisé dans la première version est de $O(k.N^3)$. k étant la taille du vecteur objectifs

Distance de crowding

La dernière critique faite sur le NSGA est l'utilisation du sharing. Une méthode qui exige le réglage d'un ou plusieurs paramètre(s). Dans NSGA-II, Deb et al remplacent la procédure de sharing par une procédure de crowding, basée sur un calcul de distance (distance de crowding) qui ne nécessite aucun paramétrage et qui est également d'une complexité algorithmique moindre que celle de sharing. La distance de crowding d'une solution particulière i se calcule en fonction du périmètre de l'hypercube ayant comme sommets les points les plus proches de i sur chaque objectif. Sur la figure 1.11, est représenté l'hypercube en deux dimensions associé au point i. Le calcul de la distance de crowding nécessite, avant tout, le tri des solutions selon chaque objectif, dans un ordre ascendant. Ensuite, pour chaque objectif, les individus possédant des valeurs limites se voient associés une distance infinie. Pour les autres solutions intermédiaires, on calcule une distance de crowding égale à la différence normalisée des valeurs des fonctions objectifs de deux solutions adjacentes. Ce calcul est réalisé pour chaque objectif. La distance de crowding d'une solution est obtenue en sommant les distances correspondantes à chaque objectif.

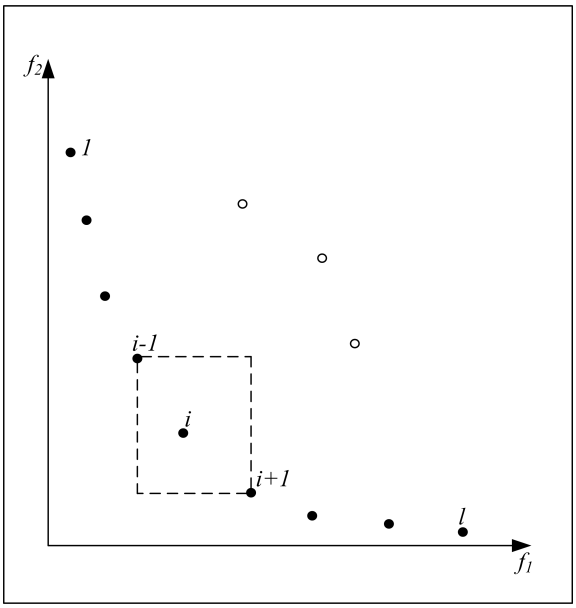

FIG. 1.11 – *Distance de crowding (les points noirs sont des solutions appartenant au même front)*

L'algorithme 1 reprend toutes les étapes décrites ci-dessus.

Algorithm 1 Calcul de la distance de crowding pour chaque solution d'un front

1: $l = |I|$, nombre de solutions dans le Front I

2: Pour chaque solution i poser $I[i]_{distance} = 0$, Initialisation des distances.

3: Pour chaque objectif m:

-$I = trier(I,m)$, trier I par ordre croissant selon le critère m

-$I[1]_{distance} = I[l]_{distance} = +\infty$

-Pour i=2 jusqu'à $l - 1$ faire

$I[i]_{distance} = I[i]_{distance} + (\frac{f_m^{i+1} - f_m^{i-1}}{f_m^{Max} - f_m^{Min}})$

Dans cet algorithme, f_m^{i+1} et f_m^{i-1} représentent respectivement les valeurs de la *m*ième fonction objectif des solutions $i + 1$ et $i - 1$ alors que les paramètres f_m^{Max} et f_m^{Min} désignent les valeurs maximale et minimale de la *m*ième fonction objectif.

Opérateur de crowding de comparaison($\prec_n$)

Cet opérateur est utilisé pour guider le processus de sélection comme suit : chaque solution i de la population est identifiée par son rang i_{rank} et sa distance de crowding $i_{distance}$.

L'opérateur $\prec_n$, défini ci-dessous, permet d'établir un ordre de préférence entre deux solutions:

$$\begin{aligned} i \prec_n j \quad & si \quad (i_{rank} < j_{rank}) \\ & ou \quad (i_{rank} = j_{rank}) \; et \; (i_{distance} > j_{distance}) \end{aligned}$$

Entre deux solutions de fronts différents, on préfère la solution avec le plus petit front. Pour deux solutions qui appartiennent au même front, on préfère la solution située dans une région dépeuplée, c'est-à-dire la solution possédant la plus grande valeur de distance de crowding.

1.5 Conclusion

Dans ce chapitre, nous avons décrit le fonctionnement et les différents opérateurs d'un algorithme génétique standard, technique d'optimisation mono-objectif. Nous avons vu que ces algorithmes présentent des inconvénients lorsqu'on s'intéresse à l'optimisation de la topologie d'un système plutôt qu'à l'optimisation seule des ses paramètres. Cela nous a conduits à introduire une autre version d'algorithmes mieux adaptée à cette classe de problèmes nommée "algorithmes génétiques hiérarchisés". Nous avons également décrit dans ce chapitre le problème d'optimisation multi-objectifs et quelques algorithmes génétiques de résolution rapportés dans la littérature. Parmi les algorithmes examinés, le NSGA-II semble être aujourd'hui l'une des techniques de référence qui garantit la diversité des solutions Pareto-optimal sans qu'il soit nécessaire de connaître le rayon de niche.

Chapitre 2

Réseaux de neurones et systèmes flous

2.1 Introduction

L'utilisation des réseaux de neurones (RN) et des systèmes flous (SF) pour la synthèse des systèmes de commande a connu un essor important au cours de ces dernières années. Dans ce chapitre, nous allons décrire ces outils ainsi que les modalités de leur utilisation. Le chapitre est organisé en trois parties. La première partie fait l'objet d'une étude détaillée sur l'emploi des réseaux de neurones pour la commande. Après une brève présentation de quelques notions générales sur les réseaux de neurones, nous présentons les deux grandes familles de structures neuronales les plus utilisées. Nous abordons également le problème d'apprentissage des paramètres de ces structures, nous nous intéressons particulièrement à l'algorithme de rétropropagation. Nous exposons dans un deuxième temps, la classification des principales approches de la commande par réseaux de neurones rencontrées dans la littérature. Notre but n'est pas de dresser une liste complète mais plutôt de dégager les lignes directrices communes. Certaines architectures sont ainsi absentes dans cette classification mais on peut généralement les rattacher à l'une des classes proposées. La deuxième partie sera consacrée aux systèmes flous. Après une présentation générale de quelques notions sur les ensembles flous, nous présentons en détail la structure interne d'un contrôleur flou. Ce dernier est considéré comme l'une des applications les plus importantes des systèmes flous. Dans la troisième partie, nous aborderons la technique d'hybridation entre ces deux paradigmes (SF et RN), employée fréquemment pour la modélisation et la commande des systèmes complexes.

2.2 Réseaux de neurones

2.2.1 Le neurone formel

Un neurone formel est un automate très simple imitant grossièrement la structure et le fonctionnement d'un neurone biologique. La première version de ce dernier est celle de Mc Culloch et W. Pitts et date de 1943. S'inspirant de leurs travaux sur les neurones biologiques, ils ont proposé le modèle du neurone formel qui se voit comme un opérateur effectuant une somme pondérée de ses entrées suivie d'une fonction d'activation (ou de transfert) comme indiqué par la figure 2.1.

– O_i représente la somme pondérée des entrées du neurone, elle est donnée par :

$$U_i = \sum_j \omega_{ij} \times x_j + b_i \tag{2.1}$$

où: x_j représente l'entrée j connectée au neurone i. b_i le seuil interne du neurone. ω_{ij} désigne le poids de la connexion reliant l'entrée j au neurone.

– $O_i = g(U_i)$ est la sortie du neurone et g sa fonction d'activation.

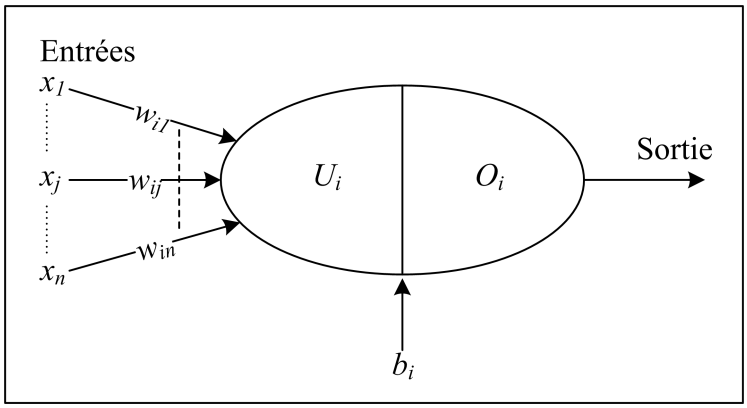

FIG. 2.1 – *Modèle de base d'un neurone formel*

La fonction d'activation de chaque neurone détermine ses propres caractéristiques. Par conséquent, le type du neurone est caractérisé par sa fonction d'activation. Conformément au neurone biologique, les fonctions d'activation sont généralement croissantes et continues. Les fonctions les plus utilisées sont la fonction linéaire et la fonction sigmoïde. Leur choix revêt une importance capitale et dépend souvent du type de l'application et du domaine de variation des variables d'entrée/sortie.

Un réseau de neurones (RN) est un système informatique qui a des caractéristiques semblables aux réseaux de neurones biologiques. Il est constitué de plusieurs unités (neurones) organisées sous forme de niveaux différents appelés couches du réseau. Les neurones appartenant à la même couche possèdent les mêmes caractéristiques et utilisent le même type de fonction d'activation. Entre deux couches voisines les connexions se font par l'intermédiaire de poids qui jouent le rôle des synapses. L'information est portée par la valeur de ses poids, tandis que la structure du réseau de neurones ne sert qu'à traiter l'information et l'acheminer vers la sortie. La structure ou la topologie d'un réseau de neurones est la manière dont les neurones sont connectés. Les structures résultantes peuvent être très variées mais elles sont souvent réparties en deux grandes familles à savoir : les réseaux de neurones non bouclés et les réseaux de neurones bouclés (Burns, 2001).

2.2.2 Réseaux non bouclés

Dans ce type de structure dite feedforward, la propagation de l'information se fait uniquement de l'entrée vers la sortie. Les neurones de la même couche peuvent se connecter uniquement avec les neurones de la couche suivante. L'architecture la plus utilisée est le Perceptron multicouches. Les neurones composant ce réseau s'organisent en N couches successives ($N \geq 3$). Dans l'exemple suivant (figure 2.2), nous présentons un perceptron à trois couches. Les neurones de la première couche, nommée couche d'entrée, voient leur activation forcée à la valeur d'entrée. La dernière couche est appelée couche de sortie. Elle regroupe les neurones dont les fonctions d'activation sont généralement de type linéaire. Les couches intermédiaires sont appelées couches cachées. Elles constituent le cœur du réseau. Les fonctions d'activation utilisées sont de type sigmoïde.

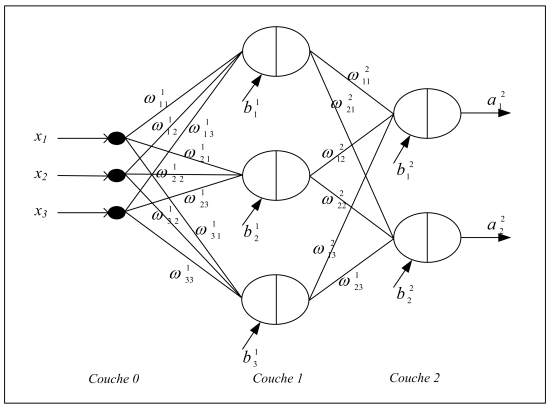

FIG. 2.2 – *Perceptron à une couche cachée*

Sur la figure 2.2, les termes b_i^l et ω_{ij}^l désignent respectivement le biais du neurone i de la couche l et le poids de connexion entre le neurone j de la couche $l-1$ et le neurone i de la couche l.
Tenant compte de ces notations, la sortie du neurone i dans la couche l est peut être donnée par :

$$U_i^l = \sum_j^{N_{l-1}} \omega_{ij}^l \times O_j^{l-1} + b_i^l \quad (2.2)$$

$$O_i^l = g^l(U_i^l) \quad (2.3)$$

$$l = 1,2$$

où $g^l(.)$ est la fonction d'activation des neurones de la couche l.
On peut réécrire les équations ci-dessus sous forme matricielle comme suit :

$$\underline{U^l} = W^l \times \underline{O}^{l-1} + \underline{b}^l \quad (2.4)$$

$$\underline{O^l} = \underline{g}^l(\underline{U^l}) \quad (2.5)$$

avec : $\underline{U^l} = (U_1^l, U_2^l, ..., U_{N_l}^l)^T$, $\underline{O^l} = (O_1^l, O_2^l, ..., O_{N_l}^l)^T$, $\underline{b}^l = (b_1^l, b_2^l, ..., b_{N_l}^l)^T$ et

$$W^l = \begin{pmatrix} \omega_{11}^l & \omega_{12}^l & \cdots & \omega_{1N_{l-1}}^l \\ \omega_{21}^l & \omega_{22}^l & \cdots & \omega_{2N_{l-1}}^l \\ \vdots & \vdots & \ddots & \cdots \\ \omega_{N_l 1}^l & \omega_{12}^l & \cdots & \omega_{N_l N_{l-1}}^l \end{pmatrix}$$

Le perceptron multicouche présente une alternative prometteuse pour la modélisation des systèmes complexes. Avec une seule couche cachée, il constitue un approximateur universel. Les études menées dans (Hornik *et al.*, 1989; Cybenko, 1989) montrent qu'il peut être entrainé de manière à approximer n'importe quelle fonction sous réserve de mettre suffisamment de neurones dans la couche cachée et d'utiliser des sigmoïdes comme fonctions d'activation.

2.2.3 Réseaux bouclés

Un réseau dynamique ou récurrent possède la même structure qu'un réseau multicouches munie de rétroactions. Les connexions rétroactives peuvent exister entre tous les neurones du réseau sans distinction, ou seulement entre certains neurones (les neurones de la couche de sortie et les neurones de la couche d'entrée ou les neurones de la même couche par exemple). La figure 2.3 montre deux exemples de réseaux récurrents. Le premier est un simple multicouches qui utilise un vecteur d'entrée qui contient les copies des activations de la couche de sortie du réseau et le deuxième est un réseau à mémoire se distingue du premier par la présence des unités mémoires (Sastry *et al.*, 1994).

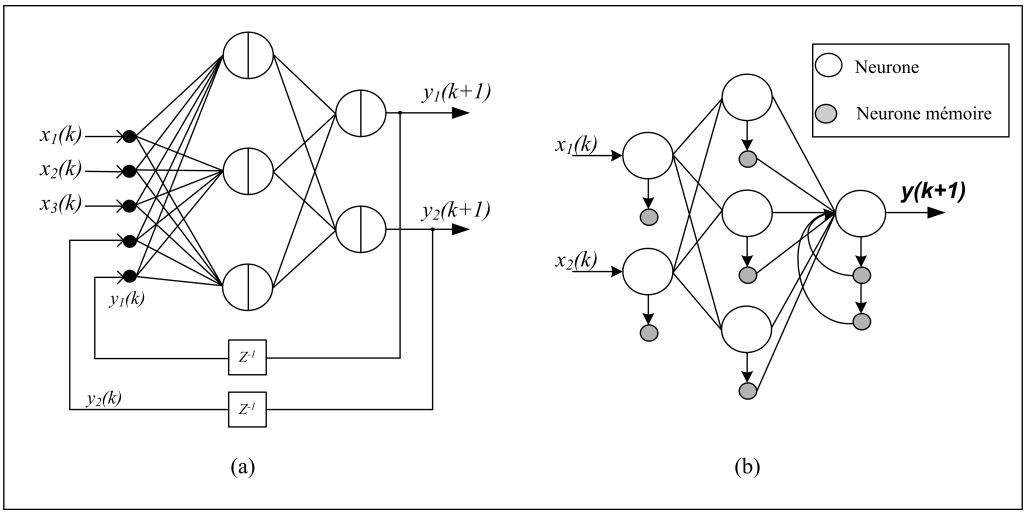

FIG. 2.3 – *Réseaux de neurones récurrents*

2.2.4 Apprentissage dans les réseaux de neurones

L'apprentissage dans le contexte des réseaux de neurones, est le processus de modification des poids de connexions (y compris les biais) ou plus rarement du nombre de couches et de neurones (Man et Halang, 1997), afin d'adapter le traitement effectué par le réseau à une tache particulière.
On distingue trois familles d'apprentissage :

– *apprentissage supervisé*: Dans ce cas, un superviseur(ou expert humain) fournit une valeur ou un vecteur de sortie (appelé cible ou sortie désirée), que le réseau de neurones doit associer à un vecteur d'entrée. L'apprentissage consiste dans ce cas à

ajuster les paramètres du réseau afin de minimiser l'erreur entre la sortie désirée et la sortie réelle du réseau.

– *apprentissage semi-supervisé ou apprentissage par renforcement*: Ce mode d'apprentissage, suppose qu'un comportement de référence n'est pas possible, mais en revanche, il est possible d'obtenir des indications qualitatives (vrai, faux, ...) sur les performances du réseau.
– *apprentissage non supervisé*: Dans cet apprentissage, les données ne contiennent pas d'information sur la sortie désirée. Il n'y a pas de superviseur. La tâche du réseau consiste, par exemple dans ce cas, à créer des regroupements de données selon des propriétés communes (classification).

2.2.4.1 Algorithme de rétropropagation du gradient

L'algorithme de rétropropagation (backpropagation) est l'un des algorithmes supervisé les plus utilisés pour l'apprentissage des réseaux de neurones. C'est d'ailleurs à sa découverte au début des années 80 (Rumelhart et McClelland, 1986) que l'on doit le renouveau d'intérêt pour les réseaux de neurones. L'objectif de cet algorithme est de modifier les poids du réseau dans le sens contraire du gradient du critère de performance.
Dans ce qui suit, nous allons présenter les équations constituant l'algorithme en utilisant un réseau multicouches. Une mise sous forme matricielle sera aussi faite afin de faciliter l'implantation de l'algorithme sous un logiciel bien adapté aux calculs matriciels.
Considérons le réseau multicouche décrit précédemment. Pour alléger l'exposé, on suppose que l'apprentissage se fait à chaque présentation d'un couple entrée/sortie de l'ensemble d'apprentissage. Le critère de performance à minimiser peut être alors exprimé par :

$$J(t) = 0.5 \times \sum_{i=1}^{N_L} (O_i^L(t) - d_i(t))^2 \tag{2.6}$$

avec:
$J(t)$est la valeur du critère à l'instant t.
$d_i(t)$ est la ième sortie désirée à l'instant t.
Les paramètres du réseau sont modifiés suivant la règle du gradient comme suit:

$$\omega_{ij}^l(t+1) = \omega_{ij}^l(t) - \eta \frac{\partial J(t)}{\partial \omega_{ij}^l(t)} \tag{2.7}$$

$$b_i^l(t+1) = b_i^l(t) - \eta \frac{\partial J(t)}{\partial b_i^l(t)} \tag{2.8}$$

avec η est une constante positive appelée taux d'apprentissage.
Le calcul des quantités $\frac{\partial J}{\partial \omega}$ et $\frac{\partial J}{\partial b}$ fait intervenir les décompositions ci-dessous:

$$\frac{\partial J(t)}{\partial \omega_{ij}^l(t)} = \frac{\partial J(t)}{\partial U_i^l(t)} \times \frac{\partial U_i^l(t)}{\partial \omega_{ij}^l(t)} \tag{2.9}$$

$$\frac{\partial J(t)}{\partial b_i^l(t)} = \frac{\partial J(t)}{\partial U_i^l(t)} \times \frac{\partial U_i^l(t)}{\partial b_i^l(t)} \tag{2.10}$$

De l'équation (2.2) on déduit que:

$$\frac{\partial U_i^l(t)}{\partial \omega_{ij}^l(t)} = O_j^{l-1} \tag{2.11}$$

$$\frac{\partial U_i^l(t)}{\partial b_i^l(t)} = 1 \tag{2.12}$$

En posant, $\delta_i^l(t) = \frac{\partial J(t)}{\partial U_i^l}$ on obitent:

$$\frac{\partial J(t)}{\partial \omega_{ij}^l(t)} = \delta_i^l(t) \times O_j^{l-1} \tag{2.13}$$

$$\frac{\partial J(t)}{\partial b_i^l(t)} = \delta_i^l(t) \tag{2.14}$$

La quantité δ_i^l exprime la sensibilité du critère de performance aux changements du *potentiel* U_i^l du neurone i de la couche l. Dans le cas où i est l'indice d'un neurone de sortie ($l = L$), on obtient :

$$\delta_i^L(t) = \frac{\partial J(t)}{\partial U_i^L} = \frac{\partial J(t)}{\partial O_i^L} \times \frac{\partial O_i^L}{\partial U_i^L} = (O_i^L(t) - d_i(t)) \times \dot{g}^l(U_i^L(t)) \tag{2.15}$$

avec
$\dot{g}^L(U_i^L(t)) = \frac{dg^L(U_i^L(t))}{dU_i^L(t)}$
Dans le cas où i est l'indice d'un neurone caché ($1 < l < L-1$), on peut vérifier aisément que les fonctions de sensibilité satisfont la relation récurrente ci-dessous (Hagan et Menhaj, 1994):

$$\underline{\delta^l} = \dot{G}^l(\underline{U^l}) \times (W^{l+1})^T \times \underline{\delta}^{l+1} \tag{2.16}$$

où

$$\dot{G}^l(\underline{U^l}) = \begin{pmatrix} \dot{g}^l(U_1^l(t)) & 0 & \cdots & 0 \\ 0 & \dot{g}^l(U_2^l(t)) & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \dot{g}^l(U_1^l(t)) \end{pmatrix} \tag{2.17}$$

Pour résumer, l'algorithme de mise à jour des paramètres du réseau se déroule comme suit : premièrement le vecteur d'entrée $\underline{U}^0 = (x_1, x_2, \cdots, x_{N_0})^T$ est propagé vers la sortie en utilisant les équations (2.2) et (2.3). Ensuite, on calcule les fonctions de sensibilités par rétropropagation de l'erreur de sortie à l'aide des équations (2.15) et (2.16). Finalement on modifie les poids et les biais en utilisant les équations (2.13), (2.14), (2.7) et (2.8).
Cet algorithme, présenté ici dans sa version la plus simple, possède de nombreuses variantes. Elles correspondent pour la plupart à l'adaptation du coefficient d'apprentissage η (Magoulasb *et al.*, 1999), ou à l'utilisation de méthodes du deuxième ordre pour le calcul du gradient (Hagan et Menhaj, 1994). Nous pouvons également citer parmi ces variantes la méthode avec un terme de momentum (Vogl *et al.*, 1988), qui utilise une version légèrement différente des équations (3.7) et (3.8) pour ajuster les paramètres du réseau:

$$p(t+1) = p(t) - \eta \frac{\partial J(t)}{\partial p(t)} + \alpha \cdot \Delta p(t) \tag{2.18}$$

où α est une constante appelée *momentum* et p représente un paramètre du réseau qui peut être soit un poids de connexion ou un biais. Cette version introduit un autre terme proportionnel à la dernière adaptation Δp du paramètre p.

2.2.5 Identification par réseaux de neurones

L'identification des systèmes linéaires est actuellement un domaine bien maîtrisé par l'automaticien. Cependant, de nombreux processus réels sont complexes et présentent des dynamiques non linéaires. De plus, sur ces mêmes systèmes, les connaissances des phénomènes physico-chimiques mises en jeu ne sont pas forcément mesurées. Le recours à des modèles de type "boite noire" devient nécessaire et l'identification du modèle est réalisée autour d'un ensemble de données entrées/sorties. Un réseau de neurone, qualifié de boîte noire, présente un outil incontournable pour l'identification de tels systèmes.
La figure 2.4 montre le schéma général d'identification neuronale. Sur cette figure, le réseau de neurones est utilisé en parallèle avec un processus de type boite noire à identifier. La sortie y du processus est comparée à la sortie $\hat{y}$ du réseau de neurones puis l'erreur $e_i = y - \hat{y}$ est utilisée par un algorithme d'apprentissage approprié (exemple la rétropropagation) pour ajuster les paramètres du réseau neuronal.

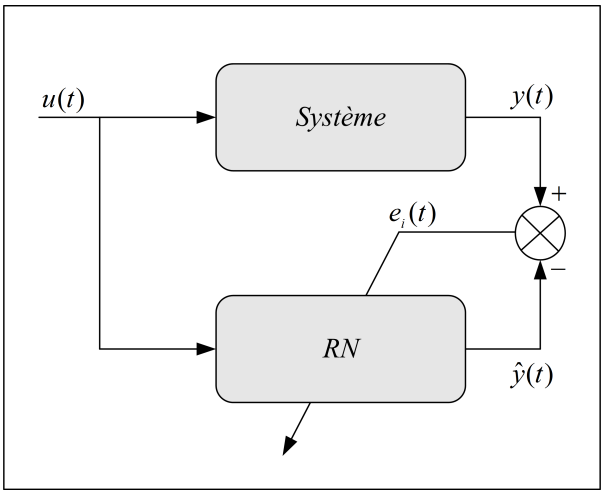

FIG. 2.4 – *Schéma de principe d'identification par réseau de neurones*

Pour tenir compte de l'aspect dynamique du processus, le vecteur d'entrée du réseau est souvent renforcé par des signaux correspondants aux valeurs antérieures des sorties du système ou du réseau. Si l'on utilise les sorties du processus (figure 2.5), la structure d'identification est dite série parallèle, par opposition à la structure parallèle qui exploite les sorties du réseau (figure 2.6). Le modèle parallèle est un cas particulier de la structure générale d'un réseau récurrent, par conséquent il est intéressent de remplacer la boucle donnant l'état du système par une connexion récurrente afin de réduire la taille du vecteur d'entrée. Cependant, il est difficile de vérifier dans quelles conditions le modèle obtenu est stable sans oublier la charge de calcul supplémentaire nécessaire dans l'algorithme d'apprentissage. Pour ces raisons, le modèle série parallèle est couramment utilisé.

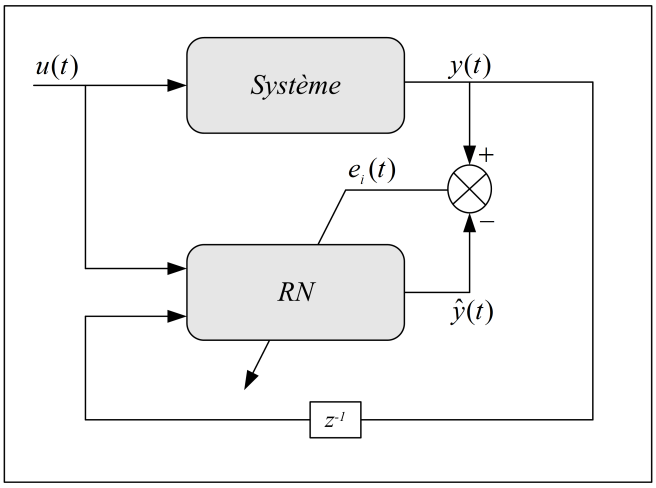

FIG. 2.5 – *Structure d'identification série parallèle*

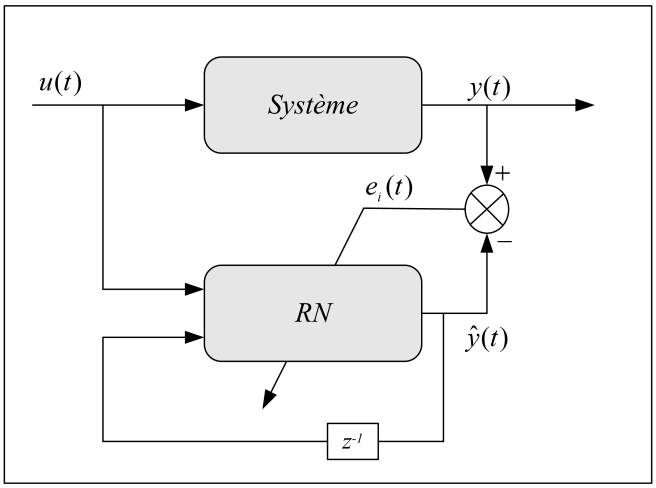

FIG. 2.6 – *Structure d'identification parallèle*

2.3 Réseaux de neurones pour la commande

2.3.1 Imitation d'un contrôleur existant

La première méthode utilisée pour la conception d'un contrôleur neuronal consiste simplement à imiter un système de commande existant. Même si cette approche semble, au premier abord, peut intéressante puisqu'elle nécessite l'existence d'un autre contrôleur. Elle peut s'avérer utile si ce dernier est trop complexe ou présente des difficultés d'implémentation matérielle.

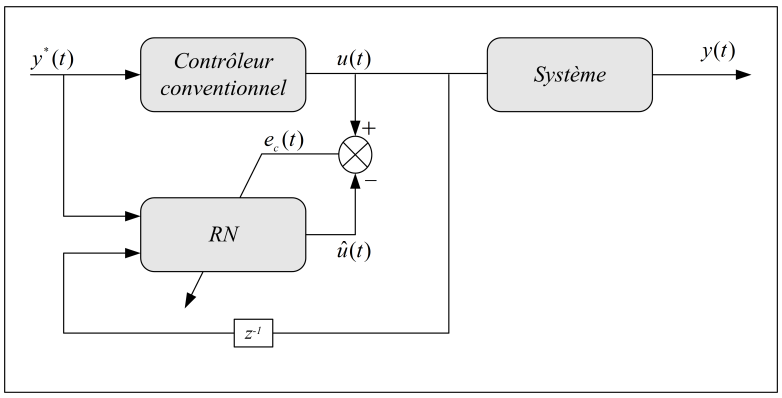

FIG. 2.7 – *Commande neuronale par reproduction d'un contrôleur conventionnel*

La figure 2.7 présente l'architecture générale de ce type de commande. L'erreur e_c entre la sortie du système de commande et celle du réseau est utilisée directement par l'algorithme de rétropropagation pour ajuster les poids de connexion du contrôleur neuronal.
Bien que les premières applications de cette approche remontent déjà à plusieurs années, il existe néanmoins des travaux récents qui exploitent cette approche.

2.3.2 Utilisation directe de l'erreur en sortie du procédé

Dans cette méthode, le réseau neuronal utilise l'erreur e, mesurée en sortie du processus, pour adapter ses paramètres en ligne. Plusieurs stratégies sont possibles. La plus simple consiste à utiliser directement cette erreur comme s'il s'agissait de l'erreur e_c, en sortie du contrôleur. Cette approche ne peut donner de bons résultats que si ces deux erreurs sont fortement corrélées. La méthode la plus utilisée consiste à considérer le processus comme une couche supplémentaire du réseau à travers laquelle on rétropropage l'erreur.

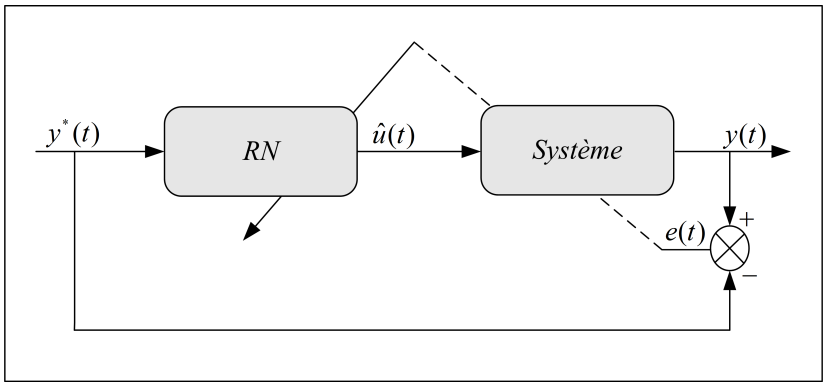

FIG. 2.8 – *Apprentissage spécialisé*

Pour pouvoir utiliser cette stratégie, il est nécessaire de connaitre le *Jacobien* du processus, c'est-à-dire la quantité $\frac{\partial y}{\partial u}$. Il est alors possible d'utiliser l'algorithme de rétropropagation en considérant que le processus est une extension fixe du réseau. Le calcul du gradient du critère d'optimisation J ($J = f(e)$) par rapport aux poids du réseau se fait en appliquant la règle:

$$\frac{\partial J}{\partial p} = \frac{\partial J}{\partial y} \times \frac{\partial y}{\partial u} \times \frac{\partial u}{\partial p} \tag{2.19}$$

Le terme $\frac{\partial J}{\partial y}$ dépend du critère d'optimisation choisi, $\frac{\partial y}{\partial u}$ est le Jacobien du processus et $\frac{\partial u}{\partial p}$ est obtenu par l'algorithme de rétropropagation du gradient. Comme l'obtention du Jacobien $\frac{\partial y}{\partial u}$ est souvent difficile, Psaltis (Psaltis *et al.*, 1988) propose de l'approximer par une méthode basée sur la perturbation des entrées, alors que Saerens (Saerens et Soquet, 1991) le remplace par celui d'un système linéaire approchant le procédé pour lequel on déduit aisément le Jacobien. Il est également possible de n'utiliser que le signe du Jacobien, plus facile à connaître que le Jacobien lui-même.

2.3.3 Identification du modèle inverse

Cette approche s'exécute en deux étapes séparées : étape d'apprentissage et étape d'utilisation. Durant l'apprentissage, le réseau est entraîné par rétropropagation de manière à identifier le modèle inverse du procédé. Comme l'indique la figure 2.9, le réseau se place en parallèle avec le procédé, il reçoit par ses entrées la sortie actuelle $y(t)$ du procédé et fournit en sortie $\hat{u}(t)$ une estimation de la commande $u(t)$. Le but de l'apprentissage est l'ajustement des paramètres du réseau afin de produire en sortie une commande $\hat{u}(t)$ proche de $u(t)$. Après cette phase, le réseau se place devant le procédé afin de fournir à chaque instant la commande $u(t)$ nécessaire pour atteindre la sortie désirée $y^*(t)$ qui lui est donnée en entrée.

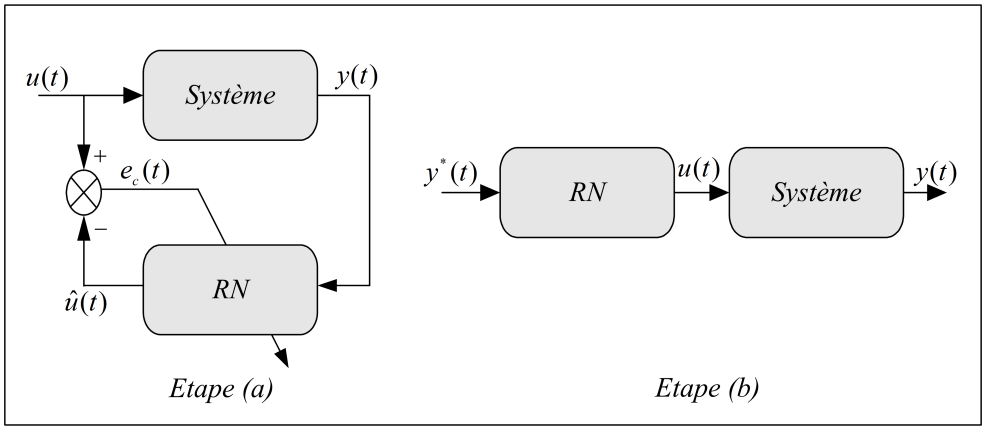

FIG. 2.9 – *Commande neuronale par identification du modèle inverse*

La méthode présente un inconvénient majeur lié à l'étape d'apprentissage: le modèle appris ne garantit pas forcément la réalisation correcte d'un comportement désiré du

moment que tous les états désirés possibles, non pas nécessairement, été rencontrés lors de l'apprentissage. Une autre difficulté apparaît quand le modèle inverse du processus est mal conditionné (i.e. il existe plusieurs commandes correspondant à un comportement désiré). Les algorithmes d'apprentissage supervisés réagissent généralement à ce problème par une commande égale à la moyenne des solutions admissibles rencontrées, ce qui peut être catastrophique dans certains cas. Cette approche a cependant, été utilisée avec succès par de nombreux auteurs.

2.3.4 Architecture indirecte d'apprentissage

Cette méthode, proposée par Psaltis (Psaltis *et al.*, 1987), correspond à une mise en œuvre particulière de la commande par modèle inverse, dans laquelle, le modèle neuronal en cours d'apprentissage, sert aussi à commander le processus. La sortie désirée $y^*(t)$ est propagée à travers le premier réseau neuronal pour produire une commande $u_1(t)$ qui est appliquée au système. La sortie $y(t)$ obtenue est alors utilisée comme entrée du deuxième réseau (identique au premier réseau) qui produit une commande $u_2(t)$. La différence entre $u_1(t)$ et $u_2(t)$ sert de signal d'erreur $e_c(t)$ afin d'assurer l'apprentissage des paramètres des réseaux par rétropropagation. Le point positif de l'approche est que l'entraînement est réalisé dans l'espace objectif, permettant ainsi de faire parcourir au processus l'ensemble de ses états possibles, ou tout au moins l'ensemble des états qui seront utilisés lors du fonctionnement. Néanmoins, l'idée exploitée par cette approche (la minimisation de l'erreur commise sur la commande entraîne la minimisation de l'erreur en sortie du processus) n'est pas souvent juste, ce qui rend l'approche peu utilisable.

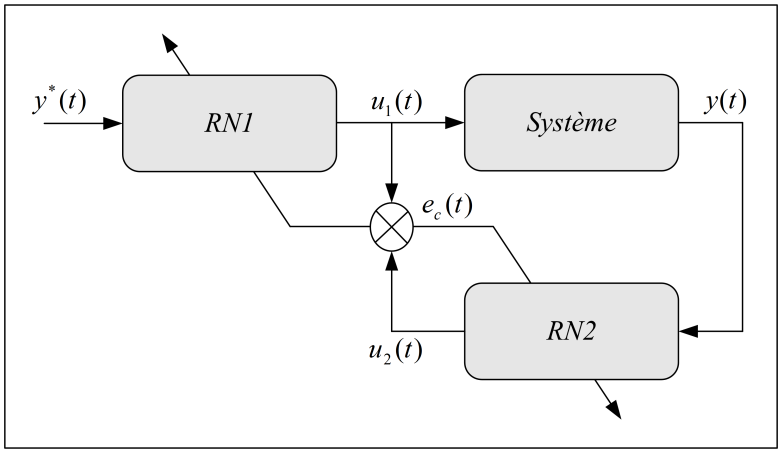

FIG. 2.10 – *Architecture d'apprentissage généralisé*

2.3.5 Apprentissage indirect du système de commande

Cette méthode diffère des approches présentées précédemment par l'usage d'un modèle neuronal du procédé dans la structure de commande présentée par la figure 2.11. La méthode nécessite, donc au préalable, une étape de modélisation du procédé par un réseau de neurones. On cherche ainsi à obtenir, par un réseau neuronal, une bonne approximation de la sortie du procédé $y(t)$ lorsqu'une commande $u(t)$ lui est appliquée.
Le principe de l'approche est alors, puisque l'on dispose d'un modèle différenciable du procédé, d'utiliser l'algorithme de rétropropagation en considérant le contrôleur et le modèle comme un seul réseau. L'erreur est propagée à travers le modèle sans modifier ses poids, et seuls ceux du réseau contrôleur subissent une mise à jour. On peut alors se ramener au cas de l'équation 3.19 en faisant l'approximation

$$\frac{\partial y}{\partial u} = \frac{\partial \hat{y}}{\partial u} \tag{2.20}$$

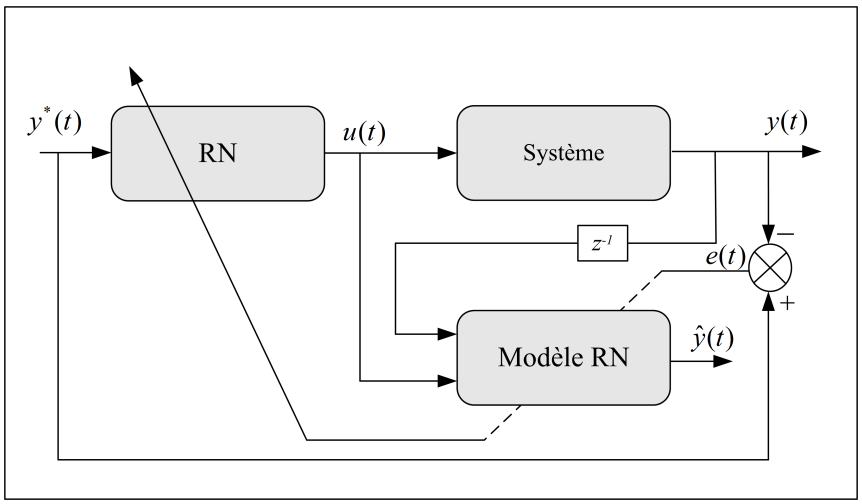

FIG. 2.11 – *Architecture de commande indirecte*

Contrairement à la méthode basée sur l'identification du modèle inverse présentée précédemment, où le système de commande tend, pour amener le procédé dans un état désiré, à moyenner l'ensemble des solutions possibles, cette architecture conduit à sélectionner une solution particulière (Jordan, 1992). On aura remarqué que cette architecture de base n'utilise pas la sortie du modèle neuronal du procédé. Lorsque cela est nécessaire, il est cependant possible d'utiliser cette sortie pour réaliser une adaptation en ligne du modèle du procédé. On se trouve alors avec un diagramme typique de commande adaptative

neuronale. L'article de Narendra (Narendra et Parthasarathy, 1990) constitue une bonne introduction d'utilisation des réseaux de neurones dans le cadre adaptatif. Dans leur architecture, le contrôleur neuronal est entrainé de manière à minimiser l'écart entre la sortie réelle du système et celle générée par un modèle de référence traduisant les performances recherchées.

2.4 Systèmes flous

Les systèmes flous peuvent être considérés comme des systèmes logiques qui utilisent des règles linguistiques pour établir des relations entre leurs variables d'entrée et de sortie. Ils sont apparus pour la première fois dans les années soixante dix avec des applications dans le domaine du contrôle des processus (Mamdani et Assilian, 1975). Aujourd'hui, les applications des systèmes flous sont très nombreuses outre la commande, ils sont largement utilisés pour la modélisation (Kim *et al.*, 1997; Wang et Langari, 1996), le diagnostic et la reconnaissance de formes. Pour une meilleure compréhension de leur fonctionnement, nous présentons brièvement quelques notions de base de ces systèmes, notamment les variables linguistiques.

2.4.1 Ensembles flous

La notion d'ensemble flou a été proposée par Zadeh (Zadeh, 1965) en introduisant un caractère graduel de l'appartenance d'un élément à un ensemble donné. Cela permet une meilleure représentation des termes et des connaissances vagues que nous, les humains, manipulons au quotidien.
Mathématiquement, un ensemble flou A d'un univers de discours U, est caractérisé par une fonction d'appartenance, notée μ_A, à valeur dans l'intervalle $[0,1]$ et qui associe à chaque élément x de U un degré d'appartenance $\mu_A(x)$ indiquant le niveau d'appartenance de x à A. $\mu_A(x) = 1$ et $\mu_A(x) = 0$ correspondent respectivement à l'appartenance et la non-appartenance.
Exemple: Evaluation de la température d'un corps (figure 2.12):

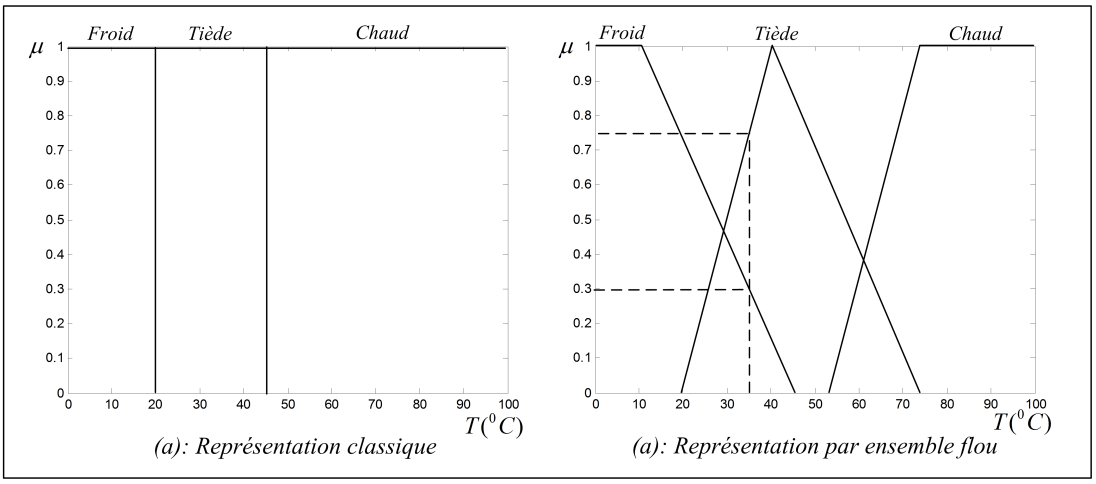

FIG. 2.12 – *Représentation de la température d'un corps par les ensembles classiques et flous*

a) En logique booléenne (Fig 3.12.a), le degré d'appartenance μ ne peut prendre que deux valeurs (0 ou 1). Dans ce cas le corps peut être :

- Froid:$\mu_{Froid} = 1$, $\mu_{Tide} = 0$, $\mu_{Chaud} = 0$
- Tiède:$\mu_{Froid} = 0$, $\mu_{Tide} = 1$, $\mu_{Chaud} = 0$
- Chaud:$\mu_{Froid} = 0$, $\mu_{Tide} = 0$, $\mu_{Chaud} = 1$

La température du corps ne peut pas prendre deux qualificatifs à la fois.
b) En logique floue, le degré d'appartenance devient une fonction qui peut prendre une valeur réelle intermédiaire comprise entre 0 et 1 inclus. Dans ce cas, pour le qualificatif tiède, le corps peut être considéré à la fois, comme froid avec un degré d'appartenance de 0.3 et comme tiède avec un degré d'appartenance de 0.75 (figure 2.12.b).
Pour T=35 ^0C : $\mu_{froid}(T) = 0.3$, $\mu_{Tide}(T) = 0.75$, $\mu_{Chaud}(T) = 0$.

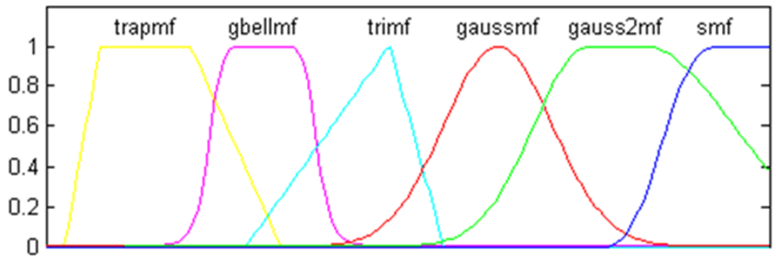

FIG. 2.13 – *Différents types de fonctions d'appartenance utilisées*

L'allure de la fonction d'appartenance est à choisir selon l'application traitée. La figure 2.13, illustre les différentes formes de fonctions d'appartenance les plus utilisées. Cependant, dans certaines applications où l'on doit dériver la fonction d'appartenance, on choisira plutôt des fonctions en S (sigmoïde) ou des fonctions de type gaussienne, continuement dérivables sur leur support.

2.4.2 Variables linguistiques

Une variable linguistique appelée aussi attribut linguistique peut être définie à partir du triplet (x,U,T_x) où x est une variable définie sur l'univers de discours U et $T_x = A1,A2,..$ est un ensemble composé de sous ensembles flous de U qui caractérise x. On associe souvent à chaque sous ensemble flou de T_X une valeur ou un terme linguistique (étiquette). La figure 2.14 illustre un exemple de la variable linguistique *'vitesse'* avec trois termes linguistiques: petite, moyenne et grande.

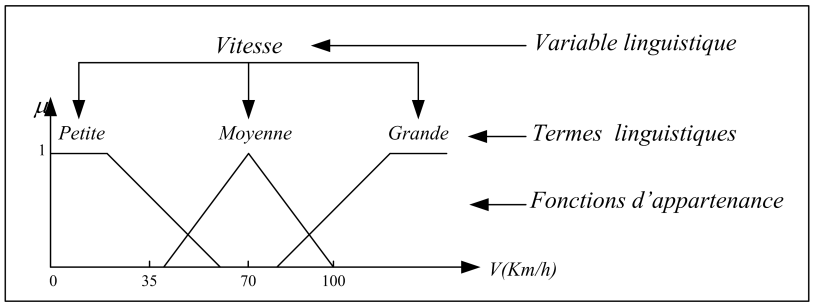

FIG. 2.14 – *Variable linguistique*

Il est généralement imposé que les ensembles flous A_i doivent satisfaire la condition suivante:

$$\forall x, \exists i, \mu_{A_i}(x) \neq 0 \tag{2.21}$$

Cette condition qu'on appelle dans la littérature, propriété d'assurance (coverage property) (Pedrycz et Zadeh, 1995), exige que chaque élément soit affecté à au moins à un ensemble flou avec un degré d'appartenance non nul. A cette condition, on ajoute souvent une propriété supplémentaire qui est le respect de la sémantique : les sous ensembles doivent interpréter réellement les termes linguistiques qui leurs ont associés. Dans la partie inférieure de la figure 2.15, le recouvrement entre les ensembles flous est tel qu'ils

peuvent être ordonnés, et donc interprétés en termes linguistiques, par exemple de la très lente jusqu'à la très rapide. La partie supérieure de la même figure montre un bel exemple d'une partition ininterprétable: il est impossible d'étiqueter les trois sous ensembles flous centraux avec des termes linguistiques.
Le respect de ces deux propriétés confère aux variables linguistiques une meilleure modélisation des connaissances imprécises en réalisant une répartition de l'espace de connaissance. Cette granulation est définie comme une décomposition d'un ensemble de référence, comprenant des informations vagues ou imprécises, en plusieurs sous ensembles flous pour former des répartitions de connaissance.

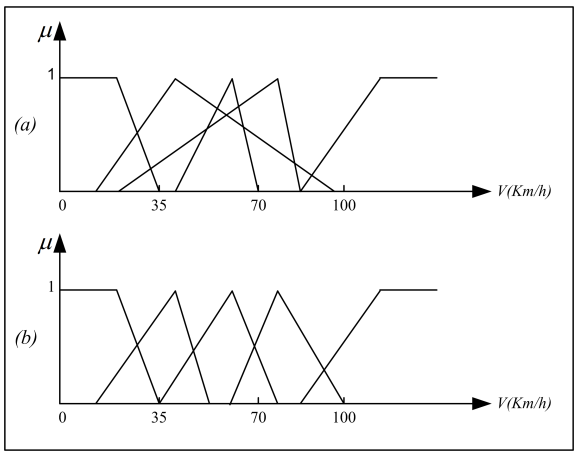

FIG. 2.15 – *La partition supérieure ne peut s'interpréter en termes linguistiques*

2.4.3 Règles et opérateurs flous

On appelle proposition floue élémentaire, une proposition de type $X\ est\ A$ où (X,U,T_x) est une variable linguistique et A un sous ensemble de T_x. Une telle proposition possède un degré de vérité égal à $\mu_A(x)$ où x est une valeur réelle de X. D'une manière générale, on peut combiner ces propositions élémentaires à l'aide des opérateurs logiques de conjonction et de disjonction ('et' et 'ou') mis en œuvre respectivement par des T-normes et T-conormes (Klir et Yuan, 1994). Le degré de vérité des nouvelles propositions obtenues peut être calculé entre autre par les équations suivantes:
Conjonction: (X est A) ET (Y est B)

- *minimum* $(\mu_A(x),\mu_B(y))$
- *produit* $\mu_A(x) \times \mu_B(y)$

Disjonction: (X est A) OU (Y est B)

- *maximum*$(\mu_A(x), \mu_B(y))$
- *somme*$\mu_A(x) + \mu_B(y) - \mu_A(x) \times \mu_B(y)$

L'opérateur d'implication permet d'introduire la notion de règle floue qui caractérise les relations de dépendance entre plusieurs propositions floues:

$$(X_1 \; est \; A_1) \; ET \; (X_2 \; est \; A_2) \Longrightarrow (Y \; est \; B) \tag{2.22}$$

où X_1, X_2 et Y sont des variables linguistiques et A_1 et A_2 et B sont des sous ensembles flous. Une telle règle se trouve habituellement dans les systèmes flous avec une formulation légèrement différente :

$$Si \; (X_1 \; est \; A_1) \; ET \; (X_2 \; est \; A_2) \; Alors \; (Y \; est \; B) \tag{2.23}$$

Dans cette dernière formulation la partie $(X_1 \; est \; A_1) \; ET \; (X_2 \; est \; A_2)$ est appelée prémisse de la règle et la partie $(Y \; est \; B)$ est appelée conclusion (conséquent).

2.4.4 Structure interne d'un système flou

De manière classique, le fonctionnement interne d'un système flou repose sur la structure présentée par la figure 2.16 qui inclut quatre blocs:

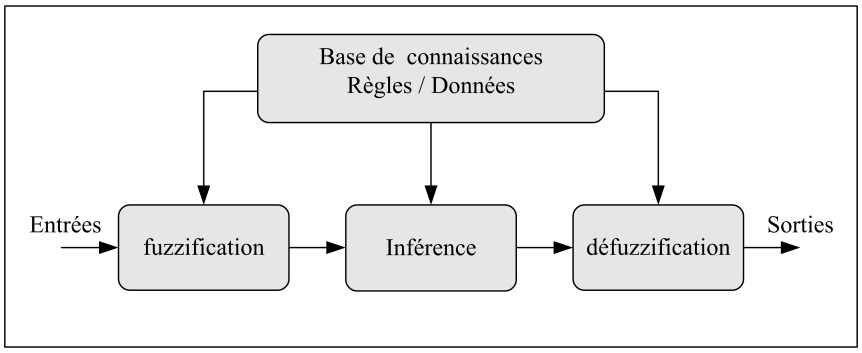

FIG. 2.16 – *Structure interne d'un système flou*

- La base de connaissances
- La fuzzification des variables d'entrée, avec éventuellement un prétraitement de l'information.

– L'inférence à partir d'une base de connaissance
– La défuzzification, avec éventuellement un post-traitement de l'information.

La base de connaissances : elle contient les définitions des fonctions d'appartenance (formes et paramètres) associées aux variables d'entrée/sortie ainsi que l'ensemble des règles floues.
La fuzzification consiste à calculer, pour chaque valeur d'entrée numérique, les degrés d'appartenance aux ensembles flous associés et prédéfinis dans la base de données du système flou. Ce bloc réalise la transformation des entrées numériques en informations symboliques floues utilisables par le mécanisme d'inférence.
Le mécanisme d'inférence consiste d'une part à calculer le degré de vérité des différentes règles du système et d'autre part à associer à chacune de ces règles une valeur de sortie. Cette valeur de sortie dépend de la partie conclusion des règles qui peut prendre plusieurs formes. Il peut s'agir d'une proposition floue, et l'on parlera dans ce cas de règle de type Mamdani:

$$Si\ (.....)\ Alors\ Y\ estB,\ Bensemble\ flou$$

Il peut également s'agir d'une fonction réelle des entrées, et l'on parlera dans ce cas de *règle de type Sugeno*:

$$Si\ (.....)\ Alors\ y = f(x_1,x_1,....,x_n)$$

où x_1,...,x_n sont les valeurs réelles des variables d'entrées.
Dans ce dernier cas, la valeur de sortie de la règle est tout simplement donnée par: $\omega \times f(x_1,x_1,....,x_n)$.
où ω représente le degré de vérité de la règle (i.e de la prémisse) qu'on peut calculer tous simplement par l'une des formules décrites précédemment sur la détermination des valeurs de vérité des propositions floues. Dans le cas d'une règle de type Mamdani, la sortie est un sous ensemble flou obtenu à partir de celui présent dans la conclusion de la règle, soit en lui appliquant un facteur d'échelle égal au degré de vérité de la permisse, on parle alors dans ce cas de la méthode d'inférence PRODUIT, soit en le tronquant à la valeur de ce degré de vérité et on parle dans ce cas de la méthode d'inférence MINIMUM (Lee, 1990).

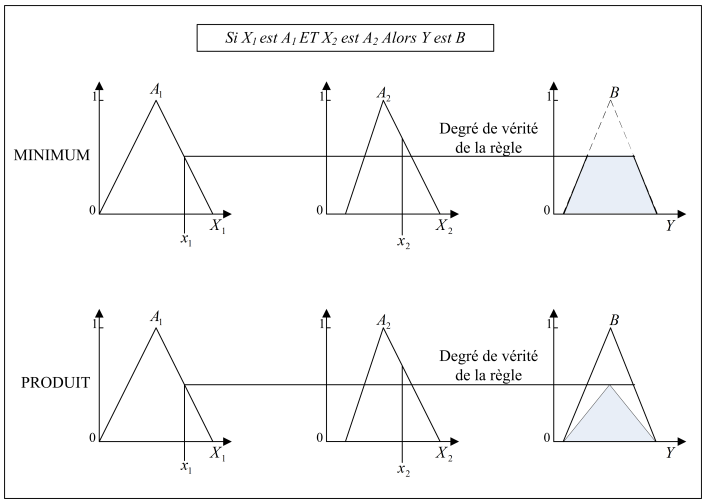

FIG. 2.17 – *Inférence: MINIMUM et PRODUIT*

La défuzzification consiste à remplacer l'ensemble des valeurs de sorties des différentes règles résultant de l'inférence par une valeur numérique unique représentative de cet ensemble. Dans le cas des règles de type Sugeno, le calcul se fait simplement par une somme normalisée des valeurs associées aux règles floues.
Dans le cas de règles de Mamdani, le calcul de la valeur numérique de sortie s'effectue en deux étapes:

1. **Composition des règles**
   Une fois la phase d'inférence terminée, il s'agit de regrouper (par union) les sous ensemble flous issus de l'inférence pour en obtenir un seul ensemble représentatif des différentes conclusions des règles floues. Comme méthode de composition, on peut citer en particulier les compositions MAXIMUM (en général couplée avec l'inférence MINIMUM) et SOMME (en général couplée avec l'inférence PRODUIT)[1]. La première consiste à caractériser l'ensemble de sorties par une fonction d'appartenance égale au maximum des fonctions d'appartenance des sous ensembles flous. La deuxième consiste à faire la somme de fonctions d'appartenance des sous ensembles issus de l'inférence (figure 2.18).

1. Dans le littérature, on regroupe souvent les phases d'inférence et de composition sous le vocable générique d'inférence. Les qualitatifs MIN-MAX et PRODUIT-SOMME, couramment employés, caractérisent les opérations retenues pour ces deux phases.

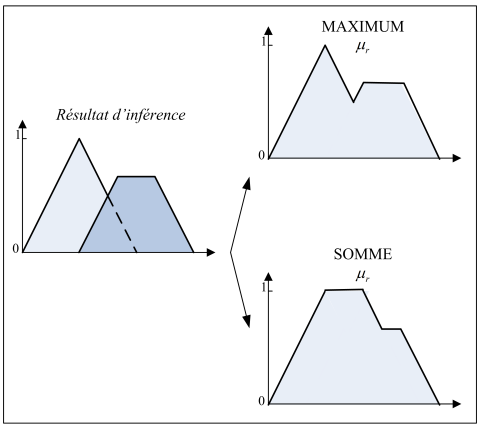

FIG. 2.18 – *Composions des ensembles flous issus de l'inférence*

2. **Passage du symbolique vers le numérique**

C'est la phase de défuzzification proprement dite qui permet de générer une valeur numérique à partir de l'ensemble obtenu par composition des règles. Il existe plusieurs méthode de défuzzification (au moins une dizaine); les plus communément employées sont:

(a) **La méthode de centre de gravité COG**

La défuzzification par centre de gravité consiste à calculer l'abscisse du centre de gravité de la fonction d'appartenance résultante μ_r de la phase de composition selon:

$$y^* = \frac{\int y \cdot \mu_r(y) dy}{\int \mu_r(y) dy}$$

En pratique, on estime le centre de gravité, en calculant la moyenne d'un certain nombre de points échantillonnés sur la fonction:

$$y^* = \frac{\sum y_i \cdot \mu_r(y_i) dy}{\sum \mu_r(y_i)}$$

(b) **La méthode de maximum**

Cette méthode, s'applique uniquement dans le cas où la fonction d'appartenance associée à l'ensemble de sortie n'admet qu'un seul maximum. On choisit comme sortie l'abscisse $y*$ correspondant à ce maximum.

(c) **La méthode de la moyenne des maxima MOM**

Dans cette méthode, la valeur de sortie est estimée par l'abscisse du point cor-

respondant au centre de l'intervalle M pour lequel la fonction d'appartenance est maximale. Cette valeur est fournie par l'expression:

$$y^* = (inf(M) + sup(M))/2$$

où $inf(M)$ et $sup(M)$ sont respectivement les bornes inférieure et supérieure de l'intervalle M.

(Lee, 1990) s'est intéressé aux contrôleurs flous, l'une des applications les plus populaires de systèmes flous, il a donné quelques conclusions relatives à ces différentes méthodes de défuzzification. De même, dans (Braae et Rutherford, 1978), les auteurs présentent une étude comparative des stratégies de défuzzification (COG, MOM) et concluent que la méthode du centre de gravité donne de meilleurs résultats. Néanmoins, on montre dans (Scharf et Mandic, 1985) que la méthode de la moyenne des maxima assure de meilleures performances de transition. Récemment l'étude menée dans (Mogharreban et Dilalla, 2006), montre que ces différentes approches de défuzzification donnent des résultats très semblables pour un problème d'analyse de données. Il en ressort à travers ces études comparatives que le choix d'une meilleure méthode de défuzzification dépend fortement de l'application considérée, ainsi c'est le cas de la méthode du maximum qui s'avère être très efficace pour les problèmes de classification mais pas autant pour les problèmes de commande.

2.5 Système neuro-flou

Les principaux avantages d'un système flou sont l'approche naturelle de la modélisation et la bonne interprétabilité de la description, en employant des règles linguistiques. Cependant, il n'y a aucune méthode formelle pour déterminer ses paramètres (fonctions d'appartenance et règles floues). Dans ce sens, il serait intéressant de disposer d'algorithmes permettant l'apprentissage automatique de ces paramètres. L'une des approches qui permette de répondre à ce besoin est les réseaux de neurones connus pour leur algorithme d'apprentissage et leur précision dans l'ajustement numérique en employant des échantillons entrée/sortie. De nombreux auteurs ont donc tout naturellement cherché à combiner ces deux approches depuis le début des années 90 et ceci de plusieurs manières : coopérative, concurrente et hybride. Nous nous portons ici notre attention sur la dernière approche (hybride) qui permet de représenter sous forme d'un réseau de neurones, les

différentes composantes d'un système flou. La structure du réseau ainsi obtenue dépend du type de règles floues et de méthodes d'inférence et de défuzzification employées par le système flou. Les paramètres du système flou (fonction d'appartenance et règles floues) peuvent ensuite être modifiés par un algorithme d'apprentissage conçu initialement pour l'ajustement des paramètres d'un réseau de neurones.

2.5.1 Architecture neuro flou hybride

Plusieurs architectures, mettant en œuvre cette approche hybride, sont décrites dans la littérature (Farag *et al.*, 1998; Lee, 2005). Ces architectures peuvent être classées en trois groupes (Wang et Lee, 2001) selon le type de règles floues qu'elles intègrent:

$$Si\ (X_1\ est\ A_1)\ ET\ (X_2\ est\ A_2)ET...ET(X_N\ est\ A_N)\ Alors\ (Y\ est\ C)$$

où:

$$C = \begin{cases} B & (typeI) \\ f(x_1,x_2,\cdots,x_n) & (typeII) \\ \theta & (typeIII) \end{cases} \tag{2.24}$$

où X_i $(i = 1,2,\cdots,n)$, Y représentent respectivement les variables d'entrée et de sortie, A_i^j les ensembles flous d'entrée; B, $f(x_1,x_2,\cdots,x_n)$ et θ représentant respectivement, l'ensemble flou de sortie,une fonction linéaire des variables d'entrée et un composant singleton.

Notons que dans le paragraphe précédent, nous avons considéré uniquement deux types de règles (les type I et II de l'équation 2.24) afin d'une part d'alléger l'exposé et d'autre part le type III est souvent considéré dans la littérature comme un cas particulier du type I (Sugeno, 1999).

La figure 2.19 montre les différentes architectures des systèmes neuro-flous hybrides. FALCON (a) et GARIC (b) interprètent la règle floue du type I avec une structure organisée en 5 couches, ANFIS (c) interprète la règle floue du type II avec une structure à 6 couches et NEFCLASS (d) interprète la règle floue de type III avec une structure à 4 couches.

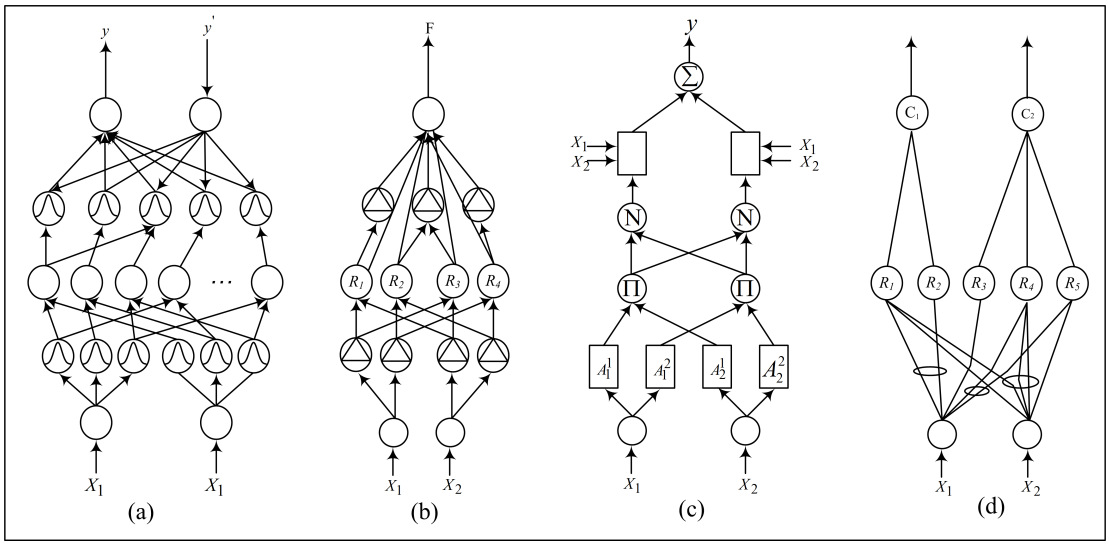

FIG. 2.19 – *Différentes architectures des systèmes neuro-flous*

2.5.1.1 ANFIS

ANFIS (Adaptive-Network-based Fuzzy Inference System) est un système flou mis en application dans le cadre des réseaux adaptatifs. Il a été proposé par Jang (Jang, 1993). Pour les utilisateurs de MATLAB, ANFIS est implanté dans la boite à outils 'Neuro-fuzzy' accompagné de son algorithme d'apprentissage hybride basé sur la rétropropagation et la méthode des moindres carrés. Ce modèle donne de très bons résultats d'approximation de fonctions non linéaires.

2.5.1.2 FALCON ET GARIC

IL s'agit de modèle à 5 couches (Lin et Lee, 1991; Berenji et Khedkar, 1992) utilisant la fuzzufication en entrée et la défuzzification en sortie. Ceci correspond à une interprétation juste de la méthode de Mamdani. La précision accrue des résultats provoque une lenteur dans l'exécution du système. Ce modèle est rarement utilisé en pratique mais il donne de meilleurs résultats en commande.

2.5.1.3 NEFCLASS

Modèle utilisé généralement pour la classification, il est constitué de 3 couches: une couche d'entrée avec les fonctions d'appartenance, une couche cachée représentée par l'ensemble des règles et une couche de sortie définissant l'ensemble des classes. Ce modèle est facile à mettre en application, il évite l'étape de défuzzification, tout en étant précis dans le résultat final, avec une rapidité supérieure aux autres modèles.

2.6 Conclusion

Les réseaux de neurones, de part la quantité des travaux de recherche et des réalisations existantes, présentent un certain nombre de points forts. En premier lieu, ils constituent des approximateurs universels capables de modéliser des systèmes complexes avec la précision voulue à partir d'un jeu de données entrées /sorties et un processus d'optimisation. Leur capacité d'apprentissage permet de simplifier la synthèse d'un contrôleur dans la mesure où l'on obtient celui-ci par un simple réglage de paramètres sans qu'il soit nécessaire de faire appel au modèle mathématique du système à commander. Dans le cas de systèmes complexes, cela représente un avantage indéniable par rapport à la plupart des autres méthodes qui s'attachent à utiliser un modèle mathématique souvent imprécis ou difficile à obtenir. Cependant, l'inconvénient majeur réside dans le fait que l'on ne peut pas incorporer les connaissances des experts, qualifiées d'une certaine intelligence, et qui peuvent être utiles soit pour accélérer le processus d'apprentissage, soit pour obtenir des structures interprétables. Contrairement aux réseaux de neurones, les systèmes flous fonctionnent à base de ce type de connaissances et permettent désormais une meilleure exploitation de l'intelligence humaine pour accomplir des tâches complexes. Dans ce mémoire, les contrôleurs que nous développons sont à base de ces systèmes flous. Le problème d'optimisation de la structure et des paramètres de ces contrôleurs, qui reste néanmoins plus difficile dans de nombreux cas, sera étudié dans le prochain chapitre en utilisant les algorithmes génétiques. Les réseaux de neurones vont intervenir, comme nous allons le voir, uniquement dans le cas des systèmes flous avec des règles de type Sugeno.

Chapitre 3

Conception de contrôleurs flous par algorithmes génétiques

3.1 Introduction

Les difficultés rencontrées dans la conception des contrôleurs flous(CF), ont guidé les chercheurs à s'orienter vers l'utilisation des algorithmes génétiques à cause de leur caractéristique d'exploration globale dans un environnement complexe. On rencontre souvent, dans la littérature, trois stratégies d'application des AG pour la conception des contrôleurs flous.

1. La base de règles floues est bien définie et leurs fonctions de stratégies sont optimisées par l'AG

2. Les fonctions d'appartenance associées aux variables d'entrées et de sorties sont fixées et l'AG est utilisé pour l'optimisation des conclusions des règles floues.

3. Les fonctions d'appartenance et les règles floues associées sont optimisées simultanément

Une représentation graphique de ces types de stratégies est indiquée dans la figure 3.1.

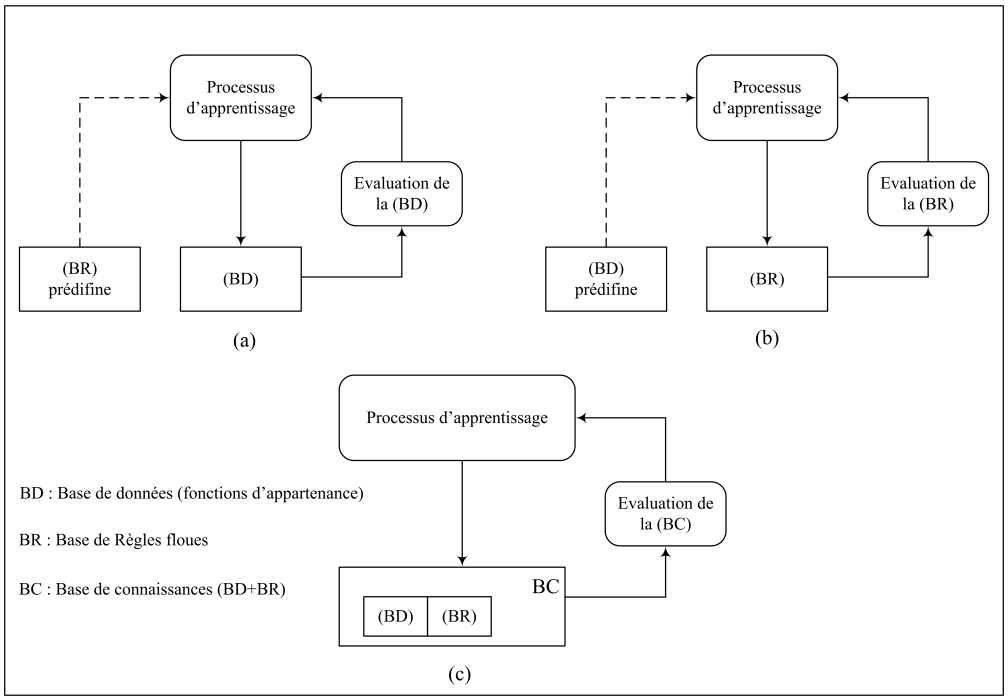

FIG. 3.1 – *Représentation graphique des différentes approches d'optimisation d'un contrôleur flou par algorithme génétique*

Dans notre travail, on s'intéresse à la troisième stratégie, qui s'avère être la plus efficace car, en toute logique, une méthode traitant globalement les différents paramètres d'un contrôleur flou devrait donner de meilleures solutions.

Le chapitre est organisé en trois parties. Dans la première partie nous étudions l'optimisation des paramètres d'un CF de type Mamdani par algorithme génétique simple. Nous examinons d'abord les différents codages qui permettent de représenter les fonctions d'appartenance et les règles floues. Ensuite, nous présentons la structure du chromosome et les opérateurs génétiques adoptés pour l'optimisation simultanée de l'ensemble des paramètres d'un CF. Enfin pour mieux illustrer la méthode, nous traitons un exemple de commande d'un système linéaire de deuxième ordre.

Dans la deuxième partie, l'optimisation des CF de Mamdani est effectuée par les AGH. Dans un premier temps, nous présentons nos motivations pour l'emploi de ce type d'algorithme. Ensuite, après la formulation de la structure du chromosome hiérarchisé, nous décrivons les opérateurs de mutation et de croisement qui s'appliquent sur les gènes de contrôle du chromosome de l'AGH. A la fin de cette deuxième partie, le même problème de commande est considéré.

La troisième partie concerne l'optimisation de contrôleurs flous de type Sugeno par un

algorithme hybride utilisant la rétropropagation et NSGA-II avec un codage hiérarchisé.

3.2 Optimisation des contrôleurs flous de type Mamdani par algorithmes génétiques simples

On considère un contrôleur à deux entrées, l'erreur $e(t)$ et sa variation $\Delta e(t)$ et une sortie $\Delta u(t)$, la variation de la commande, qui permet d'ajuster à chaque instant la commande $u(t)$ appliquée au système (figure 3.2):

$$u(t) = G_s \times \Delta u(t) + u(t-1) \tag{3.1}$$

avec G_s un facteur d'échelle (scaling factor) en sortie.
Les règles floues constituant la base du contrôleur, dans ce cas, possèdent deux prémisses

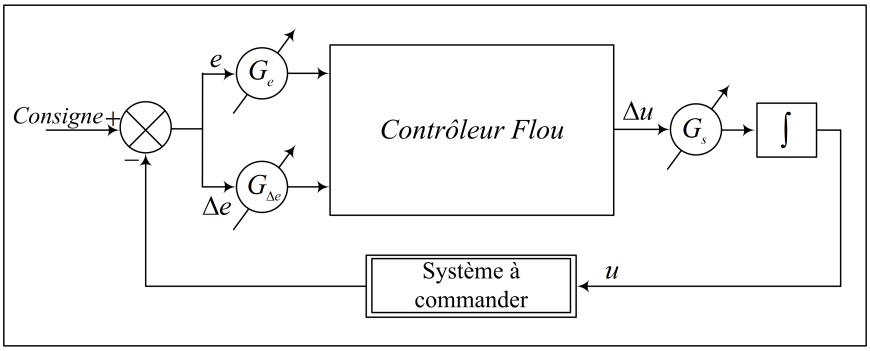

FIG. 3.2 – *Contrôleur flou à deux entrées*

et une seule conclusion:

Règle i_1,i_2: Si (e est A_{1i_1}) ET (Δe est A_{2i_2}) Alors Δu est B_j

où (A_{1i_1}| $i_1 = 1, 2,..., N_e$), (A_{2i_2}| $i_2 = 1, 2,..., N_{\Delta e}$) et B_j| $j = 1, 2,..., N_s$) sont les termes linguistiques qualifiant respectivement l'erreur e , la variation de l'erreur Δe et la variation de la commande Δu. N_e, $N_{\Delta e}$ et N_s représentent respectivement le nombre de termes linguistiques associés à e, Δe et Δu.
On définit alors $N_e \times N_{\Delta e}$ règles floues pour ce contrôleur, organisées couramment sous forme d'un tableau et dont le contenu correspond aux conclusions de toutes les configurations possibles de prémisses.
Un exemple de ce type de tableau est présenté ci-dessus: La conclusion d'une règle repé-

TAB. 3.1 – *Base de règles floues*

Δu \ Δe	A_{21}	A_{22}	$\cdots$	A_{2i_2}	$\cdots$	$A_{1N_{\Delta e}}$
A_{11}	R_{11}	R_{12}		$\cdots$		$R_{1N_{\Delta e}}$
A_{12}	R_{21}	R_{22}		$\cdots$		$R_{2N_{\Delta e}}$
e $\vdots$				$\vdots$		
A_{1i_1}		$\cdots$		$R_{i_1i_2}$		$\cdots$
$\vdots$				$\vdots$		
A_{1N_e}	R_{N_e1}	R_{N_e2}		$\cdots$		$R_{N_eN_{\Delta e}}$

rée par les indices i_1 et i_2 est alors représentée par un nombre $R_{i_1i_2}$ défini comme suit:

$$R_{i_1i_2} = j,\ j \in \{1,2,\cdots,N_s\} \text{ si la conclusion de la règle } i_1i_2 \text{ est } B_j$$

Les paramètres de réglage d'un tel contrôleur (deux entrées et une sortie) sont alors:

- Deux facteurs d'échelle en entrée (G_e et $G_{\Delta e}$) et un autre en sortie G_s.
- Les fonctions d'appartenance associées aux variables d'entrée et de sortie.
- Les $N_e \times N_{\Delta e}$ conclusions de la table de règles.

Dans cette section, le réglage de ces paramètres sera étudié par les algorithmes génétiques simples. Comme ces derniers travaillent sur une population de chromosomes sur lesquels sont codés les paramètres à optimiser, il est nécessaire de définir la structure du chromosome qui prend en compte l'ensemble des paramètres cités ci-dessus.

3.2.1 Représentation des fonctions d'appartenance

Dans un contrôleur flou, les règles floues sont appliquées sur des termes linguistiques. Ces termes, qui permettent de qualifier une variable linguistique, sont définis par l'intermédiaire de fonctions d'appartenance. Les paramètres à optimiser sont donc définis en deux étapes : on doit d'abord définir les paramètres de chaque fonction d'appartenance, puis l'ensemble des paramètres des fonctions d'appartenance qui constituent la partition floue de la variable linguistique.

3.2.1.1 Partitionnement avec des fonctions triangulaires

Une fonction d'appartenance triangulaire dans un univers du discours $[a,b]$peut être définie par:

$$\mu_A(x) = \begin{cases} \dfrac{x - x_1}{x_2 - x_1} & x_1 \leq x < x_2 \\ \dfrac{x_3 - x}{x_3 - x_2} & x_2 \leq x < x_3 \\ 0 & \text{ailleurs} \end{cases} \tag{3.2}$$

On constate que cette fonction dépend des trois paramètres x_1, x_2 et x_3 qui prennent leurs valeurs dans l'intervalle $[a,b]$, comme indiqué par la figure ci-dessous.

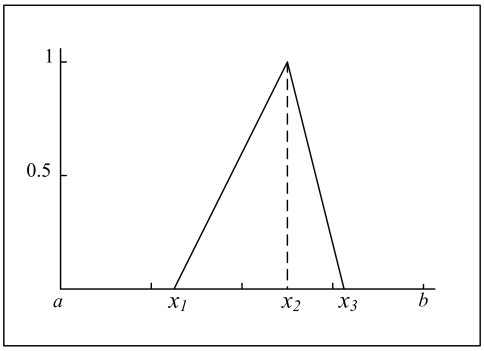

FIG. 3.3 – *Fonction d'appartenance triangulaire définie par trois paramètres*

Cette fonction étant parfaitement définie par ces trois paramètres, on peut donc utiliser la représentation suivante: où $C_{[a,b]}(x_k)$ est un code (binaire, réel, ..., etc) de la variable

$C_{[a,b]}(x_1)$	$C_{[a,b]}(x_2)$	$C_{[a,b]}(x_3)$

x_k qui prend ses valeurs dans l'intervalle $[a,b]$. Les valeurs codées doivent satisfaire la condition suivante:

$$x_1 < x_2 < x_3 \tag{3.3}$$

Dans un contrôleur fou, chaque variable linguistique est définie par un ensemble de fonctions d'appartenance des termes linguistiques, comme le montre la figure 3.4.

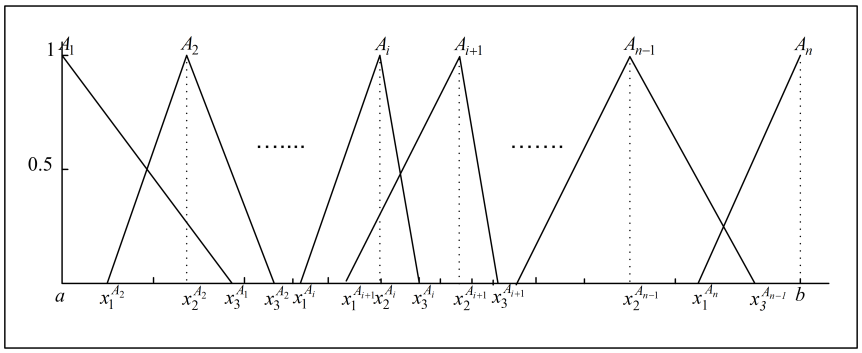

FIG. 3.4 – *Partition floue avec des fonctions d'appartenance triangulaires*

La partition floue peut être donc représentée par le codage suivant:

$c(x_3^{A_1})$	$\cdots$	$c(x_1^{A_i})$	$c(x_2^{A_i})$	$c(x_1^{A_i})$	$\cdots$	$c(x_2^{A_n})$

A la condition (3.3), on ajoute d'autres contraintes sur la plage de variation des paramètres de chaque fonction d'appartenance afin d'obtenir un partitionnement à la fois continu et interprétable. Ces contraintes sont traduites par la double condition ci-dessous:

$$\begin{cases} x_2^{A_{i-1}} < x_2^{A_i} \\ x_1^{A_i} < x_3^{A_{i-1}} < x_3^{A_i} \end{cases} \tag{3.4}$$

Dans (Guenounou et Belmehdi, 2006), en adoptant ce type de représentation pour la commande de la vitesse d'une machine asynchrone, nous avons pu obtenir des résultats satisfaisants et un contrôleur flou comprenant une base de règles lisible et sémantiquement interprétable. L'inconvénient majeur de cette représentation est qu'elle introduit un nombre important de contraintes et de paramètres qui peuvent limiter l'efficacité de la procédure de recherche de paramètres optimaux. Lee (Lee et Takagi, 1993) et Yubaziki (Yubazaki *et al.*, 1995) ont proposé une autre méthode de partitionnement pour laquelle le nombre de paramètres requis est réduit de manière significative (un rapport de 3 pour les fonctions d'appartenance de type triangulaire). Leur méthode repose sur l'idée de faire partager le même paramètre par plusieurs fonctions d'appartenance.
La figure 3.5 illustre un exemple de partitionnement avec 5 fonctions d'appartenance selon cette méthode.

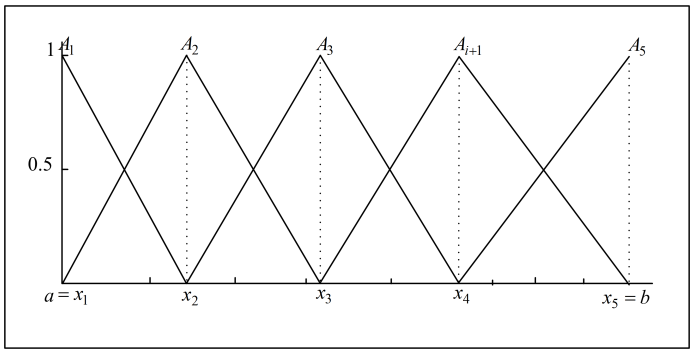

FIG. 3.5 – *Partition avec 5 fonctions d'appartenance d'après Lee et Yubaziki*

Une partition floue comportant n termes linguistiques est complètement définie par l'intermédiaire de n points ($x_1 = a$, x_2, $\cdots$, $x_n = b$) en utilisant les fonctions d'appartenance suivantes:

$$\mu_{A_1}(x) = \begin{cases} 1 & x < x_1 \\ \dfrac{x_2 - x}{x_2 - x_1} & x_1 \leq x < x_2 \\ 0 & x \geq x_2 \end{cases}$$

$$\mu_{A_i}(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & x_{i-1} \leq x < x_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & x_i \leq x < x_{i+1} \quad i = 2, \cdots, n-1 \\ 0 & ailleurs \end{cases} \tag{3.5}$$

$$\mu_{A_n}(x) = \begin{cases} 0 & x < x_{n-1} \\ \dfrac{x - x_{n-1}}{x_n - x_{n-1}} & x_{n-1} \leq x < x_n \\ 1 & x \geq x_n \end{cases}$$

La partition floue peut être donc représentée par le codage de $(n-2)$ points:

$c_{[a,b]}(x_2)$	$\cdots$	$c_{[a,b]}(x_i)$	$\cdots$	$c_{[a,b]}(x_{n-1})$

et par les $(n-1)$ inégalités:

$$a = x_1 < x_2 < \cdots < x_{n-1} < x_n = b$$

3.2.1.2 Partitionnement avec des fonctions trapézoïdales

Une fonction d'appartenance trapézoïdale dans un univers du discours fini $[a,b]$ peut être définie par:

$$\mu_A(x) = \begin{cases} \dfrac{x - x_1}{x_2 - x_1} & x_1 \leq x < x_2 \\ 1 & x_2 \leq x < x_3 \\ \dfrac{x_4 - x}{x_4 - x_3} & x_3 \leq x < x_4 \\ 0 & ailleurs \end{cases} \tag{3.6}$$

Cette fonction est définie par quatre paramètres x_1, x_2, x_3 et x_4 qui prennent leurs valeurs dans l'intervalle [a, b].
La figure 3.6 illustre cette fonction, elle nécessite un paramètre de plus par rapport à la fonction triangulaire. En effet cette dernière est un cas particulier de la fonction trapézoïdale (lorsque $x_2 = x_3$).

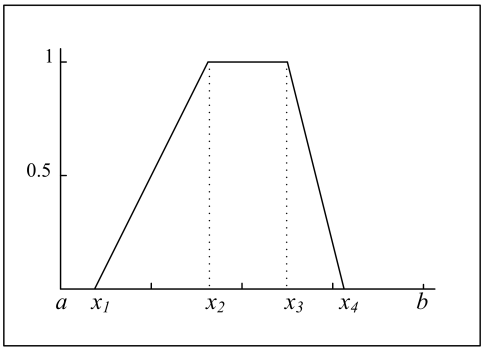

FIG. 3.6 – *Fonction d'appartenance trapézoîdale*

La représentation de la fonction trapézoïdale fait intervenir les quatre paramètres suivants:

$c_{[a,b]}(x_1)$	$c_{[a,b]}(x_2)$	$c_{[a,b]}(x_3)$	$c_{[a,b]}(x_4)$

et les conditions aux limites associées sont:

$$x_1 < x_2 \leq x_3 < x_3 \tag{3.7}$$

La figure ci-dessous illustre un exemple de partitionnement flou avec 5 fonctions d'appartenance trapézoïdales selon l'idée de Lee et Yubaziki.

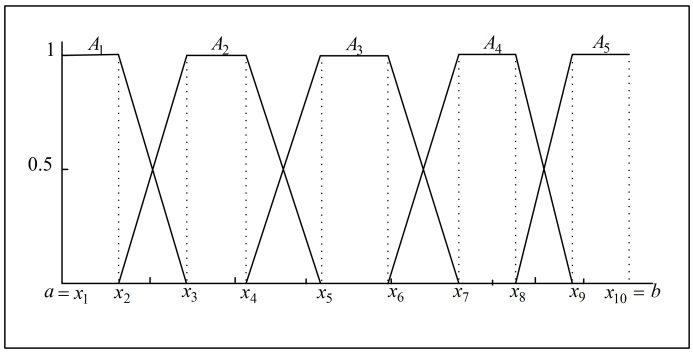

FIG. 3.7 – *Partition avec 5 fonctions d'appartenance trapézoidales d'après Lee et Yubaziki*

Plus généralement, une partition floue avec n fonctions trapézoïdales est définie par $2n$ paramètres ($a = x_1, x_2, \cdots, x_n = b$) et par les équations suivantes:

$$\mu_{A_1}(x) = \begin{cases} 1 & x_1 \leq x < x_2 \\ \dfrac{x_3 - x}{x_3 - x_2} & x_2 \leq x < x_3 \\ 0 & x \geq x_2 \end{cases}$$

$$\mu_{A_i}(x) = \begin{cases} 0 & x < x_{2i-2} \\ \dfrac{x - x_{2i-2}}{x_{2i-1} - x_{2i-2}} & x_{2i-2} \leq x < x_{2i-1} \\ 1 & x_{2i-1} \leq x < x_{2i} \\ \dfrac{x_{2i+1} - x}{x_{2i+1} - x_{2i}} & x_{2i} \leq x < x_{2i+1} \\ 0 & x \geq x_{2i+1} \end{cases} \qquad i = 2, \cdots, n-1 \tag{3.8}$$

$$\mu_{A_n}(x) = \begin{cases} 0 & x < x_{2n-2} \\ \dfrac{x - x_{2n-2}}{x_{2n-1} - x_{2n-2}} & x_{2n-2} \leq x < x_{2n-1} \\ 1 & x \geq x_{2n-1} \end{cases}$$

Dans ce cas,la représentation est:

$c_{[a,b]}(x_1)$	$\cdots$	$c_{[a,b]}(x_i)$	$\cdots$	$c_{[a,b]}(x_{2n-1})$

et les conditions aux limites sont de la forme:

$$a = x_1 < x_2 < \cdots < x_{2n-1} < x_{2n} = b$$

3.2.1.3 Fonction d'appartenance Gaussienne

Une fonction d'appartenance Gaussienne asymétrique dans l'univers de discours $[a,b]$ peut être exprimée sous la forme:

$$\mu_A(x) = \begin{cases} exp(-\dfrac{(x-c)^2}{2\sigma_g^2}) & x < c \\ exp(-\dfrac{(x-c)^2}{2\sigma_d^2}) & x \geq c \end{cases} \tag{3.9}$$

Pour calculer cette fonction d'appartenance, nous utilisons deux fonctions gaussiennes possédant le même centre c mais deux variances différentes (σ_g^2 et σ_d^2). La première gaussienne est utilisée lorsque la valeur d'entrée est inférieure à la moyenne, et la deuxième dans le cas opposé. Ceci permet de définir une fonction d'appartenance asymétrique (voir la figure 3.8) et d'être moins restrictif sur la classe des systèmes flous que nous pouvons représenter.

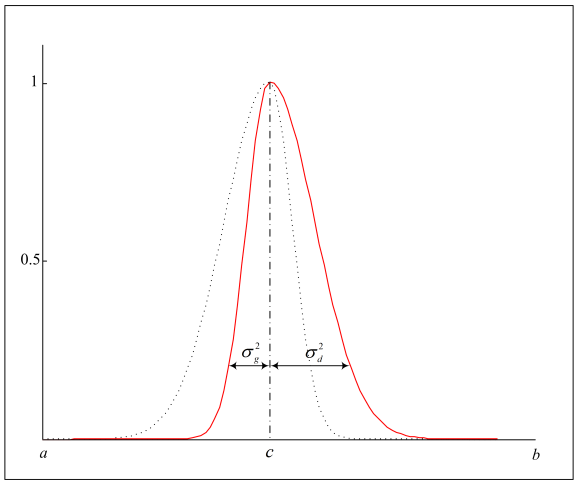

FIG. 3.8 – *Fonction d'appartenance gaussienne asymétrique(ligne continue): La moyenne est notée c et les variances σ_g^2 et σ_d^2*

La représentation d'une fonction gaussienne asymétrique est alors donnée par le code suivant:

$C(\sigma_g)$	$C_{[a,b]}(c)$	$C(\sigma_d)$

Lorsqu'on utilise des termes linguistiques caractérisés par des fonctions gaussiennes asymétriques pour représenter une variable linguistique sur un univers de discours $[a,b]$, il est possible, comme dans le cas des fonctions triangulaire et trapézoïdale, de simplifier la représentation en imposant des conditions sur les variances de chaque gaussienne:

$$\begin{cases} \sigma_{g,i} = \sigma_{d,i-1} \\ \sigma_{d,i} = \sigma_{g,i+1} \end{cases} \tag{3.10}$$

où $\sigma_{g,i}$ et $\sigma_{d,i}$ désignent respectivement les variances gauche et droite de la ième gaussienne.
La figure ci-dessous montre un exemple de partitionnement avec 5 gaussiennes asymétriques qu'on peut obtenir en tenant compte de la condition 3.10.

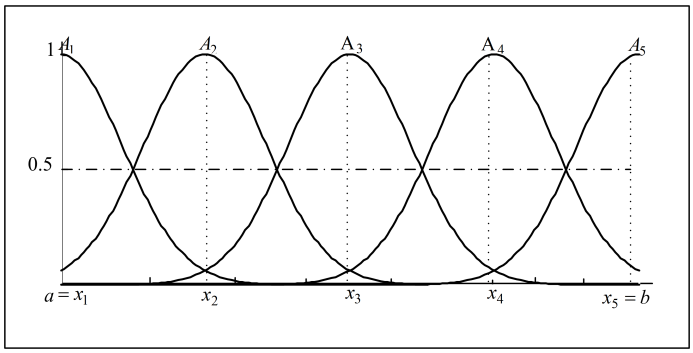

FIG. 3.9 – *Partition avec 5 gaussiennes asymétriques*

Le taux de chevauchement α entre deux fonctions d'appartenance adjacentes a été fixé à 0,5. On peut vérifier à partir des équations 3.9 et 3.10 que le taux de chevauchement relie la variance commune entre μ_{A_i} et $\mu_{A_{i+1}}$ aux centres c_i et c_{i+1} par la relation suivante:

$$\sigma_{d,i} = \sigma_{g,i+1} = \sqrt{-\frac{(c_{i+1} - c_i)^2}{8\ln(\alpha_i)}} \tag{3.11}$$

Une partition comportant n termes linguistiques peut être alors représentée par:

$C(\alpha_1)$	$C_{[a,b]}(c_2)$	$C(\alpha_2)$	$\cdots$	$C_{[a,b]}(c_i)$	$C(\alpha_i)$	$\cdots$	$C_{[a,b]}(c_{n-1})$	$C(\alpha_{n-1})$

et les conditions aux limites sont:

$$a = c_1 < c_2 < \cdots < c_{n-1} < c_n = b \quad \wedge \alpha_i \in]0,1[$$

3.2.2 Codage des paramètres des fonctions d'appartenance

Comme nous venons de le voir, une variable linguistique est représentée par un certain nombre de paramètres qui dépendent à la fois du nombre et du type de fonctions d'appartenance utilisées. Pour coder ces différents paramètres, on utilise souvent les chaînes binaires et les chaînes réelles.
Lorsque le nombre de paramètres est réduit et leurs plages de variations bien définies, un algorithme génétique avec un codage binaire est largement suffisant pour retrouver les paramètres optimaux. Par contre si le nombre de paramètres devient important, et que l'intervalle de variation de chaque paramètre n'est pas bien connu, le codage réel est le plus approprié. En ce qui concerne l'utilisation du codage en base n, il y a eu très peu de travaux sur le sujet (Homaifar et MCCormick, 1993; Ng et Li, 1994). Le seul intérêt est de disposer d'un type de codage homogène pour tous les paramètres du contrôleur que l'on souhaite optimiser. Homaifar (Homaifar et MCCormick, 1993) a opté pour ce codage afin de représenter dans un même chromosome les conclusions des règles floues et les paramètres de la fuzzification (paramètres des fonctions d'appartenance associées aux variables d'entrées). Nous verrons dans la suite de ce mémoire, qu'il est possible de regrouper les différents paramètres (paramètres des fonctions d'appartenance, règles floues et facteurs d'échelle) dans un même chromosome, sans qu'il soit nécessaire d'utiliser le même codage.

3.2.3 Représentation des règles floues

Pour représenter l'ensemble des règles du contrôleur flou de la figure 3.2, on réorganise le tableau 3.1 en une liste de valeurs en le balayant ligne par ligne. Cela fournit comme représentation:

R_{11}	R_{12}	$\cdots$	$R_{1N_{\Delta e}}$	R_{21}	R_{22}	$\cdots$	$R_{N_e N_{\Delta e}}$

Afin de permettre un traitement par les algorithmes génétiques, chaque élément R_{ij} de

la représentation ci-dessus doit être codé pour être inséré dans un chromosome. Plusieurs types de codage peuvent être adoptés.

3.2.4 Codage des règles floues

3.2.4.1 Le codage binaire

Avec ce type de codage, la base de règles est représentée par une chaine binaire, chaque élément R_{ij} de la représentation étant codé sur m bits. Nous avons vu qu'un élément R_{ij} prend ses valeurs dans l'ensemble $\{1,2,\cdots,Ns\}$, où N_s désigne le nombre de termes linguistiques associés à la variable de sortie Δu. Une sous-chaine décrivant une de ses valeurs doit donc comporter au minimum $m = E(log_2(Ns+1))$ bits, E(x) désignant la partie entière de x. Quand (N_s+1) n'est pas une puissance de deux, certaines configurations binaires ne constituent pas un codage valide. On doit alors ajouter une condition sur la valeur représentée par la sous-chaine binaire, qui doit être inférieure où égale à N_s. Cette condition, qu'il faut tester à chaque fois que l'on réalise une opération sur la chaîne binaire (croisement ou mutation), constitue un des principaux inconvénients de l'emploi du codage binaire.

3.2.4.2 Le codage réel

Ce type de codage est rarement utilisé pour l'optimisation des règles floues de Mamdani à cause de la nature des conclusions qu'elles présentent. Par contre, comme nous le verrons plus loin, ce codage s'adapte facilement aux règles de type Sugeno.

3.2.4.3 Codage en base *n*

C'est le codage le plus utilisé pour la représentation des règles floues de Mamdani (Zhou et Lai, 2000; Cheong et Lai, 2000). Dans l'article (Homaifar et MCCormick, 1993), Homaifar décrit une procédure d'optimisation des règles floues d'un contrôleur dont les sorties sont partitionnées en 5 valeurs linguistiques. Le nombre 6 (5+1) n'étant pas une puissance de deux, il propose d'utiliser un codage en base 6, ce qui permet de coder efficacement les 5 valeurs possibles de la conclusion d'une règle floue.

3.2.5 Structure du chromosome

Dans le paragraphe précédent, nous avons présenté les méthodes de codage des paramètres permettant l'optimisation des fonctions d'appartenance et la base de règles d'un contrôleur flou de Mamdani. Prenant appui sur ces différentes méthodes, nous proposons un codage mixte, qui permet de représenter simultanément les règles floues, les paramètres de fonctions d'appartenance et les facteurs d'échelle. Le chromosome de l'AG ainsi obtenu est divisé en trois parties: une première partie, codée en base n, représente les conclusions des règles floues, une deuxième partie, codée en nombres réels, représente les paramètres des fonctions d'appartenance et une troisièmes partie, codée en binaire, représente les facteurs d'échelle en entrée et en sortie. Cette structure est représentée par la figure 3.10.

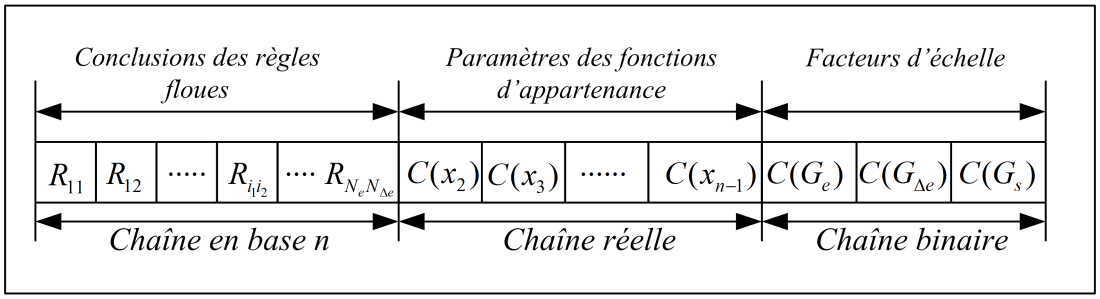

FIG. 3.10 – *Structure du chromosome de l'AGS*

3.2.6 Opérateurs génétiques

Puisque sur le même chromosome, il existe trois types de chaînes: *chaîne en base n, chaîne réelle* et *chaîne binaire*, il convient alors de spécifier pour chacune d'elle les opérateurs génétiques permettant une meilleure exploration de l'espace de recherche.

3.2.6.1 Opérateur de Croisement

Croisement en base n

Nous utilisons un croisement en deux points standard comme s'il s'agissait de chaines binaires.

Croisement des chaînes réelles

Dans ce cas, on a le choix entre un croisement standard et un croisement arithmétique.

Nous utilisons un croisement arithmétique en deux points.

Croisement des chaînes binaires

Nous utilisons aussi, ici un choisement standard en deux points. Les lieux de croisement pour toutes les chaînes sont choisis aléatoirement et changent à chaque génération.

3.2.6.2 Opérateur de Mutation

Mutation des chaines en base n

La mutation des chaînes codées en base n, consiste à remplacer l'élément de la chaîne à muter par un chiffre en base n.

L'application directe d'une telle mutation pour l'optimisation des règles floues s'avère peu efficace et peut conduire à une exploration chaotique à cause du caractère graduel présent dans les ensembles flous utilisés pour le partitionnement des variables du contrôleur. Considérant le contrôleur de la figure 3.1 dont la sortie est partitionnée en 7 valeurs linguistiques Négative Grande (NG), Négative Moyenne (NM), Négative Petite (NP), Zéro (Z), Positive Petite (PP) et Positive Moyenne (PM) et Positive grande (PG) représentées respectivement par les nombres 1, 2, 3, 4, 5, 6 et 7 de la base 8. Si maintenant le nombre 4, représentant une variation de commande égale à zéro, est ramené à 1, après mutation, la variation de commande codée devient alors négative grande, ce qui conduira probablement à un mauvais résultat. Par contre si ce même nombre (4) est ramené à 3, représentant une variation de Négative Petite, le résultat peut être amélioré, ou au pire détérioré, avec un degré moindre que dans le premier cas.

Il convient alors après la mutation d'un élément de produire des éléments voisins constituant le partitionnement de la variable de sortie. Ceci peut être obtenu en adoptant la mutation définie ci-dessous:

$$R'_{i_1 i_2} = \begin{cases} 2 & \text{si} \quad R_{i_1 i_2} = 1 \\ R_{i_1 i_2} \pm 1 & \text{si} \quad 1 < R_{i_1 i_2} < N_s \\ N_s - 1 & \text{si} \quad R_{i_1 i_2} = N_s \end{cases} \tag{3.12}$$

où $R'_{i_1 i_2}$ représente dans ce cas, le résultat de la mutation de $R_{i_1 i_2}$.

Mutation des chaînes réelles

La mutation non uniforme décrite dans le chapitre 2 est choisie.

3.2.7 Initialisation des chromosomes

Habituellement, les chromosomes de la première population sont initialisés avec des valeurs aléatoires ou avec des valeurs qui dérivent le savoir-faire des experts pour accélérer la convergence de l'algorithme d'optimisation. Nous avons utilisé une combinaison de ces deux méthodes, ainsi nous initialisons les paramètres des fonctions d'appartenance par des valeurs aléatoires prises dans leur intervalle de variation. En ce qui concerne l'initialisation de la base de règles, nous utilisons le modèle de règles proposé par Macvicar-Whelan (Macvicar-Whelan, 1976). Ce modèle est considéré comme une extension de la base de règles utilisée dans les premiers contrôleurs flous développés par Mamdani (Mamdani et Assilian, 1975) et peut être obtenu à partir des méta règles ci-dessous:

MR1 Si l'erreur $e(t)$ et sa variation $\Delta e(t)$ sont à zéro (EZ) Alors maintenir le présent réglage.

MR2 Si l'erreur e(t) tend vers zéro (EZ) avec un rythme satisfaisant Alors maintenir le présent réglage.

MR3 Si l'erreur $e(t)$ n'est pas auto corrective alors l'action de commande $\Delta u(t)$ est différente de zéro, son signe ainsi que son amplitude sont fonction de l'erreur $e(t)$ et de sa variation $\Delta e(t)$.

TAB. 3.2 – *Base de règles générée à partir du modèle de Macvicar-Whelan*

Δu / e \ Δe	NG	NM	NP	Z	PP	PM	PG
NG	1	1	1	1	2	3	4
NM	1	2	2	2	3	4	5
NP	1	2	3	3	4	5	6
Z	1	2	3	4	5	6	7
PP	2	3	4	5	5	6	7
PM	3	4	5	6	6	6	7
PG	4	5	6	7	7	7	7

Dans le tableau 3.2, nous avons représenté l'ensemble des règles floues définies par le modèle de règles de Macvicar-Whelan pour le contrôleur flou considéré (partitionnement avec 7 fonctions d'appartenance). Dans le cas général (partitionnement avec un nombre quelconque de fonctions), une description détaillée de la procédure de génération des

règles, selon l'approche de Macvicar-Whelan, peut être trouvée dans l'ouvrage de Yager (Yager et Filev, 1994).

3.2.8 Exemple d'application

Dans cette section, nous allons tester les performances de la méthode proposée sur un exemple de commande d'un système linéaire de second ordre utilisé dans (Cheong et Lai, 2000).
La fonction de transfert de ce système est donnée par :

$$G(p) = \frac{2}{p(p+1.4)+2} \tag{3.13}$$

La figure 3.11 montre la réponse indicielle du système en boucle fermée (sans contrôleur), obtenue sous Matlab en utilisant la méthode d'intégration de Runge Kutta d'ordre 4 à une période d'échantillonnage de 0.1s.

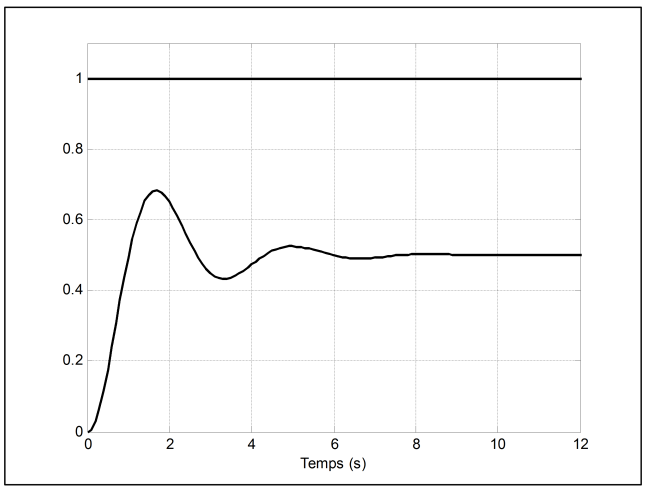

FIG. 3.11 – *Réponse indicielle du système en boucle fermée sans le contrôleur flou*

On remarque la présence d'oscillations amorties en régime transitoire qui s'éteignent au bout de 10s avec une erreur statique de 0.5.
La stratégie de commande adoptée est celle donnée en figure 3.2. Le contrôleur reçoit par ses entrées l'erreur $e(t)$ et sa variation $\Delta e(t)$ et fournit en sortie la variation de commande $\Delta u(t)$. Cette dernière sollicite l'entrée d'un intégrateur dont la sortie est l'action de commande $u(t)$. Au niveau de la structure interne, chaque variable est définie sur un

univers de discours normalisé $[+1, -1]$. Ce dernier est partitionné en sept sous ensembles flous $\{NG, NM, NP, EZ, PP, PM, PG\}$. Les fonctions d'appartenance retenues pour l'ensemble des variables du contrôleur sont de type gaussiennes asymétriques. Le traitement des règles floues, qui sont 49 au total, est effectué par la méthode MIN-MAX. La grandeur de sortie du contrôleur est générée par la méthode de centre de gravité.

3.2.8.1 Critères de performance

En générale, l'objectif d'un système de commande est de minimiser l'écart $e(t)$ entre la sortie d'un système et une valeur de consigne désirée. Cet écart peut être dû, soit à un changement de consigne, soit à des perturbations agissant sur le système. Pour choisir un bon réglage du régulateur, on prend en compte l'amplitude maximum de l'écart et la durée nécessaire pour qu'il s'annule après une perturbation ou un changement de consigne. Il existe plusiers critères numériques permettant de mesurer la qualité d'un réglage donné. Parmi ces critères on peut citer:

– Critère de l'intégrale du carré de l'erreur (ISE):

$$ISE = \sum_{n=0}^{L} [e(n.\Delta t)]^2 \tag{3.14}$$

où Δt et n sont respectivement la période d'échantillonnage et l'indice d'échantillon. $[0, L]$ est l'intervalle sur lequel est calculé le critère. En pratique, la valeur de L est choisie de manière que l'intervalle $[0, L]$ contienne suffisamment le régime transitoire

– Critère de l'intégrale de la valeur absolue de l'erreur(IAE):

$$IAE = \sum_{n=0}^{L} |e(n.\Delta t)| \tag{3.15}$$

– Critère de l'intégrale de l'erreur absolue temporelle (ITAE):

$$ITAE = \sum_{n=0}^{L} (n.\Delta t) \times |e(n.\Delta t)| \tag{3.16}$$

– Critère du pourcentage de dépassement(dep):

$$Dep(\%) = \frac{s_{max} - s_{final}}{s_{final} - s_{initial}} \times 100\% \tag{3.17}$$

Dans (Dorf et Bishop, 1995) on peut trouver une liste exhaustive de critères de performance d'un système de commande. Par la suite nous retiendrons ces deux derniers critères ($ITAE$ et Dep). Ces critères permettent de conclure que le système de commande est d'autant meilleur que les valeurs de $ITAE$ et Dep sont plus faibles. Il s'agit donc d'un problème de minimisation. Pour un algorithme génétique, ces critères vont servir pour obtenir une fonction objectif exploitable par le processus sélection.

3.2.8.2 Paramètres de l'AG

L'algorithme génétique exploitant le codage mixte des paramètres décrits précédemment doit permettre d'optimiser simultanément les paramètres des fonctions d'appartenance, les facteurs d'échelle et les conclusions des règles floues. Pour se faire, on doit choisir soigneusement les valeurs des paramètres régissant l'évolution de la population traitée par cet algorithme génétique : taille de la population, probabilitées de croisement et de mutation. Dans ce travail, après une série de tests, nous avons opté pour les paramètres du tableau 3.3

TAB. 3.3 – *Paramètres de l'AGs*

	Chromosome de l'AG standard		
	Règles floues	fonction d'appartenance	facteurs d'échelle
Représentation	En base 8 (1, 2,$\cdots$, 7)	Réelle	Binaire
Probabilité de croisement	0,65		
Probabilité de mutation	0.01	0.01	0.02
Taille de la population	40		
Nombre de générations	500		

3.2.8.3 Fonction Objectif

On fixe comme objectif, la minimisation de l'erreur $e(t)$ entre la sortie et la consigne. Cet objectif peut être défini par plusieurs indices numériques (ISE,IAE, $ITAE$, Dep.) et seul le comportement désiré peut être un paramètre prépondérant à prendre en compte pour faire un bon choix parmi ces indices. Dans notre propos, nous avons opté pour la

minimisation de l'erreur absolue temporelle ($ITAE$) et le dépassement (Dep):

$$Ob_1 = ITAE \tag{3.18}$$

$$Ob_2 = Dep \tag{3.19}$$

Par ces deux critères, il est possible de minimiser l'erreur en régime permanent par l'emploi de ($ITAE$) et de pénaliser les dépassements qui peuvent apparaitre durant le régime transitoire (en minimisant $ITAE$) par l'emploi du critère (Dep).

3.2.8.4 Méthodes de sélection

Nous avons vu, dans le chapitre 2 que la résolution d'un problème d'optimisation multi-objectif à base d'algorithmes génétiques peut se faire selon deux approches: agrégative (approche non Pareto) et non agrégative (approche de Pareto).
Si on utilise comme méthode de résolution, la première approche, il devient alors nécessaire de combiner les deux objectif Ob_1 et Ob_2 en un seul objectif Ob représentatif, soit :

$$Ob = \lambda_1 \cdot Ob_1 + \lambda_1 \cdot Ob_2 \tag{3.20}$$

où λ_1 et λ_2 sont des poids de pondération. Comme on se retrouve avec un seul objectif (Ob), une méthode de sélection telle que le tournoi ou la roulette biasiée seront suffisantes pour entamer la procédure d'optimisation. Néanmoins, il faut noter que le choix des valeurs de λ_1 et λ_2 n'est pas toujours aisé, seule la connaissance a priori des plages de variation des deux objectifs Ob_1 et Ob_2 peut être considérée pour choisir les poids appropriés.
Quant à la deuxième approche (approche de Pareto), elle présente l'avantage de ne pas nécessiter un choix de poids de pondération et permet de trouver en une seule exécution l'ensemble des solutions du front de Pareto.
Nous utilisons donc deux méthodes de sélection. La première est celle du tournoi (approche agrégative) et la deuxième est basée sur le NSGA-II, l'un des algorithmes représentatifs de l'approche non agrégative.

3.2.8.5 Résultats de l'optimisation pour la première méthode (tournoi)

Les résultats obtenus sont illustrés par les figures 3.12 à 3.15. La figure 3.12 montre les fonctions d'appartenance correspondantes au meilleur chromosome de la poulation initiale. les valeurs des facteurs d'échelle sont donnés par le tableau 3.4.

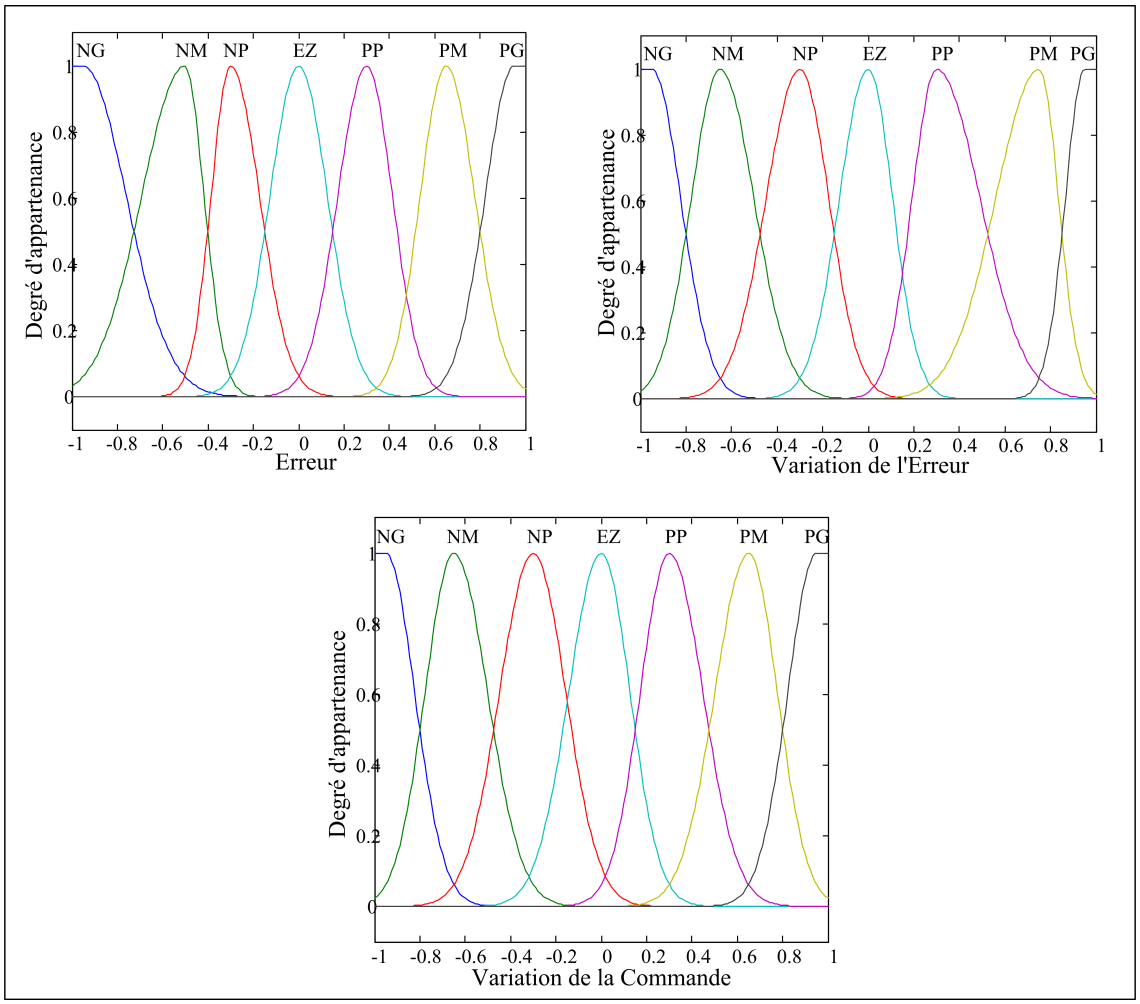

FIG. 3.12 – *Fonctions d'appartenance du meilleur chromosome de la première génération*

TAB. 3.4 – *Facteurs d'échelle du meilleure chromosome de la première génération*

G_e	$G_{\Delta e}$	G_s
1,796	19,434	0,175

Ils donnent lieu à des résultats médiocres, soit un dépassement de l'ordre de 5% et la présence d'oscillations entretenues en régime permanent(figure 3.13) pour une consigne

d'entrée constituée de deux échelons: un échelon passant de 0 à 1 (front montant) d'une durée de 10s suivi par un autre échelon de 1 à 0 (front descendent) de même durée. L'efficacité de l'AGS est ressenti dès les premières générations du processus d'optimisation, à titre d'exemple l'objectif Ob pour le meilleure chromosome de la 60^{me} génération est ramené à 34,5161 alors qu'il était initialement (première génération) de 69,3873, soit une diminution d'un rapport de 2. Les résultats sont améliorés au cours des générations (voir l'évolution Ob sur la figure 3.14) et ce n'est qu'à partir de la génération 210 ime qu'on atteint des performances très appréciables. La figure 3.15 et les tableaux 3.5 et 3.6 donnent les nouveaux paramètres correspondants au meilleur chromosome de cette génération.

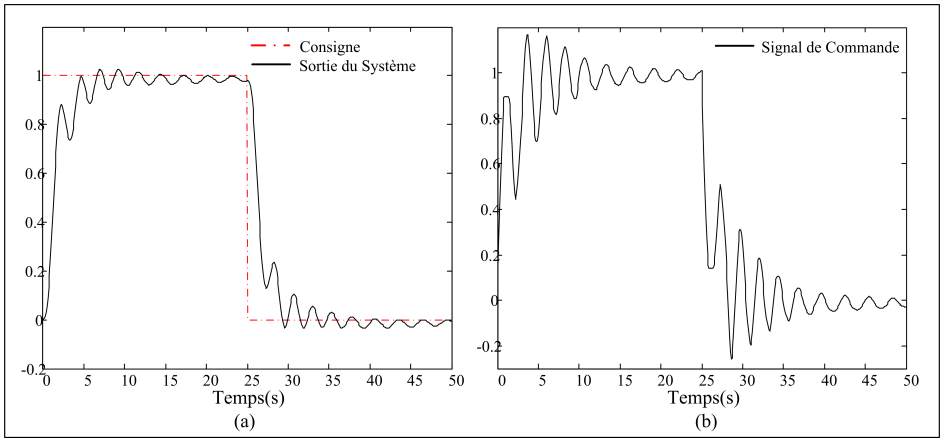

FIG. 3.13 – *Sortie du système pour le meilleur chromosome de la première génération*

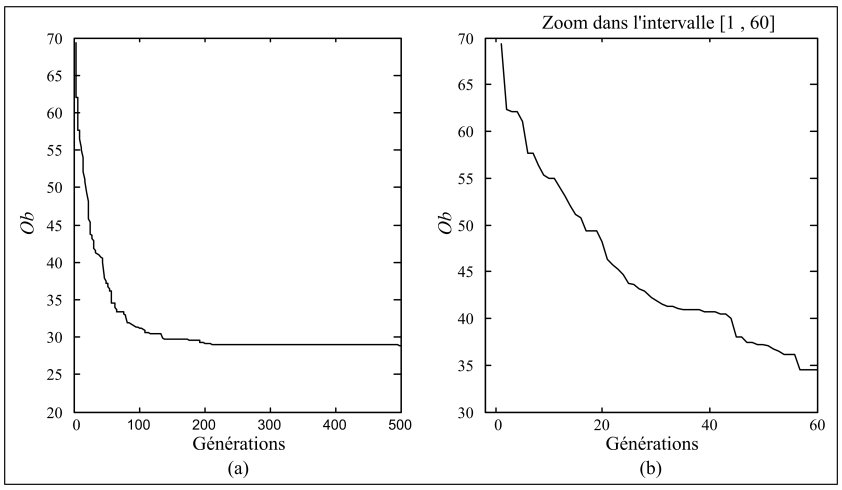

FIG. 3.14 – *Evolution de l'objectif Ob à travers les générations*

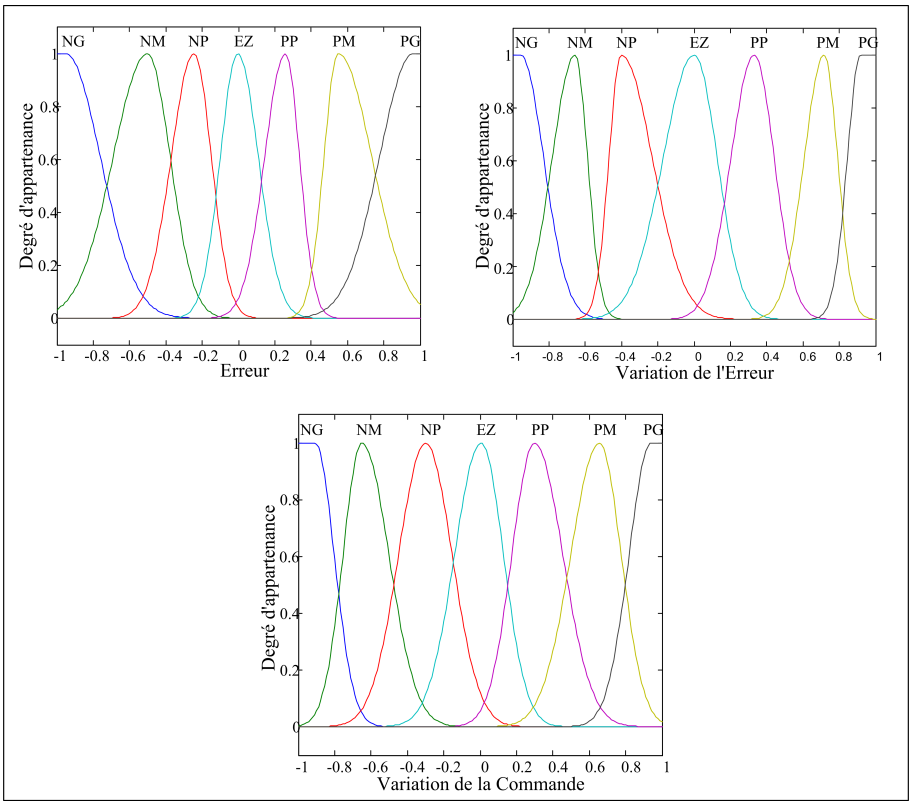

FIG. 3.15 – *Fonctions d'appartenance du meilleur chromosome de la 210 ème génération*

TAB. 3.5 – *Facteurs d'échelle du meilleur chromosome de la 210 génération*

G_e	$G_{\Delta e}$	G_s
1,998	19.623	0.181

TAB. 3.6 – *Base de règle générée à partir du meilleur chromosome de la 210 ème génération*

Δu / e \ Δe	NG	NM	NP	EZ	PP	PM	PG
NG	1	5	1	1	2	3	4
NM	6	3	2	2	2	2	5
NP	1	3	3	3	3	5	6
EZ	3	4	3	4	5	6	6
PP	4	5	5	5	4	5	7
PM	3	4	5	6	7	2	7
PG	4	6	5	7	6	7	7

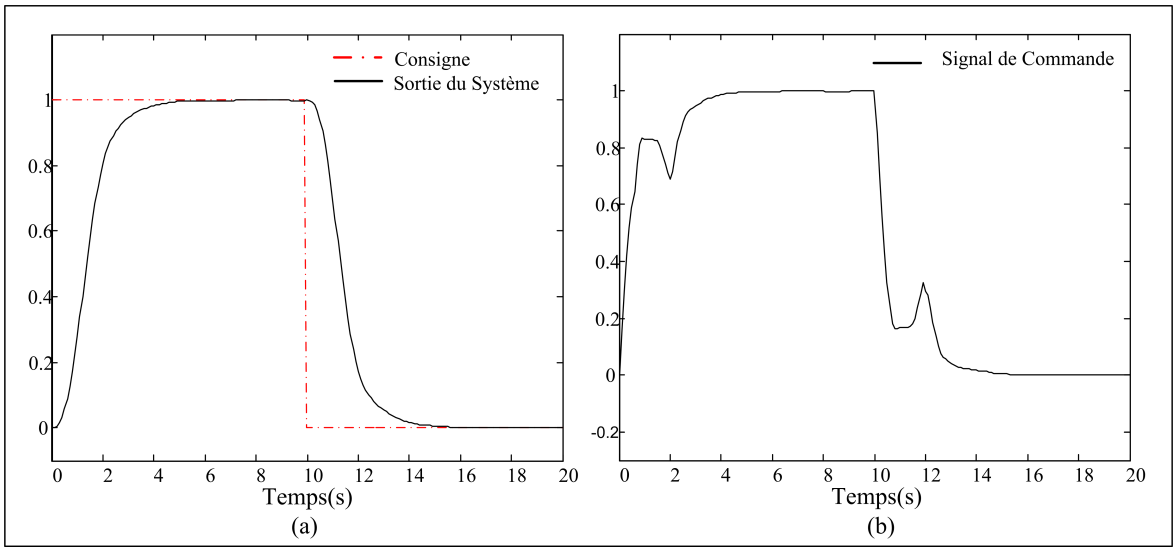

FIG. 3.16 – *Sortie du système pour le meilleur chromosome de la 210 ème génération*

On constate qu'ils sont tous a fait différents de ceux obtenus au début du processus d'optimisation. Cela se justifie par le rôle important que joue les opérateurs génétiques en explorant de nouvelles solutions dans l'espace de recherche. A la 500ème génération, nous avons toujours le même résultat (un dépassement nul et une absence d'oscillation) (voir figure 3.16-a).

3.2.8.6 Résultats de l'optimisation pour la deuxième méthode (NSGA-II)

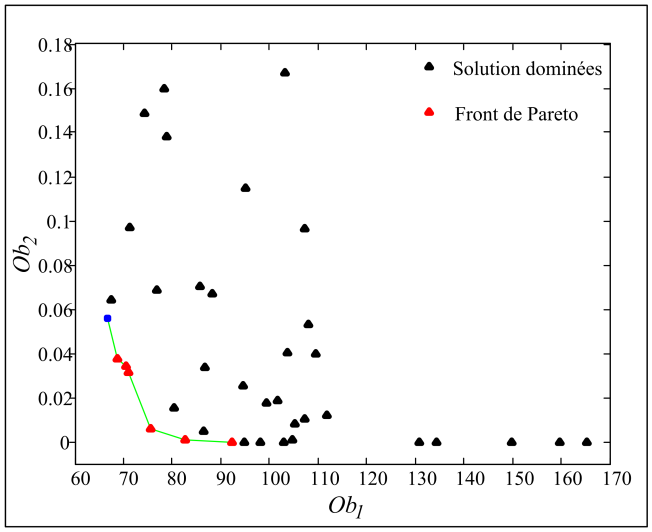

FIG. 3.17 – *Répartition de la population initiale*

La figure 3.17 montre la répartition des solutions de la première population dans l'espace objectifs (Ob_1: $ITAE$ et Ob_2: Dep).
Il faut noter que ce sont, ces mêmes solutions qui ont été utilisées par l'AGS précédent (sélection par tournoi). On retrouve alors sur la figure, la meilleure solution obtenue précédemment et qui se trouve sur l'une des extrémités du front de Pareto. De génération en génération le NSGA-II tend à regrouper les populations autour du front de Pareto, tout en minimisant les deux objectifs Ob_1 et Ob_2. Ceci est bien illustré par les figures 3.18 qui représente deux échantillons de populations obtenus au cours du processus d'optimisation.

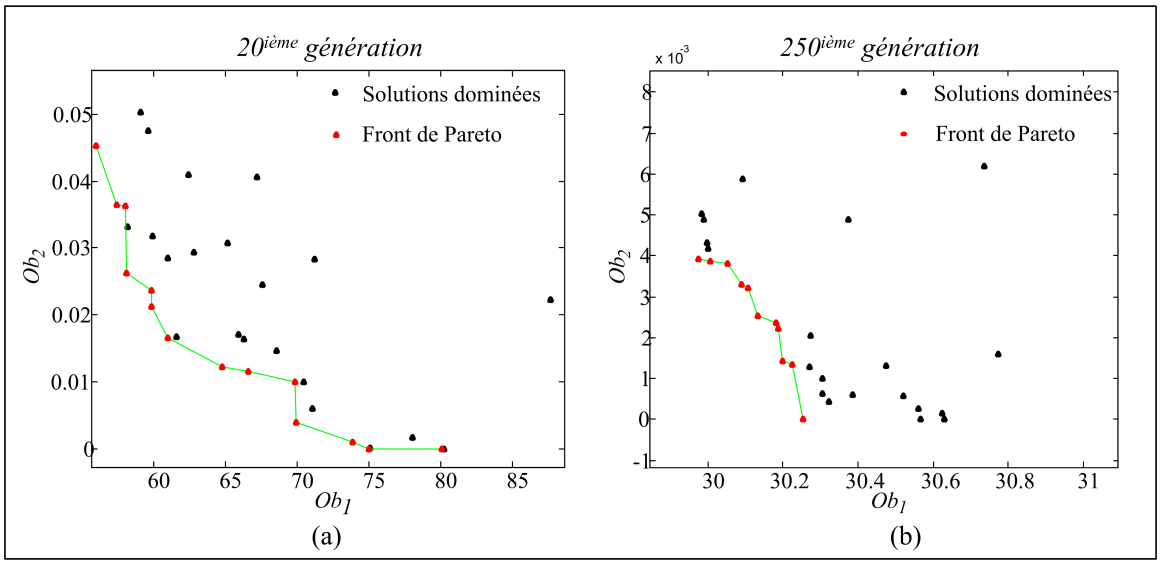

FIG. 3.18 – *Evolution de la population*

A la 500ème génération nous avons une répartition meilleure des solutions. Sur le front de Pareto, nous avons d'une part un dépassement qui est pratiquement nul (d'ordre 10^{-4}) pour toutes les solutions et d'autre part une amélioration très remarquable du critère ITAE qui atteint des valeurs optimales comprises entre 28,55 et 28,79.

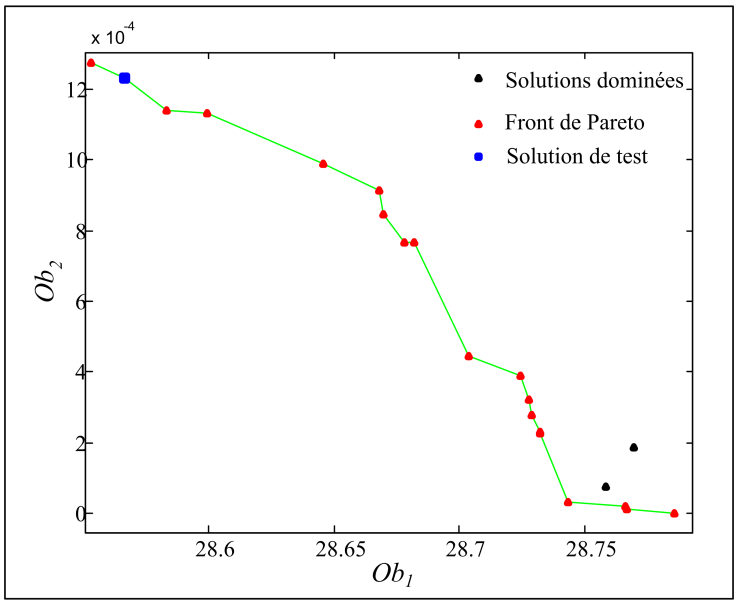

FIG. 3.19 – *Répartition de la population finale*

La figure 3.20 donne la réponse du système pour une solution de test (voir carré bleu de la figure 3.19) prise dans le front de Pareto. Nous avons un suivi de consigne satisfaisant ($ITAE = 28.56$, $Dep = 0.12\%$). Les paramètres du contrôleur correspondant sont donnés par la figure 3.21 et les tableaux 3.7 et 3.8.

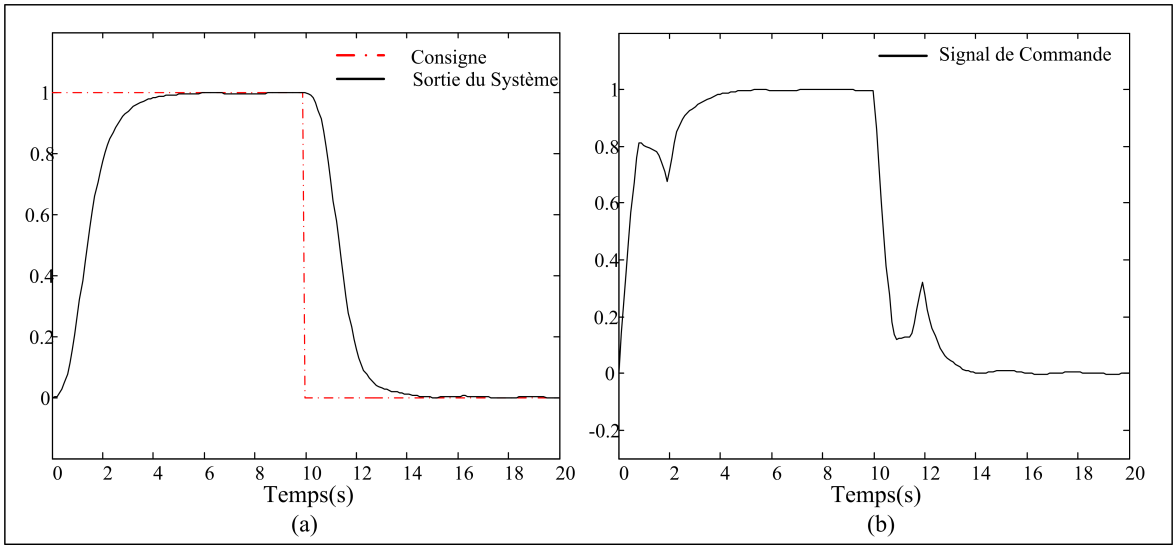

FIG. 3.20 – *Sortie du système pour la solution de test*

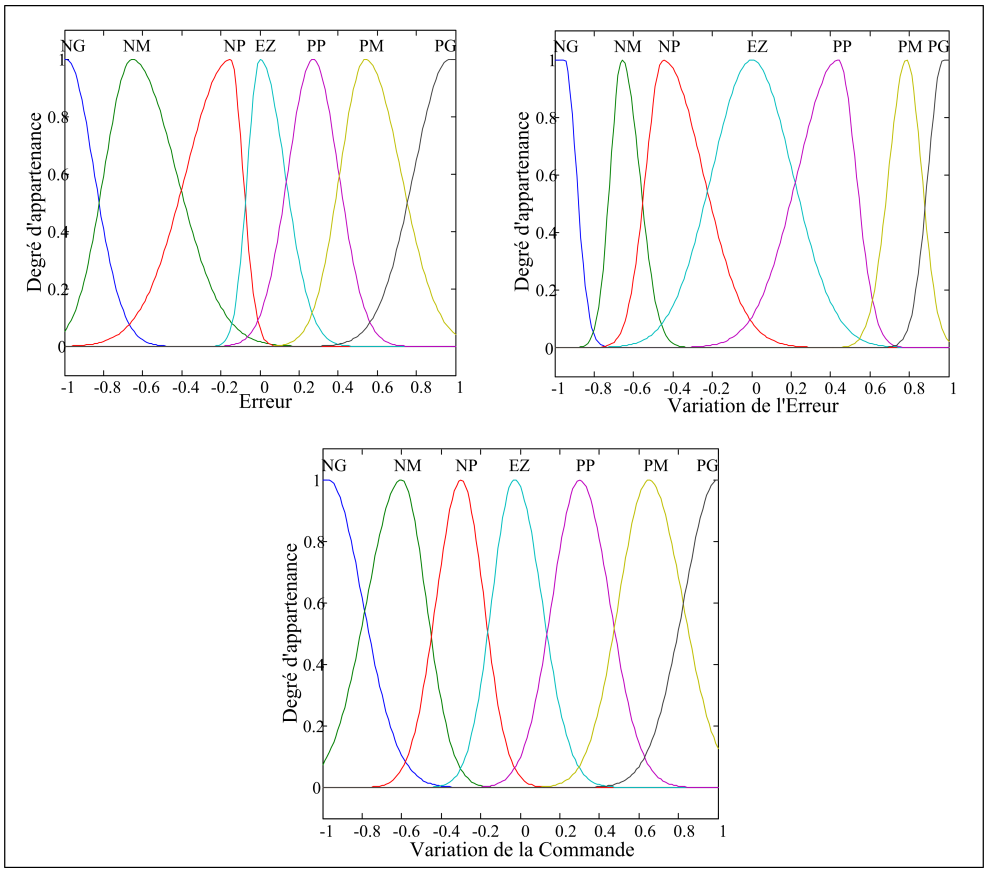

FIG. 3.21 – *Fonctions d'appartenance*

TAB. 3.7 – *Facteurs d'échelle*

G_e	$G_{\Delta e}$	G_s
1,995	18.541	0.162

TAB. 3.8 – *Base de règle générée*

Δu / e \ Δe	NG	NM	NP	EZ	PP	PM	PG
NG	1	1	1	1	2	3	4
NM	1	2	4	1	2	3	5
NP	5	3	4	3	4	3	7
EZ	4	3	3	4	5	5	7
PP	2	4	4	5	4	5	3
PM	3	5	6	6	6	6	7
PG	4	5	6	6	7	6	7

3.3 Optimisation par algorithmes génétiques hiérarchisés

Nous avons vu, dans le deuxième chapitre, que les algorithmes génétiques hiérarchisés présentent des propriétés plus intéressantes que les AG standards quand on s'intéresse à l'optimisation de la structure d'un système plutôt qu'à l'optimisation seule de ses paramètres. Pour le cas du contrôleur flou, l'optimisation structurelle consiste généralement soit à optimiser le nombre de fonctions d'appartenance de chaque variable du contrôleur (Acosta et Todorovich, 2003; Tang *et al.*, 1998), soit à optimiser le nombre de règles floues dans la base de règles (Belarbi *et al.*, 2005). Pour la première approche, nous avons vérifié à travers deux applications différentes (Guenounou et Belmehdi, 2006; Guenounou *et al.*, 2006) qu'un partitionnement avec 5 ou 7 fonctions d'appartenance est largement suffisant pour obtenir une régulation ou une poursuite de trajectoire satisfaisante. Quand à la deuxième approche, celle-ci nous semble plus importante, car la minimisation du nombre de règles permet d'une part de réduire la complexité du contrôleur et d'autre part de minimiser les charges de calcul, nécessaires pour évaluer la sortie du contrôleur, dûes essentiellement au nombre de règles qu'il faut vérifier à chaque pas d'échantillonnage. Dans la suite de ce travail nous nous intéressons à cette dernière approche, non seulement pour l'intérêt qu'elle présente, mais également pour remédier à l'inconvénient de la première méthode d'optimisation, décrite précédemment. En effet, cette méthode permet de générer dans beaucoup de cas une base de règles brouillée contenant des règles qui n'ont pas de sens.

Δu / e \ Δe

	NG	NM	NP	EZ	PP	PM	PG
NG	1	5	1	1	2	3	4
NM	6	3	2	2	2	2	5
NP	1	3	3	3	3	5	6
EZ	3	4	3	4	5	6	6
PP	4	5	5	5	4	5	7
PM	3	4	5	6	7	2	7
PG	4	6	5	7	6	7	7

Tableau (a)

Δu / e \ Δe

	NG	NM	NP	EZ	PP	PM	PG
NG	1	1	1	1	2	3	4
NM	1	2	4	1	2	3	5
NP	5	3	4	3	4	3	7
EZ	4	3	3	4	5	5	7
PP	2	4	4	5	4	5	3
PM	3	5	6	6	6	6	7
PG	4	5	6	6	7	6	7

Tableau (b)

FIG. 3.22 – *Bases de règles contenant des règles incohérentes*

Un exemple d'une telle base est représenté par les tableaux (a) et (b) de la figure 3.22. Ces deux tableaux correspondent bien aux bases de règles obtenues (tableau 3.6 et tableau 3.8) à la fin du processus d'optimisation pour les deux méthodes de sélection considérées (tournoi et NSGA-II). Les cellules entourées correspondent aux conclusions des règles contradictoires avec le sens commun et le modèle de règles de Macvicar-Whelan.

3.3.1 Structure du chromosome hiérarchisé

Le codage que nous proposons cette fois ci (voir figure 3.23) diffère de celui de la figure 3.10, par la présence de gènes de contrôle dans la nouvelle structure, et qui ont comme rôle l'activation ou la désactivation des règles floues. Le nombre de gènes de contrôle est égal à la dimension initiale de la base de règles: à chaque règle on fait correspondre un gène de contrôle. Quand " 1 " est assigné au gène de contrôle (z^i), la règle floue correspondante devient active. Dans le cas contraire (un" 0 " est assigné au gène de contrôle), cette règle devient inactive (i.e éliminée de la base de règles du contrôleur). Pour un contrôleur à deux entrées et une seule sortie (figure 3.2) le nombre de gènes de contrôle est égal à $K = N_e * N_{\Delta e}$.
Ainsi, si $z^i = 0$, $\forall i = 1,2,\cdots,K$, aucune règle n'est activée dans la base de règles du contrôleur, dans ce cas, le contrôleur ne peut fournir une commande.
Pour contourner cette singularité, nous introduisons la contrainte suivante :

$$\sum_{i=1}^{K} z^i \geq l_r \tag{3.21}$$

avec l_r le nombre minimal de règles actives dans la base de règles du contrôleur flou.

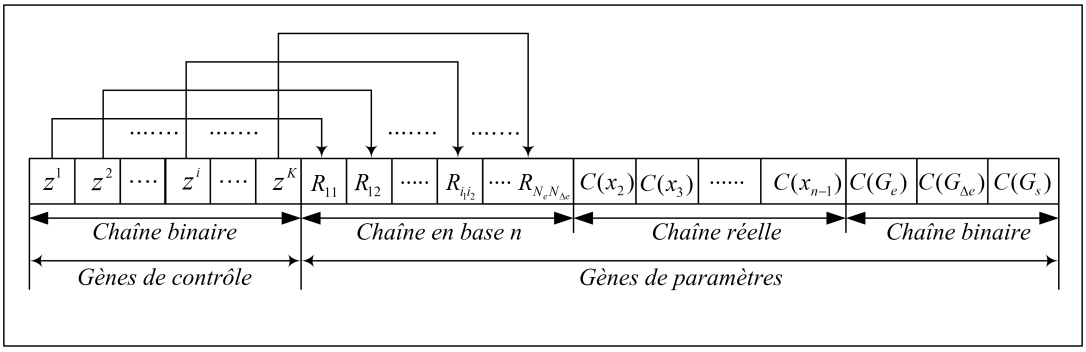

FIG. 3.23 – *Structure du chromosome de l'AGH appliqué pour l'optimisation des paramètres d'un CF*

3.3.2 Opérateurs génétiques

Les opérateurs de croisement et de mutation peuvent s'appliquer sur les gènes de paramètres (partie 2 du chromosome) comme s'il s'agissait simplement d'un algorithme génétique standard. Afin de satisfaire la contrainte (3.16), le croisement et la mutation des gènes de contrôle sont effectués de la manière suivante :

1. ***Opérateur de Croisement***: considérons les gènes de contrôle des deux chromosomes parents ch_1 et ch_2 avant de les recombiner par croisement et supposons que $l_r = 5$ (au moins cinq gènes de contrôle sont au niveau 1 pour chaque chromosome):
   - Gènes de contrôle de ch_1: $1/0000/11110$, contrainte satisfaite
   - Gènes de contrôle de ch_2: $1/1111/11000$, contrainte satisfaite.

   Si un croisement est appliqué aux points indiqués par le symbole $'/'$, les descendants suivants sont produits :
   - $ch_1^{'}$: $1/1111/11110$, contrainte satisfaite
   - $ch_2^{'}$: $1/0000/11000$, violation de contrainte.

   Afin de respecter toujours la contrainte, nous inter-changeons uniquement les premiers bits de ch_2 pour lesquels la contrainte est satisfaite, ainsi après cette correction nous obtenons:
   - $ch_2^{'}$: $1/0011/11000$, contrainte satisfaite.
2. ***Opérateur de Mutation*** : comme pour le cas de l'opérateur de croisement, la mutation de certains gènes de contrôle (quand 1 est muté à zéro), peut violer la contrainte (3.16) comme illustré ci-dessous:
   Considérons les gènes de contrôle du chromosome de ch_3: 1100001101 pour lesquels la contrainte est satisfaite ($l_r = 5$). Si maintenant une mutation est opérée sur le premier gène, on obtient la chaîne : 010001101, par conséquent la contrainte est violée. Pour la satisfaire on remet le bit muté à 1 (annulation de la mutation).

3.3.3 Fonction objectif supplémentaire

Pour remédier au problème de présence de règles floues inutiles et/ou incohérentes dans la base de règles du contrôleur optimisé, nous introduisons, en plus des objectifs précédents relatifs aux performances du contrôleur en termes d'erreur (ISE, IAE, ITAE, Dep) un autre objectif supplémentaire qui favorise les solutions correspondantes à un

nombre minimal de règles actives. Cet objectif se traduit par une simple sommation des gènes de contrôle:

$$Ob_2 = \sum_{i=1}^{K} z^i \tag{3.22}$$

3.3.4 Evaluation des performances de l'AGH

Pour évaluer les performances du contrôleur flou optimisé par les AGH, des tests de simulation ont été réalisés sur le même système traité précédemment par les AGS.
Nous avons considéré le même type et le même nombre de fonctions d'appartenance pour définir les variables du contrôleur sur leur univers de discours. Nous avons conservé également la même procédure d'initialisation des chromosomes, les gènes de contrôle qui constituent ici une partie supplémentaire et qui prennent leur valeur dans l'ensemble $\{0,1\}$ sont initialisés par un processus aléatoire où l'on accorde plus de probabilité à un qu'à zéro. De cette façon, on génère initialement des solutions qui considèrent un nombre important de règles floues de la base du contrôleur.
Deux objectifs sont considérés:

- Minimisation de l'intégrale de l'erreur absolue temporelle (ITEA) et le dépassement Dep.(equation 3.20 avec $\lambda = 50$)
- Minimisation de règles floues (équation 3.22)

Les paramètres de L'AGH sont ceux du tableau 3.3, définis précédemment pour les AGS, auxquels il faut ajouter la probabilité de mutation ($p_{mc} = 0.005$) et de croisement ($p_{cc} = 0.45$) des gènes de contrôle et le nombre minimal ($l_r = 20$) de règles à conserver dans la base de du contrôleur.
La figure 3.24 donne la répartition sur le plan des objectifs (Ob_1, Ob_2) des solutions de la dernière génération. On constate un regroupement des solutions autour du front de Paréto. Les quelques solutions qui se trouvent éloignées du front de Pareto, sont dues principalement aux effets de la mutation et du croisement des gènes de contrôle, qui peuvent être nocifs dans certains cas en désactivant des règles pertinentes et/ou en activant des règles incohérentes. C'est d'ailleurs la raison pour laquelle, les probabilités d'application des ces opérateurs ont été fixées à des valeurs relativement faibles par rapport à celle d'un algorithme génétique simple.

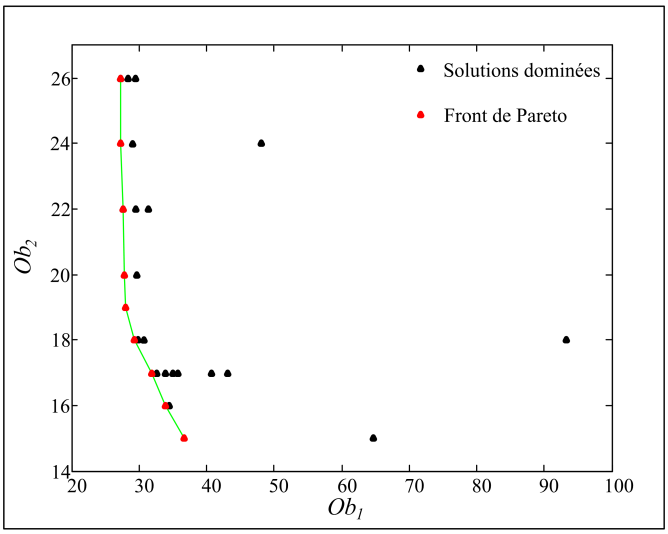

FIG. 3.24 – *Répartition de la population finale*

La figure 3.25 représente les solutions constituant le front de Pareto. Tout d'abord il y a lieu de constater que les deux objectifs Ob_1 et ob_2 sont contradictoires car en diminuant le nombre de règles, l'objectif Ob_1 tend à se détériorer, notamment à partir de moins de 20 règles. D'autre part il faut noter que certaines configurations pour lesquelles nous avons plus de 26 règles actives ($Ob_2 > 26$) ne font pas partie de ce front. Au-delà de 26 règles il est difficile d'apporter une amélioration à l'objectif Ob_1, c'est-à-dire d'obtenir une valeur de l'objectif Ob_1 inférieur à 27,367.

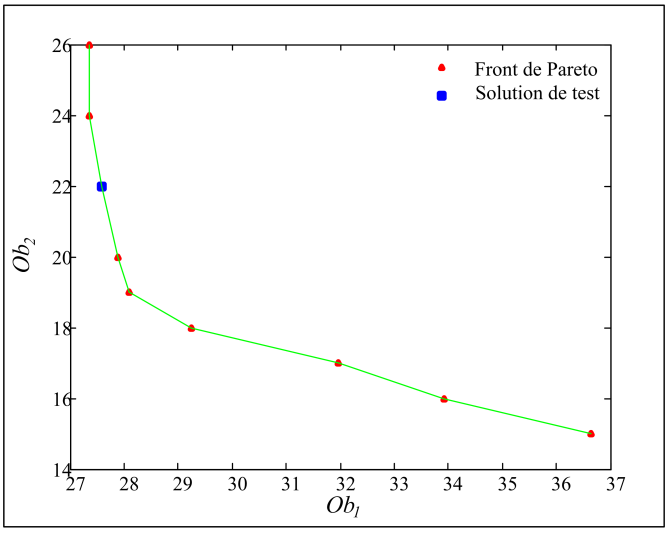

FIG. 3.25 – *Solutions constituant le front de Pareto*

La figure 3.26 donne la réponse du système correspondante à une solution prise dans le front de Pareto.

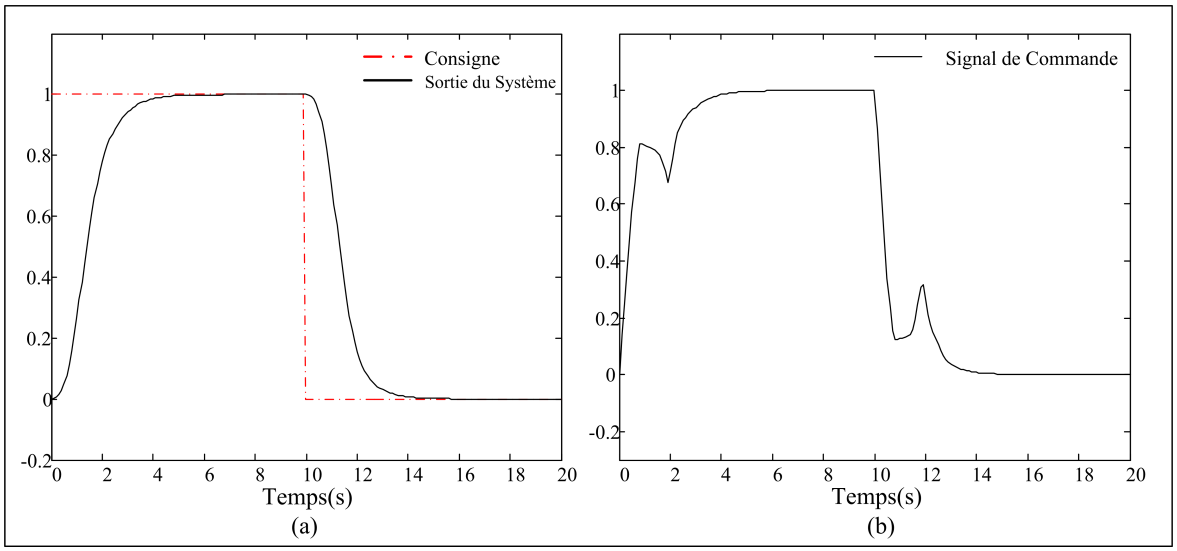

FIG. 3.26 – *Sortie du système pour la solution de test*

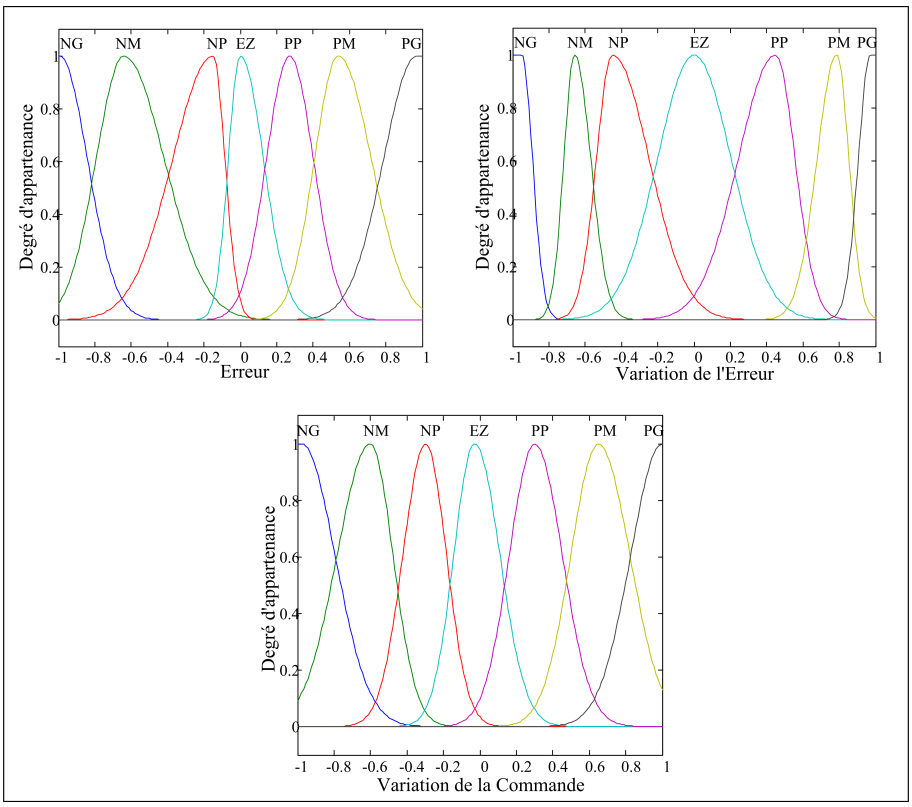

FIG. 3.27 – *Fonctions d'appartenance*

Avec uniquement 22 règles (voir le carré bleu de la figure 3.25), nous avons des performances très semblables à celles obtenues en considérant la totalité des règles floues de la base du contrôleur. Les différents paramètres correspondants à cette solution sont illustrés par la figure 3.27 et les tableaux 3.9 et 3.10

TAB. 3.9 – *Facteurs d'échelle*

G_e	$G_{\Delta e}$	G_s
2,015	18.541	0.162

TAB. 3.10 – *Base de règles avec 22 règles floues*

Δu / Δe	NG	NM	NP	EZ	PP	PM	PG
NG	1			1			4
NM					2	3	5
NP					4		6
e EZ			3	4			
PP	2	4	5	5			
PM	3	5	6				
PG	4	5	6	6			7

3.4 Optimisation des contrôleurs flous de type Sugeno

La conception de contrôleurs flous de types Sugeno par algorithmes génétiques (AGS ou AGH) peut se faire de la manière exposée précédemment pour le cas du contrôleur de type Mamdani. Tout ce qui change est le codage des conclusions des règles floues qui sont de nature différente pour les deux types de contrôleurs. En effet avec un contrôleur flou à deux entrées et une seule sortie tel qu'illustré par la figure 3.2, les règles floues de Sugeno sont décrites par:

Règle i_1,i_2: Si (e est A_{1i_1}) ET (Δe est A_{2i_2}) Alors $\Delta u = a_0^{i_1 i_2} + a_1^{i_1 i_2} \times e + a_2^{i_1 i_2} \times \Delta e$

où $a_0^{i_1 i_2}$, $a_1^{i_1 i_2}$et $a_2^{i_1 i_2}$ sont les paramètres associés à la *Règle* i_1,i_2.
Dans ce cas, la conclusion d'une règle peut être définie par les trois paramètres: a_0, a_1 et

a_2.

Tenant compte de cette considération, l'optimisation par algorithme génétique de l'ensemble des paramètres du CF de Sugeno peut être obtenue à partir des structures illustrées par la figure ci-dessous.

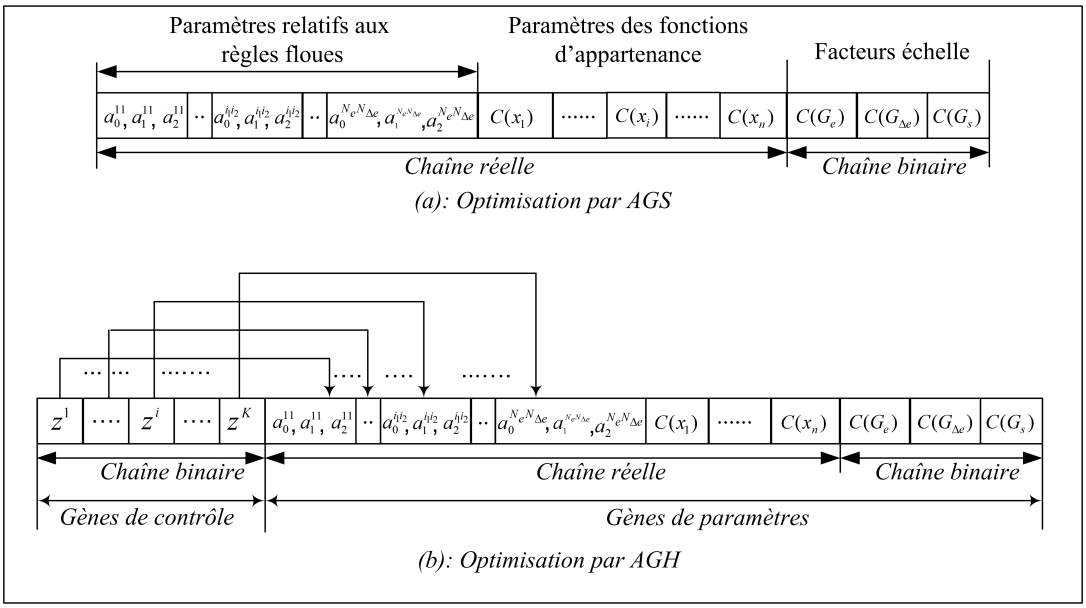

FIG. 3.28 – *Structures du chromosome optimisant le contrôleur de Sugeno*

La partie supérieure de la figure correspond à une structure simple qui fait intervenir deux type de représentation: binaire pour les facteurs d'échelle et réelle pour le reste des paramètres (paramètres des fonctions d'appartenance et règles floues). Dans la partie inférieure de la figure, la structure est un peu différente, elle présente un niveau de gènes de contrôle supplémentaire qui a pour rôle l'activation ou la désactivation des règles floues. En exploitant ces structures, il est tout à fait possible d'aboutir à des résultats semblables à ceux présentés dans les deux sections précédentes. Toutefois l'inconvénient majeur de cette méthode réside dans le fait que l'on ne dispose pas de modèle de règles comme pour le cas des contrôleurs de Mamdani, pour initialiser la base de règles du contrôleur et qui peut, par conséquent, rendre la procédure de recherche couteuse en temps. Pour remédier à cet inconvénient, nous avons proposé dans (Guenounou *et al.*, 2009) une méthode d'initialisation en utilisant l'algorithme de rétropropagation du gradient. Comme ce dernier s'adapte mieux à des structures multicouches (réseaux de neurones), le contrôleur flou est donc transformé en un réseau multicouche (réseau neuro-flou) avant toute procédure d'optimisation. Dans ce qui suit, nous allons présenter la structure du réseau neuro-flou

proposée ainsi que son algorithme d'apprentissage.

3.4.1 Structure du réseau Neuro-flou

Le réseau neuro-flou proposé est une architecture à cinq couches qui comprend les éléments d'un système flou de type Sugeno. Pour ne pas rester dans le cas d'un contrôleur à deux entrées, nous considérons cette fois ci un cas plus général avec un réseau à r entrées $[x_1, x_2, x_3, ..., x_r]$ et à une sortie y. En considérant la figure 3.29, explicitons le fonctionnement du réseau, couche par couche.

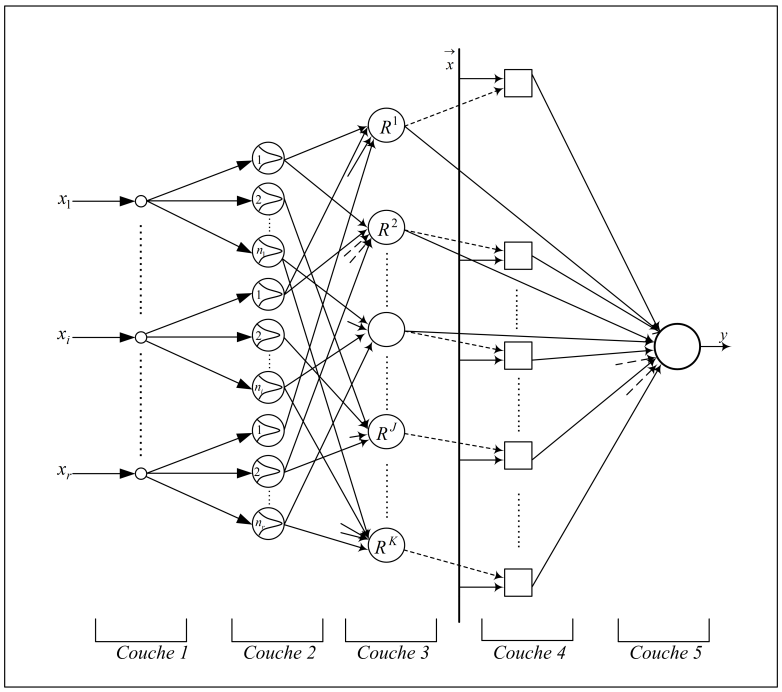

FIG. 3.29 – *Structure du réseau neuro-flou*

3.4.1.1 Première couche (couche d'entrée)

Aucune fonction n'est réalisée avec les neurones de cette couche. Chaque neurone transmet le signal d'entrée vers la deuxième couche.

$$O_i^1 = u_i^1, \qquad u_i^1 = x_i \tag{3.23}$$

3.4.1.2 Deuxième couche (couche des fonctions d'appartenance)

Les neurones de cette couche correspondent aux termes linguistiques (sous ensembles flous) associés aux variables d'entrée du contrôleur. La sortie de chaque neurone fournit le degré d'appartenance d'une variable d'entrée au sous ensemble flou correspondant à ce neurone. Elle est donnée par :

$$O_i^2 = exp(-\frac{(u_i^2 - c_i)^2}{2\sigma_i^2}) \tag{3.24}$$

$$u_i^2 = \begin{cases} O_1^1 & \text{if } i = 1,2,...,n_1 \\ O_2^1 & \text{if } i = n_1 + 1,...,n_1 + n_2 \\ O_j^1 & \text{if } i = n_1 + .. + n_{j-1} + 1,...,n_1 + .. + n_j \\ . & \\ O_r^1 & \text{if } i = n_1 + .. + n_{r-1} + 1,...,n_1 + .. + n_r \end{cases} \tag{3.25}$$

où n_1, n_2, ... et n_r représentent respectivement le nombre de sous ensembles flous associés aux varaibles du contrôleur. c_i et σ_i sont les paramètres de la gaussienne du ième neurone.

3.4.1.3 Troisième couche(couche des prémisses)

Chaque neurone de cette couche correspond à une règle floue de la base de règles. Ses entrées proviennent de tous les neurones de la deuxième couche qui participent dans la partie prémisse de la règle. La sortie de chaque neurone représente le degré de vérité d'une règle, qui peut être obtenu à partir de:

$$O_i^3 = \prod_j u_j^3, u_j^3 = O_j^2, i = 1,2...K \tag{3.26}$$

avec $K = n_1 \times n_2 \times \cdots \times n_r$ le nombre total de règles floues.

3.4.1.4 Quatrième couche(couche des conclusions)

Chaque neurone de cette couche correspond à la conclusion d'une règle floue. Il y a donc autant de neurones dans cette couche que dans la troisième couche. la sortie de chaque neurone est obtenue par une simple somme pondérée des variables d'entrée :

$$O_i^4 = a_0^i + \sum_{j=1}^{r} a_j^i \times x_j \tag{3.27}$$

où a_j^i sont les paramètres correspondant à la conclusion de la ième règle floue.

3.4.1.5 Cinquième couche (couche de sortie)

Le neurone de cette couche ou neurone de sortie, correspond à la variable de sortie du contrôleur. Il intègre toutes les actions préconisées par les couches 3 et 4 de manière à réaliser l'étape de défuzzification:

$$O^5 = \frac{\sum_{j=1}^{K} O_j^3 O_j^4}{\sum_{j=1}^{K} O_j^3} \tag{3.28}$$

3.4.2 Optimisation des paramètres du réseau neuro-flou

Dans cette section, nous allons présenter l'algorithme hybride proposé pour l'apprentissage (i.e optimisation) de la structure du réseau neuro-flou. Le principe de cet algorithme est très simple: il consiste à utiliser dans un premier temps l'algorithme de rétropropagation pour un ajustement de paramètres et dans un second temps un algorithme génétique hiérarchisé pour un deuxième ajustement de paramètres tout en minimisant le nombre de règles floues.

3.4.2.1 Optimisation par l'algorithme de rétropropagation du gradient

Pour entraîner le réseau, nous utilisons l'algorithme de rétropropagation du gradient déjà décrit dans le chapitre 2 pour le cas d'un perceptron multicouches. Les équations présentées doivent cependant être adaptées aux fonctions particulières des neurones utilisés. Le critère à minimiser pour la mesure de performance reste l'erreur quadratique donnée par l'équation:

$$J(t) = 0.5\,(y(t) - y^d(t))^2 \tag{3.29}$$

où $y(t)^d$ et $y(t)$ sont respectivement la sortie désirée et la sortie actuelle du réseau neuro-flou.
L'adaptation des paramètres a_j^i correspondant aux conclusions des règles floues peut être obtenue par l'équation ci-dessous:

$$a_j^i(t+1) = a_j^i(t) - \eta_1 \frac{\partial J(t)}{\partial a_j^i(t)} \tag{3.30}$$

avec

$$\frac{\partial J(t)}{\partial a_j^i(t)} = (y(t) - y^d(t)) \frac{O_i^3 x_j}{\sum_{k=1}^{K} O_k^3} \tag{3.31}$$

La loi d'adaptation du paramètre c_i, (centre de la i fonction d'appartenance du réseau) est:

$$c_i(t+1) = c_i(t) - \eta_2 \frac{\partial J(t)}{\partial c_i(t)} \tag{3.32}$$

la dérivé du crière $J(t)$ par apport à c_i peuvent être obtenue à partir de la décomposition ci-dessous :

$$\begin{aligned} \frac{\partial J(t)}{\partial c_i(t)} &= \frac{\partial J(t)}{\partial O_i^2} \frac{\partial O_i^2}{\partial c_i(t)} \\ &= (\textstyle\sum_k \delta_k^3 \frac{\partial O_k^3}{\partial O_i^2})(\frac{(O_l^1 - c_i) O_i^2}{\sigma_i^2}) \end{aligned} \tag{3.33}$$

avec

$$\delta_k^3 = \frac{\partial E(t+1)}{\partial O_k^3} = (y(t+1) - y^d(t+1)) \frac{O_k^4 \sum_{j=1}^{K} O_j^3 - \sum_{j=1}^{K} O_j^3 O_j^4}{(\sum_{j=1}^{K} O_j^3)^2} \tag{3.34}$$

et

$$\frac{\partial O_k^3}{\partial O_i^2} = \prod_{j \neq i} O_j^2 \tag{3.35}$$

Les équations d'adaptation du paramètre σ_i peut être obtenues de la même manière que celles de c_i et elles sont omises.

3.4.2.2 Optimisation par algorithme génétique

Une fois que les paramètres des fonctions d'appartenance et ceux des règles floues sont obtenus, un algorithme génétique hiérarchisé peut être appliqué pour un deuxième réglage des paramètres tout en minimisant le nombre de règles floues initiales. L'algorithme fonctionne de manière identique à celui décrit dans la section 3.3 pour le cas d'un contrôleur flou de type Mamdani. La seule différence réside dans l'étape de génération des chromosomes composants la première population. Pour mieux comprendre cette étape, dans ce cas, nous la détaillons ci-dessous à travers un exemple d'application.

3.4.3 Exemple

On ce propose, ici, de tester les performances de l'algorithme d'apprentissage hybride à travers un problème classique de modélisation d'un système non linéaire proposé par Narendra et Pathasarathy (1990).

3.4.3.1 Modèle de simulation

Le système non linéaire à modéliser est décrit par l'équation aux différences suivante :

$$y(k) = [\frac{y(k-1)y(k-2)(y(k-1)+2.5)}{1+y^2(k-1)+y^2(k-2)} + u(k-1)] \tag{3.36}$$

Nous avons choisi l'architecture série-parallèle car elle est considérée comme plus stable vu que le réseau est régulièrement recalé en utilisant l'état réel du processus. Le modèle possède trois entrées $u(k-1)$, $y(k-1)$ et $y(k-2)$ de trois ensembles flous $\{N,Z,P\}$ chacune, 27 règles floues correspondantes aux différentes combinaisons des ensembles flous et une sortie $y(k)$.

3.4.3.2 Résultats de simulations

Première phase

Dans cette première phase, l'algorithme de retropropagation est utilisé pour l'adaptation des paramètres des fonctions d'appartenance et des règles floues. Une base de données de 500 points, générés à partir de l'équation (3.36) et une entrée $u(k)$ uniformément répartie dans l'intervalle $[-2,2]$, a été considérée. Les paramètres de l'algorithme sont fixés comme suit :$\eta_1 = 0.12$ et $\eta_2 = 0.05$. Les paramètres des fonctions d'appartenance et des règles floues résultants sont rapportés dans les tableaux 3.11 et 3.12 respectivement.

TAB. 3.11 – *Paramètres des fonctions d'appartenance obtenus par rétropropagation*

	N		Z		P	
	c	σ	c	σ	c	σ
$u(k-1)$	-1.5183	1.2645	-0.0560	1.0001	1.3972	1.3857
$y(k-1)$	-2.2635	1.1603	2.1290	1.2138	5.2837	1.8343
$y(k-2)$	-2.1265	1.1331	2.6931	1.5531	5.8363	1.6604

TAB. 3.12 – *Base de règles floues obtenue par rétropropagation*

Règle	Prémisses des règles floues			Paramètres relatifs aux règles floues			
	$u(k-1)$	$y(k-1)$	$y(k-2)$	a_0^i	a_1^i	a_2^i	a_3^i
1	N	N	N	0.3819	0.5643	0.5282	0.1331
2	N	N	Z	-1.0990	0.9036	-0.3076	0.0405
3	N	N	P	0.0375	-0.2100	0.4020	0.1568
4	N	Z	N	-1.2384	1.2754	-0.2871	0.0209
5	N	Z	Z	-0.1550	0.7130	0.8397	-0.3301
6	N	Z	P	-0.3562	0.1647	-0.0110	-0.1022
7	N	P	N	-0.2726	0.2397	-0.0472	-0.1973
8	N	P	Z	0.2389	-0.0889	0.1119	0.0297
9	N	P	P	0.3798	0.5519	0.1797	0.2498
10	Z	N	N	0.8673	0.6258	0.2077	-0.0758
11	Z	N	Z	-0.6847	1.0613	-0.0703	-0.2491
12	Z	N	P	0.0319	-0.4731	0.1814	-0.1433
13	Z	Z	N	-0.9350	0.3343	-0.8489	-0.2357
14	Z	Z	Z	0.5132	0.5598	0.8888	0.0130
15	Z	Z	P	0.3603	0.3775	0.3822	0.2313
16	Z	P	N	-0.2939	-0.1890	0.2667	-0.3359
17	Z	P	Z	0.0990	-0.1465	0.1275	0.6254
18	Z	P	P	0.1049	0.0452	0.4170	0.1453
19	P	N	N	1.5562	0.8578	0.4061	0.0614
20	P	N	Z	0.2563	0.6218	0.0918	-0.1722
21	P	N	P	-0.1096	-0.0101	-0.0462	-0.3373
22	P	Z	N	-0.1032	1.0483	-0.5471	0.4047
23	P	Z	Z	0.9238	0.7215	0.5852	0.3663
24	P	Z	P	0.0615	0.7278	0.3912	0.0250
25	P	P	N	0.2687	0.2973	-0.2713	-0.0743
26	P	P	Z	-0.2067	-0.0493	0.5624	1.1994
27	P	P	P	0.2293	0.1330	0.2190	0.7235

Deuxième phase

Une fois la première phase terminée, le NSGA-II est utilisé pour un deuxième ajustement des paramètres. Deux objectifs ont été fixés :

1. Minimisation de l'erreur quadratique moyenne définie par :

$$Ob_1 = \frac{1}{N}\sum_{k=1}^{N}(y^d(k) - y(k))^2 \tag{3.37}$$

2. Minimisation du nombre de règles floues (équation 3.22).

La figure 3.30 montre la distribution des objectifs correspondants aux chromosomes de la première population. On constate que l'objectif 1 prend ses valeurs dans l'intervalle $[3.10^{-3}$,$28.10^{-3}]$ proche de la solution obtenue par rétropropagation ($Ob1 = 3{,}063.10^{-3}$, voir figure 3.30) tandis que l'objectif 2 prend ses valeurs autour de 27. Cette répartition ne

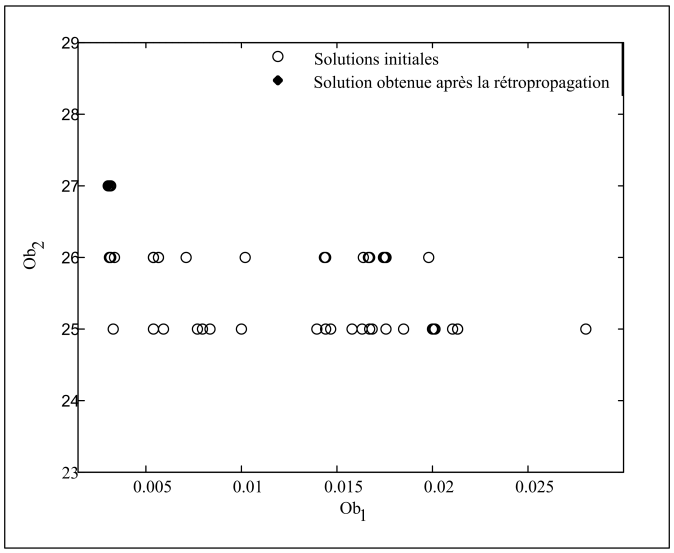

FIG. 3.30 – *Répartition des solutions de la première génération*

reflète que la méthode adoptée ici pour l'initialisation des chromosomes. Contrairement aux méthodes traditionnelles qui se basent sur un processus aléatoire, dans notre cas nous utilisons le résultat obtenu par retropropagation pour générer la population initiale. Au début un premier chromosome mère est créé par codage des paramètres du réseau conformément à la structure illustrée par la figure 3.28-b. Pour activer l'ensemble des règles, les gènes de contrôle sont forcés à 1. Ensuite les autres chromosomes sont générés par mutation avec des taux faibles des gènes de paramètres et de contrôle de ce premier chromosome. Cela permet comme l'indique la figure 3.30 de créer des chromosomes puissants au sens de Ob_1. Nous rappelons que la première phase ne permet d'améliorer que l'objectif 1, l'objectif 2 étant fixé à 27 (toutes les règles sont activées). Comme le NSGA-II, est un algorithme d'optimisation multi-objectif élitiste, le résultat obtenu à la fin de la première phase ne risque pas de se perdre à travers les générations, s'il n'est pas amélioré à la dernière génération, il sera au moins préservé. La figure 3.31 montre la répartition des objectifs relatifs aux chromosomes de la dernière génération. On constate, un regroupement au tour du front de Pareto (figure 3.32) et une amélioration significative des deux objectifs. Une fois l'apprentissage accompli, les performances du réseau neur-flou sont alors évaluées sur un ensemble de test de 75 points générés à partir d'un signal d'entrée sinusoïdal de la forme:

$$u(k) = \sin(2\pi k/25) \tag{3.38}$$

Le résultat de simulation obtenu est donné par la figure 3.33. On constate que le réseau neuro-flou réalise une bonne approximation du système avec une erreur quadratique mo-

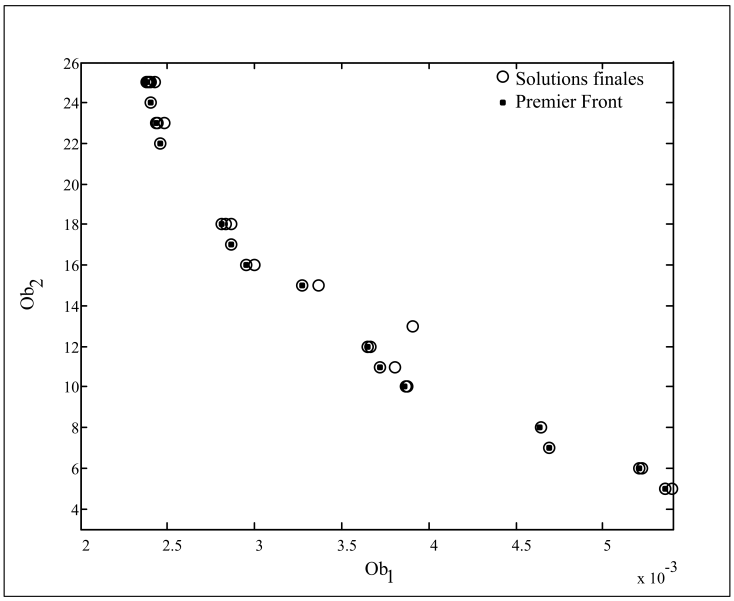

FIG. 3.31 – *Répartition des solutions de la dernière génération*

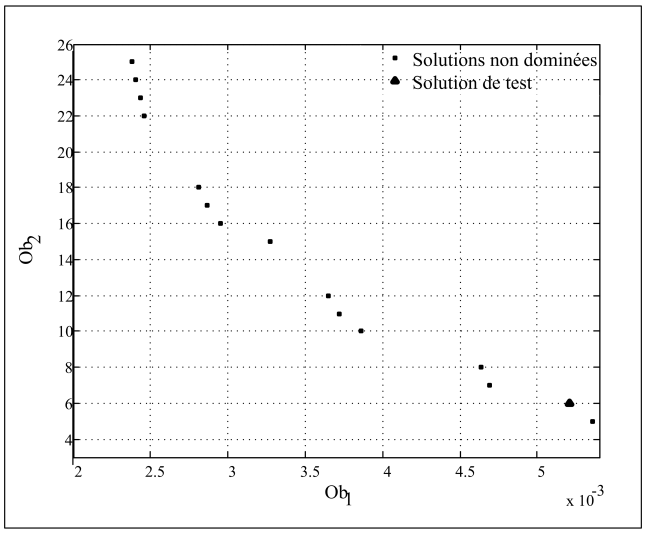

FIG. 3.32 – *Répartition des solutions du front de Pareto*

yenne de $2{,}69.10^{-3}$. Pour ce résultat, nous avons choisi les paramètres relatifs au chromosome appartenant au front de Paréto de la population finale et indiqué par un triangle. Dans ce chromosome, uniquement 6 règles floues ont été considérées, le reste (21 règles) est désactivé par les valeurs prises par les gènes de contrôle (les gènes de contrôle correspondant sont nuls).

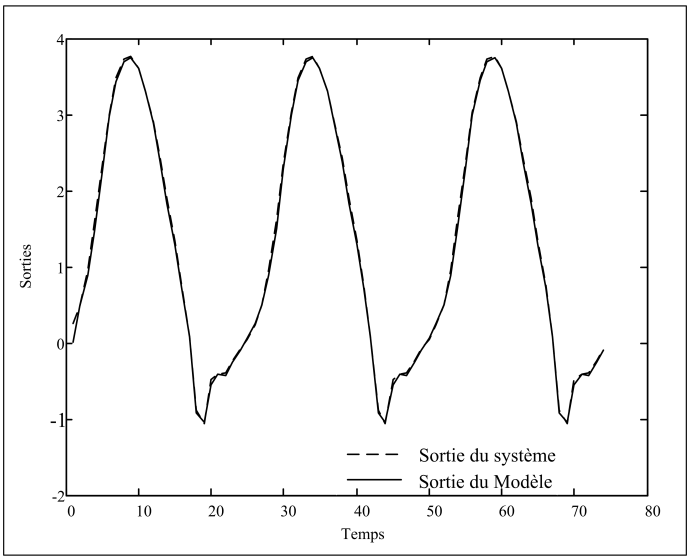

FIG. 3.33 – *Test du modèle*

TAB. 3.13 – *Paramètres des fonctions d'appartenance relatifs à la solution de test*

	N		Z		P	
	c	σ	c	σ	c	σ
$u(k-1)$	-1.4652	1.7419	-0.2416	1.9084	0.9720	1.5988
$y(k-1)$	-2.0775	1.1299	2.1737	1.0939	-	-
$y(k-2)$	-1.6944	1.1538	2.7835	1.7626	5.6253	2.1412

TAB. 3.14 – *Base de règles relative à la solution de test*

Règle	Prémisses des règles floues			Paramètres relatifs aux règles floues			
	$u(k-1)$	$y(k-1)$	$y(k-2)$	a_0^i	a_1^i	a_2^i	a_3^i
2	N	N	Z	-1.1995	0.7939	-0.1907	0.0211
5	N	Z	Z	0.1951	0.7252	0.8052	-0.2866
10	Z	N	N	0.9245	0.9832	0.2813	0.0423
13	Z	Z	N	-1.2289	1.0474	-0.5949	-0.1234
21	P	N	P	-0.0420	0.1193	-0.3753	0.2551
23	P	Z	Z	1.2833	0.6592	0.4041	0.2518

Afin de montrer l'efficacité de l'algorithme hybride proposé, le même problème de modélisation est traité avec un autre réseau neuro-flou qui exploite des règles floues de types Mamdani et qui a été proposé par Farag *et al.* (1998). Dans leur modèle les auteurs proposent un algorithme d'apprentissage réparti en trois phases. Ils utilisent durant la première phase l'algorithme de Kohonen pour trouver les fonctions d'appartenance initiales et dans la seconde phase l'algorithme MMFA (Maximum Matching-Factor Algorithm) pour l'optimisation des règles floues et enfin un algorithme génétique pour un deuxième

réglage des fonctions d'appartenance. Aussi le tableau (3.15) présente une étude comparative avec d'autres approches de modélisation utilisant les règles floues de type Takagi Sugeno. Dans (Wang et Langari, 1996), les fonctions d'appartenance sont obtenues par l'algorithme de classification floue fuzzy c-mean alors que les paramètres relatifs aux règles floues sont estimés par la méthode des moindres carrés. C'est encore cette dernière méthode qui est combinée avec l'algorithme de rétropropagation dans l'approche de Jang (1993). Les résultats obtenus, montrent que les performances de notre approche sont supérieures à celles des autres modèles même avec un nombre réduit de règles floues. Il est aussi important de noter que notre algorithme hybride aboutit à un ensemble de solutions optimales en sens de Pareto au lieu d'une unique solution, ensemble qui pourra être filtré et affiné par l'utilisateur.

TAB. 3.15 – *Comparaison avec d'autres modèles*

	Structure		EQM	EQM
	Variables d'entrée	Nombre de règles	(Apprentissage)	(test)
Notre modèle	$u(k-1)$ $y(k-1)$ $y(k-2)$	6 7 11 16 22	0.00520 0.00469 0.00372 0.00295 0.00245	0.00269 0.00260 0.00250 0.00189 0.00179
Modèle de Mamdani (Farag *et al.*, 1998)	$u(k-1)$ $y(k-1)$ $y(k-2)$	22	0.00357	0.00214
Modèle de Sugeno (Jang, 1993)	$u(k-1)$ $y(k-1)$ $y(k-2)$	12	0.01321	0.01200
Modèle de Sugeno (Wang et Langari, 1996)	$u(k-1)$ $y(k-1)$ $y(k-2)$	8	0.00980	0.01150

3.5 Conclusion

Dans ce chapitre nous avons décrit les trois méthodes proposées pour la conception d'un contrôleur flou par algorithmes génétiques.

Nous avons donc présenté dans un premier temps la méthode de synthèse d'un contrôleur flou de type Mamdani basée sur un algorithme génétique simple. Cette méthode consiste en l'optimisation simultanée des fonctions d'appartenance, de la base de règles et des facteurs d'échelle. Pour cela, nous avons d'abord proposé une méthode de représentation de l'ensemble des paramètres du contrôleur. Cette représentation fait intervenir trois types de

codage à savoir un codage en base n, un codage en nombre réels et un codage en binaire. Ensuite nous avons décrit l'étape d'initialisation des chromosomes en s'attardant sur la partie du chromosome correspondant aux règles floues. Enfin pour tester l'efficacité de la méthode, un problème de commande d'un système de deuxième ordre a été considéré. Les résultats obtenus montrent l'importance et l'intérêt des AG dans l'amélioration des performances du contrôleur. Néanmoins l'analyse des la base de règles, résultante à la fin du processus d'optimisation, montre la présence de règles incohérentes. Pour palier cet inconvénient, nous avons utilisé les AGH à cause de la structure de leurs chromosomes qui ont permis en plus du codage des paramètres du contrôleur, l'activation et la désactivation des règles floues. En introduisant alors un deuxième objectif (minimisation du nombre de règles) qui défavorise les solutions avec un nombre important de règles, il est possible d'éliminer les règles incohérentes comme le montre les résultats de simulations qui ont été faites sur le même système du deuxième ordre. Il faut noter que les deux méthodes d'optimisation par AGS et AGH peuvent être couteuses en temps de calcul (parcourir un nombre important de générations), dans le cas d'un contrôleur à plus de deux variables d'entrée, vu le manque de modèles de règles à utiliser pour l'initialisation des chromosomes. Nous nous sommes donc intéressés au contrôleur de Sugeno, qui facilite l'exploitation des algorithmes d'apprentissage classiques (moindres carrés, descente du gradient) pour l'ajustement de leurs paramètres, incluant ceux correspondant aux règles floues. Nous avons proposé pour ce type de contrôleurs, une méthode d'apprentissage hybride qui utilise l'algorithme de rétropropagation pour l'initialisation des chromosomes et le NSGA-II pour un l'ajustement des paramètres tout en minimisant le nombre de règles floues

Chapitre 4

Application à un bioprocédé

4.1 Introduction

Dans ce chapitre, nous allons d'abord décrire le procédé de fermentation sujet de notre application et le modèle retenu pour sa conduite. Ensuite nous allons procéder à l'application des approches de commande développées dans le chapitre précédent.
La bio-industrie a connu ces dernières années un progrès considérable. Une grande diversité de produits (alcools, protéines, enzymes, pesticides, antibiotiques,..etc.) sont aujourd'hui obtenus par fermentation.
Le but recherché dans ce domaine suivant d'un point de vue automatique est l'optimisation de la productivité, l'amélioration de la qualité des produits et la détection précoce des anomalies de fonctionnement, par la surveillance et la conduite automatique des bioréacteurs.
Les procédés biotechnologiques tels que la fermentation sont des procédés complexes à cause du caractère vivant du processus. Ils sont non linéaires, non stationnaires et caractérisés par un manque de capteurs fiables et /ou peu coûteux. Cependant, les performances des méthodes classiques de commande se dégradent d'autant plus que la dynamique du modèle utilisé s'éloigne de celle du procédé.

4.2 Procédé de fermentation

La fermentation microbienne est un procédé qui a pour finalité de faire croître une population de micro-organismes (levures, bactéries, champignons, etc.) aux dépens de certains substrats carbonés, à l'intérieur d'un bioréacteur soumis à des conditions environnementales (température, pH, aération, etc.) variable celon le résultat voulu. Chaque type de micro-organismes possède des caractéristiques liées à son matériel génétique et à ses mécanismes de régulation interne. Ainsi, la fermentation peut avoir différentes utilisation:

- *Production de micro-organismes*: l'objectif premier est la croissance de la masse cellulaire elle-même. C'est le cas des cultures visant à produire la levure de boulanger.
- *Production métabolique*: dans ce cas, c'est la production de substances organiques résultantes de l'activité métabolique qui est favorisée (alcools, antibiotiques, etc.)
- *Consommation du substrat*: ici, c'est la dégradation du substrat qui est privilégiée. On retrouvera dans cette catégorie les procédés de dépollution (traitement biologique des eaux usées)

A titre d'exemple, la fermentation éthylique sous certaines conditions d'environnement, consiste à dégrader biologiquement du glucose (substrat) par des levures (biomasse) en anaérobie, de manière à produire de l'éthanol (produit). Cette transformation est illustrée par la figure (4.1).

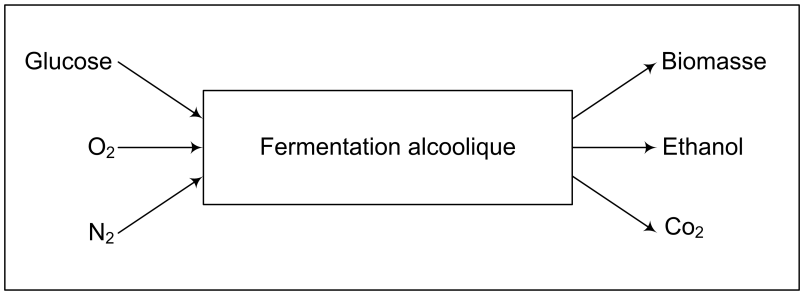

FIG. 4.1 – *Schéma simplifié d'une culture de levure*

Trois modes de fonctionnement des bioprocédés sont habituellement considérés dans la littérature (Dochain, 2001).

– le mode discontinu: le substrat est introduit initialement dans le bioréacteur contenant les souches microbiennes. Les réactions s'effectuent à volume constant et le procédé s'arrête par épuisement du substrat ou inhibition de la biomasse ou le produit.
– le mode semi-continu: le substrat est introduit dans le bioréacteur au fur à mesure des besoins des micro-organismes et le volume est donc variable.
– le mode continu: le réacteur est alimenté en substrat avec un flux constant et le milieu réactionnel est soutiré en permanence avec le même flux, le volume est donc constant dans le réacteur.

4.3 Modélisation des bioprocédés

Contrairement à une large gamme de systèmes physiques (machines électriques, organes mécaniques, systèmes hydrauliques, etc.) où il existe des lois connues depuis des siècles, la plupart des modèles en biologie reposent sur des lois empiriques. De ce fait, la modélisation des systèmes biologiques n'est pas bien maîtrisée et on trouve actuellement une bibliographie abondante de modèles mentionnés dans la littérature.
Pour donner une description physique satisfaisante du processus de fermentation, il faut combiner les équations cinétiques qui représentent le modèle microbien avec les équations de bilans de matières. Le modèle microbien qui décrit la réponse des micro-organismes au changement de son environnement peut être soit structuré soit non structuré. Dans le cas des modèles non structurés, la constante de temps des mécanismes est très faible et l'adaptation est donc très rapide. Les modèles structurés sont utilisés pour tenir compte de l'évolution de la composition et du métabolisme interne des micro-organismes qui ont des constantes de temps plus importantes.

4.3.1 Modèles non structurés

Les modèles non structurés sont basés sur l'observation des cinétiques macroscopiques au sein du bioréacteur. Ils sont dit non structurés car ils ne tiennent pas compte de la composition intracellulaire des cellules. La biomasse est représentée par une seule

variable qui est sa concentration dans le milieu réactionnel. Ces modèles sont généralement basés sur l'équation de Monod (Monod, 1942) ou des équations du même type globalisant les différentes réactions enzymatiques. Les constantes apparaissant dans ces équations sont empiriques et souvent déterminées par optimisation à partir de données expérimentales. D'autres équations décrivant la consommation du substrat et la production des produits sont nécessaires pour décrire le procédé.
Ce sont les modèles non structurés qui sont exclusivement employés, au LAAS (DISCO), dans le cas de la fermentation alcooliques (Maher, 1995; Vasilache, 2000; Kabbaj, 2004; Li, 2006).

4.3.2 Modèles structurés

Les modèles structurés ont été développés pour tenir compte de l'évolution de la structure du métabolisme interne des micro-organismes. Dans ces modèles, les composés de la biomasse sont décrits par des variables clefs qui décrivent le comportement cellulaire. Par ce moyen, l'activité microbienne devient non seulement une fonction de variables abiotiques (température, pH, etc.) qui peuvent changer avec des faibles constantes de temps mais aussi une fonction de la composition cellulaire qui dépend de l'historique de la cellule (Bideaux, 2000).
Ces modèles sont très riches en informations et peuvent être utilisés à la place des modèles non structurés dans le cas où ceux-ci sont insuffisants. Toutefois, la difficulté de la vérification expérimentale qui requiert parfois la mesure des composants intracellulaires selon différentes conditions opératoires constitue l'un des freins à l'utilisation de ce type de modèles. Dans notre travail, c'est le modèle non structuré qui sera utilisé pour décrire la dynamique du procédé de fermentation alcoolique.

4.4 Modélisation de la fermentation alcoolique

Considérons un bioréacteur infiniment mélangé au sein duquel une population de micro-organismes C se développe en consommant du substrat S et en produisant un produit P. Généralement le modèle dynamique d'un tel procédé découle directement de l'expression de bilans de matières appliquée aux différentes variables (biomasse, substrat et produit) du bioréacteur, auquel s'ajoute l'équation décrivant la variation du volume réac-

tionnel:

$$\frac{dC}{dt} = \mu C - k_m C - \frac{F_s}{V} C \tag{4.1}$$

$$\frac{dS}{dt} = -\frac{1}{y_{c/s}} \mu C - mC + \frac{F_s}{V} S_a - \frac{F_s}{V} S \tag{4.2}$$

$$\frac{dP}{dt} = -\frac{y_{p/s}}{y_{c/s}} \mu C - \frac{F_s}{V} P \tag{4.3}$$

$$\frac{dV}{dt} = F_e - F_s \tag{4.4}$$

où C, S, P représentent respectivement les concentrations en biomasse, en substrat et en produit dans le bioréacteur, S_a est la concentration en substrat dans l'alimentation, k_m le taux de mortalité naturelle des cellules,m le coefficient de maintenance de la biomasse sur le substrat, F_e le débit d'alimentation en substrat et F_s le débit de soutirage.
Le taux de croissance μ est un paramètre nécessaire pour décrire l'évolution de la biomasse. Même si ce paramètre dépend fortement de plusieurs facteurs tels que les conditions opératoires (température, pH, etc.) et le milieu opératoire (concentration en composés carbonés, azotés, phosphorés, en produits), l'expression la plus couramment utilisée est le modèle de Monod, introduit dès le début du siècle par Michaëlis-Menten pour décrire une réaction enzymatique :

$$\mu = \mu_{max} \frac{S}{K_s + S} \tag{4.5}$$

où μ_{max} est le taux maximum de croissance et K_s, appelée constante de Michaëlis-Menten, correspond à l'affinité des micro-organismes pour le substrat limitant S. Les valeurs numériques des ces paramètres ainsi que celles des rendements de conversion($y_{c/s}$ et $y_{p/s}$) sont déterminées en fonction des résultats expérimentaux.
A partir de ces équations générales, on retrouve les trois modes de fonctionnement de la fermentation décrits précédemment :

4.4.1 Mode discontinu

Au départ, tout le milieu de culture est introduit dans le fermenteur en présence des micro-organismes. Ces derniers se développent en consommant le substrat du milieu de culture et à la fin de la fermentation on extrait le produit intéressant. Cela se traduit par :

$$F_e = F_s = 0 \tag{4.6}$$

4.4.2 Mode semi-continu

La fermentation se déroule avec un volume initial V_0 inférieur au volume maximal V_m du fermenteur. Le milieu nutritif est apporté, avec un débit contrôlé en cours de la fermentation au fur et à mesure de la consommation du substrat par les micro-organismes. On arrête la fermentation lorsqu'on atteint le volume maximal puis les produits intéressants sont extraits. Cela se traduit par :

$$F_e \neq 0 \; et \; F_s = 0 \tag{4.7}$$

On utilise une variable $D(t)$, appelée taux de dilution, pour exprimer le débit par unité de volume.

$$D(t) = \frac{F_e(t)}{V(t)} \quad (V_0 \leq V(t) \leq V_m) \tag{4.8}$$

4.4.3 Mode continu

Le procédé fermentaire continu n'entraîne aucune variation du volume réactionnel. Le fermenteur est alimenté en milieu de culture et soutiré avec le même débit en permanence.

$$F_e = F_s = F_c \tag{4.9}$$

et le taux de dilution est donné par:

$$D(t) = \frac{F_c}{V} \tag{4.10}$$

avec $V(t) = V_c =$ Constante
Les variables d'environnement, sont maintenues fixes suivant le mode de fonctionnement considéré. Elles sont données par le tableau suivant:

TAB. 4.1 – *Variables d'environnement maintenues au cours de la fermentation*

Variables	Valeurs		
	Discontinu	Semi-continu	Continu
Température ($°C$)	30	30	30
pH	3,8	3,8	3,8
Volume actif (l)	30	30	30
Agitation (tr/m)	200	300	200
Aération (l/h)	3	32	3

4.4.4 Modèle dynamique

Le modèle retenu pour une fermentation alcoolique en mode continu est donné par les équations suivantes :

$$\frac{dC}{dt} = \mu C(t) - D(t)C(t) \tag{4.11}$$

$$\frac{dS}{dt} = -\frac{1}{y_{c/s}}\mu C(t) - mC(t) + D(t)S_a - D(t)S(t) \tag{4.12}$$

$$\frac{dP}{dt} = -\frac{Y_{p/s}}{y_{c/s}}\mu C(t) - D(t)P(t) \tag{4.13}$$

$$\frac{dV}{dt} = F_e - F_s = 0 \tag{4.14}$$

$$\mu(t) = \mu_{max}(t)\frac{S(t)}{K_s + S(t)} \tag{4.15}$$

Nous avons simulé le modèle pour différents taux de dilution. La résolution des équations du modèle est faite par une méthode d'intégration de Runge-Kutta d'ordre 4 sous Matlab 6.5.

4.4.5 Simulation du modèle en mode discontinu

$D(t) = 0$ caractérise une fermentation en mode batch (discontinu) ce qui n'est pas représentatif de la culture continue. Le modèle en discontinu, est simulé sous les conditions et les paramètres de synthèse suivants: $C(0) = 0.2g/l$, $S(0) = 90g/l$, $P(0) = 0.2g/l$, $\mu_{max} = 0.38h^{-1}$ $S_a = 100g/l$, $K_s = 2$, $Y_{c/s} = 0.07$ $Y_{p/s} = 0.44$

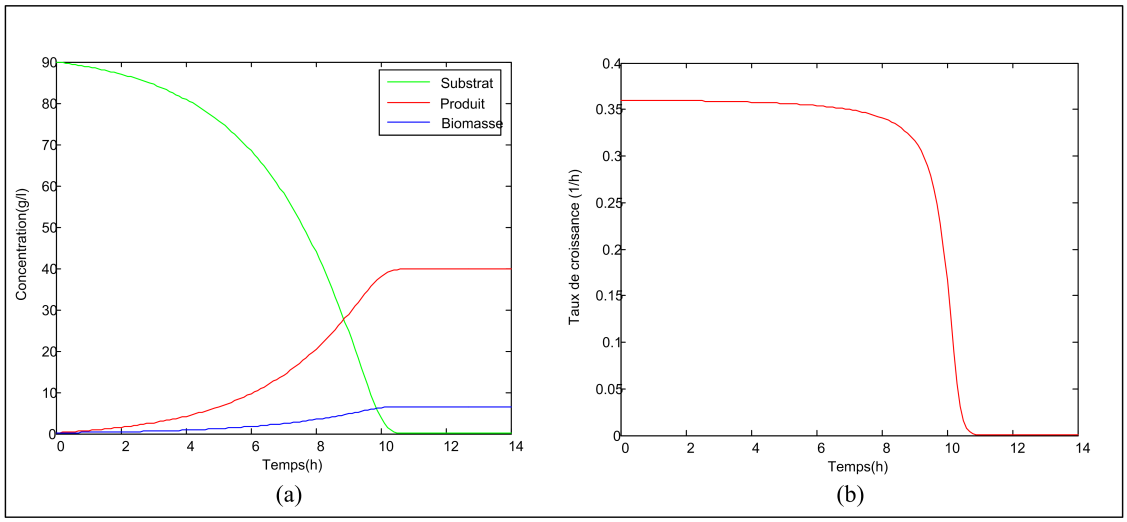

FIG. 4.2 – *Evolution des concentrations en substrat, biomasse, produit et du taux de croissance pour une fermentation en mode batch*

D'après les résultats obtenus, on constate pour de fortes concentrations en substrat, entraînent une inhibition de la croissance en biomasse et en produit. Cependant, son absence (substrat) à un effet limitant sur ces derniers (Figure 4.2-a). Parallèlement, le taux de croissance décrit par le modèle de Monod, est en fonction de la concentration en substrat. Une forte concentration de celui-ci limite le taux de croissance (Fig. 4.2-b), et l'inhibe pour des faibles concentrations.

4.4.6 Simulation du modèle en mode continu

Pour avoir un bon fonctionnement en mode continu, il faut appliquer un taux de dilution positif et inférieur à la borne supérieure de la croissance $(D(t) \prec \mu_{max})$, cela, pour éviter le phénomène de lessivage ou rinçage, qui entraîne à l'anéantissement et à l'extinction de la population cellulaire. De même, si le taux de dilution est très faible (au voisinage de zéro), le temps de résidence des cellules dans le fermenteur croît et engendre une mortalité cellulaire. En tenant compte des contraintes du taux de dilution nous avons simulé le modèle du bioprocédé en mode continu. Les conditions initiales et les paramètres de synthèse utilisés sont: $C(0) = 5g/l$, $S(0) = 5g/l$, $P(0) = 0.2g/l$, $\mu_{max} = 0.38h^{-1}$ $S_a = 100g/l$, $K_s = 2$, $Y_{c/s} = 0.07$ $Y_{p/s} = 0.44$
L'évolution des différentes variables du système est donnée par la figure (4.3).

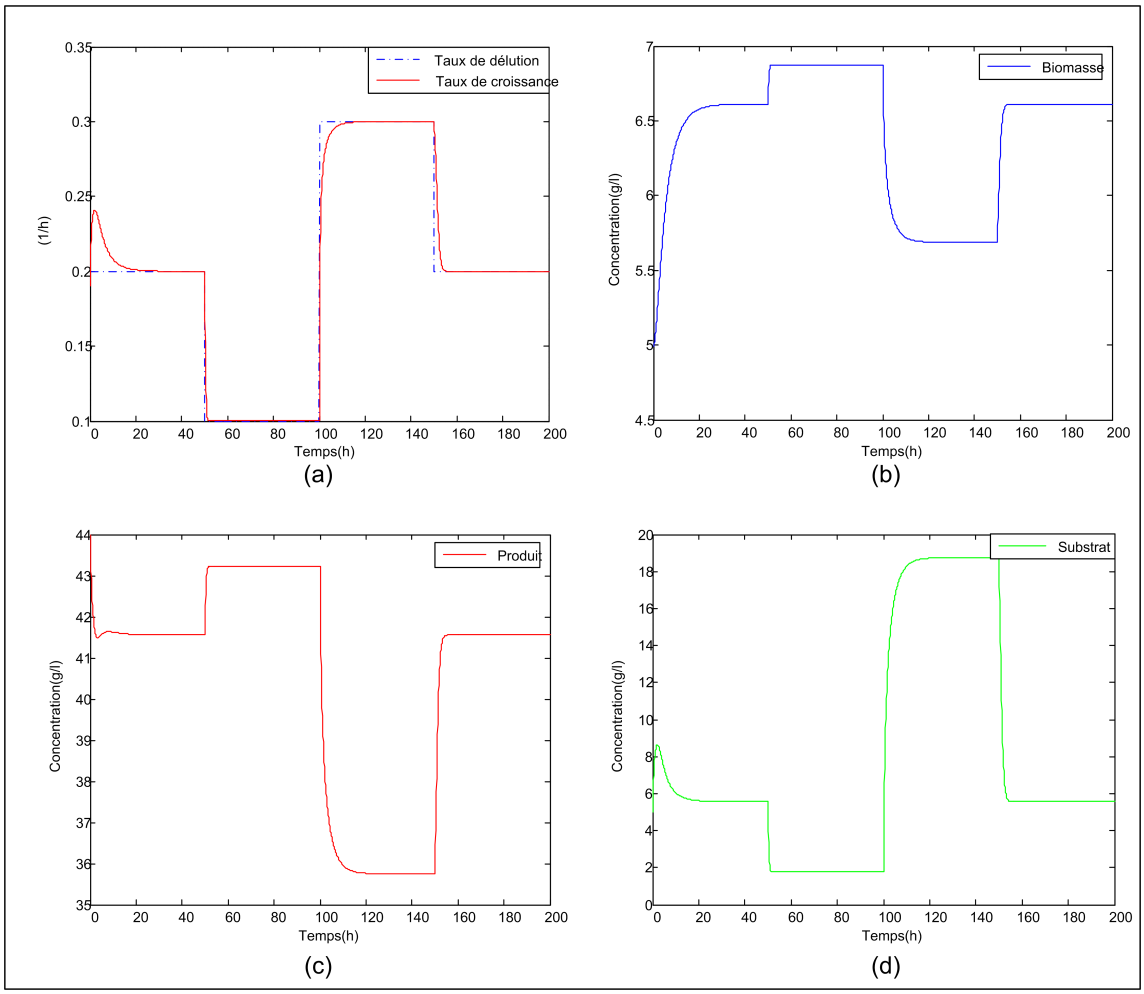

FIG. 4.3 – *Evolution des variables d'état pour une fermentation en mode continu*

Nous avons appliqué un taux de dilution avec changement de consigne donné sur la figure 4.3-a, pour étudier la cinétique des variables d'état du bioprocédé. On remarque, que pour les conditions initiales utilisées, le taux spécifique de croissance suit approximativement le taux de dilution appliqué.

Conjointement, l'évolution de la concentration en biomasse et celle du produit (Figure 4.3-b) et (Figure 4.3-c), sont disproportionnelles au taux spécifique de croissance et au taux de dilution appliqué (Figure 4.3-a). Cela se traduit, par une croissance de la biomasse et du produit au détriment (avec dégradation) du substrat (Figure 4.3-d), qui est une fonction du taux spécifique de croissance.

4.5 Application des approches intelligentes pour la conduite du bioprocédé

La commande des bioprocédés, a pour objectif la maximisation de la productivité, la minimisation des dépenses énergétiques et l'amélioration de la qualité des produits obtenus.
Les performances d'un bioprocédé fonctionnant en mode continu, peuvent êtres obtenues en deux étapes. D'abord, on définit expérimentalement des profils optimaux pour les variables d'intérêt, ensuite, on développe une commande qui se charge de réguler ces variables d'intérêt autour des profils prédéfinis.
Dans notre cas, l'objectif de la commande est de déterminer le taux de dilution $D(t)$ permettant de réguler la concentration en substrat suivant une consigne (profil) de référence $S_r(t)$.

4.5.1 Optimisation du contrôleur flou par algorithme génétique simple

Pour réguler la concentration du substrat à l'intérieur du réacteur, nous utilisons la structure de réglage illustrée ci-dessous.

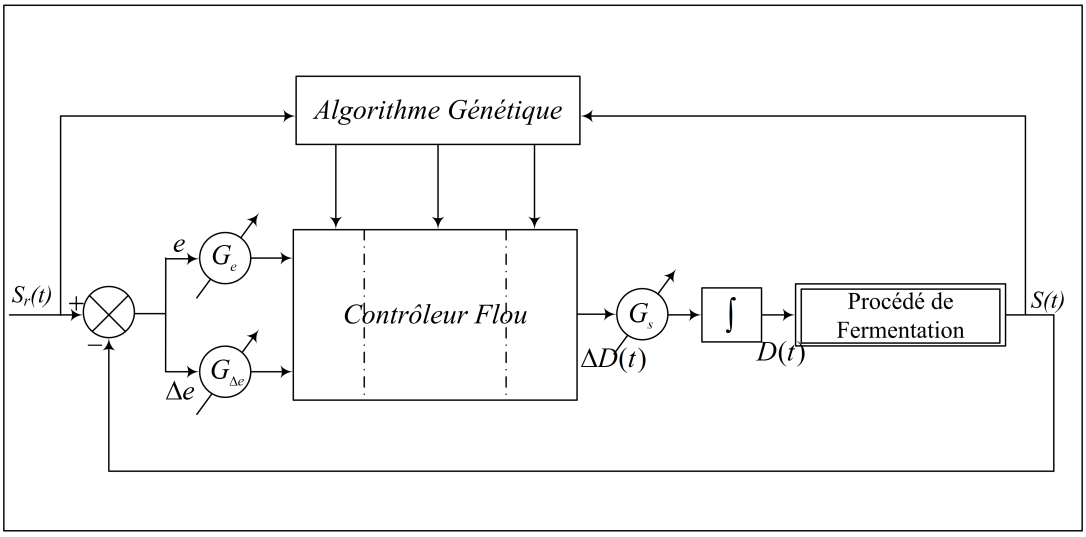

FIG. 4.4 – *Principe de la commande floue optimisée par algorithme génétique*

Le contrôleur flou reçoit dans ses deux entrées, l'erreur de la concentration en substrat et sa variation et fournit en sortie la variation du taux de dilution $\Delta D(t)$ qui sert à ajuster

le signal de commande $D(t)$:

$$u(t) = D(t) = G_s \times \Delta D(t) + u(t-1) \tag{4.16}$$

Afin d'obtenir une réponse optimale, les paramètres du contrôleur flou (fonction d'appartenance, règles floues et facteurs d'échelle) sont optimisés par un algorithme génétique simple.

4.5.1.1 Résultats de simulation

Le bioprocédé en mode continu est simulé à partir des équations (4.11-4.15) en utilisant la méthode d'intégration de Runge-Kutta d'ordre 4.
Les conditions initiales et les paramètres de synthèse utilisés sont: $C(0) = 5g/l$, $S(0) = 5g/l$, $P(0) = 44g/l$, $\mu_{max} = 0.38g/l$, $Sa = 100g/l$, $Ks = 5$, $Y_{c/s} = 0.07$, $Y_{p/s} = 0.44$.
Deux structures du contrôleur flou ont été testées.
La première correspond à un partitionnement avec 5 fonctions d'appartenance de type gaussiennes {NG: Négatif Grand, NM: Négatif Moyen, EZ: Zéro, P: Positif, PG: Positif Grand} pour chacune des variables du contrôleur et 25 règles floues.
Dans la deuxième, nous utilisons également le même type de fonctions d'appartenance mais avec un partitionnement plus fin de 7 ensembles flous {NG, NM, NP, EZ, PP, PM, PG}. La base de règles correspondante possède alors une taille de 49 règles. Nous rappelons que la taille de la base de règles demeure constante durant tout le processus d'optimisation.
Les intervalles de variation des centres des fonctions d'appartenance sont donnés dans le tableau 4.2.

TAB. 4.2 – *Intervalles de variation des centres des fonctions d'appartenance*

		Première structure		Deuxième structure	
		Min	Max	Min	Max
Ensembles Flous	NG	-1	-0,5	-1	-0,70
	NM	-0,35	-0,05	-0,69	-0,40
	NP	-	-	-0,39	-0,05
	EZ	-0,04	0,04	-0,04	0,04
	PP	-	-	0,05	0,39
	PM	0,05	0,35	0,40	0,69
	PG	0,50	1	0,70	1

Ils sont déterminés de manière à obtenir à la fin du processus d'optimisation des partitions lisibles dont les ensembles flous interprètent réellement les termes linguistiques qui leur sont associés. Le niveau de chevauchement entre deux fonctions voisines est défini dans l'intervalle $[0.05, 0.6]$

Les intervalles de variations des facteurs d'échelle en entrée du contrôleur flou sont obtenus à partir du profil de référence. En effet la connaissance de la variation maximale du profil de référence (voir figure 4.7) et de la variation de la concentration en substrat conduit à la définition des intervalles suivant:

TAB. 4.3 – *Intervalles de variation des facteurs d'échelle*

	Facteurs d'échelle		
	G_e	$G_{\Delta e}$	$G_{\Delta u}$
Min	1	3	0,05
Max	2	10	0,15

Quant à l'intervalle de variation du gain $G_{\Delta u}$ (tableau), il est choisi de manière à faire un compromis entre rapidité et stabilité.

a.Fonction objectif

La mise au point du contrôleur flou est réalisée en minimisant un critère qui comprend l'intégrale de l'erreur absolue (IAE) et l'intégrale de l'erreur absolue temporelle (ITAE):

$$Ob = IAE + ITAE \tag{4.17}$$

Par cette combinaison, il est possible de minimiser l'erreur durant la période transitoire (IAE), et en régime permanent par l'emploi de (ITAE).

b.Paramètres de l'AG

Comme pour beaucoup d'algorithmes, les paramètres de l'algorithme génétique doivent être soigneusement choisis pour obtenir de bons résultats. Dans notre cas, nous avons utilisé les paramètres suivants:

- Longueur du chromosome:
  1. Première structure: 76 (27 pour les fonctions d'appartenance, 25 pour les règles floues et 24 (3*8) pour les facteurs d'échelle)
  2. Deuxième structure: 112 (39 pour les fonctions d'appartenance, 49 pour les règles floues et 24 pour les facteurs d'échelle)
- Probabilité de croisement: 0,85

- Probabilité de mutation: 0,01
- Taille de la population: 30
- Maximum de générations: 500
- Méthode de sélection: tournoi combiné aves une approche élitiste.

Première structure

Les résultats obtenus pour cette première structure sont portés sur les figures 4.5-4.8.

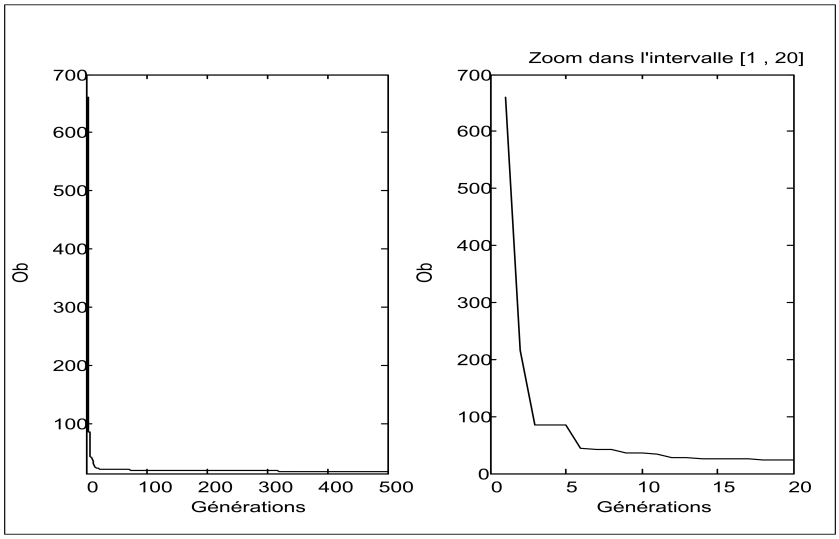

FIG. 4.5 – *Evolution de l'objectif (Ob) à travers les générations-structure 5*5*5*

Nous remarquons que les résultats diffèrent de génération en génération. La valeur du critère de performance Ob est rapidement améliorée au cours des premières générations, et est ramenée de 661,01 à 23,65 au bout de 20 générations seulement. Elle atteint sa valeur optimale ($Ob = 17{,}85$) après 463 générations puisque c'est toujours cette valeur qui revient jusqu'à la fin du processus d'optimisation.
Les figure 4.6 donnant l'évolution des paramètres des fonctions d'appartenance justifie bien le rôle des opérateurs génétiques (sélection, croisement et mutation) en exploitant des points différents dans l'espace de recherche des paramètres. Il est important de noter que les variations des paramètres sont relativement importantes au cours des premières générations par rapport au reste des générations.

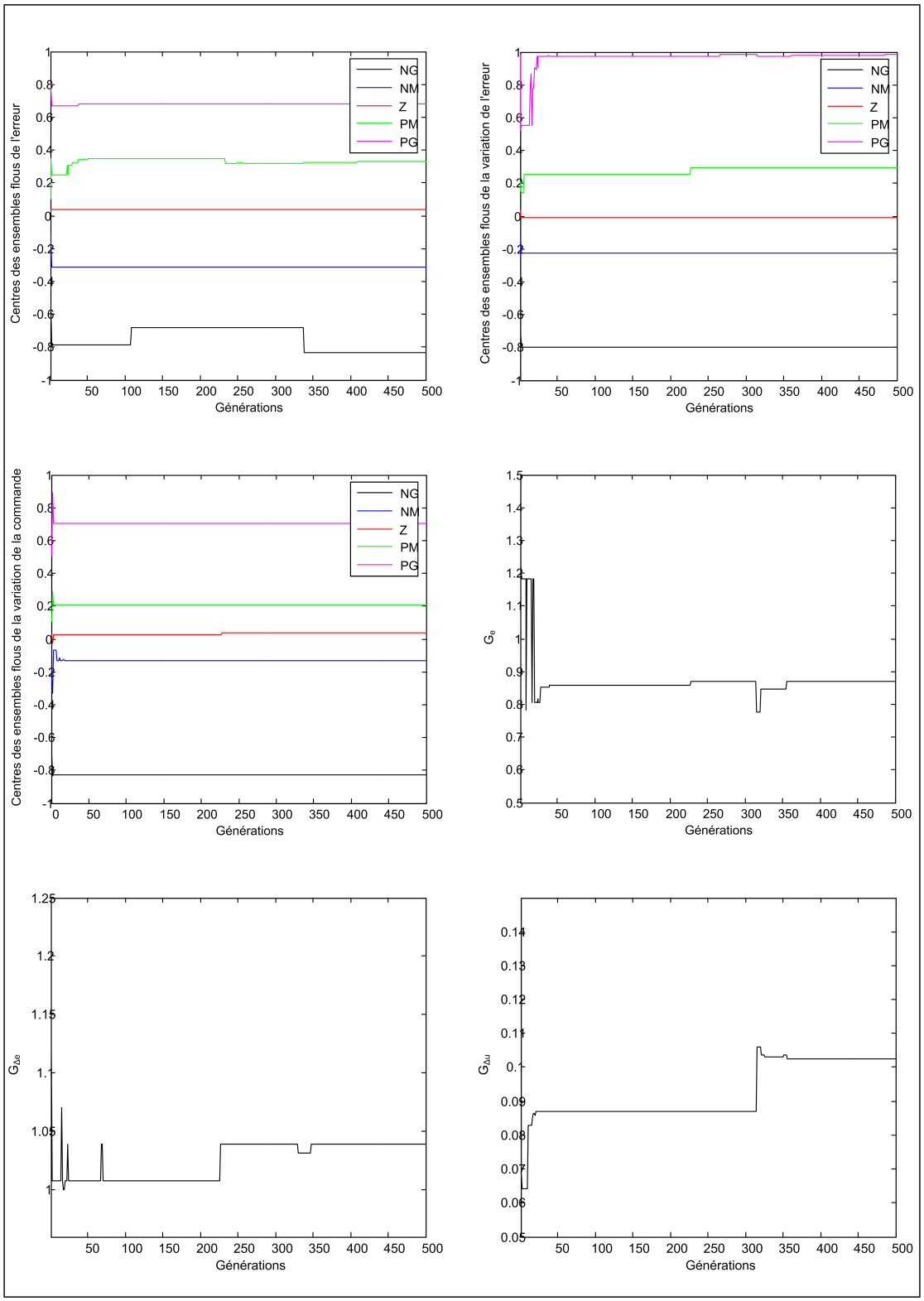

FIG. 4.6 – *Evolution des paramètres du contrôleur flou à travers les générations-structure 5*5*5*

Pour mieux illustrer l'évolution des résultats, nous portons dans les figures ci-après l'évolution de la concentration en substrat, du signal de commande et des paramètres du contrôleur, obtenus lors de la première et de la dernière génération (1, 500).

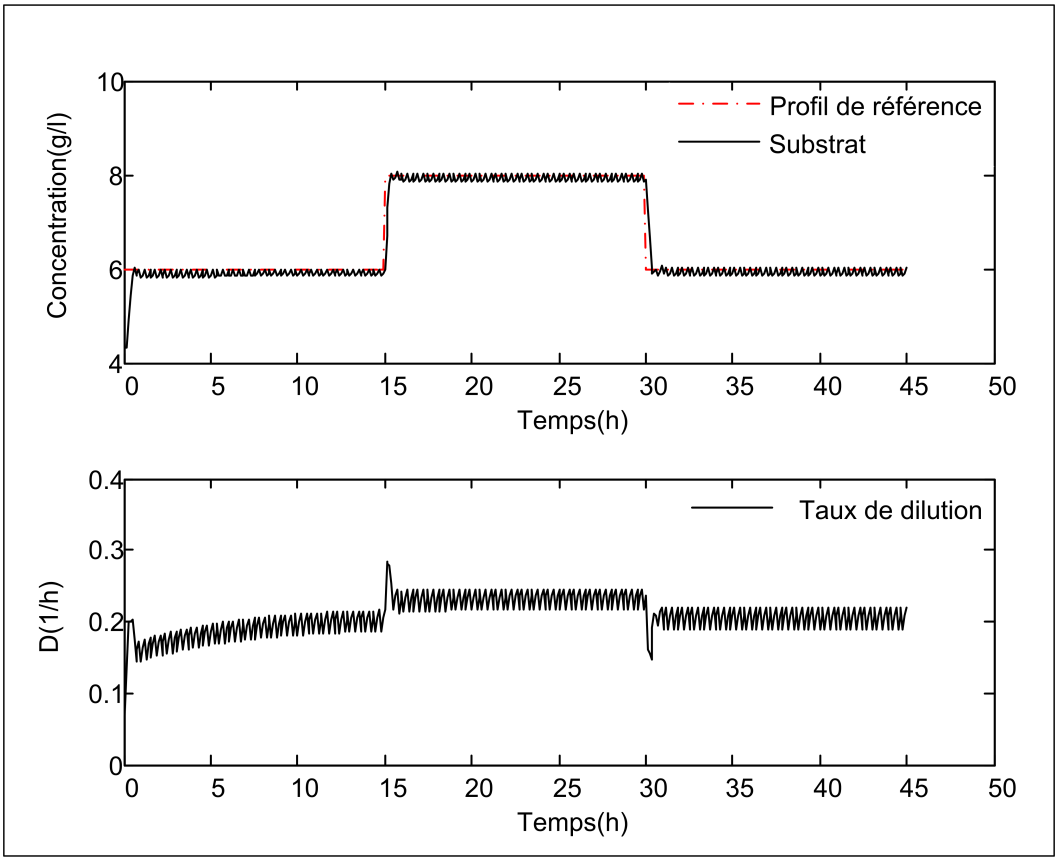

FIG. 4.7 – *Evolution de la concentration en Substrat pour le meilleur chromosome de la première génération-structure 5*5*5*

La figure (4.7) présente l'évolution de la concentration en substrat pendant 45 heures de fermentation continue, correspondant au meilleur chromosome de la première génération. Nous remarquons la présence d'oscillations qui s'étalent sur des intervalles de temps très importants.

TAB. 4.4 – *Facteur d'échelle correspondants au meilleur chromosome de la première génération-structure 5*5*5*

	Facteurs d'échelle		
	G_e	$G_{\Delta e}$	$G_{\Delta u}$
Valeur	1,6059	1,2431	0,1000

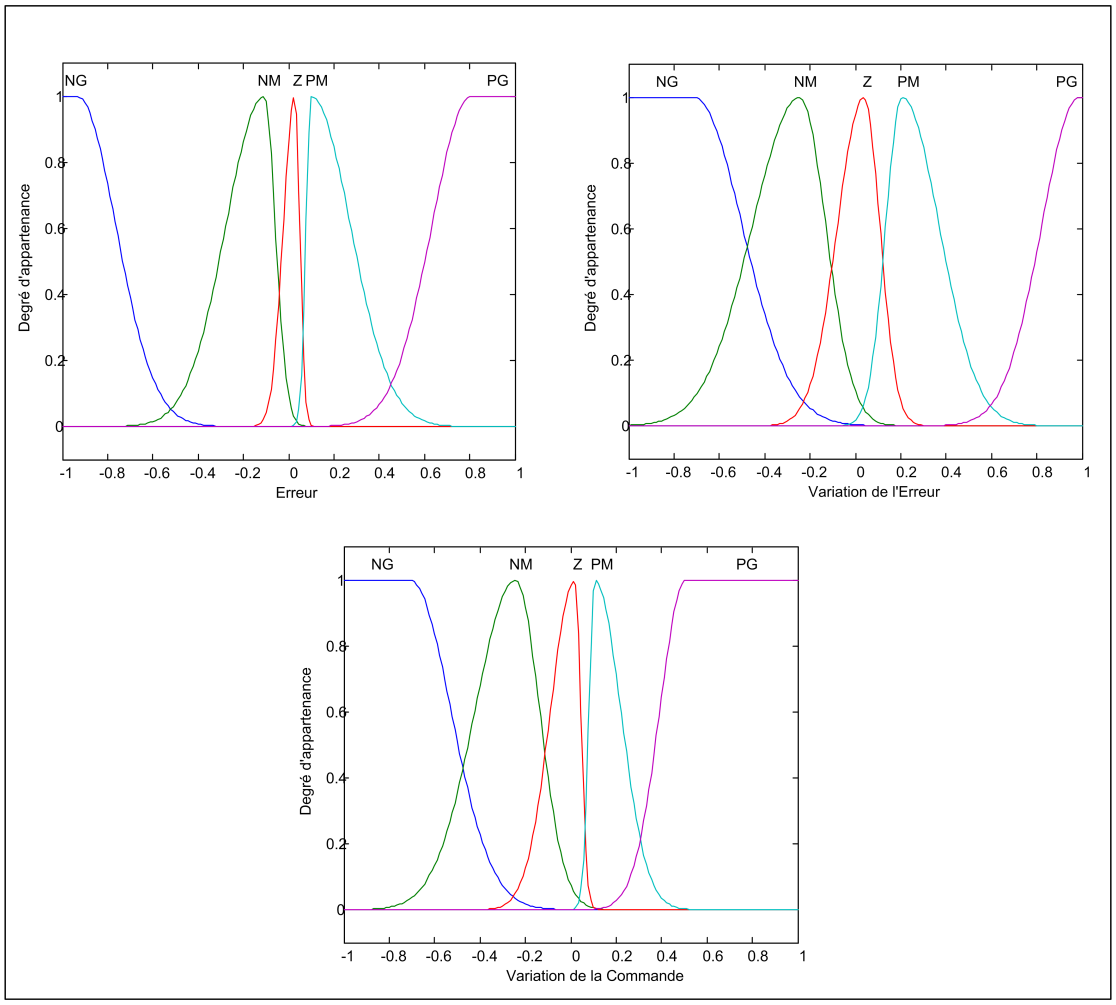

FIG. 4.8 – *Fonctions d'appartenance correspondantes au meilleur chromosome de la première génération-structure 5*5*5*

TAB. 4.5 – *Base de règles floues correspondante au meilleur chromosome de la première génération-structure 5*5*5*

Δu / Δe	NG	NM	EZ	PM	PG
NG	1	1	1	2	3
NP	1	2	2	3	4
e EZ	1	2	3	4	5
PM	2	3	4	4	5
PG	3	4	5	5	5

Le résultat de la dernière génération est nettement meilleur. On peut constater la disparation des oscillations à l'exception de quelques-unes qui apparaissent aux instants des changements de consignes. Ces dernières sont rapidement atténuées par une action optimale du contrôleur flou et les paramètres (figure 4.10 , tableaux 4.6 et 4.7) sont très différents de ceux obtenus au début du processus d'optimisation.

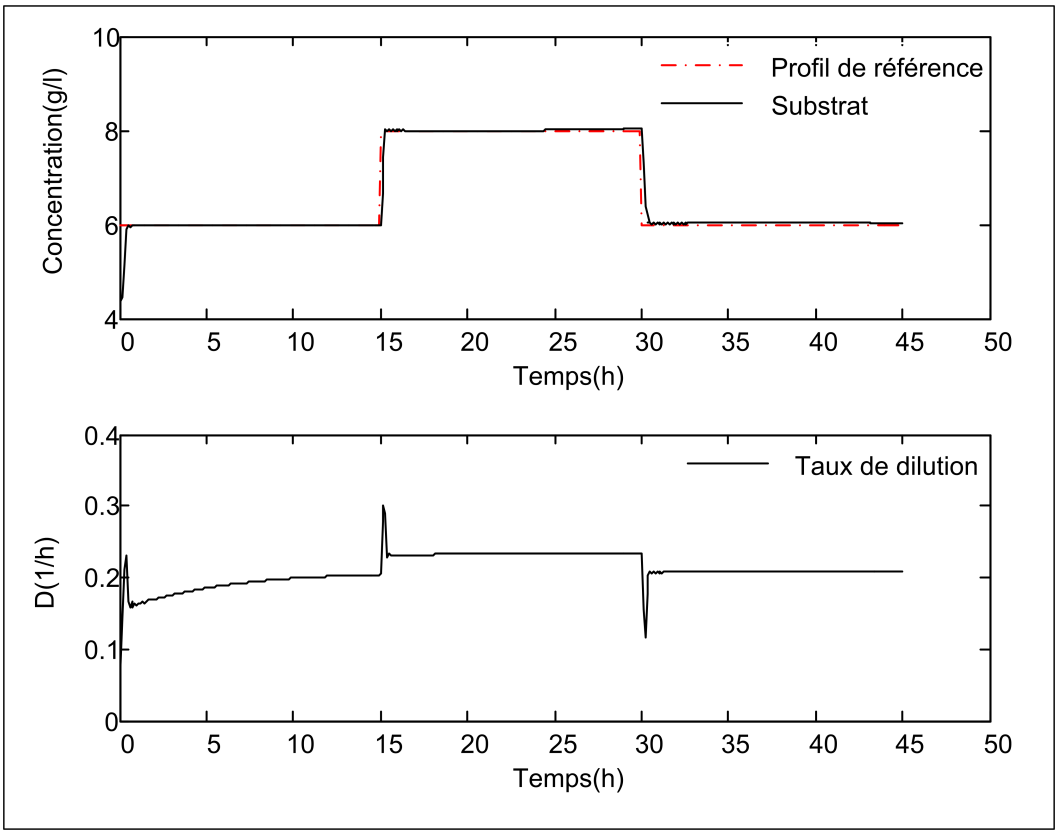

FIG. 4.9 – *Evolution de la concentration en Substrat et le taux de dilution pour la dernière génération-structure 5*5*5*

TAB. 4.6 – *Facteur d'échelle correspondants au meilleur chromosome de la dernière génération-structure 5*5*5*

	Facteurs d'échelle		
	G_e	$G_{\Delta e}$	$G_{\Delta u}$
Valeur	0,8706	1,0392	0,1024

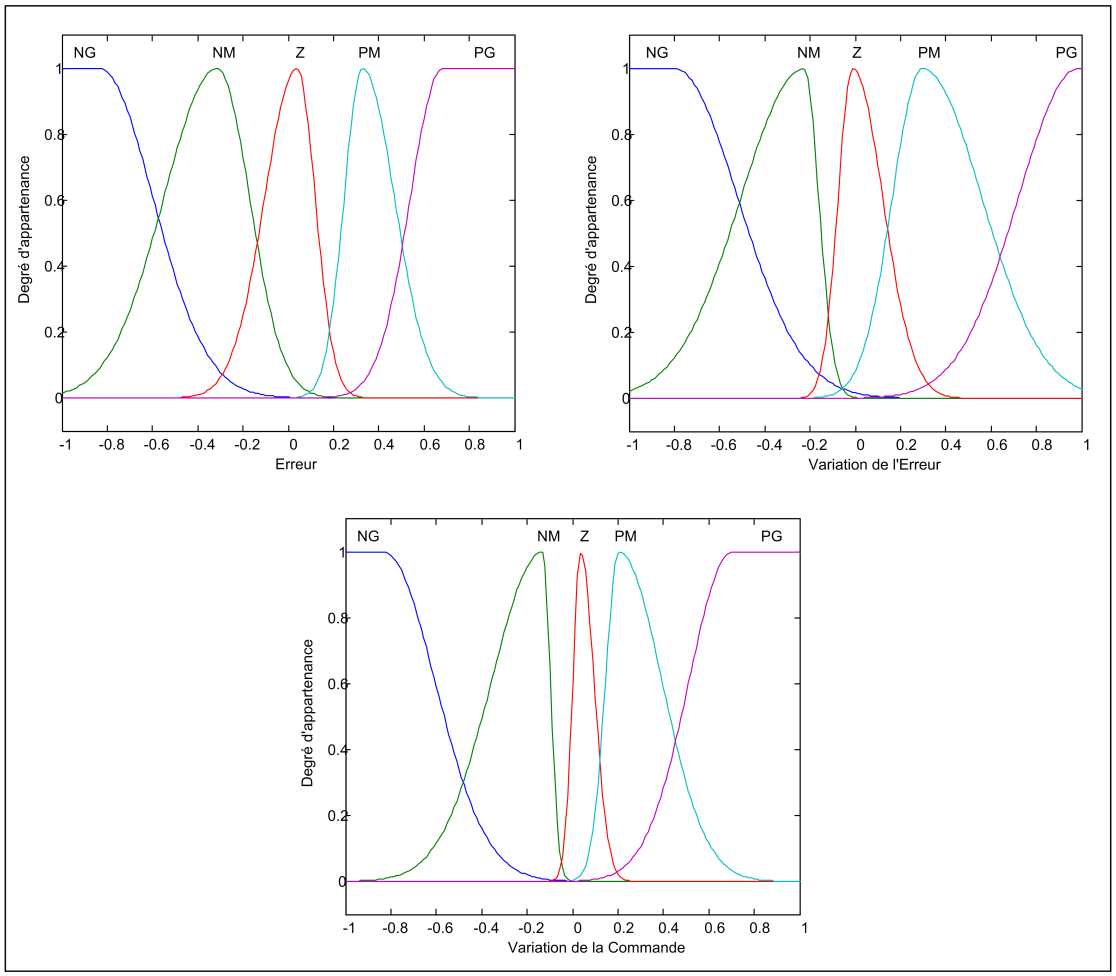

FIG. 4.10 – *Fonctions d'appartenance correspondantes au meilleur chromosome de la dernière génération-structure 5*5*5*

TAB. 4.7 – *Base de règles floues correspondante au meilleur chromosome de la dernière génération-structure 5*5*5*

Δu / Δe	NG	NM	EZ	PM	PG
NG	1	1	1	1	3
N	2	2	2	4	5
EZ	1	3	3	4	2
P	2	4	1	5	4
PG	3	5	5	5	5

(e)

Deuxième structure

Pour cette partie et par manque d'espace, nous représentons uniquement l'évolution du critère de performances à travers les générations, les paramètres du contrôleur flou et la sortie du procédé correspondants au meilleur chromosome de la dernière génération.

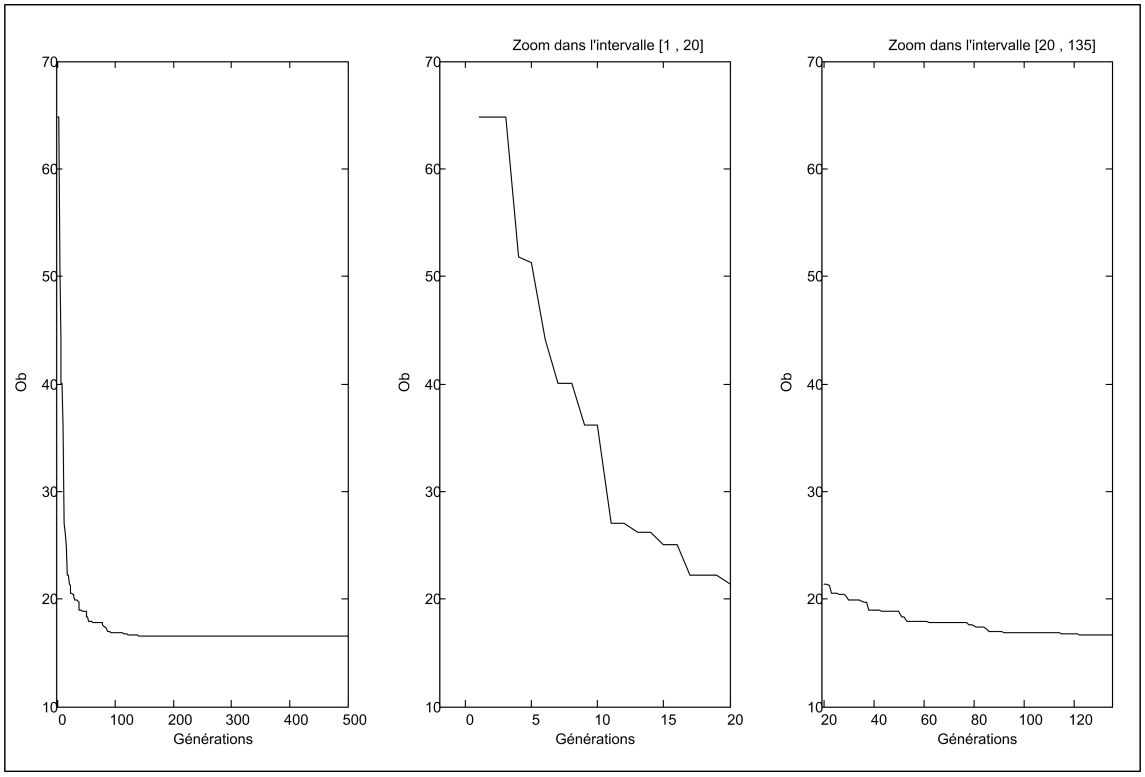

FIG. 4.11 – *Evolution de l'objectif* (Ob) *à travers les générations-structure 7*7*7*

La figure (4.11) décrit l'évolution de l'objectif Ob en fonction du nombre de générations parcourues. De même que pour la première structure, on remarque que le critère de performance diminue rapidement durant les premières générations ([0, 20] et [20,135]). Il continue à diminuer lentement jusqu'à atteindre la valeur 16,56 à la 200ème génération. A partir de cette génération, on enregistre de très faibles variations de la fonction objectif (Ob) soit une différence de 0,02 jusqu'à la 500ème génération.
Les concentrations en substrat et du taux de dilution obtenus pour la 500ème génération sont donnés sur la figure 4.12.

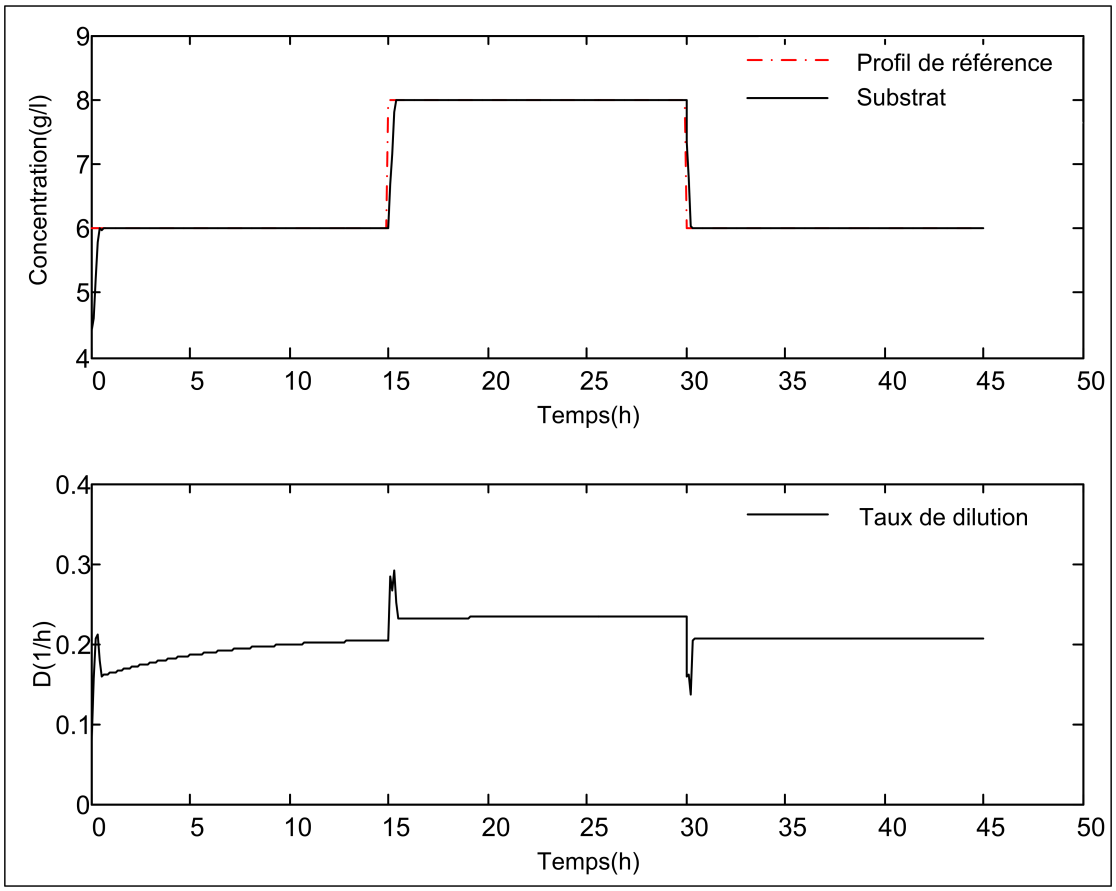

FIG. 4.12 – *Evolution de la concentration en Substrat et le taux de dilution pour la dernière génération-structure 7*7*7*

Cette variable suit bien le profil désiré, soit une erreur nulle en régime permanent et un temps de réponse très réduit sans qu'il y ait de dépassements et d'oscillations aux instants de changements de consigne.
Les facteurs d'échelle, les paramètres des fonctions d'appartenance et la base de règles correspondants sont respectivement donnés par le tableau 4.8, la figure 4.13 et le tableau 4.9.

TAB. 4.8 – *Facteur d'échelle correspondants au meilleur chromosome de la dernière génération-structure 7*7*7*

	Facteurs d'échelle		
	G_e	$G_{\Delta e}$	$G_{\Delta u}$
Valeur	1,0529	1,0078	0,0871

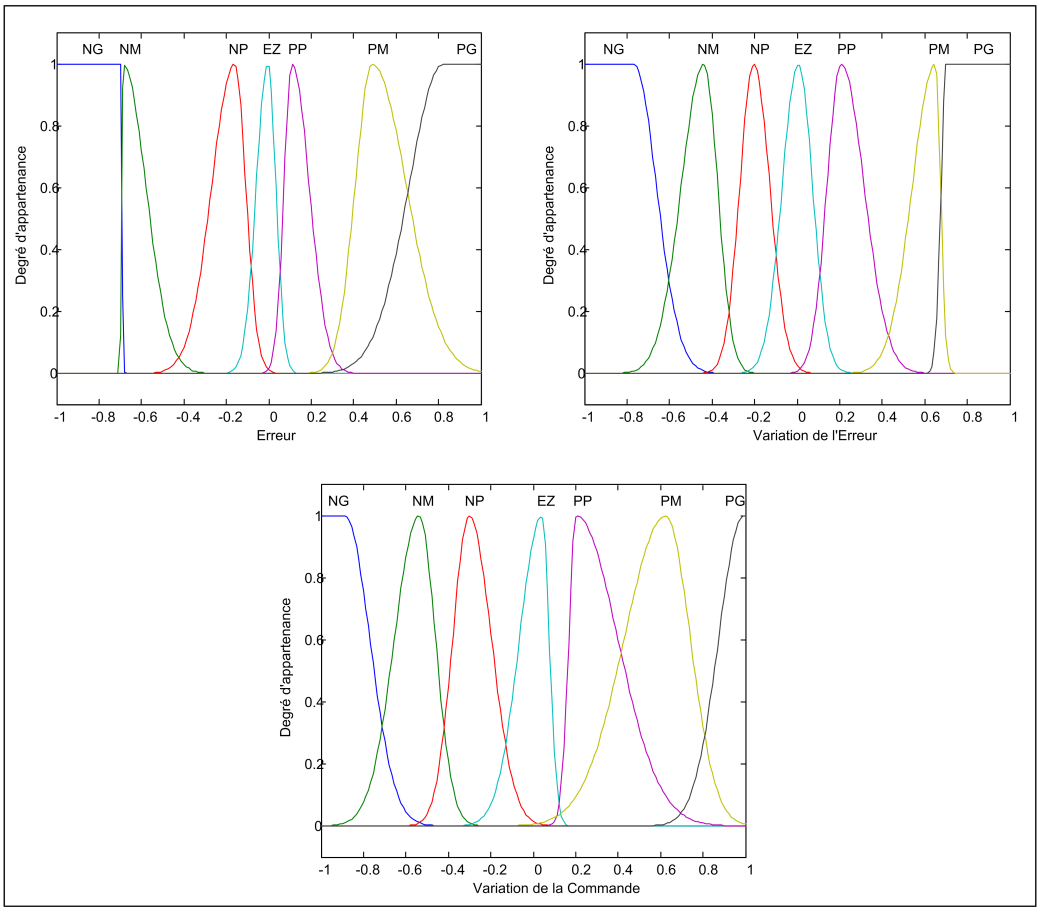

FIG. 4.13 – *Fonctions d'appartenance correspondantes au meilleur chromosome de la dernière génération-structure 7*7*7*

TAB. 4.9 – *Base de règles floues correspondante au meilleur chromosome de la dernière génération-structure 7*7*7*

Δu / e \ Δe	NG	NM	NP	Z	PP	PM	PG
NG	1	3	2	3	2	3	5
NM	4	1	3	1	2	4	4
NP	2	2	4	3	7	4	6
Z	2	2	3	4	5	5	7
PP	2	3	4	5	4	7	6
PM	5	4	6	5	6	5	7
PG	3	5	6	7	3	7	7

Notons bien qu'une attention particulière doit être portée sur les bases de règles obtenues, pour les deux structures, en fin du processus d'optimisation. Celles-ci présentent des règles incohérentes (cellules noires des tableaux 4.7 et 4.9) qui peuvent avoir un effet négatif sur la réponse du contrôleur. Pour l'AGS, la présence de ces règles ne peut être ressenti tant que le critère numérique à base duquel les individus sont évalués, n'est pas détérioré. Pour palier à cet inconvénient, nous introduisons donc un deuxième objectif qui est la minimisation du nombre de règles floues. Ce dernier permet de privilégier, durant le processus d'optimisation, les contrôleurs flous avec un minimum de règles ce qui permettra d'éliminer les règles inutiles.

4.5.2 Optimisation du contrôleur flou par algorithme génétique hiérarchisé

La stratégie de commande reste la même (fig.4.4), seule la méthode d'optimisation des paramètres du contrôleur flou diffère. On utilise cette fois ci un AGH à la place d'un AGS ce qui permet d'une part de coder par le biais des gènes des paramètres, les paramètres des fonctions d'appartenance, les facteurs d'échelle et les conclusions des règles floues, comme s'il s'agissait d'un algorithme génétique simple et d'autre part l'activation ou la désactivation des règles floues par le biais des gènes de contrôle, chose qui diffère d'un AG simple. En effet, dans les deux cas précédents (première et deuxième structure) toutes les règles floues et parmi elles les règles incohérentes, sont restées actives.

4.5.2.1 Objectifs de la stratégie de commande

Deux objectifs sont considéré le premier est l'amélioration des performances du système de commande et le deuxième concerne la réduction de la complexité du contrôleur :

$$Ob_1 = IAE + ITAE \tag{4.18}$$

$$Ob_2 = \sum_{i=1}^{49} z^i \tag{4.19}$$

Comme il s'agit d'un problème d'optimisation multi-objectif (Ob_1 et Ob_2), nous avons opté pour le NSGA-II comme algorithme de résolution (méthode de sélection).

4.5.2.2 Résultats de simulation

Pou vérifier les performances de la méthode d'optimisation, nous utilisons un contrôleur flou configuré selon la deuxième structure déjà présentée précédemment.
Les intervalles de variation de l'ensemble des paramètres du contrôleur restent les mêmes. On garde également le même profil de consigne.
Les paramètres du NSGA-II sont résumés ainsi:

- Longueur du chromosome: 161(49 pour les gènes de contrôle, 39 pour les fonctions d'appartenance, 49 pour les règles floues et 24 (3*8) pour les facteurs d'échelle)
- probabilité de croisement des gènes de paramètres : 0.85
- Probabilité de croisement des gènes de contrôle : 0.45
- Probabilité de mutation des gènes de paramètres : 0.01
- Probabilité de mutation des gènes de contrôle: 0.005.
- Taille de la population: 40
- Maximum de génération : 1000

La figure (4.14) donne la répartition des chromosomes de la première génération dans l'espace des objectifs.

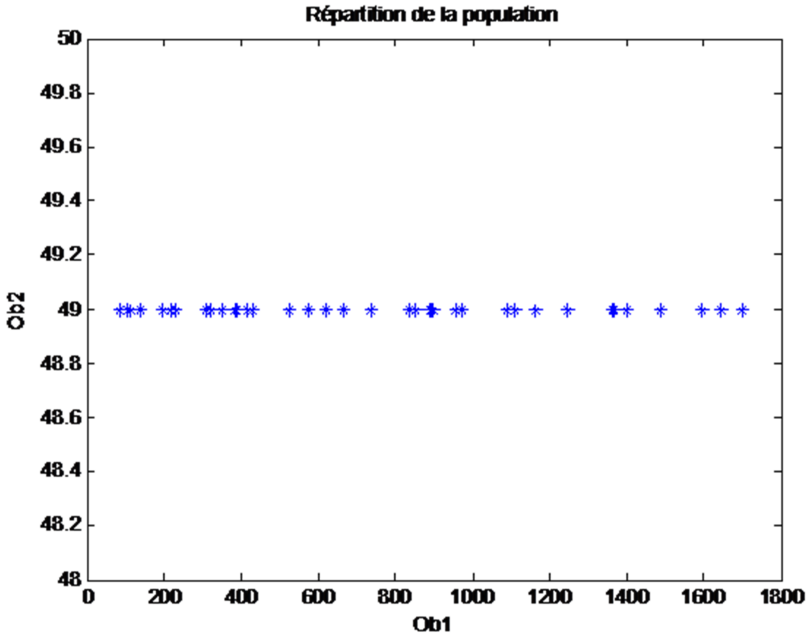

FIG. 4.14 – *Répartition de solutions de la première génération*

On constate que la valeur de l'objectif Ob_2 est maintenue à 49 pour tous les chromosomes. Cela permet de considérer, pour chaque solution, la totalité des règles floues durant la première génération (tous les gènes de contrôle sont initialisés à 1). Les différentes valeurs de l'objectif Ob_1 sont dues à l'initialisation aléatoire des gènes de paramètres correspondants aux fonctions d'appartenance et aux facteurs d'échelle.
De génération en génération, la valeur d'Ob_1 est considérablement améliorée, toute en minimisant le nombre de règles floues actives.
Nous représentons ci-après la répartition des chromosomes obtenue à la fin du processus d'optimisation (figure 4.15). Il est important de noter les améliorations apportées simultanément aux deux objectifs Ob_1 et Ob_2.
Nous avons pour la plupart des solutions, notamment celles appartenant au premier front de Pareto, un résultat meilleur que celui correspondant au meilleur chromosome de la population initiale.

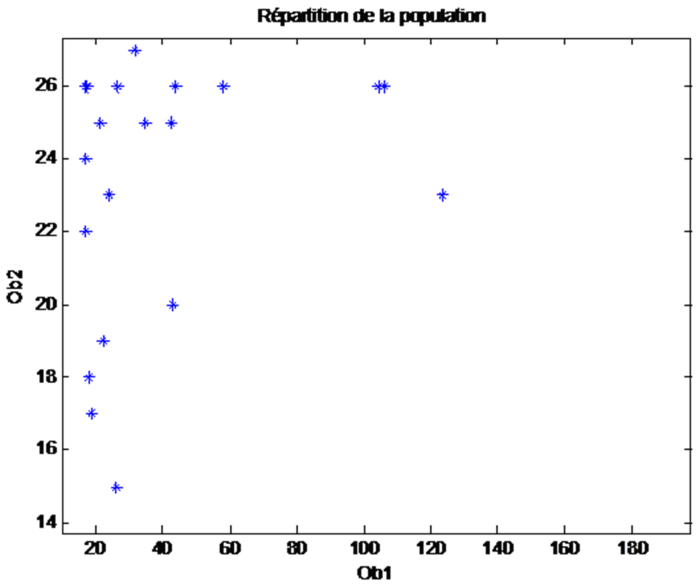

FIG. 4.15 – *Répartition de solutions de la dernière génération*

La figure 4.16 donne l'évolution de la concentration en substrat et le signal de commande obtenus pour une solution de test choisie du front de Paréto.
Pour cette solution uniquement 26 règles sont considérées (tableau 4.10), les autres règles (23 règles) sont annulées par les gènes de contrôle.

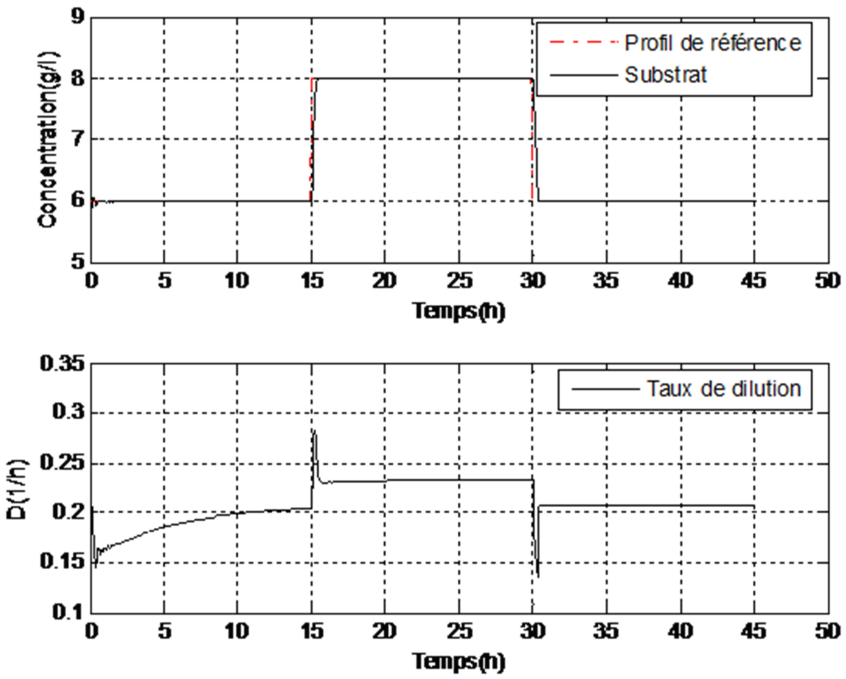

FIG. 4.16 – *Evolution de la concentration en Substrat et le taux de dilution*

On a un bon suivi du profil de référence et des performances très comparables avec celles obtenues par algorithme génétique simple (figures 4.9 et 4.12) en considérant toutes les 49 règles floues.

TAB. 4.10 – *Base de règles correspondant à la solution de test*

Δu / e	NG	NM	NP	EZ	PP	PM	PG
NG	1	1	1		1		2
NM							6
NP		2		4	5	6	7
EZ		2		4	5	6	7
PP	3	3	3	5			
PM	2		6				
PG	6		7		7		7

4.5.3 Optimisation du contrôleur flou de type Sugeno

Le schéma de principe utilisé pour l'optimisation du contrôleur flou de type Sugeno est représenté dans la figure 4.17

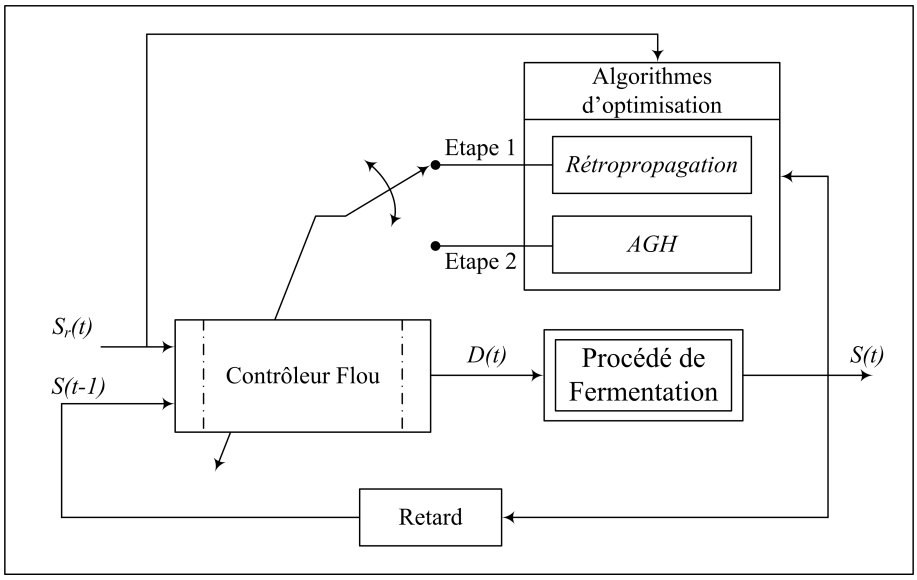

FIG. 4.17 – *Schéma de principe de l'optimisation d'un contrôleur flou de type Sugeno*

On distingue deux étapes d'optimisation dans ce schéma: la rétropropagation qui réalise un premier ajustement des paramètres (première étape) et un AGH permettant de modifier encore une deuxième fois les paramètres et de minimiser le nombre de règles floues (deuxième étape).
La séquence de référence désirée $S_r(t)$ est générée par le modèle

$$\frac{dS_r(t)}{dt} + 0.33S_r(t) = 0.33S^*(t) \tag{4.20}$$

avec $S^*(t)$ est la consigne de la concentration en substrat et dont les valeurs sont prises comme suit:

$$S^*(t) = \begin{cases} 6g/l & \text{pour} \quad 0h < t \leq 22{,}5h \\ 8g/l & \text{pour} \quad 22{,}5h < t \leq 45h \\ 6g/l & \text{pour} \quad 22{,}5h < t \leq 67{,}5h \end{cases}$$

Le contrôleur flou reçoit par ses deux entrées la concentration désirée en substrat $S_r(t)$ et la valeur passée de la concentration en substrat $S(t-1)$. Il fournit en sortie le taux de dilution $D(t)$ qui alimente le bioréacteur avec une période d'échantillonnage de 9 minutes

($0{,}15h$).
Au niveau de la structure interne, chaque variable du contrôleur est partitionnée en 7 ensembles flous (A_1, A_2, .., A_7 pour $S_r(t)$ et B_1, B_2, .., B_7 pour $S(t-1)$) et définie sur l'univers de discours [4.8, 8,2]. La table de règles associée, comporte alors 49 règles de la forme:

$$R_i : \text{Si } S_r(t) \text{ est } A_i \text{ et } S(t-1) \text{ est } B_i \text{ Alors } D(t) = a_0^i + a_1^i.S_r(t) + a_2^i.S(t-1) \quad (4.21)$$

avec a_0^i, a_1^i et a_2^i sont les paramètres relatifs à la règle R_i.

4.5.3.1 Première étape: optimisation par rétropropagation

L'adaptation des paramètres du contrôleur est réalisée en minimisant à chaque instant d'échantillonnage le critère quadratique:

$$J(t) = \frac{1}{2}(S(t) - S_r(t))^2 \quad (4.22)$$

à l'aide d'une technique de descente du gradient:

$$\begin{cases} p(t+1) & = p(t) + \Delta p(t) \\ \Delta p(t) & = -\eta \dfrac{\partial J(t)}{\partial p(t)} = -\eta \dfrac{\partial J(t)}{\partial S(t)} \dfrac{\partial S(t)}{\partial D(t)} \dfrac{\partial D(t)}{\partial p(t)} \end{cases} \quad (4.23)$$

avec $\dfrac{\partial J(t)}{\partial S(t)} = S(t) - S_r(t)$ et $\dfrac{\partial S(t)}{\partial D(t)}$ est grossièrement approximé par: $signe(\dfrac{S(t) - S(t-1)}{D(t) - D(t-1)})$.
Le terme $\dfrac{\partial D(t)}{\partial p(t)}$ se calcule selon l'algorithme de rétropropagation décrit dans la section 4 du chapitre 3.
Comme notre objectif, en plus de la minimisation du critère numérique $J(t)$, est d'obtenir un contrôleur flou avec des partitionnements interprétables, l'adaptation par rétropropagation est limitée uniquement aux paramètres a_j^i relatifs aux conclusions des règles floues. Les centres est les variances des fonctions d'appartenance restent donc constants durant cette première étape d'optimisation. Ils sont initialisés de manière à obtenir des ensembles flous uniformément répartis sur leur univers de discours ([4.8, 8.2]).
Le suivi de la trajectoire de référence à l'issue de cette première étape est déjà satisfaisant et est représenté dans la figure 4.18

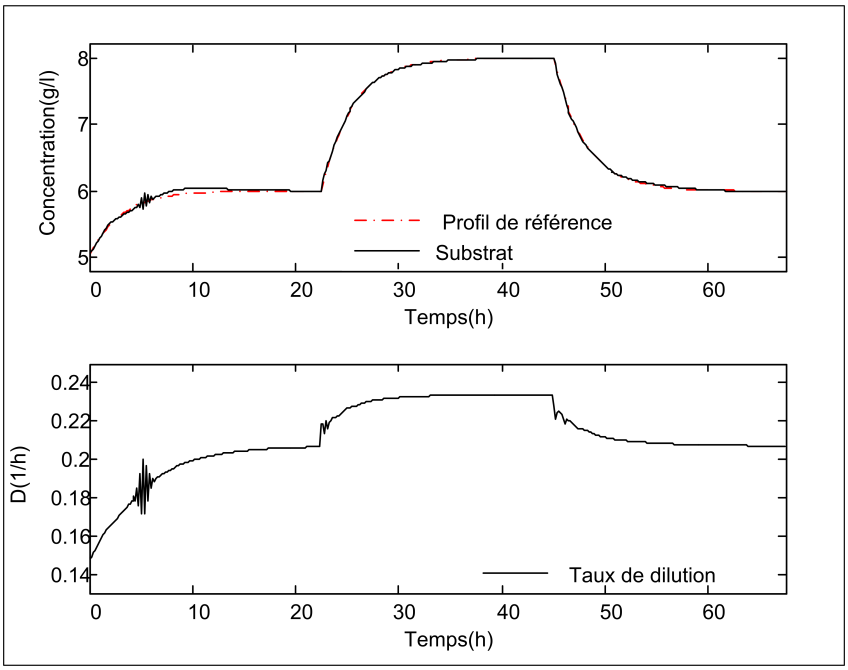

FIG. 4.18 – *Evolution de la concentration en Substrat et le taux de dilution après la première phase d'optimisation*

Les fonctions d'appartenance et la base de règles du contrôleur flou sont données respectivement par la figure 4.19 et le tableau 4.11

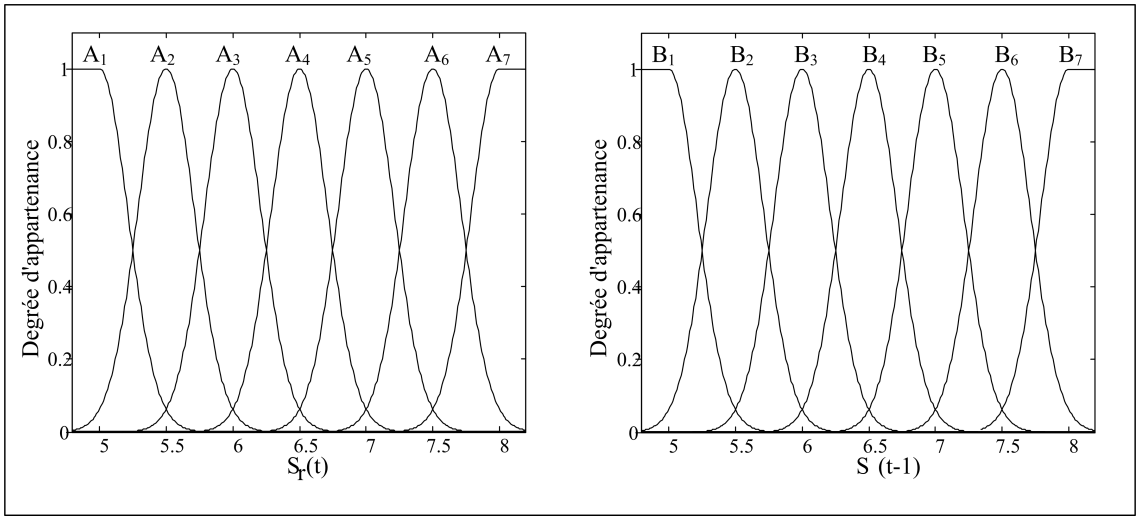

FIG. 4.19 – *Fonctions d'appartenance initiales*

TAB. 4.11 – *Base de règles obtenue après la rétropropagation*

Règle	Prémisses des règles floues		Paramètres relatifs aux règles floues			Règle	Prémisses des règles floues		Paramètres relatifs aux règles floues		
	$S_r(t)$	$S(t-1)$	a_0^i	a_1^i	a_2^i		$S_r(t)$	$S(t-1)$	a_0^i	a_1^i	a_2^i
1	A_1	B_1	0.0039	0.0205	0.0085	26	A_4	B_5	0.0070	0.0129	0.0123
2	A_1	B_2	0.0052	0.0021	0.0099	27	A_4	B_6	0.0029	0.0052	0.0098
3	A_1	B_3	0.0008	0.0084	0.0089	28	A_4	B_7	0.0019	0.0061	0.0102
4	A_1	B_4	0.0035	0.0057	0.0026	29	A_5	B_1	0.0084	0.0473	0.0316
5	A_1	B_5	0.0069	0.0040	0.0090	30	A_5	B_2	0.0068	0.0268	0.0203
6	A_1	B_6	0.0001	0.0038	0.0100	31	A_5	B_3	0.0133	0.0304	0.0319
7	A_1	B_7	0.0008	0.0036	0.0082	32	A_5	B_4	0.0041	0.0208	0.0163
8	A_2	B_1	0.0070	0.0246	0.0117	33	A_5	B_5	0.0057	0.0162	0.0148
9	A_2	B_2	0.0090	0.0109	0.0187	34	A_5	B_6	0.0011	0.0110	0.0125
10	A_2	B_3	0.0072	0.0036	0.0017	35	A_5	B_7	0.0083	0.0078	0.0126
11	A_2	B_4	0.0068	0.0038	0.0096	36	A_6	B_1	0.0081	0.0458	0.0344
12	A_2	B_5	0.0064	0.0008	0.0072	37	A_6	B_2	0.0046	0.0256	0.0215
13	A_2	B_6	0.0065	0.0005	0.0034	38	A_6	B_3	0.0046	0.0343	0.0269
14	A_2	B_7	0.0063	0.0063	0.0058	39	A_6	B_4	0.0102	0.0237	0.0262
15	A_3	B_1	0.0077	0.0149	0.0175	40	A_6	B_5	0.0049	0.0216	0.0152
16	A_3	B_2	0.0057	0.0232	0.0192	41	A_6	B_6	0.0046	0.0162	0.0136
17	A_3	B_3	0.0071	0.0182	0.0165	42	A_6	B_7	0.0103	0.0086	0.0134
18	A_3	B_4	0.0051	0.0114	0.0121	43	A_7	B_1	0.0047	0.0091	0.0085
19	A_3	B_5	0.0013	0.0075	0.0011	44	A_7	B_2	0.0075	0.0041	0.0095
20	A_3	B_6	0.0094	0.0007	0.0051	45	A_7	B_3	0.0023	0.0124	0.0111
21	A_3	B_7	0.0042	0.0067	0.0038	46	A_7	B_4	0.0086	0.0195	0.0238
22	A_4	B_1	0.0087	0.0552	0.0349	47	A_7	B_5	0.0020	0.0186	0.0136
23	A_4	B_2	0.0068	0.0199	0.0208	48	A_7	B_6	0.0114	0.0155	0.0189
24	A_4	B_3	0.0076	0.0184	0.0229	49	A_7	B_7	0.0030	0.0188	0.0100
25	A_4	B_4	0.0122	0.0156	0.0156						

4.5.3.2 Deuxième étape: optimisation par AGH

Au cours de la seconde et dernière étape d'optimisation, l'ensemble des paramètres du contrôleur (paramètres relatifs aux fonctions d'appartenance et aux conclusions des règles floues) sont optimisés tout en minimisant le nombre de règles floues.
Deux objectifs sont considérés:

1. Minimisation de l'erreur quadratique moyenne (EQM) de la concentration en substrat:

$$Ob_1 = EQM = \frac{1}{N}(S(t) - S_r(t))^2 \tag{4.24}$$

   avec N est la taille du profil de référence $S_r(t)$

2. Minimisation du nombre de règles floues (équation 3.22)

4.5.4 Paramètres de l'AGH

Les paramètres utilisés sont donnes dans le tableau ci-dessous:

TAB. 4.12 – *Paramètres de l'AGH*

	Chromosome de l'AGH		
	Gènes de contrôle	Gènes de paramètres	
		fonctions d'appartenance	paramètres a_j^i
Représentation	binaire	réelle	
Type de croisement	en deux points		
Probabilité de croisement	0.2	0,85	
Types de mutation	standard	non uniforme équations 1.2 et 1.3	gaussienne équation 1.4
Probabilité de mutation	0.005	0.01	0.008
Méthode de sélection	NSGA-II		
Taille de la population	40		
Nombre de générations	1500		
Contrainte	$l_r = 15$		

La figure 4.20 donne, dans l'espace objectifs, la répartition des solutions de la première génération dont laquelle on retrouve la solution obtenue par rétropropagation (cercle noire: $Ob_1 = 7{,}209.10^{-4}$, $Ob_2 = 49$).

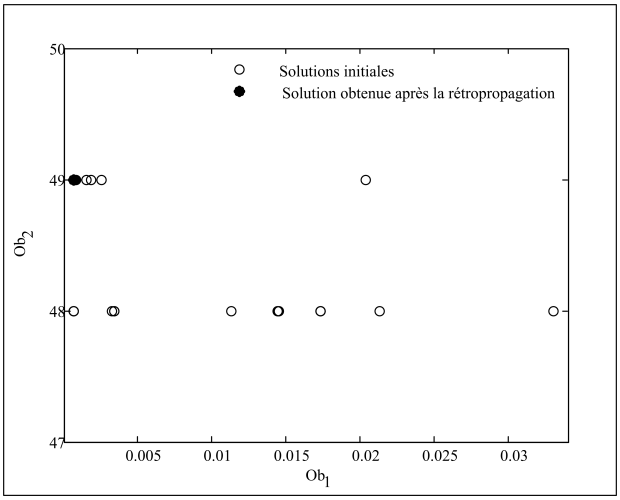

FIG. 4.20 – *Solutions de la première génération*

La figure 4.21 illustre la répartition des solutions de la dernière génération. Nous remarquons pour la plupart des solutions une amélioration considérable des deux objectifs.

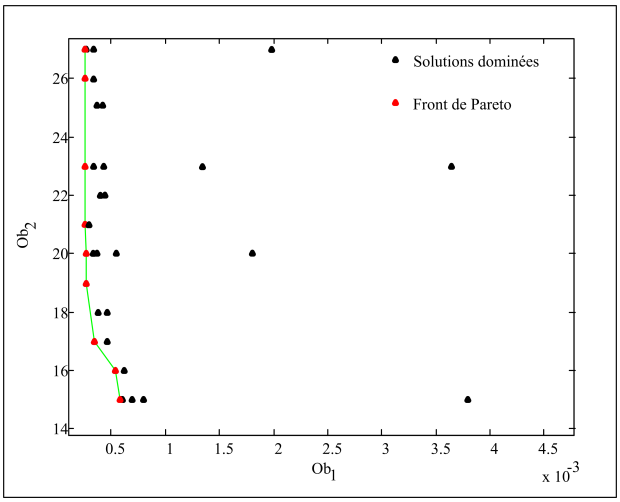

FIG. 4.21 – *Solutions de la dernière génération*

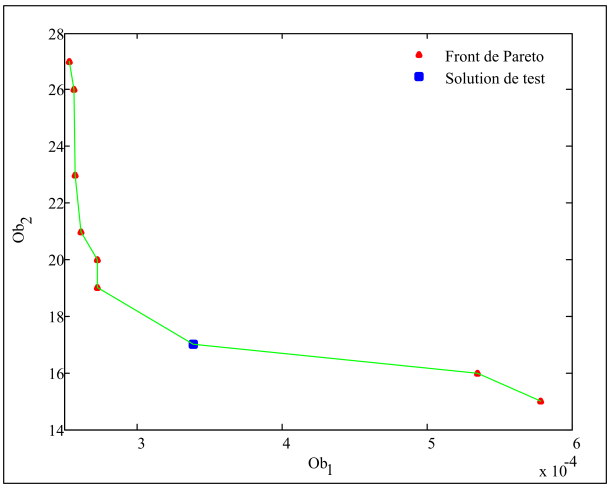

FIG. 4.22 – *Solutions du front de Pareto*

Pour toutes les solutions du front de Pareto (figure 4.22), nous avons un bon suivi de la trajectoire de référence ($ob_1 \in [2{,}538.10^{-4}, 5{,}784.10^{-4}]$). Comme exemple illustratif, nous avons représenté dans la figure 4.23 l'évolution de la concentration en substrat pour une solution du front de Pareto (carré bleu de la figure 4.22). Le suivi de la trajectoire de référence est meilleur que celui obtenu après la rétropropagation. La commande appliquée

au bioréacteur est douce et ne dépasse pas la valeur de μ_{max} ($D(t) < 0{,}38h^{-1}$, condition qu'il faut toujours satisfaire, sans quoi, le bioréacteur risque d'être lavé).

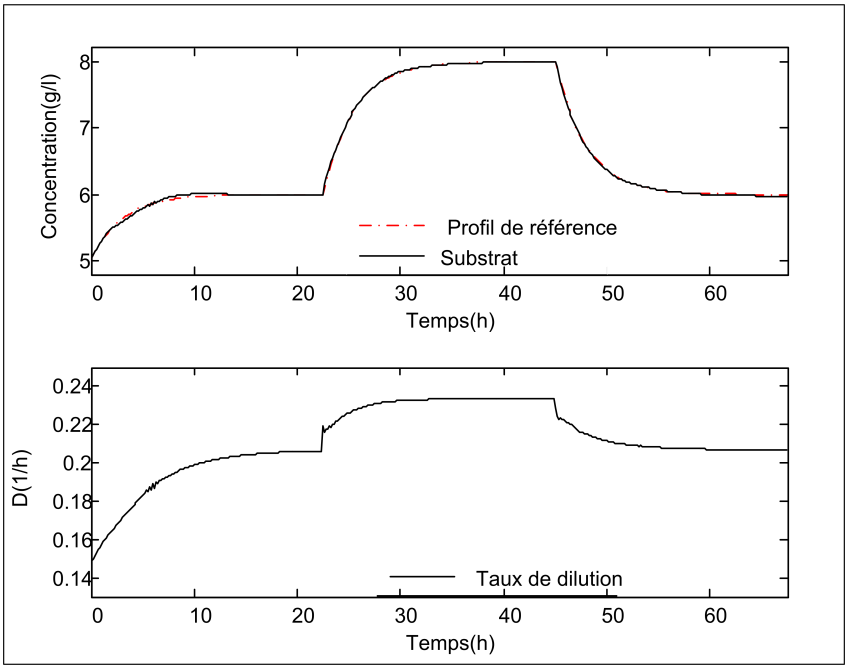

FIG. 4.23 – *Evolution de la concentration en Substrat et le taux de dilution pour la solution de test*

Pour cet exemple, uniquement 17 règles floues sont considérées (tableau 4.13), les autres règles (32 au total) sont annulées par les gènes de contrôle. Les fonctions d'appartenance trouvées pour cet exemple, sont données par la figure 4.24

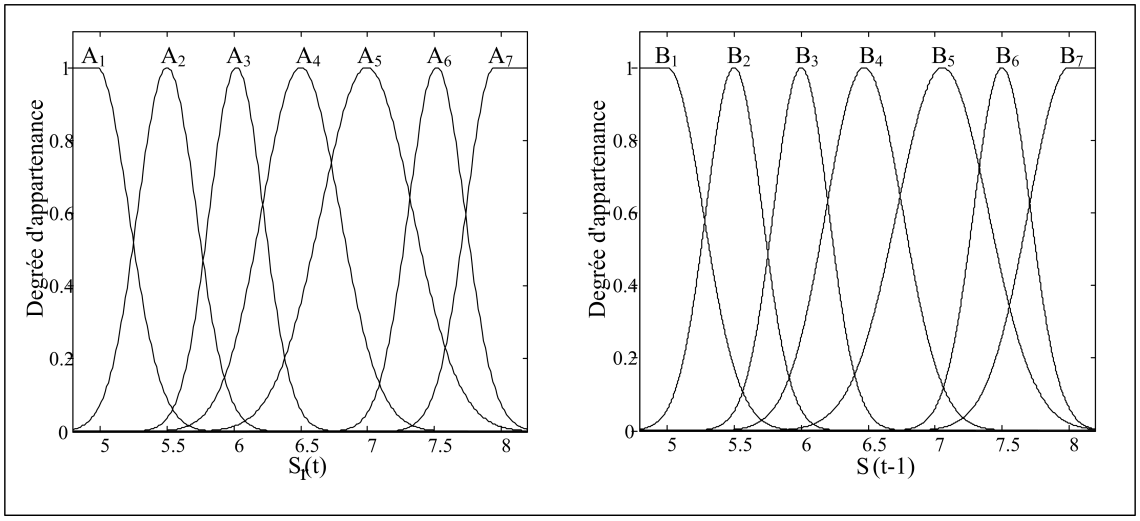

FIG. 4.24 – *Fonctions d'appartenance correspondant à la solution de test*

TAB. 4.13 – *Base de règles relative à la solution de test*

Règle	Prémisses des règles floues		Paramètres relatifs aux règles floues			Règle	Prémisses des règles floues		Paramètres relatifs aux règles floues		
	$S_r(t)$	$S(t-1)$	a_0^i	a_1^i	a_2^i		$S_r(t)$	$S(t-1)$	a_0^i	a_1^i	a_2^i
1	A_1	B_1	0.0043	0.0193	0.0089	26	A_4	B_5	0.0074	0.0110	0.0128
2	A_1	B_2	0.0032	0.0013	0.0099	32	A_5	B_4	0.0047	0.0210	0.0176
8	A_2	B_1	0.0069	0.0256	0.0117	33	A_5	B_5	0.0068	0.0164	0.0148
9	A_2	B_2	0.0090	0.0104	0.0180	34	A_5	B_6	0.0012	0.0116	0.0119
10	A_2	B_3	0.0072	0.0012	0.0020	40	A_6	B_5	0.0049	0.0214	0.0142
16	A_3	B_2	0.0061	0.0265	0.0194	42	A_6	B_7	0.0100	0.0095	0.0142
17	A_3	B_3	0.0064	0.0181	0.0170	48	A_7	B_6	0.0082	0.0169	0.0186
18	A_3	B_4	0.0057	0.0098	0.0121	49	A_7	B_7	0.0025	0.0185	0.0102
24	A_4	B_3	0.0062	0.0203	0.0235						

4.6 Conclusion

Ce chapitre a montré les différentes applications de la commande hybride GA/CF sur un procédé de fermentation en mode continu. Nous avons d'abord présenté le modèle mathématique du procédé établi à partir des équations de bilan de matière. Ensuite des simulations en mode Batch et continu ont permi de retrouver les courbes classiques pour les différentes variables, ce qui nous a permi de vérifier l'exactitude des programmes écrits sous l'environnement Matlab. La première hybridation concerne l'optimisation des paramètres d'un contrôleur flou de type Mamdani par algorithme génétique simple. Les résultats obtenus sont satisfaisants dans la mesure où la sortie du procédé (concentration en substrat) suit bien le profil de référence. Néanmoins, l'inconvénient de la méthode réside en niveau de la base de règles obtenue à la fin du processus d'optimisation. Cette dernière présente des règles incohérentes difficiles à détecter par l'AG. C'est entre autres, pour remédier à ce problème que nous avons proposé une deuxième hybridation du contrôleur flou avec le NSGA-II et un codage hiérarchisé. Un autre objectif supplémentaire peut être alors considéré, celui de la minimisation du nombre de règles floues. L'efficacité de l'approche est bien justifiée à travers les simulations effectuées pour le même profil de référence. Nous avons toujours un bon suivi de consignes même avec un nombre réduit de règles.

La troisième et la dernière hybridation concerne le contrôleur flou de type Sugeno. La mise au point de ce dernier (optimisation des fonctions d'appartenance et des règles floues) est réalisée à travers la poursuite d'une trajectoire de référence, et ce, à l'aide d'un algorithme hybride basé sur une technique de descente du gradient (rétropropaga-

tion) et un algorithme génétique hiérarchisé.
Là aussi, les résultats de simulations ont confirmé l'intérêt de la méthode: le nombre de règles floues est considérablement réduit (15 à 27 règles) pour l'ensemble des solutions du front de Pareto et la poursuite de la trajectoire de référence est toujours satisfaisante.

Conclusion et perspectives

Nous nous sommes intéressés dans cette thèse, au problème de la conception de contrôleurs flous par algorithmes génétiques. La difficulté de l'obtention de la base de règles et des fonctions d'appartenance est en effet très pénalisante lors de l'utilisation des techniques floues.
Cette thèse s'inscrit donc dans la continuité des travaux existants visant à simplifier l'étape de conception en proposant quelques méthodes par hybridation des techniques du soft compting. Ainsi, nous avons présenté dans un premier temps la méthode de conception par algorithme génétique simple qui optimise simultanément:

- Les fonctions d'appartenance des variables d'entrées et de sortie du contrôleur,
- Les facteurs d'échelle en entrée et en sortie,
- Les conclusions des règles floues

Nous avons décrit le principe de codage mixte des paramètres du contrôleur dans un seul chromosome divisé en trois parties dont la première est codée en base n, la seconde en nombres réels et la troisième en binaire. Nous avons ensuite présenté les opérateurs génétiques permettant d'exploiter le codage mixte. La méthode de sélection est celle du tournoi. Pour ne pas rester dans un cadre descriptif, nous avons présenté une application de cette méthode à partir d'un problème de commande d'un système linéaire du second ordre. Bien que les résultats obtenus soient satisfaisants en terme d'erreur, la méthode, comme la plupart des méthodes proposées dans la littérature, présente un inconvénient majeur qui est celui de l'obtention de règles floues incohérentes à la fin du processus d'optimisation. Pour remédier à cet inconvénient nous avons introduit dans un second temps les AGH. Ces derniers permettent par le biais de leurs chromosomes un codage plus complet qui offre, en plus du codage des paramètres, l'activation ou la désactivation des règles floues. Pour éviter d'obtenir, comme pour le cas des AGS, une base de

règles comprenant des règles incohérntes, un autre objectif a été considéré (minimisation du nombre de règles floues), ce qui constitue un problème d'optimisation multiobjectifs. Devant un tel problème nous avons utilisé le NSGA-II comme méthode de résolution à cause des caractéristiques qu'elle présente en particulier la rapidité de tri et l'élitisme. Les résultats de simulation obtenus sur le même système sont très concluants quant à l'efficacité de la méthode du moment que nous avons toujours les même performances en terme d'erreur entre la sortie désirée et la sortie du système, tout en considérant dans le contrôleur un nombre réduit de règles floues et en éliminant d'éventuelles règles incohérentes qui peuvent se présenter.

Pour obtenir de bonnes performances en un nombre réduit de générations, l'initialisation de la partie correspondante aux règles floues, dans la structure des chromosomes, est effectuée par le modèle de règles de Macvicar Whelan. Ce modèle, exploité par la plupart des contrôleurs flous, permet de créer des chromosomes performants et donc de débuter le processus d'optimisation avec une bonne population initiale. Néanmoins ce modèle n'est valable que pour des contrôleurs à deux entrées (l'erreur et sa variation (ie. dérivée de l'erreur)). Pour un cas plus général nous avons préféré les contrôleurs de Sugeno, à cause de la nature des règles qu'il présente (expressions analytiques des variables d'entrée) et offrant une meilleure possibilité d'exploitation des techniques d'optimisation classiques (descente du gradient, moindres carrés, etc). Nous avons donc proposé une méthode de synthèse hybride des CF de type Sugeno qui utilise d'abord l'algorithme de rétropropagation pour un premier ajustement de l'ensemble des paramètres du contrôleur, puis l'algorithme génétique hiérarchisé pour un deuxième ajustement des paramètres, tout en minimisant le nombre de règles floues. Les performances de l'algorithme ont été vérifiées à travers un exemple de modélisation d'un système non linéaire. La comparaison, de point de vue performance, entre cet algorithme et d'autres algorithmes reportés dans la littérature a permi de montrer que l'algorithme proposé donne lieu à de meilleurs résultats. L'ensemble de ces hybridations a été appliqué, dans le dernier chapitre de ce mémoire, à un modèle de simulation d'une fermentation alcoolique. Le contrôleur flou optimisé permet de réguler la concentration en substrat à l'intérieur du bioréacteur en agissant sur le taux de dilution.

En conclusion, nous pouvons dire que les algorithmes génétiques représentent un outil très puissant pour la conception de contrôleurs intelligents. Pour cela il faut d'abord définir les variables d'entrée et de sortie du contrôleur ainsi que les critères de performances permettant de distinguer les bonnes et les mauvaises solutions. Ensuite, vient le choix du codage et de la méthode de sélection. Le premier choix est conditionné par plusieurs

facteurs à savoir le type des variables à optimiser (continus ou discrets) et la précision recherchée. Quant au choix de la méthode de sélection, il faut tenir compte du nombre de critères de performance défini, s'il s'agit d'un seul critère, la méthode de sélection par tournoi peut être considérée dans un premier coup. Si par contre, le nombre de critère est supérieur à un et s'ils sont non commensurables ou contradictoires, il faut opter pour une méthode de sélection utilisant le principe de dominance au sens de Pareto.

Perspectives

Les travaux décrits dans cette thèse peuvent se poursuivre sur plusieurs voies de recherche. Tout d'abord, il serait intéressant d'étendre la première et la deuxième méthode de synthèse des contrôleurs de Mamdani sur un espace d'entrée supérieur à deux. Pour éviter d'obtenir durant le processus d'optimisation des solutions pour lesquelles aucune règle n'est active, nous avons imposé une contrainte qui consiste à garder un nombre minimal de règles l_r active et qu'il faut fixer dès la première génération.
Une étude visant à modifier la valeur de ce nombre l_r au cours des générations est également une problématique intéressante qu'il faut aborder.

Bibliographie

Acosta, G. et E. Todorovich (2003). Genetic algorithms and fuzzy control: a practical synergism for industrial applications. *Computers in Industry* **52**, 183–195.

Belarbi, K., F. Titela, W. Bourebiaa et K. Benmahammed (2005). Engineering applications of artificial intelligence. *Design of mamdani fuzzy logic controllers with rule base minimisation using genetic algorithm* **18**, 875–880.

Berenji, H.R. et P. Khedkar (1992). Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks* **3**(5), 724–740.

Bideaux, C. (2000). Modélisation stœchiométrique des productions microbiennes par descripteur métabolique au moyen du calcul formel. Validation sur le modèle Kluyveromyces marxianus. Thèse de doctorat. Institut National des Sciences Appliquées de Toulouse.

Braae, M. et D.A. Rutherford (1978). Fuzzy relations in control setting. *Kybernetes* **7**(3), 185–188.

Burns, R.S. (2001). *Advanced Control Engineering*. butterworth heinemann ed.

Chen, C.C. et C.C. Wong (2002). Self-generating rule-mapping fuzzy controller design using a genetic algorithm. In: *IEE Proceedings on Control Theory and Applications*. Vol. 49. pp. 143–148.

Cheong, F. et R. Lai (2000). Constraining the optimization of a fuzzy logic controller using an enhanced genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics-Part B:Cybernetics* **30**(1), 31–46.

Cordon, O., F. Herrera et P. Villar (2001). Generating the knowledge base of a fuzzy rule-based system by thegenetic learning of the data base. *IEEE Transactions on Fuzzy Systems* **9**(4), 667–674.

Corne, D., J.D. Knowles et M.J. Oates (2000). The pareto envelope-based selection algorithm for multi-objective optimization. In: *Proceedings of the sixth International Conference on Parallel Problem Solving from Nature*. pp. 839–848.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* **2**, 303–314.

Deb, K., A. Pratap, S. Agarwal et T. Meyarivan (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197.

Dochain, D. (2001). *Automatique des bioprocédés*. Hermes Science.

Dorf, R.C. et R.H. Bishop (1995). *Modern control systems*. Addison Wesley.

Farag, W.A., V.H. Quintana et G.Lambert-Torres (1998). A genetic-based neuro-fuzzy approach for modeling and control of dynamical systems. *IEEE Transactions on Neural Networks* **9**(5), 756–767.

Fonseca, C.M. et P.J. Fleming (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, Ed.). Morgan Kaufmann. San Mateo,California. pp. 416–423.

Glover, F. (1989). Tabu search-part 1. *ORSA Journal on Computing* **1**, 190–206.

Goldberg, D.E. (1994). *Algorithmes génétiques. Exploration, optimisation et apprentissage automatique*. Addison-Wesley. France.

Goldberg, D.E. et J. Richardson (1987). Genetic algorithms with sharing for multimodal function optimization. In: *Second International Conference on Genetic Algorithms and their application*. Cambridge, Massachusetts, United States. pp. 41–49.

Guenounou, O., A. Belmehdi et B. Dahhou (2006). Optimisation des contrôleurs flous par algorithmes génétiques hiérarchisés-application à la commande d'un bioprocédé. In: *Seventh International Conference on Sciences and Techniques of Automatic control STA 2006*. Hammamet,Tunisia. pp. 1–10.

Guenounou, O., A. Belmehdi et B. Dahhou (2009). Multi-objective optimization of tsk fuzzy model. *Expert Systems with Applications* **36**(4), 7416–7423.

Guenounou, O. et A. Belmehdi (2006). Optimisation des contrôleurs flous par algorithmes génétiques hiérarchisés-application sur actionneur asynchrone. In: *Proceedings de la conférence internationale sur le génie électrique*. Batna,Algérie. pp. 600–605.

Hagan, M.T. et M.B. Menhaj (1994). Training feedforward networks with marquardt algorithm. *IEEE Transaction On Neural Networks* **5**(6), 989–993.

Herrera, F., M. Lozano et J.L. Verdegay (1995). Tuning fuzzy logic controllers by genetic algorithms. *International Journal of Approximate Reasoning* **12**, 299–315.

Herrera, F., M. Lozano et J.L. Verdegay (1998). A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems* **100**, 143–158.

Holland, J. (1992). *Adaptation in natural and artificial and systems*. 2^{nd} edition. IT Press.

Homaifar, A. et E. MCCormick (1993). Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE transactions on fuzzy systems* **3**(2), 129–139.

Horn, J. et N. Napfliotis (1994). Multiobjective optimization using the niched pareto genetic algorithm. In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*. Vol. 1. pp. 82–87.

Hornik, K., M. Stinchcombe et H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* **2**(5), 359–366.

Hwang, H.S. (1999). Automatic design of fuzzy rule base for modelling and control using evolutionary programming. *IEE Proceedings on Control Theory and Applications* pp. 9–16.

Jang, J.-S.R. (1993). Anfis: adaptive-network-based fuzzy inference system. *IEEE Transaction on System Man and Cybernetics* **23**(3), 665–685.

Jordan, M.I. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science* **16**, 307–354.

Kabbaj, M.N. (2004). Développement d'amgorithmes de détection et d'isolation de défauts pour la supervision des bioprocédés. Thèse de doctorat. Université de Perpignan.

Karr, C.L. (1991). Applying genetics to fuzzy logic. *AI Expert* **6**(3), 38Ű43.

Karr, C.L. et E.J. Gentry (1993). Fuzzy control of ph using genetic algorithms. *IEEE Transactions on Fuzzy Systems* **1**(1), 46–53.

Kim, E., M. Park, S. Ji et M. Park (1997). A new approach to fuzzy modeling. *IEEE Transactions on Fuzzy Systems* **5**(3), 328–337.

Kirkpatrick, S., C. Gelatt et M. Vecchi (1983). Optimization by simulated annealing. *Science* **220**(4598), 671–680.

Klir, G.J. et B. Yuan (1994). *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall. Inc. Upper Saddle River, NJ, USA.

Lee, C.C. (1990). Fuzzy logic in control systems: Fuzzy logic controller, part ii. *IEEE Transactions on Systems, Man. and Cybernetics* **20**(2), 419–435.

Lee, C.S. Ouyang W.J. Lee S.J. (2005). A tsk-type neurofuzzy network approach to system modeling problems. *IEEE Transactions on Systems, Man and Cybernetics* **35**(4), 751–767.

Lee, M. A. et H. Takagi (1993). Dynamic control of genetic algorithms using fuzzy logic techniques. In: *Proceedings of International conference on Genetic Algorithms*. San Mateo, CA. pp. 76–83.

Li, Z. (2006). Contribution à l'élaboration d'algorithmes d'isolation et d'identification de défauts dans les systèmes non linéaires. Thèse de doctorat. Université Paul Sabatier de Toulouse.

Lin, C.T. et C.S.G. Lee (1991). Neural-network-based fuzzy logic control and decision system. *IEEE Transactions on Computers* **40**(12), 1320.

Liska, J. et S.S. Melsheimer (1994). Complete design of fuzzy logic systems using genetic algorithms. In: *Proceedings of the Third IEEE International Conference on Fuzzy Systems*. Vol. 2. Orlando, FL, USA. pp. 1377Ű–1382.

Macvicar-Whelan, P.J. (1976). Fuzzy sets for man-machine interaction. *International Journal of Man-Machine Studies* **8**, 687–697.

Magoulasb, G.D., M.N. Vrahatis et G.S. Androulakis (1999). Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Computation* **11**, 1769–1796.

Maher, M. (1995). Modélisation et élaboration d'algorithmes d'estimation et de commande: Application à un bioprocédé. Thèse de doctorat. Université Paul Sabatier de Toulouse.

Mamdani, E.H. et S. Assilian (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies* **7**(1), 1–15.

Man, K. F. et W.A. Halang (1997). Genetic algorithms for control and signal processing. In: *Proceedings of the 23rd International Conference on Industrial Electronics,Control and Instrumentation*. Vol. 4. New Orleans, LA, USA. pp. 1541–1555.

Man, K.F., K.S. Tang et S. Kwong (1996). Genetic algorithms: Concepts and applications. *IEEE Transaction on industrial electronics* **43**, 519–534.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Mogharreban, N. et L.F. Dilalla (2006). Comparison of defuzzification techniques for analysis of non-interval data. In: *Fuzzy Information Processing Society, 2006. NAFIPS 2006.Annual meeting of the North American*. pp. 257–260.

Monod, J. (1942). *Recherche sur la croissance des cultures bactériennes*. Hermann, Paris.

Narendra, K.S. et K. Parthasarathy (1990). Identification and control of dynamical systems using neural networks. *IEEE Transaction on Neural Networks* **1**(1), 4–27.

Ng, K.C. et Y. Li (1994). Design of sophisticated fuzzy logic controllers using genetic algorithms. In: *IEEE World Congress on Computational*. Vol. 3. USA. pp. 1708–1712.

Pedrycz, W. et L.A. Zadeh (1995). *Fuzzy Set Engineering*. CRC Press. Boca Raton,USA.

Psaltis, D., A. Sideris et A. Yamamura (1987). Neural controllers. In: *Proceedings of the International Conference on Neural Networks*. Vol. 4. San Diego. pp. 551–558.

Psaltis, D., A. Sideris et A.A. Yamamura (1988). A multilayered neural network controller. *IEEE Control Systems Magazine* **8**, 17–21.

Rumelhart, D.E. et J. L. McClelland (1986). *Parallel Distributed Processing:Explorations in the Microstructure of Cognition*. Vol. 1. M.I.T. Press.

Saerens, M. et A. Soquet (1991). Neural controller based on back-propagation algorithm. In: *IEE Proceedings-F on Radar and Signal Processing*. Vol. 138. pp. 55–62.

Sastry, P.S., G. Santharam et K.P. Unnikrishnan (1994). Memory neuron networks for identification and control of dynamical systems. *IEEE Transactions on Neural Networks* **5**(2), 306–319.

Schaffer, J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. pp. 93–100.

Scharf, E.M. et N.J. Mandic (1985). The application of a fuzzy controller to the control of a multi-degree-freedom robot arm. *Industrial Application of fuzzy control* pp. 41–61.

Srinivas, V. et K. Deb (1993). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* **2**(3), 221–248.

Sugeno, M. (1999). On stability of fuzzy systems expressed by fuzzy rules with singleton consequents. *IEEE Transactions on Fuzzy Systems* **7**(2), 201–224.

Tang, K.S., C.Y. Chan, K. F. Man et S. Kwong (1995). Genetic structure for nn topology and weights optimization. In: *Conference on Genetic Algorithms in Engineering Systems:Innovations and Applications*. IEE. pp. 250–255.

Tang, K.S., K.F. Man, Z.F. Liu et S. Kwong (1998). Minimal fuzzy memberships and rules using hierarchical genetic algorithms. *IEEE Transactions on Industrial Electronics* **45**(1), 162–169.

Thrift, P. (1991). Fuzzy logic synthesis with genetic algorithms. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*. pp. 509Ű–513.

Vasilache, A. (2000). Ler Réseaux de neurones pour la modélisation et la conduite des procédés biotechnologiques. Thèse de doctorat. Institut National des Sciences Appliquées de Toulouse.

Vogl, T.P., J.K. Mangis, A.K. Zigler, W.T. Zink et D.L. Alkon (1988). Accelerating the convergence of the back-propagation method. *Biological Cybernetics* **59**, 256–264.

Wang, J.S. et C.S.G. Lee (2001). Efficient neuro-fuzzy control systems for autonomous underwater vehicle control. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 3. Seoul, Korea. pp. 2986–2991.

Wang, L. et R. Langari (1996). Complex systems modeling via fuzzy logic. *IEEE Transactions On Systems, Man, And Cybernetics-Part B: Cybernetics* **26**(1), 100–106.

Wu, C.J. et G.Y. Lin (1999). Design of fuzzy logic controllers using genetic algorithms. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 6. SAAA. pp. 104–109.

Xiang, T., K.F. Man, K.M. Luk et C.H. Chan (2006). Design of multiband miniature handset antenna by mom and hga. *IEEE Antennas and wireless propagation letters* **5**, 179–182.

Yager, R.R. et D.P. Filev (1994). *Essentials of fuzzy modeling and control*. Wiley-Interscience. New York, NY, USA.

Yubazaki, N., M. Otani, T. Ashida et K. Hirota (1995). Dynamic fuzzy control method and its application to positioning of induction motor. In: *Proceedings of Fourth IEEE International Conference on Fuzzy Systems*. Japan. pp. 1095–1102.

Zadeh, L.A (1965). fuzzy sets. *Informatic and Control* **8**, 338–353.

Zhou, Y.S. et L.Y. Lai (2000). Optimal design for fuzzy controllers by genetic algorithms. *IEEE Transactions on Industry Applications* **36**(1), 93–97.

Zitzler, E. et L. Thiele (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation* **3**(4), 257–271.

Méthodologie de conception de contrôleurs intelligents par l'approche génétique - application à un bioprocédé

Résumé : Dans ce travail, le problème de conception de contrôleurs flous est étudié. Dans une première partie, on présente un état de l'art sur les techniques utilisées à savoir les algorithmes génétiques et ses différentes variantes, les réseaux de neurones, la logique floue et leurs hybridations. Prenant appui sur cet état de l'art nous proposons une première méthode de conception des contrôleurs flous de Mamdani par algorithmes génétiques simples.

Cette méthode est en suite améliorée par l'emploi des algorithmes génétiques hiérarchisés. Ces derniers permettent par le biais de la structure de leurs chromosomes, une meilleure optimisation des paramètres du contrôleur tout en éliminant les règles incohérentes qui peuvent se présenter, comme pour la première méthode, à la fin du processus d'optimisation.

La dernière méthode proposée concerne la synthèse des contrôleurs flous de Sugeno. Elle est basée sur une procédure d'apprentissage hybride qui se déroule en deux étapes. Durant la première étape, le contrôleur flou est représenté sous forme d'un réseau de neurones multicouches dont les paramètres sont optimisés par l'algorithme de rétropropagation. Dans la deuxième étape, les paramètres obtenus à l'issue de la première phase sont extraits et optimisés par le NSGA-II suivant un codage hiérarchisé.

L'ensemble des ces méthodes est appliqué pour la conduite d'un procédé de fermentation alcoolique en mode continu.

Mots-clés : Algorithmes génétiques simples, Algorithmes génétiques hiérarchisés,contrôleur flou, réseaux de neurones, NSGA-II, bioprocédé.

Design methodology of intelligent controllers based on genetic approach - Application to a bioprocess

Abstract : In this work, the problem of design of fuzzy controllers is studied. In a first part, we present a state of art on the techniques used, namely the genetic algorithms and its various alternatives, the neural networks, fuzzy logic and their hybridizations. Taking support on this state of art, we propose a first design method of the fuzzy controllers of Mamdani by simple genetic algorithms.

Thereafter, this method is improved by the use of the hierarchical genetic algorithms. These algorithms allow, by the means of the structure of their chromosomes, a better optimization of the controller parameters while eliminating the incoherent rules which can arise, as well as for the first method, at the end of the optimization process.

The last method suggested relates to the synthesis of the fuzzy controllers of Sugeno. It is based on a hybrid procedure of training which proceeds in two stages. During the first stage, the fuzzy controller is represented in the form of a network of multi-layer neural networks, whose parameters are optimized by the algorithm of retro propagation. In the second phase, the parameters obtained at the end of the first phase are extracted and optimized by the NSGA-II according to a coding arranged hierarchically. These methods are applied for control of an alcoholic fermentation process in continuous mode.

Keywords: Genetic algorithms, Hierarchical genetic algorithms, Fuzzy controller, neural networks, NSGA-II, bioprocess