République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École Nationale Polytechnique
Département d'Automatique
Laboratoire de Commande des Processus

THESE

En vue de l'obtention du titre de

Docteur en Sciences

en Génie Electrique

Option : Automatique

Présenté par :

ALLOUANI FOUAD

Magister en Électronique de l'université Mohamed Boudiaf M'sila

Intitulé :

Contributions à l'Optimisation par Métaheuristiques des Lois de Commandes Non Linéaires

Soutenue le 12-11-2015 devant le du jury composé de :

Président	Mohamed TADJINE	Professeur à l'ENP
Rapporteurs	Djamel BOUKHETALA	Professeur à l'ENP
	Farès BOUDJEMA	Professeur à l'ENP/ESDAT
Examinateurs	Habiba DRIAS	Professeur à l'USTHB
	Mohamed BOUAMAR	Professeur à UMBM'Sila
	Lazhar RAHMANI	Professeur UFASétif

ENP 2015

Résumé

ملخص- في هذه الأطروحة، قمنا باقتراح بديلين بالنسبة لكل من خوارزميات مستعمرات النمل (*Ant colony optimization* ACO) و وخوارزمية البحث عن الانسجام (*Harmony Search* HS). الهدف من هذه المساهمة هو تحسين هذه الخوارزميات بغية تجنب مشكلة التقارب المبكر و سرعة التقارب البطيئة. الطريقة الاولى المقترحة والتي تخص خوارزميات مستعمرات النمل ACO، تعتمد على إدخال بعض التغييرات في الخوارزمية الاصلية C-ACO *Conventional ACO*. الطريقة الثانية المقترحة والتي تسمى GHSACO، نشأت من عملية تهجين بين إصدار من خوارزمية HS تدعى خوارزمية البحث عن الانسجام العام (GHS *Global Best Harmony search*) و خوارزمية C-ACO. بعد ذلك، قمنا باستخدام الخوارزميات المقترحة (ACO المحسن و GHSACO) على التوالي في التطبيقين التاليين، الاولى في عملية تعديل وسائط بنية مراقبة لا مركزية تعتمد على طريقة وضع الانزلاق (modes glissants) والموجهة لمراقبة الأنظمة الغير خطية المترابطة، اما الخوارزمية الثانية فقد استغلت في عملية تعديل غير مباشرة (offline) للإعدادات الشبكات العصبونية الضبابية المتكررة (*Recurrent Fuzzy Neural Networks* RFNN*s*) المندمجة في بنية مراقبة متكيفة والموجهة لمراقبة قسم من الأنظمة الغير خطية. نتائج المحاكاة المتحصل عليها أظهرت فعالية جميع الطرق المقترحة

كلمات مفتاحية : خوارزميات مستعمرات النمل، خوارزمية البحث عن الانسجام، تحكم لا مركزي انزلاقي، تحكم تلاؤمي، الشبكات العصبونية الضبابية المتكررة.

Résumé- Dans cette thèse, deux variantes de *métaheuristiques* sont proposées pour les algorithmes de colonies de fourmis (*Ant Colony Cptimization* ACO) et pour l'algorithme de recherche d'harmonie (*Harmony Search* HS) respectivement. En effet, le but de cette contribution est d'améliorer la vitesse convergence et d'éviter le problème de convergence prématurée afin d'éviter les optimums locaux. La première méthode, relative aux algorithmes ACO, est basée sur l'introduction de quelques modifications dans la structure algorithmique d'algorithme ACO classique (*Conventional ACO* C-ACO). La seconde méthode proposée, dite,GHSACO, est issue d'une hybridation entre une variante d'algorithme HS, appelée algorithme de recherche d'harmonie globale (*Global Best Harmony Search* GHS) et un algorithme de type C-ACO. Les deux algorithmes développés, sont appliqués respectivement au problème de calcul des paramètres d'une commande par mode de glissement décentralisée des systèmes non linéaires interconnectés et au problème d'apprentissage des réseaux de neurones flous récurrents (*Recurrent Fuzzy Neural Networks* RFNN*s*) intégrés dans une structure de commande adaptative d'une classe de systèmes non linéaires. Les résultats de simulations, obtenus montrent l'efficacité de l'ensemble des méthodes proposées.

Mots clés : Algorithmes de colonies de fourmis, Algorithme de recherche d'harmonie, Commande par mode de glissement décentralisée, Commande adaptative, Réseaux de neurones flous récurrents.

Abstract- In this thesis, two variants of *metaheuristics* are proposed for the *Ant Colony Optimization* ACO algorithms and for the *Harmony Search* HS method respectively. Indeed, the aim of this contribution is to improve the speed of convergence and to overcome the premature convergence problem. The first method, relating to ACO algorithms, is based on the introduction of some improvement in the basic algorithmic structure of the *Conventional ACO* C-ACO, the second proposed method, so-called GHSACO, is obtained by merging the Global-Best Harmony Search (GHS) and C-ACO together. The two developed algorithms are used, respectively, to optimize the parameters values of a decentralized sliding mode controller of nonlinear interconnected systems and in the training problem of *Recurrent Fuzzy Neural Networks* RFNN*s* used in adaptive control structure, of a class of nonlinear systems. The obtained simulation results show clearly the effectiveness of all proposed methods.

Key words: Ant colony optimization algorithms, Harmony Search algorithm, Decentralized sliding mode controller, Adaptive control, Recurrent Fuzzy Neural Networks.

Avant-propos

Je tiens à remercier avant tout, nôtre dieu pour m'avoir orienté vers le bon chemin et de m'avoir offert le climat et les conditions pour atteindre ce niveau d'étude et la réalisation de ce travail.

Je remercie mon directeur de thèse Professeur Djamel BOUKHETALA et mon co-directeur de thèse Professeur Farès BOUDJEMA pour leurs conseils et leurs soutiens tout le long de la préparation de cette thèse.

Je remercie aussi les membres du jury, Professeur Mohamed TADJINE, Professeur Habiba DRIAS, Professeurs Mohamed BOUAMAR et Lazhar RAHMANI pour avoir accepté l'évaluation de ce travail.

Un grand merci au Professeur Kai ZENGER pour son accueil au département de génie électrique et automatique à l'université Aalto, Finlande ainsi que pour ses conseils et ses orientations durant mes séjours à l'université.

Je remercie infiniment le docteur XIAO-ZHI Gao de m'avoir offert une assistance et un aide permanant pour la réussite de mon travail.

Enfin, une gratitude et des remerciements particuliers à tous les membres de ma famille pour leur soutien permanant le long de mes études.

Table des matières

Introduction générale 1

CHAPITRE 1
Métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoire mono-objectifs : Etat de l'art

1.1 Introduction 5
1.2 Métaheuristiques pour l'optimisation mono-objectif difficile 6
1.2.1 Problème d'optimisation mono-objectif 6
1.2.2 Optimisation difficile 7
1.2.3 Algorithmes d'optimisation approchée 8
1.2.3.1 Heuristiques 8
1.2.3.2 Métaheuristiques 8
1.3 Métaheuristiques à population de solutions issues de la théorie de l'évolution (algorithmes évolutionnaires) 10
1.3.1 Les algorithmes génétiques (Genetic Algorithms) 11
a. Représentation des solutions 11
b. Population des solutions candidates 12
c. Fonction objectif 12
d. Sélection 12
e. Croisement 12
f. Mutation 13
g. Remplacement 14
1.3.2 Programmation génétique 14
1.3.3 Stratégies d'évolution 15
1.3.4 Programmation évolutive 15
1.3.5 Autres algorithmes évolutionnaires 16
1.3.5.1 Algorithme à évolution différentielle 16
a – Mutation 17
b – Croisement 18
c – Sélection 18
1.3.5.2 Algorithmes à estimation de distribution 18
1.3.5.3 Les algorithmes culturels 19
1.3.5.4 Les systèmes immunitaires artificiels 21
1.3.5.5 Algorithme de recherche d'harmonie 22
1.4 Métaheuristiques à population de solutions issues de l'intelligence en essaim 22
1.4.1 Optimisation par Essaim Particulaire 23
1.4.2 Algorithme de recherche de nourriture bactérienne 26
1.4.3 Algorithmes de colonies de fourmis 29
1.5 Conclusion 29

CHAPITRE 2
Contribution à l'amélioration des algorithmes de colonies de fourmis : Algorithme ACO amélioré

2.1 Introduction . . . 31
2.2 Optimisation par colonies de fourmis (ACO) . . . 32
2.2.1 Origines de l'approche . . . 32
2.2.2 Optimisation naturelle : pistes de phéromones . . . 33
2.2.3 Optimisation par colonies de fourmis et problème du voyageur de commerce . 35
2.2.3.1 Algorithme de colonie de fourmis de base, le AS . . . 35
2.2.3.2 Principaux schémas ACO pour le TSP . . . 37
2.2.3.3 Choix des paramètres . . . 39
2.2.4 Formalisation et propriétés d'un algorithme ACO . . . 40
2.2.4.1 Formalisation . . . 40
2.2.4.2 Phéromones et mémoire . . . 42
2.2.4.3 Intensification/diversification . . . 42
2.2.4.4 ACO et la recherche locale . . . 43
2.2.4.5 Parallélisme . . . 44
2.2.4.6 Convergence . . . 44
2.3 Amélioration de ACO – proposition d'une nouvelle variante . . . 45
2.3.1 Principe . . . 45
2.3.2 Formalisme et méthode d'adaptation de la variante proposée aux problèmes d'optimisation combinatoire . . . 45
2.3.3 Règle de déplacement des fourmis . . . 47
2.3.4 Règle de mise-à-jour des pistes de phéromone . . . 48
2.4 Conclusion . . . 50

CHAPITRE 3
Elaboration d'une nouvelle variante d'algorithme de recherche d'harmonie : le GHSACO

3.1 Introduction . . . 51
3.2 Algorithme HS est ses variantes, IHS et GHS . . . 52
3.2.1 Algorithme HS . . . 52
3.2.2 Algorithme IHS . . . 55
3.2.3 Algorithme GHS . . . 55
3.3 Caractéristiques fondamentales d'algorithme HS . . . 56
3.4 Nouvelle variante d'algorithme HS, le GHSACO . . . 57
3.4.2 Principe de l'algorithme proposé . . . 57
3.5 Résultats et analyse . . . 60
3.5.1 Problèmes de test . . . 60
3.5.2 Paramétrage . . . 61
3.5.3 Résultats et discussions . . . 62
3.6 Conclusion . . . 68

CHAPITRE 4
Application de l'algorithme ACO amélioré à l'optimisation des paramètres de la loi de commande par mode de glissement décentralisée

4.1 Introduction . . . 70
4.2 Commande par modes glissants décentralisée à base d'une approche coopérative et un algorithme ACO amélioré . . . 71
4.2.1 Commande par modes glissants . . . 71
4.2.1.1 Définitions et concepts de base . . . 71
4.2.1.2 Synthèse de la surface de glissement . . . 72
4.2.1.3 Synthèse de la loi de commande . . . 73
4.2.1.4 Commande équivalente . . . 75
4.2.1.5 Le phénomène du *chattering* . . . 76
4.2.2 Méthode de commande proposée . . . 79
4.2.2.1 Classe de systèmes interconnectés . . . 79
4.2.2.2 Synthèse de la commande . . . 79
4.2.2.3 Schéma global de la commande décentralisée proposée . . . 80
4.2.2.4 Calcul de la commande équivalente . . . 82
4.2.2.4 Calcul de la commande corrective . . . 85
4.2.2.5 Preuve de la stabilité . . . 87
4.2.3 Régalage des paramètres de la commande décentralisée par l'algorithme ACO amélioré . . . 88
4.3 Simulations . . . 89
4.3.1 Exemple 1 : Commande d'un TRMS . . . 90
4.3.2 Interprétation des résultats . . . 96
4.3.3 Exemple 2 : Commande d'un système double pendule inversé . . . 97
4.4 Conclusions . . . 102

CHAPITRE 5
Application de l'algorithme GHSACO au problème d'apprentissage des RFNNs en commande adaptative

5.1 Introduction 103
5.1 Commande adaptative indirecte basée sur un contrôleur superviseur et les RFNNs . . . 103
5.1.1 Objectifs de la commande proposée . . . 103
5.1.2 Synthèse de la commande adaptative indirecte . . . 104
5.1.3 Contrôleur superviseur . . . 105
5.2 Apprentissage des RFNNs intégrés dans une structure de commande adaptive par algorithme GHSACO . . . 106
5.3 Simulations . . . 108
5.4 Conclusions . . . 116

Conclusion générale . . . 117
Références bibliographiques . . . 119

Liste des principales abréviations

ACO	-Algorithmes de colonies de fourmis
ACO - SP	- Algorithmes de colonies de fourmis à deux modèles de phéromone (bonne et mauvaise)- ACO with stench pheromone
ACS	- Algorithme Ant Colony System
AIS	- Systèmes immunitaires artificiels
AS	- Algorithme Ant System
BCO	- Algorithme d'optimisation par colonie d'abeilles
BFO	- Algorithme de recherche de nourriture bactérienne
BPA	- Algorithme de rétro-propagation du gradient
BWAS	- Algorithme Ant System meilleure-mauvaise (Best-Worst Ant System)
C-ACO	- Algorithme ACO classique
CAs	- Algorithmes culturels
DE	- Algorithme à évolution différentielle
EAs	- Algorithmes évolutionnaires
EDA	- Algorithme à estimation de distribution
EP	- Programmation évolutive
ES	- Stratégies d'évolution
F_{obj}	- Fonction objectif
FNNs	- Réseaux de neurones flous
FS	- Système flou
FSM	- Machines à état fini
GAs	- Algorithmes génétiques
GBAS	- Algorithme Graph-Based Ant System
GP	- Programmation génétique
GHS	- Algorithme de recherche d'harmonie globale
GHSACO	-Nouvelle variante de HS, née d'une hybridation entre le GHS et un C-ACO
HS	-Algorithme de recherche d'harmonie
IHS	- Algorithme de recherche d'harmonie amélioré
MMAS	- Algorithme Max-Min Ant System
MSE	- Moyenne du carrée de l'erreur
NNs	- Réseaux de neurones
RBAS	- Algorithme Ant System basé sur un classement cyclique de bonnes solutions (Rank-Based Ant System)
RFNN*s*	- Réseaux de neurones flous récurrents
RFNN-FSMC	- Contrôleur par modes glissants basé sur un RFNN et un FS
PSO	- Algorithme d'optimisation par essaim particulaire
SBX	- Version de croisement à un point (cas d'un codage réel)
SI	- Intelligence en essaim
TRMS	- Simulateur de vol d'hélicoptère, le Twin Rotor MIMO system
TSP	-Problème du voyageur de commerce

Liste des Figures

CHAPITRE 1

Figure 1.1 Différence entre un optimum global et des optima locaux 7
Figure 1.2 Exemple de croisement en représentation binaire 13
Figure 1.3 Exemple de mutation en représentation par permutation 14
Figure 1.4 Les composants principaux d'un algorithme culturel 20
Figure 1.5 Déplacement d'une particule 24
Figure 1.6 Méthode de déplacement des bactéries 27

CHAPITRE 2

Figure 2.1 Des fourmis autour de leur nid 33
Figure 2.2 Illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. (a) Recherche sans obstacle, (b) Apparition d'un obstacle, (c) Recherche du chemin optimal, (d) Chemin optimal trouvé 34
Figure 2.3 Le problème TSP optimisé par l'algorithme AS, les points représentent les villes et l'épaisseur des arêtes la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi, (b) au début du calcul, tous les chemins sont explorés, (c) le chemin le plus court est plus renforcé que les autres, (d) l'évaporation permet d'éliminer les autres chemins les moins renforcé 37
Figure 2.4 Dans un algorithme ACO, les pistes de phéromone peuvent être associées aux composants (a) ou aux connexions (b) du graphe représentant le problème à résoudre 41
Figure 2.5 Représentation graphique du problème d'optimisation par ACO 47

CHAPITRE 3

Figure 3.1 Représentation graphique de la matrice HM, d'où le chemin effectué par une fourmi est marqué par un trait gras 59
Figure 3.2 Organigramme de l'algorithme GHSACO 60
Figure 3.3 Evolution des solutions obtenues pour la fonction *Ackley* 65
Figure 3.4 Evolution des solutions obtenues pour la fonction *Griewank* 65
Figure 3.5 Evolution des solutions obtenues pour la fonction *Quartic function with noise* 66
Figure 3.6 Evolution des solutions obtenues pour la fonction *Generalized Schwefel Problem 2.26* 66
Figure 3.7 Evolution des solutions obtenues pour la fonction *Rosenbrock* 67
Figure 3.8 Evolution des solutions obtenues pour la fonction *Step* 67
Figure 3.9 Evolution des solutions obtenues pour la fonction *Schwefel problem* 2.22 68

CHAPITRE 4

Figure 4.1 Surface de glissement 72
Figure 4.2 Phénomène de *chattering* 77
Figure 4.3 Une couche limite (Boundary Layer) enveloppe une surface de glissement 78
Figure 4.4 Schéma global de la commande décentralisée 81
Figure 4.5 Configuration interne de chaque sous-contrôleur 82
Figure 4.6 Structure du RFNN, utilisé pour le calcul de la commande équivalente 84
Figure 4.7 Fuzzification de la variable σ_i 86
Figure 4.8 Fuzzification du terme correctif $u_{Fuzzy\,i}$ 86
Figure 4.9 Le TRMS 90
Figure 4.10 Structure de commande décentralisée optimisée par ACO pour le TRMS 91
Figure 4.11 Evolution de la fonction objectif pour l'algorithme ACO proposée et le C-ACO 93
Figure 4.12 Poursuite d'une trajectoire sinusoïdale ; réponses du TRMS (en bleu réponse du système et en rouge la trajectoire désirée) ; commandes générées $u_{i=1,2}$ 94
Figure 4.13 Test de stabilisation du TRMS à un angle $\alpha_{hd} = 1\,rad$ (en bleu discontinu réponse réelle et en rouge valeur désirée) ; commande u_1 94
Figure 4.14 Test stabilisation du TRMS à un angle $\alpha_{vd} = 0\,rad$ (en bleu discontinu réponse réelle et en rouge valeur désirée) ; commande u_1 95
Figure 4.15 Poursuite d'une trajectoire sinusoïdale avec commande décentralisée par modes glissants classiques ; réponses du TRMS (en bleu réponse du système et en rouge la trajectoire désirée) ; commandes générées $u_{i=1,2}$ 95
Figure 4.16 Angle α_h avec la présence d'une perturbation externe ($\alpha_{p_h} = 1.5\,rad$) appliquée à l'instant $t = 10s$; commande u_1 générée 96
Figure 4.17 Angle α_v avec la présence d'une perturbation externe ($\alpha_{p_v} = 1.5\,rad$) appliquée à l'instant $t = 10s$; commande u_2 générée 96
Figure 4.18 Système double pendule inversé 98
Figure 4.19 Evolution de la fonction objectif pour l'algorithme ACO proposée et le C-ACO 99
Figure 4.20 Poursuite d'une trajectoire sinusoïdale ; réponses du système double pendule inversé (en rouge réponse du système et en bleu la trajectoire désirée) ; commandes générées $u_{i=1,2}$ 100
Figure 4.21 Test de stabilisation du système double pendule inversé à un angle $\theta_1 = 0\,rad$ (en bleu réponse réelle et en rouge valeur désirée) ; commande u_1 100
Figure 4.22 Test de stabilisation du système double pendule inversé à un angle $\theta_2 = 0\,rad$ (en bleu réponse réelle et en rouge valeur désirée) ; commande u_1 101
Figure 4.23 Poursuite d'une trajectoire sinusoïdale avec une commande décentralisée par modes glissants classiques ; réponses du système double pendule inversé (en rouge réponse du système et en bleu la trajectoire désirée) ; commandes générées $u_{i=1,2}$ 101

CHAPITRE 5

Figure 5.1 Structure de commande globale proposée 108

Figure 5.2 Système simple pendule inversé 109
Figure 5.3 Réponses du système (positions angulaire du pendule) $(y(t) = x_1)$ en bleu et trajectoire désirée y_m en rouge, obtenues en utilisant les commandes adaptatives synthétisées, basées respectivement sur les algorithmes (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs . . 112
Figure 5.4 Réponses du système (vitesse angulaire du pendule) x_2 en bleu et trajectoire désirée $\dot{y}_m$ en rouge, obtenues en utilisant les commandes adaptatives synthétisées, basées respectivement sur les algorithmes (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs 113
Figure 5.5 Signaux de commandes générés relatifs aux commandes adaptatives synthétisées, basées respectivement sur les algorithmes (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs 114
Figure 5.6 Evolution de la fonction objectif F_{obj} en fonction des itérations, pour les algorithmes,GHSACO, HS, IHS, GHS, PSO et GAs respectivement . . 115

Liste des Tableaux

CHAPITRE 3

Tableau 3.1 Caractéristiques des fonctions du benchmark utilisées dans notre étude expérimentale, avec S correspond au domaine de définition de chaque fonction et D est le nombre de dimensions . 61

Tableau 3.2 Comparaison entre GHSACO, GHS, IHS et HS en termes de valeur moyenne des minimum globaux et l'écart type (*std*.) obtenus durant les 30 essais, avec $D = 30$ 63

Tableau 3.3 Résultat du test Wilcoxon entre GHSACO et GHS, IHS et HS respectivement, avec $D = 30$ 64

Tableau 3.4 Comparaison entre GHSACO, GHS, IHS et HS en termes de valeur moyenne des minimum globaux et l'écart type (*std*.) obtenus durant les 25 essais avec $D = 50$ 64

Tableau 3.5 Résultat du test Wilcoxon entre GHSACO et GHS, IHS et HS respectivement, avec $D = 50$ 64

CHAPITRE 4

Tableau 4.1 Définition des bornes de paramètres 92

Tableau 4.2 Les valeurs finales des paramètres 93

Tableau 4.3 Comparaison entre l'algorithme ACO proposé et le C-ACO 93

CHAPITRE 5

Tableau 5.1 Comparaison entre GHSACO, HS, IHS, GHS, PSO et GAs en termes de valeurs moyennes, mauvaises et meilleures des optimums globaux et l'écart type (*std*.) obtenus dans les 20 essais, ainsi que les six commandes adaptatives synthétisées en terme de critères de performance J_1 and J_2 111

Introduction générale

À l'heure actuelle, les problèmes d'optimisation constituent l'une des préoccupations majeures des chercheurs dans le monde. En effet, la complexité des problèmes d'optimisation rendent leurs solutions difficiles et parfois impossibles par les approches conventionnelles.

Parmi les domaines favoris pour l'application des algorithmes d'optimisation, on peut citer entre autres ; la conception des systèmes de commande automatique, le traitement du signal, la recherche opérationnelle, etc…

Généralement, un problème d'optimisation est défini par un ensemble de variables, une fonction objectif et un ensemble de contraintes. Les variables du problème dites aussi variables de conception ou de décision peuvent être de différentes nature (réelle, entière, booléenne. etc.) et peuvent exprimer des données qualitatives ou quantitatives. L'espace d'état, appelé aussi domaine de recherche, représente l'enveloppe de toutes les solutions possibles du problème. Il est en général fini, puisque les méthodes de résolution opèrent dans des espaces bornés. Pour des raisons pratiques et de temps de calcul, cet espace doit être fini. Cette limitation n'est pas gênante, puisqu'en général le décideur défini a priori le domaine de définition de chaque variable. Enfin, même dans le cas des problèmes à variables continues, une certaine granularité est définie [Soch 08, Bilch 95]. La fonction objectif définit le but à atteindre, on cherche à minimiser ou à maximiser celle-ci. L'ensemble des contraintes est en général un ensemble d'égalités ou d'inégalités que les variables de l'espace de recherche doivent satisfaire. Ces contraintes imposent les limites de l'espace de recherche.

Les méthodes d'optimisation recherchent un point ou un ensemble de points dans l'espace de recherche satisfaisant l'ensemble des contraintes, et maximisant ou minimisant la fonction objectif. Parmi ces méthodes, les *métaheuristiques* qui sont des algorithmes généraux d'optimisation applicables à une grande variété de problèmes. Elles sont utilisées généralement quand les méthodes classiques ne sont plus efficaces. Les métaheuristiques ont comme caractéristiques communes, de part leurs caractères stochastiques, c.-à-d. qu'une partie de la recherche est conduite de façon aléatoire, elles sont inspirées d'analogies avec la réalité : physique (recuit simulé,...), biologie (algorithmes évolutionnaires, recherche tabou,...) ou éthologie (colonies de fourmis,...). En plus de leur aspect stochastique, les métaheuristiques sont généralement itératives, c.-à-d. qu'un même schéma de recherche est appliqué plusieurs fois au cours de l'optimisation, et d'une manière directe, c.-à-d. qu'elles n'utilisent pas l'information du gradient de la fonction objectif. Ainsi, l'un des points clés de ces approches et leur capacité à réaliser un équilibre entre la *diversification* et l'*intensification* de la recherche. D'un côté, la

diversification vise à échantillonner l'espace de recherche de façon large de sorte que la recherche ne soit pas concentrée sur une zone particulière de l'espace de recherche. De l'autre côté, l'intensification a pour objectif d'augmenter l'effort de recherche dans les zones les plus prometteuses, aux alentours des bonnes solutions, pour ainsi atteindre la solution optimale. De plus, les métaheuristiques tirent en particulier leur intérêt de leur capacité relative à éviter les optimums locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de points comme méthode de recherche.

Parmi les métaheuristiques qui ont connues une évolution remarquable durant les deux dernières décennies par le nombre important de travaux publiés, on peut citer ; les algorithmes de colonies de fourmis (*Ant colony optimization* ACO) [Dori 91, Dori 92, Dori 96] et l'algorithme de recherche d'harmonie (*Harmony Search* HS) [Geem 01]. En effet, ces deux méthodes d'optimisation constituent l'objet principale du travail présenté dans cette thèse [Allo 12a, Allo 15a, Allo 15b].

Les algorithmes ACO, ont été initialement introduits par Marco Dorigo et ses collaborateurs [Dori 91, Dori 92, Dori 96]. Ils s'inspirent du comportement des fourmis réelles à la recherche de nourriture [Dene 90]. En effet, celles-ci parviennent à trouver le chemin le plus court entre leur nid et une source de nourriture, sans pour autant avoir des capacités cognitives individuelles très développées. Les fourmis explorent d'abord les environs de leur nid en effectuant une marche aléatoire. Le long de leur chemin entre la source de nourriture et le nid, elles déposent sur le sol une substance chimique volatile appelée *phéromone* afin de marquer certains chemins favorables qui devraient guider leurs congénères à la source de nourriture [Dori 05]. Après un certain temps, le chemin le plus court entre le nid et la source de nourriture présente une plus forte concentration de phéromone et, par conséquent, attire plus de fourmis. Les colonies de fourmis artificielles exploitent cette caractéristique pour construire des solutions à un problème d'optimisation.

Les algorithmes ACO ont été principalement utilisés pour produire des solutions quasi-optimales au problème du voyageur de commerce (*Traveling Salesman Problem* TSP [Colo 91, Dori 97b]), puis, plus généralement, aux problèmes d'optimisation combinatoire [Yang 10]. Récemment, leur emploi se généralise à plusieurs domaines, depuis l'optimisation continue [Jing 11, Tian 14] jusqu'à la classification [Mart 07, Khal 14], la commande des processus [Baoj 07, Cast 15, Osha 15, Allo 12b] ou encore le traitement d'image [Chen 14, Chen 13].

L'algorithme HS (Harmony Search) est une métaheuristique relativement récente. Elle a été proposée par le professeur Geem et ses collaborateurs, dans leur célèbre article [Geem 01]. À l'opposé des autres métaheuristiques qui s'inspirent des phénomènes naturels, cette méthode d'optimisation s'inspire du processus de recherche de la meilleure harmonie musicale. L'algorithme HS a été appliqué avec succès dans divers domaines : dans des applications

industrielles [Geem 08], dans les systèmes électriques [Kudi 11], dans les sciences médicales [Panc 09], dans la commande des systèmes [Shar 10, Shar 13, Wang 13a, Wang 13b, Allo 15b], dans le génie civil [Dege 12], dans les technologies de l'information [Zhan 09], etc…

Cependant, comme toutes les méthodes stochastiques, les algorithmes ACO et HS présentent des inconvénients communs. Parmi ceux-ci, nous citons le problème de convergence prématurée, qui peut conduire ces algorithmes à stagner dans des optimums locaux ainsi leur vitesse de convergence devient lente.

Le travail présenté dans cette thèse constitue une contribution pour l'amélioration des deux algorithmes sus cités. Ainsi, nous proposons deux nouvelles méthodes permettant l'amélioration de ces deux métaheuristiques afin de remédier aux problèmes cités ci-dessus, notamment celui de la convergence prématurée. En effet, de nouvelles versions pour les algorithmes ACO et HS sont proposées. La première méthode proposée [Allo 12a], relative aux algorithmes ACO, est basée sur l'introduction de quelques modifications dans la structure de l'algorithme ACO classique (*conventional ACO* C-ACO) [Dori 96], plus précisément sur les deux règles suivantes ; la règle de déplacement des fourmis ainsi la règle de mise-à-jour des pistes de phéromone.

La seconde méthode proposée dite GHSACO [Allo 15a], est issue d'une hybridation entre une variante d'algorithme HS, appelée algorithme de recherche de d'harmonie globale (*Global Best Harmony Search* GHS) [Omra 08] et un algorithme de type C-ACO.

Afin de tester leur efficacité, les deux algorithmes développés (ACO amélioré et le GHSACO) sont appliqués pour résoudre des problèmes de calcul optimal des paramètres d'ajustement d'une commande décentralisée par modes glissants des systèmes non linéaires interconnectés [Allo 12a] et pour l'apprentissage des réseaux de neurones flous récurrents (*Recurrent Fuzzy Neural Networks* RFNN*s*)[Lee 00] intégrés dans une structure de commande adaptative d' une classe de systèmes non-linéaires [Allo 15a].

Le mémoire de thèse est organisé comme suit:

Le premier chapitre, à caractère introductif et bibliographique, dresse un état de l'art non exhaustif des métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoire mono-objectifs. Ces méthodes, peuvent être classées en deux grandes catégories ; celles issues de la théorie de l'évolution dits algorithmes évolutionnaires et celles issues de l'intelligence en essaim. Les méthodes appartenant à ces deux classes et les plus citées dans la littérature seront données.

Le second chapitre, présente d'une manière détaillée tout ce qui concerne les algorithmes ACO, leurs origines, leurs variantes les plus célèbres appliquées au problème TSP, leur formalisme et finalement leurs propriétés fondamentales. Une partie importante de ce

chapitre est consacrée à la nouvelle variante proposée, elle comprend la présentation de toutes les adaptations et modifications que nous avons apportées.

Dans le troisième chapitre, nous décrivons l'algorithme GHSACO, que nous avons élaboré, à travers une hybridation faite entre les deux algorithmes GHS et C-ACO. L'algorithme développé, est caractérisé par un mécanisme d'improvisation différent de celui de GHS. Une étude par simulation et des comparaisons à différentes méthodes de la littérature, via différents benchmarks classiques, sont également présentées.

Le quatrième chapitre est consacré à l'application de la variante des algorithmes ACO développée dans le deuxième chapitre pour résoudre le problème de calcul des paramètres d'ajustement d'une commande décentralisée par modes glissants appliquée aux systèmes non linéaires interconnectés. Les fondements théoriques de commande par modes glissants, le principe de la méthode de commande proposée ainsi les résultats obtenus, sont également illustrés.

Le cinquième et dernier chapitre présente l'application de l'algorithme GHSACO, dans le problème d'apprentissage des RFNN*s* intégrés dans une structure de commande adaptative indirecte basée sur un contrôleur superviseur et appliquée à une classe de systèmes non-linéaires.

Nous terminons finalement ce manuscrit par une conclusion dans laquelle nous récapitulons nos contributions puis nous donnons quelques perspectives que nous avons pu dégagé suite aux travaux effectués.

CHAPITRE 1

Métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoire mono-objectifs : Etat de l'art

1.1 Introduction

La recherche de solution optimale d'un problème constitue une préoccupation importante dans le monde actuel, qu'il s'agisse d'optimiser le temps, le confort, la sécurité, les coûts ou les gains. En effet, de nombreux problèmes scientifiques, sociaux-économiques et techniques contiennent des paramètres devant être ajustés pour donner un résultat souhaitable. Ceux-ci peuvent être considérés comme des problèmes d'optimisation et leur résolution représente un sujet central en recherche opérationnelle.

Plusieurs techniques ont été proposées pour résoudre ce genre de problèmes, notamment les problèmes dits « difficiles », en déterminant des solutions qui ne sont pas strictement optimales, mais qui s'en approchent. Ces méthodes, appelées heuristiques et métaheuristiques. Elles sont généralement inspirées de métaphores biologiques (algorithmes évolutionnaires), d'évolution culturelle des populations (algorithmes culturels [Reyn 94]) ou du comportement collectif des d'espèces vivantes spécifiques (insectes sociaux, oiseaux migrateurs, poisson,..).

Dans se premier chapitre nous présentons un état de l'art sur les métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoire mono-objectifs.
On distingue, généralement, deux grandes familles de métaheuristiques : celles basées sur l'évolution itérative d'une solution unique et celles qui manipulent en parallèle toute une population de solutions.

Les approches de la première famille, appelées aussi méthodes de trajectoire, commencent toujours leur processus de recherche avec une seule solution initiale et s'en éloignent progressivement, en construisant une trajectoire dans l'espace de recherche. Ces méthodes englobent essentiellement la méthode de descente [Papa 82], la méthode du recuit simulé [Kirk 83], la méthode de recherche avec tabous [Glov 86], la méthode de recherche gloutonne aléatoire adaptative (*Greedy Randomized Adaptive Search Procedure* GRASP) [Feo 89, Feo 95], La recherche à voisinage variable [Mlad 95, Mlad 97], la recherche locale itérée [Stüt 98], et leurs variantes.

Contrairement aux algorithmes partant d'une solution singulière, les métaheuristiques à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. On distingue dans cette catégorie, les algorithmes évolutionnaires, qui sont une famille d'algorithmes issus de la théorie de l'évolution par la sélection naturelle et les algorithmes d'intelligence en essaim qui, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels.

1.2 Métaheuristiques pour l'optimisation mono-objectif difficile

1.2.1 Problème d'optimisation mono-objectif

Un problème d'optimisation mono-objectif au sens général est défini comme la recherche, parmi un ensemble de solutions possibles S (appelé aussi espace de décision ou espace de recherche), de la (ou des) solution(s) x^* qui rend(ent) minimale (ou maximale) une fonction mesurant la qualité de cette solution. Cette fonction dite fonction objectif ou fonction coût.
Si l'on pose $f: S \rightarrow \mathbb{R}$ la fonction objectif à minimiser (respectivement à maximiser) à valeurs dans $\mathbb{R}$, le problème revient alors à trouver l'optimum $x^* \in S$ tel que $f(x^*)$ soit minimal (respectivement maximal).

La résolution d'un problème d'optimisation comprend principalement la recherche de la meilleure solution possible à ce problème, c.-à-d. l'optimum global. Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : on parle alors d'optimums locaux.

Cette notion est illustrée dans la figure 1.1. La seule hypothèse faite sur S est qu'il s'agit d'un espace topologique, c.-à-d. sur lequel est définie une notion de voisinage. Cette hypothèse est nécessaire pour définir la notion de solutions locales du problème d'optimisation. On peut alors définir un optimum local (relativement au voisinage V) comme la solution x^* de S telle que $f(x^*) \leq f(x);\ \forall\, x \in V(x^*)$.

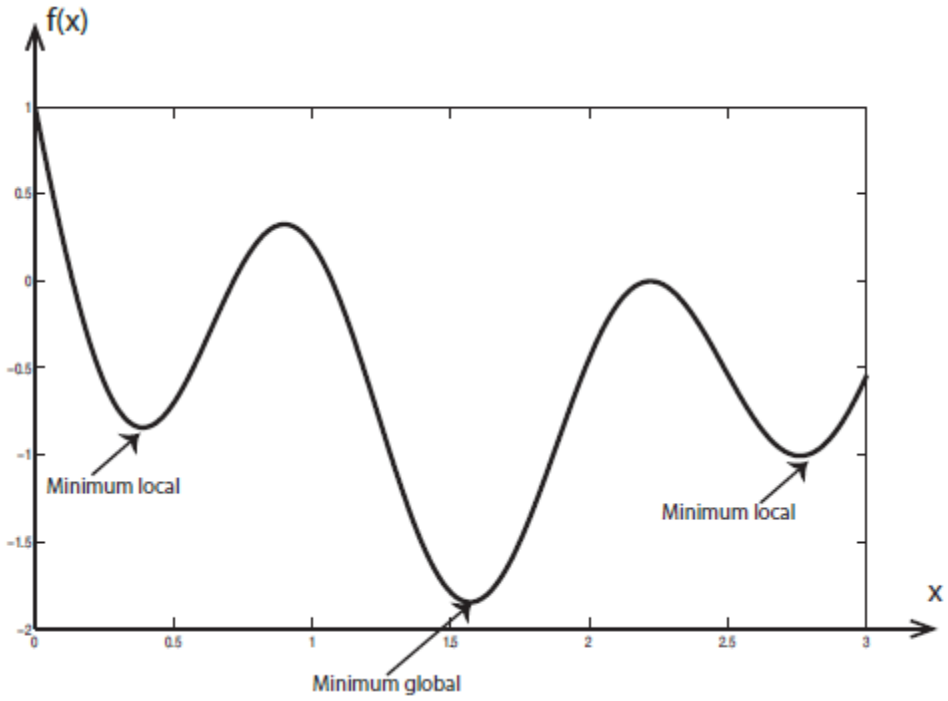

Figure 1.1 : Différence entre un optimum global et des optima locaux [Bous 13b].

1.2.2 Optimisation difficile

La pratique des problèmes d'optimisations, montre que les méthodes d'optimisation déterministes (ou exactes) ne sont pas adaptées à toutes les problématiques, et donc certains problèmes sont trop complexes à résoudre par ces méthodes. Parmi ces problématiques, nous pouvons citer l'existence de discontinuités, l'absence de convexité stricte (multimodalité), la non-dérivabilité, la présence de bruit ou encore la fonction objectif peut ne pas être définie précisément. En outre, ces méthodes peuvent avoir un temps de résolution trop long. Dans ce cas, le problème d'optimisation est dit difficile, car aucune méthode exacte n'est capable de le résoudre en un temps raisonnable. Il est alors nécessaire d'avoir recours à des heuristiques de résolution dites méthodes approchées, qui fournissent un résultat sans garantie de l'optimalité.

L'optimisation difficile peut se diviser en deux catégories : les problèmes à variables discrètes et les problèmes à variables continues. De façon générale, un problème d'optimisation à variables discrètes, dit aussi combinatoire consiste à trouver, dans un ensemble discret, une solution réalisable. Le problème majeur réside ici dans le fait que le nombre de solutions réalisables est généralement très élevé, donc il est très difficile de trouver la meilleure solution dans un temps « raisonnable ». Les problèmes à variables discrètes rassemblent les problèmes de type NP-complets (*Non-deterministic Polynomial time* NP-complets ; Problèmes Non-déterministes Polynomiaux-complets), tels que le problème TSP. L'utilisation d'algorithmes

d'optimisation stochastiques, tels que les métaheuristiques, permet de trouver une solution approchée en un temps raisonnable.

Dans le second type, les variables du problème d'optimisation sont continues. C'est le cas p. ex. des problèmes d'identification, où l'on cherche à minimiser les/l'erreur(s) entre la/les sortie(s) du système réel et celle/celles du modèle modélisant ce dernier. Ce type de problèmes est moins formalisé que le précédent. En effet, la grande majorité des métaheuristiques existantes ont été créées pour résoudre des problèmes à variables discrètes [Bous 13a]. Cependant, la nécessité croissante de méthodes pouvant résoudre ce type de problèmes a poussé les chercheurs à adapter leurs méthodes au cas continu. Les difficultés les plus habituelles de ce type de problèmes sont généralement les corrélations non identifiées entre les variables, le caractère bruité des fonctions ou encore des fonctions objectifs qui ne sont accessibles que par dispositif expérimental.

Il est à noter, qu'il existe des problèmes à variables mixtes (c.-à-d. le problème présent à la fois des variables discrètes et continues) [Yunq 12].

1.2.3 Algorithmes d'optimisation approchée

1.2.3.1 Heuristiques

L'utilisation de méthodes déterministes n'est pas toujours possible pour un problème donné à cause d'un certain nombre de contraintes, telles que le temps de calcul souvent important ou bien la difficulté, voire l'impossibilité dans certains cas, d'une définition séparée du problème. Pour surmonter ces contraintes, des méthodes spécifiques dites : heuristiques ont été utilisées. Le mot heuristique dérive du grec ancien *heuriskêin* qui signifie « trouver ». Il qualifie tout ce qui sert à la découverte et à l'exploitation. Il est à souligner ici qu'une méthode heuristique peut être déterministe ou stochastique.

Une heuristique, ou méthode approximative, est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation difficile. On oppose les méthodes approchées aux méthodes exactes, qui trouvent toujours l'optimum si on leur en laisse le temps (énumération complète, méthodes arborescentes, programmation dynamique). Les approches heuristiques, contournent le problème de l'explosion combinatoire [Comb 98] en faisant délibérément des impasses et n'explorent qu'une partie de l'espace des combinaisons. Une méthode heuristique est généralement conçue pour un problème particulier, en s'appuyant sur sa structure propre.

1.2.3.2 Métaheuristiques

Sans de profonds changements dans leurs structures algorithmiques, plusieurs heuristiques plus poussées, adaptables à un grand nombre de problèmes différents, ont été mises au point et

ont donné naissance à une nouvelle famille d'algorithmes d'optimisation stochastiques : les métaheuristiques. Le terme métaheuristique a été inventé par Fred Glover en 1986, lors de la conception de la recherche tabou [Glov 86].

Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile, pour lesquels les méthodes classiques échouent. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, d'où le qualificatif méta. Leur capacité à optimiser un problème à partir d'un nombre minimal d'informations est contrebalancée par le fait qu'elles n'offrent aucune garantie quant à l'optimalité de la meilleure solution trouvée. Cependant, du point de vue de la recherche opérationnelle, ce constat n'est pas forcément un désavantage, puisque l'on préfère toujours une approximation de l'optimum global trouvée rapidement à une valeur exacte trouvée dans un temps rédhibitoire.

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. La plupart des métaheuristiques utilisent des processus aléatoires comme moyens de rechercher l'information et de faire face à des problèmes comme l'explosion combinatoire [Comb 98].

Les métaheuristiques peuvent être considérées comme des algorithmes aléatoires itératifs, où elles manipulent une ou plusieurs solutions à la recherche de l'optimum global. Les itérations successives doivent permettre de passer progressivement d'une mauvaise solution à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti ou en une précision demandée. Ces méthodes tirent leur efficacité du fait qu'elles sont moins facilement piégeables dans des optima locaux, car elles acceptent, au cours du traitement, des dégradations de la fonction objectif et la recherche est souvent menée par une population de points et non un point unique.

Les métaheuristiques sont majoritairement conçues pour résoudre des problèmes à variables discrètes, mais font de plus en plus l'objet d'adaptations aux problèmes à variables continues. De plus, de par leur variété et leur capacité à résoudre des problèmes très divers, les métaheuristiques sont assez facilement sujettes à extensions. Parmi celles-ci, on trouve des métaheuristiques pour : optimisation multi-objectif, optimisation multimodale, optimisation de problèmes bruités, optimisation dynamique, …etc.

Le monde des métaheuristiques est un monde en constante évolution. De nombreuses méthodes sont proposées chaque année pour tenter d'améliorer la résolution des problèmes les plus complexes. Du fait de cette activité permanente, un grand nombre de classes de métaheuristiques existe actuellement [Bous 13a].

1.3 Métaheuristiques à population de solutions issues de la théorie de l'évolution (algorithmes évolutionnaires)

Les algorithmes évolutionnaires (*Evolutionary Algorithms* EAs) sont des techniques de recherche inspirées de l'évolution biologique des espèces, apparues à la fin des années 1950. Ces algorithme, se basent sur les grands principes rencontrés dans la nature et notamment celui de l'évolution des espèces et de la sélection naturelle énoncés par Charles Darwin [Darw 1859].

Le principe de base d'un algorithme évolutionnaire est très simple. Un ensemble de N points dans un espace de recherche, choisi a priori au hasard, constituent la population initiale ; chaque individu x de la population possède une certaine performance, qui mesure son degré d'adaptation à l'objectif visé : dans le cas de la minimisation p. ex. d'une fonction objectif F_{obj}, x est d'autant plus performant que $F_{obj}(x)$ est plus petit. Un EA consiste à faire évoluer progressivement, par générations successives, la composition de la population, en maintenant sa taille constante. Au cours des générations, l'objectif est d'améliorer globalement la performance des individus ; le but étant d'obtenir un tel résultat en imitant les deux principaux mécanismes qui régissent l'évolution des êtres vivants, selon la théorie de Darwin :

– la sélection, qui favorise la reproduction et la survie des individus les plus performants,

– la reproduction, qui permet le brassage, la recombinaison et les variations des caractères héréditaires des parents, pour former des descendants aux potentialités nouvelles.

Selon la façon de l'emploi et de la présentation des deux principes Darwiniens cités ci-dessus dans le modèle artificiel, quatre approches différentes ont été proposées [Bäck 97a] : les algorithmes génétiques (*Genetic Algorithms* GAs) [Holl 75], la programmation génétique (*Genetic Programming* GP) [Koza 89, Koza 90], les stratégies d'évolution (*Evolutionary Strategy* ES) [Rech 65] et la programmation évolutive (*Evolutionary Programming* EP) [Foge 62, Foge 66] que nous allons décrire par la suite.

De plus, plusieurs autres modèles d'algorithmes évolutionnaires ont été proposés dans la littérature [Bous 13a]. Nous allons présenter les plus connus dans la littérature. Parmi ceux-ci, nous nous intéressons principalement à l'algorithme HS. Ce dernier est considéré comme un algorithme évolutionnaire, bien qu'il n'utilise pas tous les concepts.

La structure générale d'un algorithme évolutionnaire est donnée par le pseudo code (Algorithme 1.1) illustré ci-dessous.

Algorithme 1.1 Structure de base d'un algorithme évolutionnaire [Bena 10]

$t \longleftarrow 0$
Initialiser la population $P(t)$
Evaluer $P(t)$
Répéter
$t \longleftarrow t + 1$
Sélectionner les parents
Appliquer les opérateurs génétiques
Evaluer la population des enfants crées
Créer la nouvelle population $P(t)$ en utilisant une stratégie de sélection
Tant que (condition d'arrêt n'est pas satisfaite)

1.3.1 Les algorithmes génétiques (Genetic Algorithms)

La théorie des algorithmes génétiques (*Genetic Algorithms* GAs) a été initialement développée par John Holland en 1960 et a été entièrement élaborée dans son livre *"Adaptation in Natural and Artificial Systems"*, publié en 1975 [Holl 75]. Son objectif initial n'était pas le développement d'un algorithme d'optimisation, mais plutôt la modélisation du processus d'adaptation, et montrer comment ce processus pourrait être utile au sein d'un système de calcul. Les GAs sont des techniques de recherche stochastique qui utilisent les concepts de la génétique mendélienne et ceux de l'évolution darwinienne.

Une population d'individus choisis dans l'espace de recherche, souvent d'une façon aléatoire, sert de solutions candidates au problème à optimiser [Talb 97]. Les individus de cette population sont évalués grâce à une fonction d'adaptation dite ("fitness"). Un mécanisme de sélection est ensuite utilisé pour sélectionner les individus à utiliser comme parents pour ceux de la prochaine génération. Ces individus seront ensuite croisés puis mutés pour constituer la nouvelle progéniture. La prochaine génération est enfin constituée par un mécanisme de remplacement entre les parents et leur progéniture [Reev 02]. Ce processus est répété jusqu'à satisfaction d'une certaine condition d'arrêt. Ces différentes étapes sont détaillées dans les paragraphes suivants.

a. Représentation des solutions

La représentation définit la façon dont les individus de la population (solutions candidates) seront codés afin d'être évalués par la fonction objectif. Dans les travaux de Hollande [Holl 75] ces solutions sont représentées par des chaînes binaires de longueur fixe. Cependant, l'utilisation de ce type de codage est très difficile dans le cas de la représentation des solutions de plusieurs problèmes d'optimisation p. ex. en industrie ou en ingénierie [Gen 00]. Différents types de codage peuvent être utilisés, en fonction du problème, p. ex., binaire, réel, entier ou

permutations des éléments. Il est important de noter que, la sélection d'un bon codage est un choix critique dans la conception des AGs [Bäck 00].

b. Population des solutions candidates

Une population est formée par un ensemble d'individus, appelés également chromosomes, typiquement d'une taille fixe. Chaque individu représente une solution possible au problème à résoudre et consiste en une séquence d'éléments plus petits, appelés gènes. Chaque gène peut prendre des valeurs différentes, ou allèles. Au début de l'évolution, une population initiale est générée de façon aléatoire, mais il existe plusieurs heuristiques pour créer cette population initiale. Cette population évolue au fil des générations. A la fin du processus, il est prévu qu'un GA converge vers une région de l'espace de recherche qui contient une solution satisfaisante. La taille d'une population est un élément important dans le paramétrage des GAs, qui influence la complexité de l'algorithme. Typiquement, la taille de la population est constante, mais il existe plusieurs versions des GAs où cette taille est dynamique [Lobo 05].

c. Fonction objectif

La fonction objectif sert à mesurer la qualité des individus de la population. Cette mesure est soit à maximiser soit à minimiser. Généralement, la définition de cette fonction représente une tâche difficile et une étape très essentielle. Une mauvaise formulation de cette fonction pourra mener les GAs vers des solutions non souhaitables.

d. Sélection

La sélection [Bäck 97b] est utilisée pour choisir les parents qui vont survivre pour produire les enfants de la prochaine génération. Les parents avec les meilleures valeurs d'adaptation (fonction objectif) ont plus de probabilité d'être choisi pour l'accouplement. Ils existent plusieurs méthodes de sélection, parmi elles : la méthode dite roulette biaisée, la sélection par rang, la sélection par tournoi, la sélection de Boltzmann et bien d'autres [Jeba 09]. La roulette biaisée sélectionne des individus en fonction de leur valeur d'adaptation relative : des individus de plus grande valeur d'adaptation ont une probabilité plus élevée d'être sélectionnés. Dans la sélection par tournoi, un certain nombre (taille du tournoi) des individus est choisi au hasard, et le meilleur individu est sélectionné. La méthode de sélection par rang est basée sur le rang des individus en fonction de leurs valeurs d'adaptation. Chaque individu a une certaine probabilité d'être choisi, en fonction de son rang.

e. Croisement

L'opérateur de croisement [Herr 03], [Bäck 00] vise à orienter la recherche vers les régions prometteuses de l'espace de recherche. Pour cela, les individus (enfants) crées par croisement héritent les meilleures caractéristiques de leurs parents et peuvent exploiter au mieux l'espace de recherche. L'exemple le plus simple de ce type d'opérateur est appelé croisement à un point. Il consiste à choisir au hasard un point de croisement pour chaque couple d'individus

sélectionné. Ce point divise le génome en deux sous-chaînes. On échange ensuite les deux sous-chaînes terminales de chacun des deux chromosomes, ce qui produit deux enfants. La figure 1.2 illustre ce fonctionnement.

Un autre opérateur de croisement bien connu est appelé croisement uniforme, où les gènes des deux parents sont entrelacées selon un masque aléatoire. On trouve également l'opérateur appelé "*simulated bianry crossover* SBX" qui est une autre version de croisement à un point dans le cas d'un codage réel.

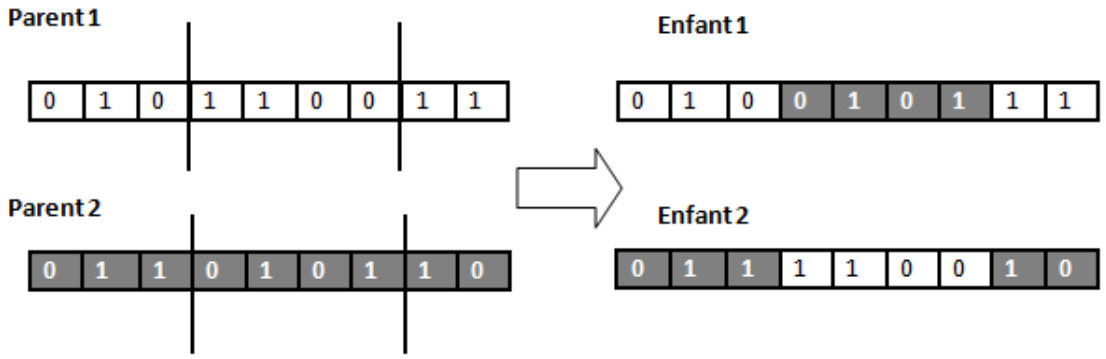

Figure 1.2 : Exemple de croisement en représentation binaire.

f. Mutation

La mutation [Wort 01] consiste à altérer la composition génétique de quelques chromosomes pour créer une diversité génétique au sein de la population. Cet opérateur permet d'explorer de nouveaux espaces de recherches et permet à GA de donner chance à d'autres solutions candidates. La figure 1.3 montre un exemple de mutation pour un individu codé en représentation par permutation. Cependant, les deux opérateurs génétiques, croisement et mutation sont appliqués avec une certaine probabilité. En général une faible probabilité est accordée à la mutation alors que le croisement se voit accordé une forte probabilité. En effet, un taux de mutation élevé va rendre GA comme une recherche aléatoire et ne va pas converger vers la solution optimale.

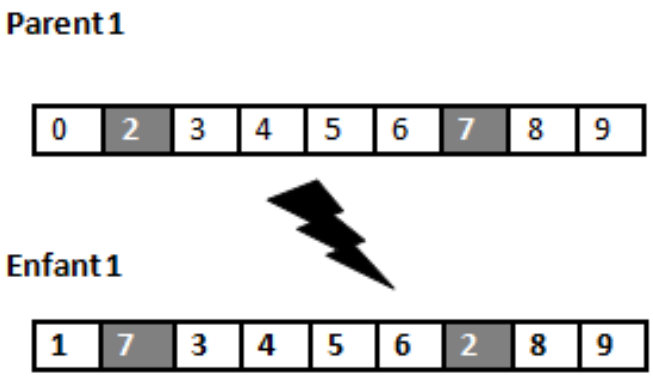

Figure 1.3 : Exemple de mutation en représentation par permutation.

g. Remplacement

Le remplacement est l'opération qui consiste à passer d'une génération à une autre [Bäck 97b]. Il peut se faire de deux façons : la première consiste à garder uniquement les descendants, c'est l'approche générationnelle ; la deuxième consiste à fusionner les deux populations en choisissant les meilleures de leurs individus (parents et fils). Une version possible de ce fusionnement est de garder une petite fraction des meilleurs parents et de compléter la population par la meilleure progéniture, cette approche est appelée *élitisme* [Eibe 04].

Algorithme 1.2 Pseudo-code de l'algorithme génétique standard [Enge 10]

Entrées : NP : Taille de la population ; p_x : Probabilité de croisement ; p_m: Probabilité de mutation
Sorties : Le meilleur individu
$t \longleftarrow 0$
Choisir NP individus (I_i) d'une façon aléatoire pour former la population $P(t)$
Tant que *Condition d'arrêt* $\neq$ *Vrai* faire
  Évaluer ($P(t)$)
  $P_1(t) \longleftarrow$ *Selection* ($P(t)$)
  $P_2(t) \longleftarrow$ *Croisement* ($P_1(t)$)
  $P_3(t) \longleftarrow$ *Mutation* ($P_2(t)$)
  $P(t) \longleftarrow$ *Remplacement* ($P_1(t), P_3(t)$)
  $t \longleftarrow t + 1$
Fin

1.3.2 Programmation génétique

La programmation génétique (*Genetic Programming* GP) initialement proposé par Koza [Koza 89, Koza 90] s'applique sur un ensemble de programmes informatiques : la *population*. Sa dynamique est une répétition de plusieurs étapes, qui forment une *génération*. D'une génération

à l'autre, la GP transforme la population de programmes de manière stochastique, en une population de programmes de meilleure qualité. Tout comme le processus de l'évolution naturelle, la GP est un processus stochastique, qui intègre des évènements aléatoires. Le résultat ne peut donc en être garanti. Cependant, c'est précisément l'essence stochastique de cette heuristique qui lui permet d'échapper aux inconvénients des méthodes déterministes. Ainsi la GP peut trouver des solutions issues de régions de l'espace de recherche inexplorées (par les méthodes classiques).

1.3.3 Stratégies d'évolution

Les stratégies d'évolution (*Evolution strategy* ES) forment une famille de métaheuristiques d'optimisation. Elles sont inspirées de la théorie de l'évolution. Ce modèle fut initialement proposé par Rechenberg [Rech 65]. Il constitue, à ce titre, la première véritable métaheuristique et le premier algorithme évolutionnaire. Dans sa version de base, l'algorithme manipule itérativement un ensemble de vecteurs de variables réelles, à l'aide d'opérateurs de mutation et de sélection. L'étape de mutation est classiquement effectuée par l'ajout d'une valeur aléatoire, tirée au sein d'une distribution normale. La sélection s'effectue par un choix déterministe des meilleurs individus, selon la valeur de la fonction objectif. Ces méthodes utilisent un ensemble de μ "parents" pour produire λ "enfants". La production de chaque enfant nécessite la "recombinaison" de ρ parents. Une fois produits, les enfants sont mutés. L'étape de sélection peut s'appliquer, soit uniquement aux enfants, soit à l'ensemble *(enfants + parents)*. Dans le premier cas, l'algorithme est noté $(\mu, \lambda) - \text{ES}$ et $(\mu + \lambda) - \text{ES}$ dans le second [Scho 97].

Une itération de l'algorithme général procède comme suit :

1- À partir d'un ensemble de μ parents,
2- Produire une population de λ enfants :
   a. Choisir ρ parents,
   b. Recombiner les parents pour former un unique individu,
   c. Muter l'individu ainsi crée,
3- Sélectionner les μ meilleurs individus.

1.3.4 Programmation évolutive

La programmation évolutive (*Evolutionary Programming* EP) a été introduite par Laurence Fogel en 1966 [Foge 62, Foge 66] dans la perspective de créer des machines à état fini (*Finite State Machine* FSM) dans le but de prédire des événements futurs sur la base d'observations antérieures. La programmation évolutive suit le schéma classique des algorithmes évolutifs comme le montre le pseudo-code (Algorithme 1.3) illustré ci-dessous :

Algorithme 1.3 Pseudo-code de l'algorithme Programmation Evolutive de base [Enge 10]

$t \longleftarrow 0$
Initialiser les paramètres de l'algorithme
Générer aléatoirement (initialisation) une population $P(0)$ de NP individus
pour *chaque individu*, $x_i(t) \in P(t)$ faire
  Évaluer la fonction objectif $\mathrm{F}_{\mathrm{obj}}(x_i(t))$
Fin
Tant que *Condition d'arrêt* ≠ *Vrai* faire
  pour *chaque individu*, $x_i(t) \in P(t)$ faire
    Créer un enfant $x_i'(t)$, en appliquant l'opérateur de mutation
    Évaluer la fonction objectif $\mathrm{F}_{\mathrm{obj}}(x_i'(t))$
    Ajouter $x_i'(t)$ à l'ensemble d'enfants $P'(t)$
  end
  Sélectionnez la nouvelle population $P(t+1)$ de $P(t) \cup P'(t)$ en utilisant l'opérateur de sélection.
  $t \longleftarrow t+1$
Fin

1.3.5 Autres algorithmes évolutionnaires

Plusieurs autres méthodes sont également qualifiées d'algorithmes évolutionnaires, bien qu'elles n'en utilisent pas tous les concepts. C'est le cas p. ex. des algorithmes à évolution différentielle (*Differential Evolution* DE) [Storn 97], les algorithmes à estimation de distribution (*Estimation of distribution algorithms* **EDA**) [Mühl 97],les algorithmes culturels (*Cultural algorithms* **CAs**) [Reyn 94], les systèmes immunitaires artificiels (*Artificial Immune Systems* **AIS**)[Cast 99, Cast 00] et l'algorithme HS.

1.3.5.1 Algorithme à évolution différentielle

L'évolution différentielle (*Differential Evolution* DE) est une métaheuristique stochastique d'optimisation, elle a été inspirée par les algorithmes génétiques et les stratégies d'évolution combinées avec une technique géométrique de recherche. Les algorithmes génétiques changent la structure des individus en utilisant la mutation et le croisement, alors que les stratégies d'évolution réalisent l'auto-adaptation par une manipulation géométrique des individus. Ces idées ont été mises en œuvre grâce à une opération simple, mais puissante, de mutation de vecteurs, proposée en 1997 par K. Price et R. Storn [Storn 97]. Même si, à l'origine, la méthode de l'évolution différentielle était conçue pour les problèmes d'optimisation continus et sans contraintes, ses extensions actuelles peuvent permettre de traiter les problèmes à variables mixtes et gèrent les contraintes non linéaires [Lamp 99].

Dans la méthode DE, la population initiale est générée par tirage aléatoire uniforme sur l'ensemble des valeurs possibles de chaque variable. Les bornes inférieures et supérieures des variables sont spécifiées par l'utilisateur selon la nature du problème traité. Après une étape d'initialisation, l'algorithme effectue une série de transformations sur les individus, dans un processus appelé *évolution*[Enge 10].

La population contient N_P individus. Chaque individu $x_{i,G}$ est un vecteur de dimension D, où G désigne la génération :

$$x_{i,G} = \left(x_{1i,G}, x_{2i,G}, \ldots\ldots, x_{Di,G}\right) \text{ avec } i = 1,2, \ldots\ldots, N_P \tag{1.1}$$

L'algorithme DE standard est composé principalement de trois opérations (mutation, croisement et sélection) exactement comme les algorithmes génétiques. A chaque génération, l'algorithme applique successivement ces trois opérations sur chaque vecteur pour produire un vecteur d'essai (trial vector) :

$$x_{i,G+1} = \left(x_{1i,G+1}, x_{2i,G+1}, \ldots\ldots, x_{Di,G+1}\right) \text{ avec } i = 1,2, \ldots\ldots, N_P \tag{1.2}$$

Une opération de sélection permet de choisir les individus à conserver pour la nouvelle génération $(G + 1)$.

a - Mutation

Pour chaque vecteur courant $x_{i,G}$, on génère un vecteur mutant $v_{i,G+1}$ qui peut être créé en utilisant une des stratégies de mutation suivantes :

- Rand/1 :

$$v_{i,G+1} = x_{r1,G} + F \cdot \left(x_{r2,G} - x_{r3,G}\right) \tag{1.3}$$

- Best/1 :

$$v_{i,G+1} = x_{best,G} + F \cdot \left(x_{r1,G} - x_{r2,G}\right) \tag{1.4}$$

- Current to best/1 :

$$v_{i,G+1} = x_{i,G} + F \cdot \left(x_{r1,G} - x_{r2,G}\right) + F \cdot \left(x_{best,G} - x_{i,G}\right) \tag{1.5}$$

- Best/2 :

$$v_{i,G+1} = x_{best,G} + F \cdot \left(x_{r1,G} - x_{r2,G}\right) + F \cdot \left(x_{r3,G} - x_{r4,G}\right) \tag{1.6}$$

- Rand/2 :

$$v_{i,G+1} = x_{r1,G} + F \cdot (x_{r2,G} - x_{r3,G}) + F \cdot (x_{r4,G} - x_{r5G}) \quad (1.7)$$

Les indices $r1$, $r2$, $r3$, $r4$ et $r5 \in \{1, 2, \ldots, \mathrm{N_P}\}$ sont des entiers aléatoires différents. Ils sont choisis différents de l'indice courant i. $x_{best,G}$ est le meilleur individu à la $G^{ème}$ génération. $F \in [0, 2]$ est une valeur constante, appelée *differential weight*, elle contrôle la valeur de la variation différentielle de $(x_{ri,G} - x_{rj,G})$.

b – Croisement

L'opération de croisement est introduite pour augmenter la diversité des vecteurs de paramètres perturbés. Un vecteur d'essai final $u_{i,G+1}$ (nouveau vecteur) est formé durant cette opération selon les deux vecteurs $x_{i,G}$ et $v_{i,G+1}$ comme suit :

$$u_{ij,G+1} = \begin{cases} v_{1i,G+1} \text{ si } (randb(j) \leq CR) \text{ ou } j = rnbr(i) \\ x_{ji,G} \quad \text{si } (randb(j) > CR) \text{ et } j \neq rnbr(i) \end{cases} \quad \text{pour tout } j \in \{1,2, \ldots\ldots, D\} \quad (1.8)$$

D'où, $randb(j)$ représente la $j^{ème}$ valeur procurée un générateur de nombre aléatoire uniforme appartenant à l'intervalle [0,1]. CR est le coefficient de croisement qui appartient aussi à l'intervalle [0,1], il est déterminé par l'utilisateur. $rnbr(i)$ est un indice choisi au hasard dans l'ensemble $\{1,2, \ldots\ldots, \mathrm{N_P}\}$. .

c – Sélection

Pour décider quel vecteur, parmi $u_{i,G+1}$ ou $x_{i,G}$, doit être choisi dans la génération $G + 1$, on doit comparer les valeurs de la fonction objectif de ces deux vecteurs. Dans tous les cas, on garde le vecteur ayant la plus petite valeur de fonction objectif $\mathrm{F_{obj}}()$ en cas de minimisation. Le nouveau vecteur $x_{i,G+1}$ est choisi selon l'expression suivante :

$$x_{i,G+1} = \begin{cases} u_{i,G+1} \; si \; \mathrm{F_{obj}}(u_{i,G+1}) < \mathrm{F_{obj}}(x_{i,G}) \\ x_{i,G} \qquad\qquad\qquad sinon \end{cases} \quad (1.9)$$

Il est important de noter que le bon réglage des différents paramètres de cet algorithme ($\mathrm{N_P}$, F et CR) contribue de façon importante à l'efficacité de ce dernier.

1.3.5.2 Algorithmes à estimation de distribution

Les algorithmes à estimation de distribution (*Estimation of distribution algorithms* EDA) [Mühl 97] utilisent des techniques d'apprentissage pour résoudre des problèmes d'optimisation,

en essayant d'apprendre au fur et à mesure les régions les plus prometteuses de l'espace de recherche. Plus particulièrement, un modèle probabiliste est utilisé pour engendrer des solutions candidates, l'apprentissage est utilisé pour adapter le modèle probabiliste et pour explorer des régions de l'espace de plus en plus prometteuses. Chaque composante (variable) dans un individu est appelée gène. Ces dernières composantes (gènes) sont parfois indépendantes les unes des autres, et d'autres fois corrélées. Des communications et des échanges d'informations sont effectués entre individus à l'aide des opérateurs de sélection et de combinaison. Ce type d'échange permet de combiner des solutions (individus) partielles pour engendrer des solutions partielles et de haute qualité. Le comportement des GAs dépend généralement du choix des opérateurs de sélection, croisement et de mutation, des probabilités de croisement et de mutation, la taille de la population, la vitesse de reproduction d'une génération, le nombre de générations ainsi l'interaction entre les différentes variables.

Cette dernière caractéristique a pour conséquence de produire des enfants de moindre qualité à partir de la combinaison fixe des deux parents, d'où des convergences vers des optimums locaux.

Pour surmonter ce problème, le processus de combinaison des deux parents est remplacé par une génération de nouvelles solutions, selon une distribution de probabilité sur toutes les solutions prometteuses de la génération précédente. Cette nouvelle approche est la base d'un algorithme EDA. Les EDAs ont été introduits dans le domaine évolutionnaire par Mühlenbein et Paab (1996) [Mühl 97].

Un pseudo-code détaillé de cette approche est présenté dans l'algorithme 1.4.

Algorithme 1.4 Pseudo-code d'un algorithme à estimation de distribution (EDS) [Enge 10]

$t \longleftarrow 0$
Générer aléatoirement (initialisation) une population $P(0)$ de N_P individus.
Tant que *Condition d'arrêt* ≠ *Vrai* faire
  $t \longleftarrow t + 1$
  $P^{se}(t-1) \longleftarrow$ Choisir $N \leq N_P$ individus de $P(t-1)$ en utilisant une méthode de sélection.
  $p_t^{prob}(x) = p_t^{prob}(x|P^{se}(t-1)) \longleftarrow$ Estimer la probabilité de distribution des individus choisis.
  $p(t) \longleftarrow$ Échantillonner N individus en utilisant $p_t^{prob}(x)$.
Fin

1.3.5.3 Les algorithmes culturels

Les algorithmes culturels (*Cultural algorithms* CAs) sont des techniques évolutives modélisant l'évolution culturelle des populations [Reyn 94]. Ces algorithmes supportent les

mécanismes de base de changement culturel [Durh 94]. Certains sociologues ont proposé des modèles où les cultures peuvent être codées et transmises à l'intérieur et entre les populations [Durh 94]. En se basant sur cette idée, Reynolds a développé un modèle évolutif dans lequel l'évolution culturelle est considérée comme un processus d'héritage qui agit sur deux niveaux évolutifs différents : le niveau micro-évolutif en termes de transmission de matériel génétique entre individus d'une population et un niveau macro-évolutif en termes de connaissances acquises sur la base des expériences individuelles [Reyn 94].

Ces algorithmes agissent donc sur deux espaces : l'espace de population qui contient un ensemble d'individus qui évoluent en utilisant un modèle évolutif, et l'espace de connaissances qui contient toutes les informations et les connaissances, relatives au problème à résoudre, utilisées pour guider et influencer l'évolution des individus des populations au cours des générations. Ainsi, un protocole de communication est indispensable pour établir une interaction entre ces deux espaces (figure 1.4).

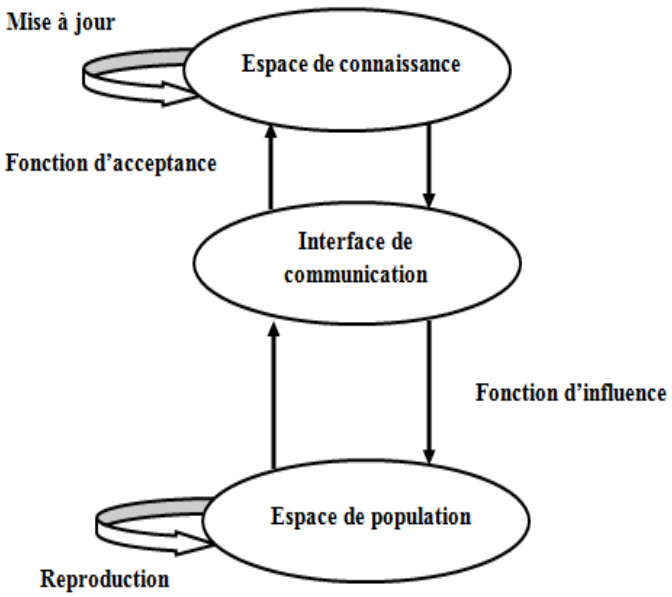

Figure 1.4 : Les composants principaux d'un algorithme culturel.

Ce protocole a deux objectifs principaux, il détermine d'une part quels individus de la population peuvent influencer l'espace de connaissances (fonction d'acceptance), et d'autre part quelles connaissances peuvent influencer l'évolution des populations (fonction d'influence). Le pseudo code illustré ci-dessous (algorithme 1.5) représente la structure de base d'un algorithme culturel.

Algorithme 1.5 Algorithme culturel [Enge 10]

$t \longleftarrow 0$
Créer et initialiser l'espace de connaissances, $B(t)$
Créer et initialiser l'espace de population, $P(t)$
Tant que *Condition d'arrêt* ≠ *Vrai* faire
  Évaluer la fonction d'objectif $\mathrm{F_{obj}}()$ pour chaque $x_i(t) \in P(t)$
  Ajuster ($B(t)$, acceptance $P(t)$)
  Evaluer ($P(t)$, influence $P(t)$)
  $t \longleftarrow t + 1$
  Sélectionnez la nouvelle population
Fin

1.3.5.4 Les systèmes immunitaires artificiels

Les systèmes immunitaires artificiels (*Artificial Immune Systems* AIS) sont apparus dans les années 90, ils sont inspirés du fonctionnement du système immunitaire humain qui est un mécanisme de défense capable d'apprendre [Schi 02]. Ils ont été proposés initialement comme étant des méthodes d'apprentissage notamment dans le domaine de la reconnaissance de formes [Cast 02].

Le système immunitaire d'un organisme est composé de cellules immunitaires nommées "anticorps". Elles sont différenciées par des récepteurs protéiques situés à leur surface. Lorsque des cellules indésirables, nommées "antigènes", sont détectées (microbes, virus, infections...) l'organisme génère un grand nombre d'anticorps différents. L'efficacité de ceux-ci est proportionnelle à leur affinité aux antigènes. Cette affinité dépend de la compatibilité entre les récepteurs protéiques des antigènes et des anticorps. L'organisme va ensuite identifier les anticorps ayant la meilleure affinité. Ceux-ci seront clonés et maturés, afin de modifier leurs marqueurs en espérant obtenir des anticorps encore plus performants, tout en continuant à générer de nouveaux anticorps. De cette façon, les anticorps correspondent de plus en plus aux antigènes, jusqu'à l'élimination de ces derniers. Les meilleurs anticorps sont gardés en mémoire en cas d'agression ultérieure [Schi 02].

Les AIS peuvent être considérés dans le cadre de l'optimisation difficile, comme une forme d'algorithme évolutionnaire présentant des opérateurs particuliers [Enge 10]. Par exemple, l'opération de sélection est basée sur une mesure d'affinité (c.-à-d. entre le récepteur d'un anticorps et un antigène). La mutation s'opère, quant à elle, via un opérateur d'hyper-mutation directement issu de la métaphore. Un exemple d'AIS basique est présenté dans l'algorithme 1.6.

Algorithme 1.6 Systèmes immunitaires artificiels [Enge 10]

$t \longleftarrow 0$
Initialiser aléatoirement une population $P(t)$ d'individus, composée de m cellules mémoires $P_m(t)$ et de r autres cellules $P_r(t)$: $P(t) = P_m(t) \cup P_r(t)$
Evaluer l'affinité de chaque individu de $P(t)$

Tant que *Condition d'arrêt* ≠ *Vrai* faire
| Sélectionner un sous-ensemble $S(t)$ d'individus de plus haute affinité dans $P(t)$
| Cloner chaque individu de $S(t)$ proportionnellement à son affinité, pour former une Population $C(t)$
| Muter chaque individu de $C(t)$ avec un taux inversement proportionnel à son affinité, pour former une population $C^*(t)$
| Evaluer l'affinité de chaque individu de $C^*(t)$
| Sélectionner les m individus de plus haute affinité dans $C^*(t) \cup P_m(t)$ pour former $P_m(t+1)$
| Remplacer un sous-ensemble d'individus de plus faible affinité dans $P_r(t)$ par des nouveaux individus, tirés aléatoirement, pour former $P_r(t+1)$
| Evaluer l'affinité de ces nouveaux individus
| $t \longleftarrow t+1$
Fin

1.3.5.5 Algorithme de recherche d'harmonie

L'algorithme de recherche d'harmonie (*Harmony Search* HS) est une métaheuristique à base de population développée en 2001 par le professeur Geem et ses collaborateurs [Geem 01], elle a été appliquée pour la première fois dans un problème d'optimisation des réseaux de distribution d'eau [Geem 01]. Son principe est très simple, il est basé sur le processus de performance musical, qui consiste à trouver une meilleure harmonie, parmi celles produites par un orchestre musical où chaque musicien joue une note spécifique. L'algorithme HS a été appliqué, dès son apparition, à une large gamme de problèmes : industriels, médicaux, dans les technologies de l'information, etc. [Manj 13].

Cette méthode d'optimisation fait l'objet d'un intérêt particulier dans ce travail de thèse. Cet intérêt comprend, une présentation détaillée de la méthode, une élaboration d'une nouvelle variante ainsi son application (la nouvelle variante) à un problème d'optimisation d'une loi de commande adaptative.

1.4 Métaheuristiques à population de solutions issues de l'intelligence en essaim

L'intelligence en essaim (*Swarm Intelligence* SI) est née de la modélisation mathématique et informatique des phénomènes biologiques rencontrés en éthologie [Bona 99]. Elle désigne les capacités cognitives d'une communauté résultant des interactions multiples entre les membres

(ou agents) de la communauté. Des agents, au comportement très simple, peuvent ainsi accomplir des tâches complexes grâce à un mécanisme fondamental appelé *synergie*. Sous certaines conditions particulières, la synergie créée, par la collaboration entre individus, fait émerger des possibilités de représentation, de création et d'apprentissage supérieures à celles des individus isolés.

Les formes d'intelligence en essaim sont très diverses selon les types de communauté et les membres qu'elles réunissent. Tout système basé sur l'intelligence en essaim a les caractéristiques principales suivantes :

1. L'information locale : Chaque individu ne possède qu'une connaissance partielle de l'environnement et n'a pas conscience de la totalité des éléments qui influencent le groupe,
2. L'ensemble de règles : Chaque individu obéit à un ensemble restreint de règles simples par rapport au comportement du système global,
3. Les interactions multiples : Chaque individu est en relation avec un ou plusieurs autres individus du groupe,
4. La collectivité : les individus trouvent un bénéfice à collaborer (parfois instinctivement) et leur performance est meilleure que s'ils avaient été seuls.

L'intelligence en essaim est observée notamment chez les insectes sociaux (fourmis, termites, abeilles,…) et les animaux en mouvement (oiseaux migrateurs, bancs de poissons,..).

Plusieurs algorithmes basés sur le principe d'intelligence en essaim ont été introduits, parmi les plus significatifs d'entre eux figurent : l'algorithmes d'optimisation par essaim particulaire (*Particle Swarm Optimization* PSO) [Kenn 95] qui s'inspire du comportement social des animaux évoluant en essaim, tels que les nuées d'oiseaux et les bancs de poissons, l'algorithme *Bacterial foraging optimization* BFO [Pass 02] inspiré du comportement de recherche de nourriture et de reproduction des bactéries, l'optimisation par colonie d'abeilles (*Bee Colony Optimization* BCO) [Kara 07] et l'algorithme de colonies de fourmis, dont il est question dans cette thèse.

1.4.1 Optimisation par Essaim Particulaire

L'optimisation par essaim particulaire (*Particle Swarm Optimization* PSO), est un algorithme évolutionnaire qui utilise une population de solutions candidates pour développer une solution optimale à un problème donné. Cet algorithme a été proposé par Russel Eberhart et James Kennedy en 1995 [Kenn 95]. Il s'inspire du comportement social des animaux évoluant en essaim, tels que les bancs de poissons et les vols groupés d'oiseaux. En effet, on peut observer chez ces animaux des dynamiques de déplacement relativement complexes, alors

qu'individuellement chaque individu a une « intelligence » limitée, et ne dispose que d'une connaissance locale de sa situation dans l'essaim.

L'information locale et la mémoire de chaque individu sont utilisées pour décider de son déplacement. Des règles simples, telles que « rester proche des autres individus », « aller dans une même direction » ou « aller à la même vitesse », suffisent pour maintenir la cohésion de l'essaim, et permettent la mise en œuvre de comportements collectifs complexes et adaptatifs.

L'essaim de particules correspond à une population d'agents simples, appelés particules. Chaque particule est considérée comme une solution potentielle du problème, où elle possède une position (le vecteur solution) et une vitesse. De plus, chaque particule possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines » dites *informatrices* ou *voisinage*.

Un essaim de particules, qui sont des solutions potentielles au problème d'optimisation, « survole » l'espace de recherche, à la recherche de l'optimum global. Le déplacement d'une particule est influencé par trois composantes (figure 1.5) : (1) Une composante d'*inertie* : la particule tend à suivre sa direction courante de déplacement ; (2) Une composante *cognitive* : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ; (3) Une composante *sociale* : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

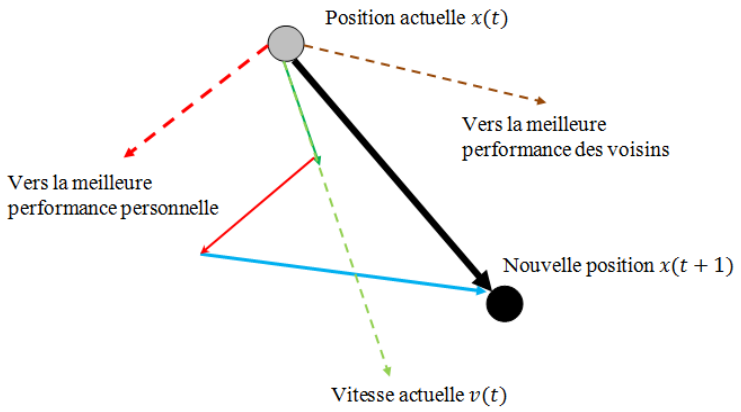

Figure 1.5 : Déplacement d'une particule.

Dans un espace de recherche de dimension D, la particule i de l'essaim (composée de N particules) est modélisée par son vecteur position $\vec{x}_i = (x_{i1}, x_{i2}, \dots, x_{iD})$ et par son vecteur vitesse $\vec{v}_i = (v_{i1}, v_{i2}, \dots, v_{iD})$. La qualité de sa position est déterminée par la valeur de la fonction objectif en ce point. Chaque particule dispose d'une mémoire lui permettant de se souvenir de sa meilleure solution, découverte dans le passé, que l'on note $\vec{P}best_i = (pbest_{i1}, pbest_{i2}, \dots, pbest_{iD})$.

La meilleure position atteinte par les particules de l'essaim est notée $\vec{G}best = (gbest_{i1}, gbest_{i2}, \dots, gbest_{iD})$.

Au départ de l'algorithme, les particules de l'essaim sont initialisées de manière aléatoire dans l'espace de recherche du problème. Ensuite, à chaque itération, chaque particule se déplace, en combinant linéairement les trois composantes citées ci-dessus. En effet, à l'itération $t+1$, le vecteur vitesse et le vecteur position sont calculés à partir de l'équation (1.10) et de l'équation (1.11), respectivement.

$$v_{i,j}^{t+1} = wv_{i,j}^{t} + c_1 r_{1_{i,j}}^{t}\left[pbest_{i,j}^{t} - x_{i,j}^{t}\right] + c_2 r_{2_{i,j}}^{t}\left[gbest_{i,j}^{t} - x_{i,j}^{t}\right] \quad (1.10)$$

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1}, j \in \{1,2, \dots, D\} \quad (1.11)$$

Où w est une constante, appelée coefficient d'inertie ; c_1 et c_2 sont deux constantes, appelées coefficients d'accélération ; r_1 et r_2 sont deux nombres aléatoires tirés uniformément dans [0,1], à chaque itération t et pour chaque dimension j avec $j \in \{1,2,.,D\}$ et $i \in \{1,2,.,N\}$.
Les trois composantes mentionnées ci-dessus (c.-à-d. d'inertie, cognitive et sociale) sont représentées dans l'équation (1.10) par les termes suivants :

1- $wv_{i,j}^{t}$ correspond à la composante d'inertie du déplacement, où le paramètre w contrôle l'influence de la direction de déplacement sur le déplacement futur,
2- $c_1 r_{1_{i,j}}^{t}\left[pbest_{i,j}^{t} - x_{i,j}^{t}\right]$ correspond à la composante cognitive du déplacement, où le paramètre c_1 contrôle le comportement cognitif de la particule,
3- $c_2 r_{2_{i,j}}^{t}\left[gbest_{i,j}^{t} - x_{i,j}^{t}\right]$ correspond à la composante sociale du déplacement, où le paramètre c_2 contrôle l'aptitude sociale de la particule.

Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées et les deux vecteurs $\vec{P}best_i$ et $\vec{G}best$ sont mis-à-jour, à l'itération $t+1$, suivant les deux équations (1.12) (dans le cas p. ex. d'une minimisation de la fonction objectif) et (1.13) respectivement.

$$\vec{P}best_i(t+1) = \begin{cases} \vec{P}best_i(t) \;\; si \quad \mathrm{F_{obj}}\left(\vec{x}_i(t+1)\right) \geq \vec{P}best_i(t) \\ \vec{x}_i(t+1) \qquad\qquad\qquad\qquad sinon \end{cases} \quad (1.12)$$

$$\vec{G}best(t+1) = \arg min_{\vec{P}best_i} \mathrm{F_{obj}}\left(\vec{P}best_i(t+1)\right), \quad (1.13)$$

Le pseudo code de l'algorithme PSO est donné par l'algorithme 1.7.

Algorithme 1.7 Pseudo code de l'algorithme PSO [Enge 10]

Initialiser aléatoirement N particules : position et vitesse.
Evaluer les positions des particules
Pour chaque particule i, $\vec{P}best_i = \vec{x}_i$
Calculer $\vec{G}best$ selon (1.13)

Tant que *Condition d'arrêt* ≠ *Vrai* faire
  Déplacer les particules selon (1.10) et (1.11)
  Evaluer les positions des particules
  Mettre à jour $\vec{P}best_i$ et $\vec{G}best$ selon (1.12) et (1.13)
Fin

1.4.2 Algorithme de recherche de nourriture bactérienne

L'algorithme de recherche de nourriture bactérienne (*Bacterial Foraging Optimization* BFO) est une métaheuristiques inspirée du modèle imitant le comportement alimentaire des bactéries. Les bactéries ont tendance à se réunir pour les zones riches en éléments nutritifs selon une activité appelée "*chimiotactisme*" [Adle 66]. Cet algorithme a été développé en 2002 par le professeur Passion [Pass 02], il a été le premier qui à penser à introduire l'algorithme alimentaire d'un type des bactéries dit : *Escherichia coli* (*E. Coli*) afin de trouver le minimum d'une fonction $J(\theta), \theta \in R_p$, où nous n'avons pas de mesures ou une description analytique du gradient $\Delta J(\theta)$[Pass 02].

D'après l'article original [Pass 02], les deux grandeurs $\theta^i(s)$ et $J(\theta)$ sont appelées respectivement la position et le coût de la fonction objectif de la bactérie *"i"*. La stratégie de recherche de nourriture des bactéries *E. Coli* comprend trois étapes principales : Le chimiotactisme, la reproduction et élimination et dispersion [Pass 02].

Le déplacement des bactéries peut être classé en deux modes. Le premier dit : mode "Run" signifie un mouvement régulier (suivant à peu près une ligne droite), et le second dit : mode "Tumble" signifie un mouvement aléatoire.

1. *Chimiotactisme* : Le chimiotactisme des bactéries est basé sur la suppression des actions en mode Tumble qui peuvent orienter les bactéries vers des directions divergentes. Le mouvement de chaque bactérie peut être décrit en termes d'intervalles des actions au cours des quels la cellule bactérienne nage approximativement suivant une ligne droite, c.-à-d. en mode " Run ", interrompue par des mouvements orientés vers des directions aléatoires, en mode "Tumble", comme le montre la (figure 1.6). La position de la i^{me} bactérie peut être représentée par

l'équation $\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$, avec $\theta^i(j,k,l)$ représente la position de la $i^{ième}$ bactérie à l'étape chimiotactique j dans la boucle k de reproduction et la $l^{ième}$ événements de suppression ou de dispersion, Δ indique un vecteur orienté vers une direction aléatoire, dont leur élément sont dans l'intervalle $[-1,1]$.

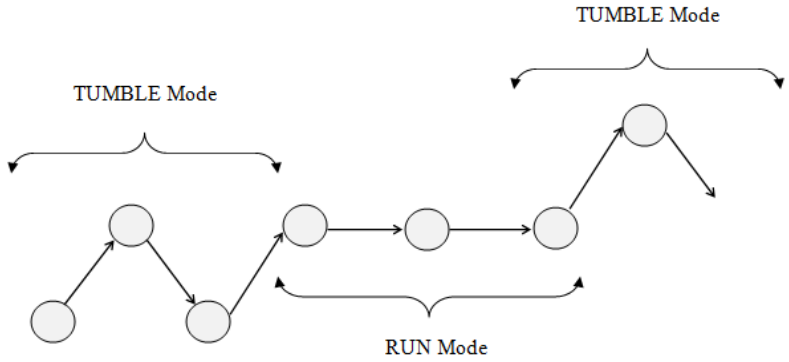

Figure 1.6 : Méthode de déplacement des bactéries.

2. *Reproduction* : La reproduction des bactéries représente une étape importante dans cet algorithme. La remise en forme totale de chaque bactérie est décidée suite à un calcul de la somme de ces aptitudes collectées au cours de sa vie, comme le montre l'équation suivante :

$$J_{health}^{i} = \sum_{j=1}^{N_C+1} J(i,j,k,l) \qquad (1.14)$$

En effet, suite a chaque étape de chimiotactisme, la valeur de la fonction objectif correspondant à la bactérie désirée, sera retenue et utiliser par la suite pour prendre la décision du maintien ou de suppression de cette dernière. Toutes les bactéries sont classées dans l'ordre inverse, en fonction de leur état physique (santé). Dans l'étape de reproduction, seule la première moitie de la population des bactéries survivantes se divise en deux modèles identiques, qui occupent les mêmes positions. Ainsi, la population (nombre d'individus) de bactéries reste constante dans chaque processus de chimiotactisme.

3. *Elimination et dispersion* : Dans le processus de l'évolution, un événement soudain et imprévu peut se produire, il peut modifier largement le bon déroulement de l'évolution et provoquer une élimination et/ou une dispersion à un nouvel espace (environnement) de l'ensemble des bactéries. Néanmoins, au lieu de perturber le bon déroulement du chimiotactisme, l'événement produit peut placer une nouvelle série de bactéries plus près de l'emplacement de la nourriture. Ces deux processus, ont l'avantage de réduire l'arrêt dans des points de solutions prématurées ou dans des optimums locaux. L'Algorithme 1.8 décrit en détail la méthode BFO.

Algorithme 1.8 Algorithme de recherche de nourriture bactérienne BFO [Chen 09]

[Etape 1] Initialisation des paramètres
$n, N, N_C, N_S, N_{re}, N_{ed}, P_{ed}, C(i)$ $(i = 1, 2, 3, 4 \ldots, N)$.
Avec :
n : dimension de l'espace de recherche,
N : taille de population (nombre de bactéries),
N_C : nombre des étapes de chimiotactisme,
N_S : longueur de natation,
N_{re} : nombre des étapes de reproduction,
N_{ed} : nombre des étapes d'élimination et de dispersion,
P_{ed} : la probabilité d'élimination-dispersion,
$C(i)$: la taille de l'étape prise dans la direction aléatoire en mode Tumble,
[Étape 2] boucle d'élimination-dispersion : $l = l + 1$
[Étape 3] boucle de reproduction : $k = k + 1$
[Étape 4] boucle de chimiotactisme: $J = J + 1$
{1} Pour $i = 1, 2, 3, 4 \ldots, N$, faire un pas de chimiotactisme pour la bactérie i comme suit. :
{2} Calculer $J(i, j, k, l) = J(i, j, k, l) + J_{CC}\left(\theta^i(j, k, l), P(j, k, l)\right)$
{3} Enregistrer le meilleur coût de la fonction objectif $J_{last}(i, j, k, l)$
{4} Mode tumble : générer un vecteur aléatoire $\Delta(i) \in R^n$ avec chaque élément $\Delta_m(i), m = 1, 2, 3, 4 \ldots, p$ est un nombre aléatoire dans $[-1,1]$
{5} changer la position de la bactérie i selon : $\theta^i(j+1, k, l) = \theta^i(j, k, l) + C(i)\,\Delta(i)/\sqrt{\Delta^{\mathrm{T}}(i)\Delta(i)}$
{6} Calculer la nouvelle valeur $J(i, j+1, k, l)$
{7} Commencer l'action "natation" – Swim
i) Initialiser le compteur de natation $(m = 0)$
ii) Si $(m < N_S)$
$m = m + 1$
Si $J(i, j+1, k, l) < J_{last}$ faire $J_{last} = J(i, j+1, k, l)$
et $\theta^i(j+1, k, l) = \theta^i(j, k, l) + C(i)\,\Delta(i)/\sqrt{\Delta^{\mathrm{T}}(i)\Delta(i)}$
et utiliser cette valeur $(\theta^i(j+1, k, l))$ pour le calcul de la nouvelle valeur de $J(i, j+1, k, l)$
Sinon poser $m = N_S$
{8} Si $(i \neq N)$ passez à la prochaine bactérie $(i = i + 1)$
[Étape 5] Si $(J < N_C)$ retourner à l'étape 3
[Étape 6] Reproduction
i) Pour k et l donnés, et pour chaque $i = 1, 2, 3, 4 \ldots, N$ calculer $J^i_{health} = \sum_{j=1}^{N_C+1} J(i, j, k, l)$. Le calcul de cette grandeur permet de connaitre la qualité (mauvaise/bonne) du démarche de cette bactérie.
ii) la dernière moitié des bactéries qui ont la plus grande valeur de J sont morts. Ainsi, la première moitié qui ont la meilleur valeur de J sont reproduit et placée dans les même positions.
[Étape 7] Si $(k < N_{se})$ retourner à l'étape 2
[Étape 8] Elimination et dispersion : éliminer et disperser avec une probabilité P_{ed} chaque bactérie i. Dans ce processus une bactérie éliminée en même temps correspond à une autre dispersée.
Si $(l < N_{ed})$ retourner à l'étape 1, sinon à la fin (génération des paramètres optimaux).

1.4.3 Algorithmes de colonies de fourmis

Une colonie de fourmis, dans son ensemble, est un système complexe stable et autorégulé capable de s'adapter très facilement aux variations environnementales les plus imprévisibles, mais aussi de résoudre des problèmes, sans contrôle externe ou mécanisme de coordination central, de manière totalement distribuée.

L'optimisation par colonies de fourmis (*Ant Colony Optimization* ACO) a été initialement introduite par Marco Dorigo et ses collègues [Dori 91, Dori 92, Dori 96]. Elle s'inspire du comportement des fourmis lorsque celles-ci sont à la recherche de nourriture [Dori 05]. En effet, celles-ci parviennent à trouver le chemin le plus court entre leur nid et une source de nourriture, sans pour autant avoir des capacités cognitives individuelles très développées. Les fourmis explorent d'abord les environs de leur nid en effectuant une marche aléatoire. Le long de leur chemin entre la source de nourriture et le nid, elles déposent sur le sol une substance chimique volatile appelée *phéromone* afin de marquer certains chemins favorables qui devraient guider leurs congénères à la source de nourriture. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court. Il en découle que la quantité de phéromones déposées par unité de temps sur ce trajet est plus importante que sur les autres. Les colonies de fourmis artificielles exploitent cette caractéristique pour la résolution de problèmes d'optimisation combinatoire surtout les problèmes du parcours des graphes.

Autant que les algorithmes de colonies de fourmis sont des métaheuristiques à population, chaque solution est représentée par une fourmi déplaçant dans l'espace de recherche. Les fourmis marquent les meilleures solutions, et tiennent compte des marquages précédents pour optimiser leur recherche. Ces algorithmes utilisent une distribution de probabilité implicite pour effectuer la transition entre chaque itération.

Dans leurs versions adaptées à des problèmes combinatoires, ils utilisent aussi une construction itérative des solutions. Les fondements théoriques essentiels de ces algorithmes, qui constituent un sujet principal dans cette thèse, seront présentés avec la proposition d'une version améliorée dans le second chapitre.

1.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoire mono-objectifs. Tout d'abord, nous avons présenté respectivement les définitions : d'un problème d'optimisation mono-objectif, de l'optimisation difficile et des algorithmes d'optimisation approchée (heuristiques et métaheuristiques). Puis, nous avons présenté les deux types de métaheuristiques à population

de solutions, celles issues de la théorie de l'évolution (algorithmes évolutionnaires) et les autres issues de l'intelligence en essaim.

Parmi les algorithmes présentés,

Dans les travaux de cette thèse nous nous intéressons, particulièrement, à l'algorithmes de colonies de fourmis ACO et celui de recherche d'harmonie HS.

Les algorithmes ACO d'origines, inspirés des comportements collectifs de dépôt et de suivi de pistes observées dans les colonies de fourmis, ont rencontré un succès remarquable depuis leur apparition. Ils présentent un ensemble de caractéristiques très intéressantes, telles que le parallélisme intrinsèque élevé (voir section 2.2.4.5 dans le chapitre 2), la robustesse (une colonie peut maintenir une recherche efficace, même si certains de ses individus sont défaillants) ou encore la décentralisation (les fourmis n'obéissent pas à une autorité centralisée). En outre, l'algorithme HS est une métaheuristique à population de solutions, basée sur l'imitation du processus d'improvisation musical, où chaque musicien joue une note avec l'ensemble de l'orchestre en utilisant son instrument pour trouver une meilleure harmonie. De nombreux travaux récents ont montré l'efficacité de cet algorithme en termes de simplicité d'analyse et de programmation (mathématiquement facile à formuler). Ces caractéristiques ont motivé la communauté scientifique pour leur utilisation dans la résolution d'un grand nombre de problèmes réels d'optimisation [Manj 13]. Néanmoins, ces algorithmes présentent certains inconvénients, entre autres ; leur faible capacité de recherche locale, leur vitesse de convergence lente. De plus, ils sont faciles à être piégés par les optima locaux (la convergence prématurée qui peut conduire l'algorithme à se stagner dans un optimum local).

De nombreux travaux de recherches ont été proposés pour améliorer les performances des versions classiques de ces deux métaheuristiques, tout en essayant de remédier à leurs défauts. En effet, à travers notre travail de thèse nous avons apporté notre contribution par le développement de deux nouvelles variantes améliorées pour ces algorithmes.

Nos contributions seront détaillées dans le deuxième et le troisième chapitre. Les deux algorithmes développés seront, ensuite, appliquées aux problèmes d'optimisation de lois de commandes adaptative et par modes glissants.

CHAPITRE 2

Contribution à l'amélioration des algorithmes de colonies de fourmis : Algorithme ACO amélioré

2.1 Introduction

Les algorithmes de colonies de fourmis (*Ant Colony Optimization* ACO) sont une métaheuristique appartenant aux méthodes d'intelligence en essaim : elles mettent en œuvre un mécanisme de mémoire collective permettant de guider la recherche en mutualisant sur les bonnes solutions déjà trouvées. Mise au point au début des années 1990 [Dori 91, Dori 92] avec l'algorithme *Ant System* (AS) [Colo 91], cette méthode est issue de l'observation des fourmis réelles qui communiquent par des signaux chimiques avec les autres membres de la colonie, afin de signaler un parcours à suivre. Elles utilisent pour cela des phéromones, une substance attractive naturellement sécrétée par les fourmis, qu'elles dispersent à leur passage, notamment lorsqu'une source de nourriture a été trouvée. Ce processus d'échange d'information utilisant l'environnement ambiant est appelé *stigmergie*. La sécrétion de phéromones permet, dans le cas des fourmis, d'aboutir à des itinéraires de distance quasi-optimale au cours du temps (entre le nid et la source de nourriture), bien que les capacités cognitives de ces insectes soient très limitées.

Comparé aux autres métaheuristiques de la même famille, C-ACO (versions classiques) présentent plusieurs caractéristiques relativement intéressantes, telles que : p.ex. leur processus de forage est basé sur une rétroaction positive (*positive feedback*) d'où la probabilité qu'une fourmi choisit une piste dont le parcours est proportionnel au nombre de fourmis qui ont déjà choisi la même piste, cette caractéristique permet l'accélérer la découverte de bonnes solutions ; la procédure de recherche d'une heuristique gloutonne qui est exploitée pour aider l'algorithme à trouver de bonnes solutions dans un nombre minimal d'itération. En outre, leur robustesse vis-à-vis de la défiance de certains (ou un petit ensemble) des individus de la colonie et la décentralisation (les fourmis sont indépendantes dans la prise de leurs décisions).

Cependant, ACO ont montré des insuffisances majeures en terme de : vitesse de convergence lente et de convergence prématurée qui conduit souvent l'algorithme à un état de piégeage dans des minimums locaux.

Pour remédier à ces deux inconvénients (et même d'autres) et afin d'améliorer les performances de C-ACO, plusieurs résultats de recherche ont été obtenus Parmi eux, on trouve

les modèles hybrides [Kefa 15, Ding 12], les mécanismes inspirés de la biologie [Yang 10] et quelques modifications de base au niveau de : la méthode de sélection des différents paramètres de l'algorithme, on parle ici des variantes adaptatives [Duan 09, Fors 07], la phase d'initialisation (génération de l'espace de recherche) [Qigu 09], les règles contrôlant le déplacement des fourmis dans leur colonie [Cheng 13] ainsi que les formules de mise-à-jour des pistes de phéromone imposées aux fourmis [Jova 11].

Dans le cadre de l'amélioration de C-ACO, nous proposons dans ce chapitre une variante permettant de remédier principalement aux problèmes de convergence lente et de la convergence prématurée.

Nous présentons, dans un premier temps, une revue sur les algorithmes ACO, leurs origines, leurs variantes les plus indiquées appliquées au problème TSP, leur formalisme et leurs propriétés fondamentales.

Dans la section 2.3 nous proposons une nouvelle variante de l'algorithme C-ACO où nous explicitons les modifications apportées à l'algorithme de base.

L'algorithme proposé, permet, tout en évitant les minimums locaux, d'améliorer la vitesse de convergence en raison d'une capacité d'exploration plus rapide due à une stratégie de recherche de solutions optimales. Cette stratégie est basée sur un mécanisme de mise-à-jour de phéromone modifié et associé à une procédure de mémorisation itérative des meilleures solutions.

La version ainsi développée de l'algorithme sera mise en œuvre pour l'optimisation des paramètres d'une commande décentralisée par modes glissants présentée dans le quatrième chapitre.

2.2 Optimisation par colonies de fourmis (ACO)

Une partie importante des informations présentées ci-après sont tirées du chapitre 4 du livre : Métaheuristiques pour l'optimisation difficile [Dréo 03].

2.2.1 Origines de l'approche

Les algorithmes de colonies de fourmis forment une classe des métaheuristiques récemment proposée pour des problèmes d'optimisation difficile. Ces algorithmes s'inspirent des comportements collectifs de dépôt et de suivi de piste observés dans les colonies de fourmis. Une colonie d'agents simples (les fourmis) communiquent indirectement via des modifications dynamiques de leur environnement (les pistes de phéromones) et construisent ainsi une solution à un problème en s'appuyant sur leur expérience collective. Inspiré de ce phénomène naturel, Dorigo et ses collaborateurs ont conçu une heuristique, baptisée *Ant System* (AS) [Colo 91], pour résoudre le problème TSP, mais n'a pas permis de produire des résultats compétitifs.

Cependant, l'intérêt pour la métaphore était lancé et de nombreux algorithmes s'en inspirant ont depuis été proposés, certains atteignant des résultats très convaincants [Dori 05].

2.2.2 Optimisation naturelle : pistes de phéromones

Les algorithmes de colonies de fourmis sont nés à la suite d'une constatation : les insectes sociaux en général, et les colonies de fourmis en particulier, résolvent naturellement des problèmes relativement complexes. Les biologistes ont étudié comment les fourmis arrivent à résoudre collectivement des problèmes trop complexes pour un seul individu, notamment les problèmes de choix lors de l'exploitation de sources de nourritures.

Les fourmis ont la particularité d'employer pour communiquer des substances volatiles appelées phéromones. Elles sont très sensibles à ces substances, qu'elles perçoivent grâce à des récepteurs situés dans leurs antennes. Ces substances sont nombreuses et varient selon les espèces. Les fourmis peuvent déposer des phéromones au sol, grâce à une glande située dans leur abdomen, et former ainsi des pistes odorantes, qui pourront être suivies par leurs congénères (figure 2.1).

Figure 2.1 : Des fourmis autour de leur nid [Willi 12].

Les fourmis utilisent les pistes de phéromone pour marquer leur trajet, par exemple entre le nid et une source de nourriture. Une colonie est ainsi capable de choisir (sous certaines conditions) le plus court chemin vers une source à exploiter [Dori 92], sans que les individus aient une vision global du trajet.

L'expérience de l'obstacle a montré comment le dépôt de phéromone mène progressivement les fourmis à emprunter le chemin le plus court. La principale illustration de cette expérience est donnée par la figure 2.2 : un obstacle est placé sur le trajet des fourmis qui, après une étape d'exploration, finiront par emprunter le plus court chemin entre le nid et la source de nourriture

[Goss 89]. Les fourmis qui sont retournées le plus rapidement au nid en passant par la source de nourriture sont celles qui ont emprunté le chemin le plus court. Il en découle que la quantité de phéromone déposée par unité de temps sur ce trajet est plus importante que sur les autres. Par ailleurs, une fourmi est d'autant plus attirée à un certain endroit que le taux de phéromones y est important. De ce fait, le plus court chemin a une probabilité plus importante d'être emprunté par les fourmis que les autres chemins et sera donc, à terme, emprunté par toutes les fourmis.

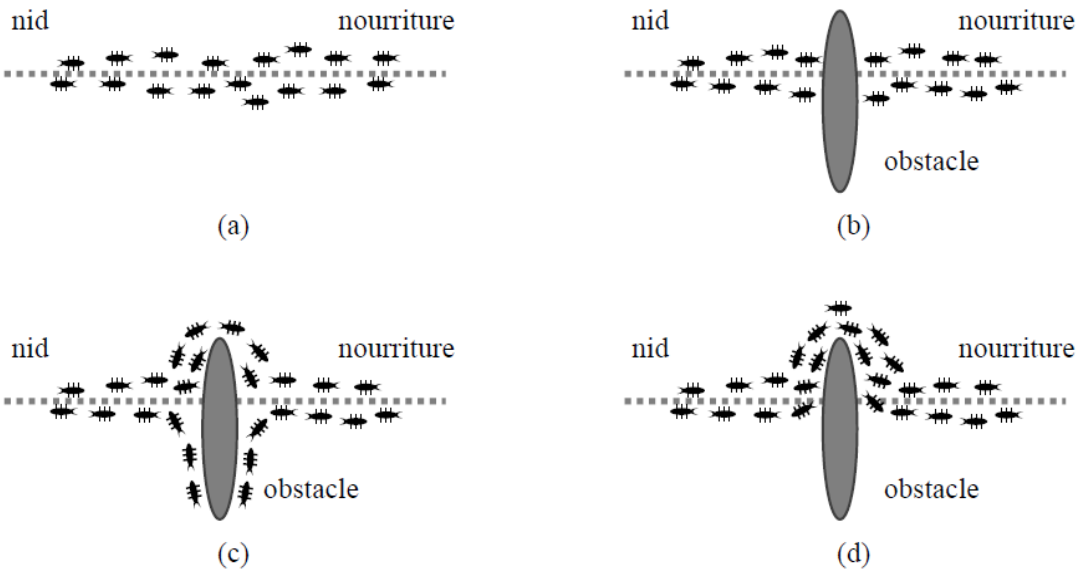

Figure 2.2 : Illustration de la capacité des fourmis à chercher de la nourriture en minimisant leur parcours. (a) Recherche sans obstacle, (b) Apparition d'un obstacle, (c) Recherche du chemin optimal, (d) Chemin optimal trouvé [Goss 89].

On constate qu'ici le choix s'opère par un mécanisme d'*amplification* d'une fluctuation initiale. Cependant, il est possible qu'en cas d'une plus grande quantité de phéromone déposée sur les grandes branches, au début de l'exploitation, la colonie choisisse le plus long parcours. D'autres expérience [Beck 89], avec une une autre espèce de fourmis, ont montré que si les fourmis sont capables d'effectuer des demi-tours, alors la colonie est plus flixsible et le risque d'être piégé sur le chemin long est plus faible.

Il est très difficile de connaitre avec précision les propriétés physico-chimiques des pistes de phéromones, qui varient en fonction des espéces et d'un grand nombre de pramètres. Cependent, les métaheuristiques d'optimisation de colonies de fourmis s'appuient en grande partie sur le phénomène d'évaporation des pistes de phérmone. Or, on constate dans la nature que les pistes s'évaporent plus lentement que ne le prévoient les modèles. Les fourmis réelles disposent en effet 'd'heuristques' leur apportant un peu plus d'information sur le problème (par

exemple une information sur la direction). En effet, il faut garder à l'esprit que l'intérêt immédiat de la colonie (trouver le plus court chemin vers une source de nourriture) peut être en concurrence avec l'intérêt *adaptatif* de tels comportements. Si l'on prend en compte l'ensemble des containtes que subit une colonie de fourmis (prédation, compétition avec d'autre colonies, etc.) un choix rapide et stable peut être meilleur, et un changement de site exploité peut entraîner des coûts trop forts pour permettre la sélection naturelle d'une option.

2.2.3 Optimisation par colonies de fourmis et problème du voyageur de commerce

Le problème TSP a fait l'objet de la première implémentation d'un algorithme ACO : le AS [Colo 91]. Le passage de la métaphore à l'algorithme est ici relativement facile et le problème du TSP est bien connu et étudié. Il est très intéressant d'approfondir le principe de ce premier algorithme pour bien comprendre le mode de fonctionnement des algorithmes ACO. Il y a deux façons d'aborder ces algorithmes. La première, la plus évidente au premier abord, est celle qui a historiquement mené au algorithme AS original ; nous avons choisi de la décrire dans cette section. La seconde est une description plus formelle des mécanismes communs aux algorithmes ACO, elle sera décrite dans la section 2.2.3.2.

Le TSP consiste à trouver le trajet le plus court (désigné par "tournée" ou plus loin par "tour") reliant n villes données, chaque ville ne devant être visitée qu'une seule fois. Le problème est plus généralement défini comme un graphe complètement connecté (N, A), où les villes sont les nœuds N et les trajets entre ces villes, les arêtes A.

2.2.3.1 Algorithme de colonie de fourmis de base, le AS

Dans l'algorithme AS, à chaque itération $(1 \leq t \leq t_{max})$, chaque fourmi k $(k = 1, \dots, m)$ parcourt le graphe et construit un trajet complet de $n = |N|$ étapes (on note $|N|$ le cardinal de l'ensemble N). Pour chaque fourmi, le trajet entre une ville i et une ville j dépend de :

1- la liste des villes déjà visitées, qui définit les mouvements possibles à chaque pas, quand la fourmi k est sur la ville i : J_i^k,

2- l'inverse de la distance entre les villes : $\eta_{ij} = \frac{1}{d_{ij}}$, appelée visibilité. Cette information "statique" est utilisée pour diriger le choix des fourmis vers des villes proches, et éviter les villes trop lointaines,

3- la quantité de phéromone déposée sur l'arête reliant les deux villes, appelée l'intensité de la piste. Ce paramètre définit l'attractivité d'une partie du trajet global et change à chaque passage d'une fourmi. C'est, en quelque sorte, une mémoire globale du système, qui évolue par apprentissage.

La règle de déplacement (appelée "règle aléatoire de transition proportionnelle" par les auteurs [Bona 99]) est la suivante :

$$p_{ij}^{k}(t) = \begin{cases} \frac{\left(\tau_{ij}(t)\right)^{\alpha}(\eta_{ij})^{B}}{\sum_{l\in J_i^k}(\tau_{il}(t))^{\alpha}(\eta_{il})^{B}} & si \quad j \in J_i^k \\ 0 & si \quad j \notin J_i^k \end{cases} \tag{2.1}$$

Où α et β sont deux paramètres contrôlant l'importance relative de l'intensité de la piste $\tau_{ij}(t)$, et de la visibilité η_{ij}.

Dans le cas où $\alpha = 0$, seule la visibilité de la ville est prise en compte ; la ville la plus proche est donc choisie à chaque pas. Au contraire, avec $\beta = 0$, seules les pistes de phéromone jouent. Pour éviter une sélection trop rapide d'un trajet, un compromis entre ces deux paramètres, jouant sur les comportements de *diversification* et d'*intensification* (voir section 2.2.4.3 de ce chapitre), est nécessaire.

Après un tour complet, chaque fourmi laisse une certaine quantité de phéromone $\Delta\tau_{ij}^{k}(t)$ sur l'ensemble de son parcours, quantité qui dépend de la qualité de la solution trouvée :

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L^k(t)} & si \quad (i,j) \in T^k(t) \\ 0 & si \quad (i,j) \notin T^k(t) \end{cases} \tag{2.2}$$

Où $T^k(t)$ est le trajet effectué par la fourmi k à l'itération t, avec $L^k(t)$ est la longueur de la tournée et Q un paramètre fixé.

L'algorithme ne serait pas complet sans le processus d'évaporation des pistes de phéromone. En effet, pour éviter d'être piégé dans des solutions sous-optimales, il est nécessaire de permettre au système "d'oublier" les mauvaises solutions. On contrebalance donc l'additivité des pistes par une décroissance constante des valeurs des arêtes à chaque itération. La règle de mise-à-jour des pistes est donc :

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t+1) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t) \tag{2.3}$$

Où m est le nombre de fourmis et ρ le taux d'évaporation. La quantité initiale de phéromone sur les arêtes est une distribution uniforme d'une petite quantité $\tau_0 \geq 0$.

La figure 2.3 présente un exemple simplifie de problème TSP optimisé par un algorithme AS dont le pseudo-code est présenté sur l'algorithme 2.1.

Algorithme 2.1 : Algorithme AS [Dréo 03]

Pour $t = 1, \dots, t_{max}$
  Pour chaque fourmi $k = 1, \dots, m$
    Choisir une ville au hasard
    Pour chaque ville non visité i
      Choisir une ville j, dans la liste J_i^k des villes restantes, selon la formule (2.1)
    Fin Pour
    Déposer une piste $\Delta\tau_{ij}^k(t)$ sur le trajet $T^k(t)$ conformément à l'équation (2.2)
  Fin Pour
  Evaporer les pistes selon la formule (2.3)
Fin Pour

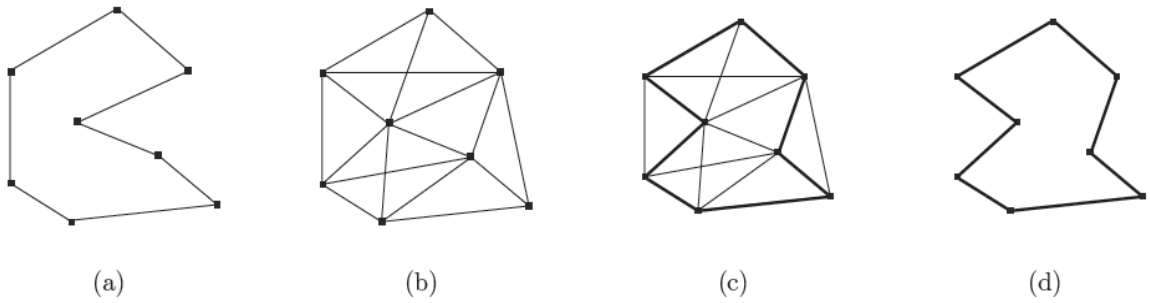

Figure 2.3 : Le problème TSP optimisé par l'algorithme AS, les points représentent les **villes** et l'épaisseur des **arêtes** la quantité de phéromone déposée. (a) exemple de trajet construit par une fourmi, (b) au début du calcul, tous les chemins sont explorés, (c) le chemin le plus court est plus renforcé que les autres, (d) l'évaporation permet d'éliminer les autres chemins les moins renforcé [Dréo 03].

2.2.3.2 Principaux schémas ACO pour le TSP

– **Ant System & élitisme (*Elitist Ant*)** Une première variante du "Système de Fourmis" à été proposée dans [Dori 96] : elle est caractérisée par l'introduction de fourmis "élitistes". Dans cette version, la meilleure fourmi (celle qui a effectué le trajet le plus court) dépose une quantité de phéromone plus grande, dans l'optique d'accroitre la probabilité des autres fourmis d'explorer la solution la plus prometteuse.

– ***Rank-Based Ant System* (*RBAS*)** [Bull 99], qui s'appuie sur *Elitist Ant* et propose pour sa part de classer les solutions de chaque cycle par ordre de qualité, pour ne renforcer que les ω premières (ω étant une variable paramétrable) avec une pondération relative au rang de la solution,

– ***Best-Worst Ant System* (*BWAS*)** [Cord 00], où seules les plus mauvaises solutions sont évaporées, tandis que les meilleures solutions locales sont renforcées.

– ***MAX-MIN Ant System* (*MMAS*)**

Cet algorithme est l'un des successeurs de AS les plus performant. Proposé dans [Stüt 97, Stüt 00], il introduit quatre innovations par rapport à AS qui sont les suivantes :

1. L'exploitation intense des meilleures solutions trouvées. Après chaque itération, les traces de phéromones sont mises-à-jour uniquement en utilisant soit la meilleure solution trouvée lors de la dernière itération soit la meilleure solution trouvée depuis le début de l'algorithme.
2. Les valeurs de phéromone possibles sur les arêtes sont restreintes à être comprises dans l'intervalle $[\tau_{min}, \tau_{max}]$ avec τ_{min} et τ_{max} sont deux paramètres de l'algorithme. Le but de restreindre les valeurs des phéromones est d'éviter la stagnation prématurée de la recherche.
3. Les valeurs initiales des phéromones τ_0 est fixée à τ_{max}.
4. À chaque fois que l'algorithme approche à un état de stagnation (les fourmis construisent les mêmes solutions) ou si la meilleure affectation trouvée depuis le début de l'algorithme n'a pas été améliorée depuis un certain nombre d'itérations, l'algorithme réinitialise toutes les traces de phéromones à τ_{max} (c.-à-d. : il fait un "restart").

Les expérimentations de cet algorithme sur le TSP ont montrées que pour les petites instances, il vaudrait mieux utiliser uniquement la meilleure solution de la dernière itération pour la mise-à-jour des phéromones, tandis que pour de plus grandes instances, il serait intéressant d'altérer entre la meilleure solution de la dernière itération et la meilleure solution trouvée depuis le début de l'algorithme.

– ***Ant Colony* System (*ACS*)**

Dans le but d'améliorer les performances de AS sur des problèmes de grandes tailles, Dorigo et Gambardella ont proposé dans [Dori 97a] une variante appelée Ant Colony System (ACS). Appliquée sur le TSP, cette nouvelle variante est essentiellement différente de AS sur les trois points suivants :

1. Une fourmi k se situant sur le nœud i, choisit le prochain noeud j en utilisant une loi proportionnelle pseudo-aléatoire formulée comme suit :

$$j = \begin{cases} argmax_{u \in J_j^k} \left[\tau_{iu}(t) \cdot \left(\eta_{iJ}\right)^B \right] & si \quad q \leq q_0 \\ J & si \quad q > q_0 \end{cases} \tag{2.4}$$

où q est une variable aléatoire uniformément distribuée dans l'intervalle [0,1], q_0 est un paramètre tel que $0 \leq q_0 \leq 1$, J est une variable aléatoire sélectionnée selon la probabilité :

$$p_{ij}^k(t) = \frac{\left(\tau_{ij}(t)\right)^{\alpha} (\eta_{ij})^B}{\sum_{l \in J_i^k} \left(\tau_{il}(t)\right)^{\alpha} (\eta_{il})^B} \qquad si \quad j \in J_i^k \tag{2.5}$$

La valeur de paramètre q_0 est très importante, car lorsque $q \leq q_0$, l'algorithme exploite au maximum les informations apprises (c.-à-d. : l'algorithme fait de l'exploitation en cherchant des solutions autour des meilleures solutions déjà trouvées) et lorsque $q > q_0$, l'algorithme utilise la règle standard (2.1) lui permet de faire un choix probabiliste et donc il explore l'espace de recherche.

2. La gestion des pistes est séparée en deux niveaux : une mise-à-jour locale et une mise-à-jour globale. Chaque fourmi dépose une piste lors de la mise-à-jour locale :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \tag{2.6}$$

Où τ_0 est la valeur initiale de la piste. À chaque passage, les arêtes visitées voient leur quantité de phéromone diminuer, ce qui favorise la diversification par la prise en compte des trajets non explorés. À chaque itération, la mise-à-jour globale s'effectue comme suit :

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}^{k}(t) \tag{2.7}$$

Où, les arêtes (i, j) appartiennent au meilleur tour T^{best} de longueur L^{best} avec $\Delta\tau_{ij}^{k}(t) = \frac{1}{L^{best}}$. Ici, seule la meilleure piste est donc mise-à-jour, ce qui participe à une intensification par sélection de la meilleure solution.

3. Le système utilise une liste de candidats. Cette liste stocke pour chaque ville les v plus proches voisines, classées par distances croissantes. Une fourmi ne prendra en compte une arête vers une ville en dehors de la liste que si celle-ci a déjà été explorée. Concrètement, si toutes les arêtes ont déjà été visitées dans la liste de candidats, le choix se fera en fonction de la règle (2.5), sinon c'est la plus proche des villes non visitées qui sera choisie.

2.2.3.3 Choix des paramètres

Le choix des paramètres de réglage des différentes variantes ACO citées préalablement, a une importance capitale sur les résultats obtenus. D'après [Dréo 03], pour l'algorithme AS, les auteurs préconisent que, bien que la valeur de Q ait peu d'influence sur le résultat final, cette valeur soit du même ordre de grandeur qu'une estimation de la longueur du meilleur trajet trouvé. D'autre part, la ville de départ de chaque fourmi est typiquement choisie par un tirage aléatoire uniforme, aucune influence significative du placement de départ n'ayant pu être démontrée.

En ce qui concerne l'algorithme ACS, les auteurs conseillent d'utiliser $\tau_0 = (n \cdot L_{nn})^{-1}$, où n est le nombre de villes et L_{nn} la longueur d'un tour trouvé par la méthode du plus proche voisin. Le nombre de fourmis m est un paramètre important ; les auteurs suggèrent d'utiliser autant de

fourmis que de villes ($i.e. m = n$) pour de bonnes performances sur le problème TSP. Il est possible de n'utiliser qu'une seule fourmi, mais l'effet d'amplification des longueurs différentes est alors perdu, de même que le parallélisme naturel de l'algorithme, ce qui peut s'avérer néfaste pour certains problèmes. En règle générale, les algorithmes ACO semblent assez peu sensibles à un réglage fin du nombre de fourmis.

2.2.4 Formalisation et propriétés d'un algorithme ACO

Un extrait très clair sur les ACO, a été écrit par [Dréo 04], il présente une petite introduction descriptive aux apports originaux des métaheuristiques ACO, son contenu est le suivant :

> *Une métaheuristique de colonie de fourmis est un processus stochastique construisant une solution, en ajoutant des composants aux solutions partielles. Ce processus prend en compte (i) une heuristique sur l'instance du problème (ii) des pistes de phéromone changeant dynamiquement pour refléter l'expérience acquise par les agents.*

Une formalisation très précise à été présentée par [Dori 03]. Elle passe par une représentation du problème, un comportement de base des fourmis et une organisation générale de la métaheuristique. Plusieurs concepts sont également à mettre en valeur pour comprendre les principes de ces algorithmes, notamment la définition des pistes de phéromone en tant que mémoire adaptative, la nécessité d'un réglage *intensification/diversification* et enfin l'utilisation d'une *recherche locale*. En se référant à [Dréo 03], nous traitons ci-après ces différents sujets.

2.2.4.1 Formalisation

Représentation du problème Selon [Dréo 03], un problème d'optimisation traité par un algorithme ACO est représenté généralement par un *jeu de solutions*, une fonction objectif assignant une valeur à chaque solution et un *jeu de contraintes*. L'objectif est de trouver l'optimum global de la fonction objectif satisfaisant les contraintes. Les différents états du problème sont caractérisés comme une séquence de composants. On peut noter que, dans certains cas, un coût peut être associé à des états autres que des solutions. Dans cette représentation, les fourmis construisent des solutions en se déplaçant sur un graphe $G = (C, L)$, où les nœuds sont les composants de C et où l'ensemble L connecte les composants de C. Les contraintes du problème sont implémentées directement dans les règles de déplacement des fourmis (soit en empêchant les mouvements qui violent les contraintes, soit en pénalisant de telles solutions).

Comportement des fourmis Les fourmis artificielles peuvent être considérées comme une procédure de construction stochastique *construisant* des solutions sur le graphe $G = (C, L)$. En

général, les fourmis tentent d'élaborer des solutions faisables mais, si nécessaire, elles peuvent produire des solutions infaisables. Les composants et les connexions peuvent être associés à des pistes de phéromone τ (mettant en place une mémoire adaptative décrivant l'état du système) et à une valeur heuristique (représentant une information à priori sur le problème, ou venant d'une source autre que celle des fourmis ; c'est bien souvent le coût de l'état en cours). Les pistes de phéromone et la valeur de l'heuristique peuvent être associées soit aux composants, soit aux connexions (figure 2.4).

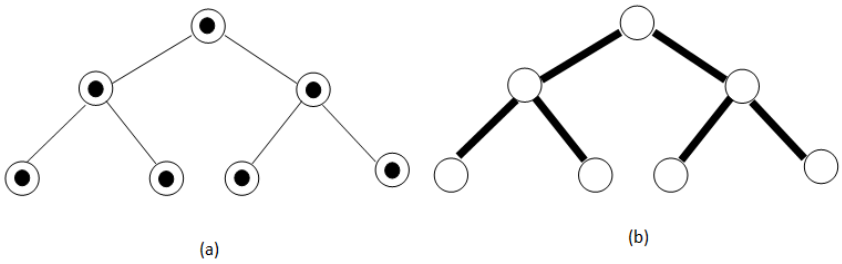

Figure 2.4 : Dans un algorithme ACO, les pistes de phéromone peuvent être associées aux composants (a) ou aux connexions (b) du graphe représentant le problème à résoudre [Dréo 03].

Chaque fourmi dispose d'une mémoire utilisée pour stocker le trajet effectué, d'un état initial et de conditions d'arrêt. Les fourmis se déplacent en utilisant une règle de décision probabiliste fonction des pistes de phéromone locales, de l'état de la fourmi et des contraintes du problème. Lors de l'ajout d'un composant à la solution en cours, les fourmis peuvent mettre à jour la piste associée au composant ou à la connexion correspondante. Une fois la solution construite, elles peuvent mettre à jour la piste de phéromone des composants ou des connexions utilisées. Enfin, une fourmi dispose au minimum de la capacité de construire une solution du problème.

Organisation de la métaheuristique En plus des règles régissant le comportement des fourmis, un autre processus majeur a cours : l'*évaporation* des pistes de phéromone. En effet, à chaque itération, la valeur des pistes de phéromone est *diminuée*. Le but de cette diminution est d'éviter une convergence rapide et le piégeage de l'algorithme dans des minimums locaux, par une forme d'oubli favorisant l'exploration de nouvelles régions.

Selon les auteurs du formalisme ACO, il est possible d'implémenter d'autres processus nécessitant un contrôle centralisé (et donc ne pouvant être directement pris en charge par des fourmis), sous la forme de processus annexes. Ce n'est, à notre sens, que peu souhaitable ; en effet, on perd alors la caractéristique décentralisée du système.

De plus, l'implémentation de processus annexes entre difficilement dans une formalisation rigoureuse.

2.2.4.2 Phéromones et mémoire

La *stigmergie* est l'un des mécanismes fondamentaux caractérisant les algorithmes ACO, Elle est précisément définie comme une forme de communication passant par le biais de modifications de l'environnement [Dréo 03]. La grande importance de ce phénomène est que les individus échangent des informations par le biais du travail en cours, de l'état d'avancement de la tache globale à accomplir. Dans les algorithmes ACO ce mécanisme est apparu clairement dans le processus de construction des pistes de phéromone, ce dernier à un grand impact sur la qualité des résultats obtenus.

Également, le choix de la méthode d'implémentation des pistes de phéromone est très important. Ce choix est en grande partie lié aux possibilités de *représentation* de l'espace de recherche, chaque représentation pouvant apporter une façon différente d'implémenter les pistes. Par exemple, pour le problème TSP, une implémentation efficace consiste à utiliser une piste τ_{ij} entre deux villes i et j comme une représentation de l'intérêt de visiter la ville j après la ville i. Une autre représentation possible, moins efficace en pratique, consiste à considérer τ_{ij} comme une représentation de l'intérêt de visiter i en tant que *jème* ville.

En effet, les pistes de phéromone décrivent à chaque pas l'état de la recherche de la solution par le système, les agents (fourmis) modifient la façon dont le problème va être représenté et perçu par les autres agents. Cette information est partagée par le biais des modifications de l'environnement des fourmis, grâce au stigmergie. L'information est donc stockée un certain temps dans le système, ce qui a amené certains auteurs à considérer ce processus comme une forme de *mémoire adaptative* [Tail 01], où la dynamique de stockage et de partage de l'information va être cruciale pour le système.

2.2.4.3 Intensification/diversification

Un point critique, lors de l'application d'un algorithme ACO, est de trouver un équilibre entre l'exploitation (*intensification*) de l'expérience de recherche acquise par les fourmis et l'exploration (*diversification*) de nouvelles zones de l'espace de recherche non encore visitées. Les algorithmes ACO possèdent plusieurs manières pour accomplir une telle balance, essentiellement par la gestion des traces de phéromone. En effet, les traces de phéromone déterminent dans quelles parties de l'espace de recherche les solutions construites auparavant sont localisées.

Une manière de mettre à jour la phéromone, et pour exploiter l'expérience de recherche acquise par les fourmis, est de déposer une quantité de phéromone fonction de la qualité de la solution trouvée par chaque fourmi. Ainsi, les zones correspondant aux meilleures solutions recevront une quantité de phéromone plus élevée et seront plus sollicitées par les fourmis durant leurs déplacements.

Pour contrôler la balance entre l'exploitation de l'espace de recherche et l'exploration de nouvelles zones, ACO dispose de plusieurs paramètres pour gérer l'importance relatives des traces de phéromone, essentiellement les deux paramètres α et ρ.

Le paramètre α détermine le poids des traces de phéromone dans le calcul des probabilités de transition, et ρ détermine l'évaporation de phéromone. Ces deux paramètres ont une influence sur le comportement exploratoire ; pour favoriser la stratégie d'exploration, on peut diminuer le coefficient α, ainsi les fourmis deviennent moins sensibles aux phéromones lors de leurs déplacements et peuvent visiter des zones non explorées, ou diminuer ρ, ainsi la phéromone s'évapore plus lentement et l'intensité des traces de phéromone devient similaire sur les arêtes et donc les fourmis deviennent moins influencées par la phéromone.

Pour favoriser la stratégie d'exploitation, on augmente les valeurs respectives de α et/ou de ρ, ainsi les fourmis seront plus guidées par la phéromone vers des zones ''prometteuses'' de l'espace de recherche. Après un certain nombre d'itérations, toutes les fourmis vont converger vers un ensemble de 'bonnes' solutions. Ainsi, on favorise une convergence plus rapide de l'algorithme.

Toutefois, il faut faire attention et éviter une intensification trop forte dans les zones qui apparaissent prometteuses de l'espace de recherche car cela peut causer une situation de stagnation : une situation dans laquelle toutes les fourmis génèrent la même solution.

Pour éviter une situation pareille de stagnation, une autre solution serait de maintenir un niveau raisonnable d'exploration de l'espace de recherche. Par exemple, dans ACS les fourmis utilisent une règle de mise-à-jour locale de phéromone durant la construction de solution pour rendre le chemin qu'elles viennent de prendre moins désirable pour les futures fourmis et ainsi, diversifier la recherche. MMAS introduit une limite inférieure pour l'intensité des traces de phéromone pour garantir toujours la présence d'un niveau minimal d'exploration. MMAS utilise aussi une réinitialisation des traces de phéromone, qui est une manière de renforcer l'exploration de l'espace de recherche.

2.2.4.4 ACO et la recherche locale

Dans plusieurs applications à des problèmes d'optimisation combinatoire difficiles, les algorithmes ACO réalisent de meilleures performances lorsqu'ils sont hybridés avec des algorithmes de recherche locale [Aart 97]. Ces algorithmes optimisent localement les solutions des fourmis et ces solutions optimisées localement sont utilisées par la suite dans la mise-à-jour de phéromone. L'utilisation de la recherche locale dans les algorithmes ACO peut être très intéressante comme les deux approches sont complémentaires. En effet, la combinaison peut améliorer largement la qualité des solutions produites par les fourmis. D'un autre côté, générer des solutions initiales pour les algorithmes de recherche locale n'est pas une tâche facile. Dans

les algorithmes de recherche locale, où les solutions initiales sont générées aléatoirement, la qualité des solutions peut être médiocre. En combinant la recherche locale avec ACO, les fourmis génèrent aléatoirement, en exploitant les traces de phéromone, des solutions initiales prometteuses pour la recherche locale.

2.2.4.5 Parallélisme

Un algorithme ACO est caractérisé principalement par une structure équipé d'un parallélisme *intrinsèque*. D'une manière générale, les solutions de bonnes qualités émergente du résultat des *interactions* indirectes ayant cours dans le système, pas d'un codage explicite d'échanges. En effet, chaque fourmi ne prend en compte que des informations locales de son environnement (les pistes de phéromone) ; il est donc facile de paralléliser un tel algorithme. Il est intéressant de noter que les différents processus en cours dans la métaheuristique (c.-à-d. le comportement des fourmis, l'évaporation et les processus annexes) peuvent également être implémentés de manière indépendante, l'utilisateur étant libre de décider de la manière dont ils vont interagir.

2.2.4.6 Convergence

Les métaheuristiques peuvent être vues comme des modifications d'un algorithme de base : une recherche aléatoire. Cet algorithme possède une propriété très intéressante, il permet de garantir que la solution optimale sera trouvée tôt ou tard, on parle alors de convergence. Cependant, puisque cet algorithme de base est biaisé, la garantie de convergence n'existe plus.

Si, dans certains cas, on peut facilement être certain de la convergence d'un algorithme de colonies de fourmis (MMAS p. ex.), le problème reste entier en ce qui concerne la convergence d'un algorithme ACO quelconque. Cependant, il existe plusieurs variantes dont la convergence à été prouvée : Le "Graph-Based Ant System" (GBAS) [Gutj 00]. La différence entre GBAS et l'algorithme AS se situe au niveau de la mise-à-jour des pistes de phéromone, qui n'est permise que si une meilleure solution est trouvée. Pour certaines valeurs de paramètres, et étant donné $\epsilon > 0$ une "faible" valeur, l'algorithme trouvera la solution optimale avec une probabilité $P_t \geq 1 - \epsilon$, après un temps $t \geq t_0$(où t_0 est fonction de ϵ).

Dans le même contexte, une nouvelle variante de ACO dite *ACO with stench pheromone* (ACO-SP) [Cong 13] à été proposée récemment avec une garantie de la convergence. Cet algorithme est crée dont le but d'optimiser la circulation dans un réseau de trafic de type dynamique. ACO-SP a deux différences fondamentales, résident principalement dans le modèle de phéromone utilisé (02 modèles de phéromone) : le premier étant une phéromone régulière produit afin d'attirer les fourmis caractérisées par un faible coût, tendis que le second étant une phéromone de mauvaise odeur (*stench pheromone*) utilisé afin d'aliéner les fourmis lors de la détection d'une stagnation quelconque.

2.3 Amélioration de ACO – proposition d'une nouvelle variante

Comme nous l'avons déjà signalé dans l'introduction de ce chapitre, C-ACO (p. ex. AS ou ACS) a montré des déficiences significatives en termes de rapidité qui s'explique par la convergence prématurée vers des régions sous-optimales (facilement piégés dans des optimums locaux). En effet, nous avons élaboré une nouvelle variante inspirée du C-ACO. Ce nouvel algorithme, devrait nous permettre, en évitant les minima locaux, d'améliorer la vitesse de convergence et par conséquent la qualité des solutions trouvées. Nous présentons dans ce qui suit le principe de l'algorithme proposé.

2.3.1 Principe

À travers la nouvelle méthode proposée, nous avons essayé d'introduire les améliorations suivantes [Allo 12a] :

- le renforcement de l'intensification de l'algorithme et l'évitement de la convergence prématurée vers des régions sous-optimales (obtenue par une ou plusieurs fourmis défaillantes) par : (i) l'introduction d'une modification au niveau de la règle aléatoire de transition (celle utilisée dans la variante ACS), (ii) l'emploi d'un mécanisme sélectif lors de la mise-à-jour des pistes de phéromone.
- l'accélération de convergence de l'algorithme par l'ajout d'un processus de stockage de bonnes solutions qui vient juste après l'opération de mise-à-jour des pistes de phéromone.

L'algorithme proposé s'exécute lors en deux cycles dépendant, le premier cycle comprend une seule itération ($t = 1$), tendis que le second contient les itérations restantes ($t = 2, \dots, t_{max}$). En effet, l'objectif de cette partition dans le temps d'exécution est l'activation d'une soustraction systématique de bonnes solutions dès la première évaluation. L'algorithme commence par une initialisation ordinaire de tous ses paramètres, comme dans les versions classiques, AS et ACS.

Dans ce qui suit, nous détaillons le formalisme et la méthode d'adaptation de cet algorithme aux problèmes d'optimisation combinatoire mon-objectif, ainsi que toutes les étapes par d déroulement complet de ce dernier.

2.3.2 Formalisme et méthode d'adaptation de la variante proposée aux problèmes d'optimisation combinatoire

Les problèmes d'optimisation traités dans ce travail de thèse, sont de type combinatoire. De ce fait, on peut représenter l'ensemble de toutes les sous-solutions (combinaisons de la solution globale) cherchées par la variante proposée, avec un vecteur P_S exprimé comme suit :

$P_S = [p_{S1}\ p_{S2}\ p_{S3}\ p_{S4,\ldots\ldots}p_{Sn}]$ avec $p_{Si}\ (i = 1, \ldots, n)$ est la sous-solution i à chercher, n représente la dimension de l'espace de recherche. Le vecteur P_S porte après un nombre d'itérations égale à t_{max} une solution optimale globale (composée de sous-solutions optimales).

Pour chaque sous-solution optimale à chercher, on attribue un ensemble de valeurs candidates. Pour simplifier, ces valeurs candidates sont uniformément distribuées sur un intervalle en utilisant les formules ci-dessous [Boube 09] dont les bornes sont fixées préalablement. Le choix de ces limites est basé généralement sur les connaissances du problème d'optimisation traité.

Supposant que $p_{Si_{min}}$ et $p_{Si_{max}}$ sont les bornes de la sous-solution p_{Si} , donc :

$$p_{Si_1} = p_{Si_{min}}, p_{Si_2} = p_{Si_1} + \frac{p_{Si_{max}} - p_{Si_{min}}}{J}, \ldots\ldots\ldots, p_{Si_J} = p_{Si_{max}} \tag{2.8}$$

Où J représente le degré de partition de l'espace de recherche relatif à chaque sous-solution p_{Si}.

Il est important de noter qu'il existe diverses méthodes de partition de l'espace de recherche (génération aléatoire), dont la plus utilisée est celle présentée par [Boube 09]. Ainsi, Le problème consiste en la recherche de meilleures sous-solutions qui permettent, après leur combinaison, de minimiser (ou maximiser) la fonction objectif $F_{obj}()$. C'est un problème d'optimisation combinatoire avec une complexité égal à J^n. La formulation du problème d'optimisation, sous la forme présentée, simplifie largement la résolution de ce dernier en utilisant un algorithme ACO.

La figure 2.5 donne une représentation graphique du problème, les n ensembles de vecteurs sous-solutions p_{Si}, sont arrangés dans n listes en colonne où chaque valeur candidate est représentée par un nœud (C_{ij}) avec $(i = 1, \ldots, n)$ et $(j = 1, \ldots, J)$.

Le tour d'une fourmi consiste en une combinaison des sous-solutions optimales du problème. À partir de son nid, une fourmi se déplace à travers les p_{Si}. Enfin, la fourmi atteigne la source d'alimentation S qui est ajoutée ici juste par analogie au monde réel des fourmis.

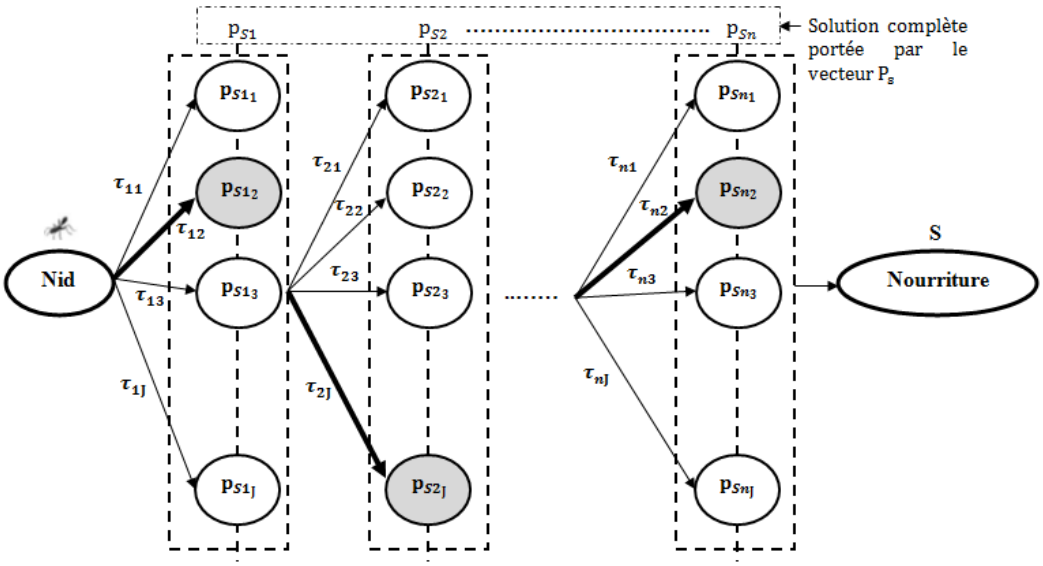

Figure 2.5 : Représentation graphique du problème d'optimisation par ACO.

2.3.3 Règle de déplacement des fourmis

Pour chaque sous-solution, le nœud visité par une fourmi est sélectionné comme une valeur de cette dernière. La sélection de la valeur d'une sous-solution est basée sur les traces de phéromone entre les vecteurs p_{Si}. La dimension de la matrice de phéromone τ est n×J et chaque élément est noté par $\tau_{ij}, (i = 1, \dots, n)$ et $(j = 1, \dots, J)$. Comme montré sur la figure 2.5, quand une fourmi sélectionne une sous-solution p_{Si_j} appartenant au vecteur p_{Si}, la sélection de la valeur de la sous-solution $p_{S(i+1)_j}$suivante dans le vecteur $p_{S(i+1)}$ dans la liste des candidates est effectué par la règle de transition aléatoire suivante qui dépond de la trace de phéromone, et un peu d'heuristique [Allo 12a].

$$j = \begin{cases} \arg max_{\ y \in J}\{\tau_{iy}(t)\} & si\ r \le r_d(t) \\ j_1 & si\ non \end{cases} \qquad (2.9)$$

Où r est une variable aléatoire uniformément distribuée sur [0,1], $r_d(t)$ est un paramètre dynamique donné par (2.10) a valeurs limitées dans [0,1], lors de l'évolution des cycles $r_d(t)$ est incrémenté avec un pas r_{pas} égal à $\frac{1-r_0}{t_{max}}$ où ; t_{max} représente le nombre maximal d'itérations. r_0 représente la valeur initiale de r_d.

$$r_d(t+1) = r_d(t) + r_{pas} = r_d(t) + \frac{1-r_0}{t_{max}} \qquad (2.10)$$

Cette incrémentation à pour but d'orienter le choix vers les sous-solutions optimales caractérisées par des pistes de phéromones condensées, chose qui force relativement l'intensification au cours de l'exécution de l'algorithme.

Le paramètre j_1 appartient à la liste des {1,2, … . , J} est sélectionné sur la base d'une règle de probabilité uniforme, l'orientation vers ce choix heuristique dépend de la valeur du paramètre r choisi aléatoirement à chaque itération t.

2.3.4 Règle de mise-à-jour des pistes de phéromone

A chaque itération $t(0 \leq t \leq t_{max})$ Chaque fourmi k $(1, \dots, m)$ suit une trajectoire qui dépend de la règle de transition (2.9), ce chemin réalisé représente en réalité le vecteur $\mathrm{P_s^k} = \left[\mathrm{p_{S1}^k}\ \mathrm{p_{S2}^k}\ \mathrm{p_{S3}^k}\ \mathrm{p_{S4}^k}, \dots \dots, \mathrm{p_{Sn}^k}\right]$ des sous-solutions optimales sélectionnées par la fourmi. La solution optimale obtenue par cette fourmi sera évaluée à travers la fonction objectif $\mathrm{F_{obj}}(\mathrm{P_s^k})$. est comparée par la suite à un seuil fixé $\mathrm{S_{seuil}}$, et aussi à la valeur de la fonction de performance $\mathrm{F_{obj}}(\mathrm{P_s^{k-1}})$ obtenue par la fourmi précédente. On réalité seule les fourmis vérifiant les deux conditions $\mathrm{F_{obj}}(\mathrm{P_s^k}) \leq \mathrm{F_{obj}}(\mathrm{P_s^{k-1}})$ et $\mathrm{F_{obj}}(\mathrm{P_s^k}) \leq \mathrm{S_{seuil}}$ sont autorisées à laisser des traces sur le graphe. La valeur $\mathrm{S_{seuil}}$ est déterminée à base d'une connaissance préalable du problème d'optimisation.

Ce mécanisme est introduit afin de contrôler l'opération de la mise-à-jour des trajectoires réalisées par chaque fourmi, ce qui permet d'éviter une convergence rapide (prématurée) vers des minimums locaux. L'équation (2.11) exprime ce mécanisme.

$$\tau_{ij}(t+1) = \begin{cases} (1-\rho)\tau_{ij}(t) + \frac{Q}{\mathrm{F_{obj}}(\mathrm{P_s^k})} & si\ \ \mathrm{F_{obj}}(\mathrm{P_s^k}) \leq \mathrm{F_{obj}}(\mathrm{P_s^{k-1}}) \text{ et } \mathrm{F_{obj}}(\mathrm{P_s^k}) \leq \mathrm{S_{seuil}} \\ pas\ de\ mise\ à\ jour & si\ non \end{cases} \tag{2.11}$$

Pour la première fourmi $(k = 1)$, la valeur de la fonction de performance $\mathrm{F_{obj}}(\mathrm{P_s^1})$ est comparée seulement au seuil $\mathrm{S_{seuil}}$, elle laisse une trace uniquement lorsque la deuxième partie de la condition (2.11) est vérifiée. En effet, ce mécanisme permet d'accélérer le processus d'exploration de bonnes solutions et d'éviter d'amener les fourmis qui ont des mauvaises solutions de laisser des traces sur le graphe.

Cet algorithme est caractérisé aussi par un mécanisme de sélection successive de bonnes solutions. Il permet de mémoriser les bonnes solutions et d'omettre en même temps les mauvaises durant tout le cycle d'exploration. On peut exprimer ce processus sélectif (en cas de minimisation de $\mathrm{F_{obj}}()$) par l'équation suivante :

$$\mathrm{F_{obj}}(\mathrm{P_s^k}) = \begin{cases} \mathrm{F_{obj}}(\mathrm{P_s^{k-1}}) & si \qquad \mathrm{F_{obj}}(\mathrm{P_s^k}) \geq \mathrm{F_{obj}}(\mathrm{P_s^{k-1}}) \\ \mathrm{F_{obj}}\left(\mathrm{P_s^k}\right) & si\ non \end{cases} \tag{2.12}$$

Ce processus permet à la colonie de fourmis de converger rapidement vers l'optimum global (meilleure solution complète). Le pseudo-code présenté dans l'algorithme 2.2 décrit la méthode développée.

Algorithme 2.2 : Algorithme ACO amélioré

Initialiser tous les paramètres de l'algorithme m, S_{seuil}, r_0, t_{max}, ρ, Q et J
Initialiser les traces de phéromone $\tau_{ij}(0) = \tau_0$, $(i = 1, \dots, n)$ et $(j = 1,2, \dots, J)$

Pour $t = 1$
  Pour la première fourmi $k = 1$
    Choisir le vecteur de paramètres P_s^1 en utilisant l'équation (2.9)
    Laisser des traces sur le graphe si : $F_{obj}(P_s^k) \leq S_{seuil}$
  Fin pour

  Pour chaque fourmi $k = 2, .., m$
    Choisir le vecteur de paramètres P_s^k en utilisant l'équation (2.9)
    Laisser des traces sur le graphe selon (2.11)
    Evaluer la fonction de performance obtenue $F_{obj}(P_s^k)$ selon (2.12)
  Fin pour

  Poser $F_{obj\ best}(t) = F_{obj}(P_s^k)$
  Incrémonter r_d selon (2.10)
Fin pour

Pour $t = 2, \dots, t_{max}$
  Pour la première fourmi $k = 1$
    Choisir le vecteur de paramètres P_s^1 en utilisant l'équation (2.9)
    Laisser des traces sur le graphe si $F_{obj}(P_s^1) \leq F_{obj\ best}(t-1)$ et $F_{obj}(P_s^1) \leq S_{seuil}$
  Fin pour
  Pour chaque fourmi $k = 2, \dots, m$
    Choisir le vecteur de paramètres P_s^k en utilisant l'équation (2.9)
    Laisser des traces sur le graphe selon (2.11)
    Evaluer la fonction de performance obtenue $F_{obj}(P_s^k)$ selon (2.12)
  Fin pour

  Poser $F_{obj\ best}(t) = F_{obj}(P_s^k)$
  Incrémonter r_d selon (2.10)
Fin pour

La performance de cette nouvelle variante sera évaluée en quatrième chapitre sur un problème d'optimisation des paramètres d'une commande décentralisée par modes glissants. En outre, les résultats obtenus seront comparés à ceux obtenus en utilisant un C-ACO. Une analyse comparative des résultats, permet d'argumenter la supériorité de l'algorithme proposé en termes de rapidité de convergence, qualité de solutions trouvées et l'évitement du problème de convergence prématurée vers des optimums locaux.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques essentiels pour une bonne conception de la méthode d'optimisation par colonies de fourmis. Nous avons expliqué les origines de l'inspiration biologique de cette approche, et nous avons présenté ses principaux schémas appliqués au problème TSP, le problème d'optimisation combinatoire le plus célèbre. En outre, la formalisation et les propriétés principales d'un algorithme ACO standard sont aussi présentées.

Ces métaheuristiques ont deux propriétés principales, lui diffèrent complètement des autres approches : (i) L'utilisation des traces de phéromone qui permet d'exploiter toutes les expériences de recherche acquissent par les fourmis et ainsi renforcer l'apprentissage pour la construction de solutions dans les itérations futures de l'algorithme. (ii) l'emploi de l'information heuristique au niveau de la règle de transition, chose qui peut guider les fourmis vers les zones prometteuses de l'espace de recherche.

Toutefois, C-ACO a ses propres inconvénients, qui résident, principalement, dans les problèmes de convergence lente et la convergence prématurée qui peut conduire l'algorithme à se stagner dans un optimum local.

Dans ce chapitre, nous avons proposé une nouvelle variante ACO, pour résoudre les problèmes mentionnés ci-dessus. La méthode proposée est basée sur l'introduction de quelques modifications sur la structure algorithmique de C-ACO notamment au niveau des règles de transition et de mise-à-jour des pistes de phéromone. La méthode proposée sera appliquée dans le quatrième chapitre pour l'optimisation des paramètres d'une commande décentralisée par modes glissants.

CHAPITRE 3

Elaboration d'une nouvelle variante d'algorithme de recherche d'harmonie : le GHSACO

3.1 Introduction

L'algorithme de recherche d'harmonie (*Harmony search* HS) développé par le professeur Geem et ses collaborateurs [Geem 01] est une métaheuristique à base de population relativement récente. Il est basé sur le processus de performance musical, qui consiste à trouver une meilleure harmonie, parmi celles produites par un orchestre musical où chaque musicien joue une note spécifique.

Les inventeurs de cet algorithme, estime qu'un processus d'improvisation musical peut être vu comme une procédure d'optimisation. Ils ont considéré, qu'un instrument musical correspond à une variable de décision dans un problème d'optimisation, dont la gamme de sa tonalité est similaire à l'intervalle à qui appartient cette variable, ainsi une harmonie correspond à un vecteur de solution.

Comparé aux autres métaheuristiques, le HS impose moins d'exigences mathématiques (très facile à programmer), il est aussi très flexible, en terme d'adaptation aux différents problèmes d'optimisation. En outre, des études comparatives ont montré que le HS est plus rapide (complexité de calcul et évolution) par rapport aux AGs [Geem 01]. Par conséquent, ces caractéristiques ont motivées les chercheures d'appliquer cet algorithme dans une large gamme de problèmes d'optimisation théoriques et réels, tels que p. ex. ; la conception des réseaux de distribution d'eau [Geem 06], l'acheminement des véhicules [Geem 05], réglage des paramètres des commandes adaptives [Shar 10, Wang 13b], conception optimale des générateurs à base d'énergie éolienne [Gao 10], conception optimale des positions d'amortisseurs structurels [Amin 13], problèmes d'ordonnancement [Chen 12, Esta 14] et plusieurs d'autres [Manj 13].

L'algorithme HS a prouvé, qu'il a la possibilité de découvrir des régions prometteuses dans l'espace de recherche dans des délais très raisonnables. Cependant, il montre une vulnérabilité significative en cas d'une recherche locale [Mahd 07]. Ainsi, plusieurs variantes de HS ont été développées dès son apparition, pour améliorer sa précision (qualité de solution) et son taux de convergence [Moh'd 11]. Les variantes proposées portent généralement des modifications sur l'algorithme original, elles peuvent être classées en deux grandes catégories : (1) Adaptation

dynamique des paramètres de l'algorithme, c.-à-d. proposition des versions adaptatives (2) Hybridation de l'algorithme HS avec d'autres métaheuristiques [Moh'd 11].

Dans ce chapitre, nous proposons une nouvelle variante d'algorithme HS appelée (GHSACO), basée sur l'hybridation entre une version de HS appelée algorithme de recherche d'harmonie globale (*Global Best Harmony Search* GHS) [Omra 08] et l'algorithme C-ACO. En effet, dans cette nouvelle variante, le but de l'utilisation de l'algorithme GHS est d'accéléré la diversification de la recherche, tandis que l'emploi de l'algorithme C-ACO avec une configuration spécifique (détaillée dans la section 3.4.2), qui conduit à une large concentration sur l'intensification, est d'ajouté une règle supplémentaire de considération de la mémoire (*Memory Consideration Selection Rule* MCSR) au processus d'improvisation global de l'algorithme HS, et ceci à travers la règle aléatoire de transition avec l'intégration d'un mécanisme de mise-à-jour des pistes de phéromone. Les modifications introduites, permettent d'obtenir une nouvelle version de HS caractérisée par un taux de convergence et une intensification (exploitation locale) améliorés. De plus, la qualité des solutions trouvées par GHSACO est meilleure comparée aux algorithmes HS, GHS ainsi la variante, recherche d'harmonie amélioré (*Improved Harmony Search* IHS) [Mahd 07], en raison d'une capacité d'exploration plus grande due à l'hybridation avec le C-ACO. Le test de cette nouvelle méthode, dans le contexte d'un ensemble de quelques fonctions de *benchmark* mono-objectifs, s'est avéré satisfaisant dans le sens où elle permet d'atteindre souvent une meilleure performance.

Cette méthode sera aussi appliquée dans un problème de réglage des paramètres d'une commande adaptative. Le détaille de cette application, sera présenté dans le chapitre 5.

3.2 Algorithme HS est ses variantes, IHS et GHS

3.2.1 Algorithme HS

L'algorithme HS est une métaheuristique à population, proposée récemment (en 2001) par le professeur Geem et ses collègues dans leur travail publié dans [Geem 01].

Cet algorithme, est basé dans son principe sur la modalisation du processus d'improvisation musicale, Dans lequel tous les membres de l'orchestre (musiciens) cherchent d'improviser leurs tonalités afin de trouver la meilleure harmonie. En effet, ce mécanisme musical est similaire à une procédure (algorithme) d'optimisation qui cherche à trouver un meilleur état (optimum global, coût minimal ou un rendement maximal) déterminé par l'évaluation d'une fonction objectif. L'état d'une estimation esthétique, résultante d'une harmonie dans le fonctionnement de l'ensemble des instruments de l'orchestre, est semblable à l'évaluation d'une fonction objectif, déterminée par l'ensemble des variables définissant le problème d'optimisation. Ainsi, la qualité d'un morceau musical, peut être améliorée au fur et à mesure au cours du temps, basant sur la

répétition et l'expérience, tout comme les valeurs des variables explorées pour l'obtention d'une meilleure évaluation de la fonction objectif, peuvent être améliorées itération par itération.

L'exécution d'une itération dans l'algorithme HS comprend cinq étapes principales, qui sont respectivement [Mahd 07] :

Étape 1. Initialisation du problème d'optimisation et paramètres de l'algorithme,
Étape 2. Initialisation de la mémoire d'harmonie (*Harmony Memory* HM),
Étape 3. Improvisation d'une nouvelle harmonie à partir de HM,
Étape 4. Mise-à-jour de HM,
Étape 5. Répétition des étapes 3 et 4 jusqu'à ce que le critère d'arrêt soit satisfait.

Dans ce qui suit, nous décrivons en détaille les étapes précédentes ;

Étape 1. *Initialisation du problème d'optimisation et paramètres de l'algorithme* :

Généralement, au départ tout problème d'optimisation comprend deux étapes principales ; (i) définition du problème, (ii) initialisation des paramètres d'algorithme d'optimisation.

$$Minimiser\ \mathrm{F}_{\mathrm{obj}}(x)\ soumise\ aux\ x_i \in [LB_i UB_i], i = 1, \dots, n, \tag{3.1}$$

Avec $\mathrm{F}_{\mathrm{obj}}(x)$ est la fonction objectif, $x = (x(1), x(2), \dots, x(n))$ c'est un ensemble de variables de décision, n est le nombre des variables de décision, LB_i et UB_i sont les limites supérieur et inferieur du variable $x(i)$.

Les paramètres de l'algorithme HS, requis à résoudre le problème d'optimisation définit dans (3.1), sont également spécifiées dans cette étape, ils sont respectivement : la taille de la mémoire d'harmonie (*Harmony Memory Size* HMS), ce paramètre représente le nombre des vecteurs de solutions dans la matrice HM, le taux de considération de la mémoire d'harmonie (*Harmony Memory Consideration Rate* HMCR), le taux d'ajustement de la tonalité (*Pitch Adjustment Rate* PAR), le paramètre distance de la bande passante (*distance bandwidth bw*) et le critère d'arrêt (nombre maximal d'itérations ou le nombre d'improvisations NI). Les paramètres HMCR et PAR sont utilisés spécifiquement à l'étape 3, dans le processus d'improvisation d'une nouvelle harmonie [Mahd 07].

Étape 2. *Initialisation de* HM

Dans cette étape, la matrice mémoire d'harmonie HM donnée dans (3.2), est remplie avec HMS vecteurs de solutions, générées aléatoirement et triés en se référant aux valeurs de $\mathrm{F}_{\mathrm{obj}}()$ obtenues en fonction de ces vecteurs [Mahd 07].

$$\text{HM} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{HMS} \end{bmatrix} = \begin{bmatrix} x_1(1) & x_1(2) & \dots & x_1(n) \\ x_2(1) & x_2(2) & \dots & x_2(n) \\ \vdots & \vdots & \dots & \vdots \\ x_{HMS}(1) & x_{HMS}(2) & \dots & x_{HMS}(n) \end{bmatrix} \quad (3.2)$$

Étape 3. *Improvisation d'une nouvelle harmonie à partir de* HM

Dans cette étape, un nouveau vecteur d'harmonie X_{new} (3.3) est généré en utilisant trois règles différentes ; (i) règle de considération de la mémoire (*memory considerations*), (ii) règle de réglage de la tonalité (*pitch adjustments*), (iii) une sélection aléatoire.

$$X_{new} = \{x_{new}(1), x_{new}(2), \dots., x_{new}(n)\} \quad (3.3)$$

Au départ, un nombre aléatoire uniforme r_0 est généré dans l'intervalle [0,1]. Si la condition $r_0 \leq$ HMCR est vérifiée, la variable de décision $x_{new}(j)(j \in [1, n])$ est choisi aléatoirement parmi tous les vecteurs d'harmonies $x_i(j)(i \in [1, \text{HMS}])$ de la matrice HM.

Par ailleurs, $x_{new}(j)$ est obtenue par une sélection aléatoire effectuée comme suit :

$$x_{new}(j) = LB_j + rand() \times \left(UB_j - LB_j\right) \quad (3.4)$$

Où $rand$ () est un nombre aléatoire uniforme généré dans l'intervalle [0,1].

D'autre part, chaque variable de décision $x_{new}(j)$ subit un ajustement de la tonalité si la condition $r_1 \leq$ PAR avec $r_1 \in [0,1]$ est vérifiée, ainsi que celle imposée par la règle de considération de la mémoire ($r_0 \leq$ HMCR).

La règle d'ajustement de la tonalité est exprimée par l'équation suivante :

$$x_{new}(j) = LB_j + rand() \times bw \quad (3.5)$$

Dans cette étape, les trois règles susmentionnées sont appliquées séquentiellement à chaque élément de la nouvelle harmonie.

Étape 4. *Mise à jour de* HM

Si, la valeur de la fonction objectif $\text{F}_{\text{obj}}(X_{new})$ obtenue à partir du nouveau vecteur d'harmonie X_{new} est meilleure que la mauvaise X_{mauvaise} dans la matrice HM , X_{mauvaise} est remplacée par X_{new} dans HM.

Étape 5. *Vérification du critère d'arrêt*

Dans cette étape, un arrêt d'exécution du programme est effectué, lorsque la condition d'arrêt est vérifiée. Souvent cette condition est associée au nombre d'itérations (Si $t \geq NI$, avec t est l'itération courante), sinon, les étapes 3 et 4 seront répétées.

3.2.2 Algorithme IHS

Afin d'améliorer la précision et la vitesse de convergence de l'algorithme HS original, Mahdavi et ses collègues ont proposé une nouvelle version de celui-ci, appelée algorithme recherche d'harmonie amélioré IHS [Mahd 07].

La différence fondamentale entre le IHS et HS réside principalement dans le fait que, dans le IHS, les deux paramètres PAR et bw sont ajustés dynamiquement en fonction des itérations t comme le montrent les deux équations (3.6) et (3.7).

$$\mathrm{PAR}(t) = \mathrm{PAR}_{\min} + (\mathrm{PAR}_{\max} - \mathrm{PAR}_{\min}) \times t/IN \tag{3.6}$$

$$bw(t) = bw_{\max} \times \exp(\mathrm{In}(bw_{\min}/bw_{\max}) \times t/IN) \tag{3.7}$$

Avec PAR_{min},$bw_{\min}$, PAR_{max} et $bw_{\max}$ sont respectivement les valeurs minimales et maximales des paramètres PAR et bw, t est l'itération courante.

Il est à noter que, selon un grand nombre d'études, par exemple dans [Mahd 07, Omra 08, Allo 15a], le IHS a monter son efficacité par rapport à HS, en terme de rapidité de convergence et de qualité des solutions trouvées.

3.2.3 Algorithme GHS

Inspiré de l'algorithme PSO, une nouvelle variante de l'algorithme HS a été proposée par Omran et Mahadavi dans leur article [Omra 08]. Dans cette nouvelle approche dénommée GHS, l'improvisation d'une nouvelle harmonie est basée sur l'utilisation du meilleur vecteur d'harmonie $X_B = \{x_B(1), x_B(2), \ldots., x_B(n)\}$ dans le HM. En effet, le changement proposé permet d'éliminer complètement la phase de réglage de la tonalité, ce qui rend le GHS entièrement indépendant du paramètre bw.

De plus, le paramètre PAR est exprimé, dans cette variante, sous une forme (dynamique) adaptable au cours des itérations comme suit :

$$\mathrm{PAR}(t) = \mathrm{PAR}_{\min} + (\mathrm{PAR}_{\max} - \mathrm{PAR}_{\min}) \times t/IN \tag{3.8}$$

Avec PAR_{min} et PAR_{max} sont respectivement les valeurs minimale et maximale du paramètre PAR, t est l'itération courante.

L'algorithme GHS a exactement les mêmes étapes d'exécution que l'algorithme IHS, sauf l'étape de la phase de réglage de la tonalité, qui est modifiée comme suit :

$$x_{new}(j) = x_B(k), k \in [1, n] \tag{3.9}$$

Avec k est un nombre entier aléatoire compris entre 1 et n.

3.3 Caractéristiques fondamentales de l'algorithme HS

L'algorithme HS, a plusieurs caractéristiques qui font de lui, l'une des métaheuristiques les plus importantes. Parmi celles-ci, on peut citer les suivantes [Moh'd 11] : (1) La génération d'un nouveau vecteur d'harmonie est effectuée, après avoir examiné tous les vecteurs existants dans le HM. L'importance de cette spécificité apparaît clairement dans le cas des AGs, dans lesquels deux vecteurs (parents) sont employés pour la production d'un nouveau vecteur.

(2) Le HS a trois règles intégrales très importantes ; règle de considération de la mémoire, règle d'adaptation de la tonalité et celle de la sélection aléatoire. En réalité, ces trois mécanismes représentent le noyau de HS. Ils permettent la réalisation de l'équilibre souhaité entre l'intensification et la diversification.

Plus précisément, la diversification de HS est contrôlée par les deux règles ; celle qui permet l'ajustement de la tonalité et celle qui assure la sélection aléatoire. Ainsi, les différentes composantes du nouveau vecteur, généré par une sélection aléatoire, sont produites avec une efficacité acceptable [Moh'd 11].

La règle d'adaptation de la tonalité est un mécanisme crucial. Elle offre une adaptation fine des solutions locales, ce qui permet la prise en compte de leurs meilleures classes. En outre, elle permet d'élargir l'exploration de l'espace de recherche, par l'ajout/soustraction d'une petite quantité aléatoire ($\in [0,1]$) dans des limites ajustées par la variable bw. Par ailleurs, cette règle peut soutenir l'intensification de HS à travers le test lié à PAR (Étape 3 dans le processus de l'exécution de HS). Un autre mécanisme caractérise le HS, n'est pas moins important aux précédents, le mécanisme (règle) de considération de la mémoire, il permet d'introduire la propriété intensification à l'algorithme HS. Ainsi, sont processus est équivalent à une attitude élitiste.

(3) La structure algorithmique de HS est relativement aisée. Ceci facilite largement sa programmation ainsi que son hybridation avec d'autres méthodes [Moh'd 11].

3.4 Nouvelle variante d'algorithme HS, le GHSACO

Dès son apparition, l'algorithme HS à attiré l'attention d'un grand nombre de chercheurs dans différents domaines afin de résoudre une large gamme de problèmes d'optimisation [Manj 13]. Par conséquent, les chercheurs se sont investis pour améliorer et développer ses performances en parallèle avec les exigences imposées par les problèmes d'optimisation posés. En effet, ces améliorations sont basées essentiellement sur les deux aspects suivants : (1) les méthodes pour l'adaptation des paramètres de l'algorithme, et (2) l'hybridation avec d'autres méthodes.

Le travail présenté dans ce chapitre rentre dans ce cadre. En effet, nous proposons dans une modification de l'algorithme HS afin d'améliorer ces performances. Une nouvelle variante nommée (GHSACO) [Allo 15a], basée sur le principe d'hybridation entre deux métaheuristiques est donc proposée. Elle consiste en la fusion entre une version de HS le GHS et l'algorithme C-ACO.

Dans ce qui suit, nous détaillons le principe de la méthode proposée ainsi qu'une évaluation complète de ses pertinences à travers une étude comparative avec d'autres versions de HS.

3.4.2 Principe de l'algorithme proposé

Généralement, la résolution d'un problème d'optimisation discrète (combinatoire) avec un algorithme ACO, revient, comme nous l'avons vu au chapitre 2, à trouver le plus court chemin reliant un ensemble de nœuds dans un graphe représentant l'espace de recherche. Par ailleurs, le mouvement de chaque fourmi entre les différents nœuds est, habituellement, contrôlé par une règle de déplacement appelée règle aléatoire de transition proportionnelle.

Au chapitre 2, dans la section 2.3.3, nous avons proposé l'utilisation d'une règle de déplacement [Allo 12a] basée sur une permutation aléatoire contrôlée par une variable q choisie aléatoirement dans l'intervalle [0,1]. La valeur de q est comparée chaque fois lors de l'appel de cette règle à un paramètre $r_d(t)$ dynamique. En réalité, la forme la plus simple de cette règle commence par le cas d'un paramètre fixe q_0 ($0 \leq q_0 \leq 1$) au lieu de $r_d(t)$ comme le montre (3.10).

$$j = \begin{cases} \arg max_{y \in J}\{\tau_{iy}(t)\} & si \quad q \leq q_0 \\ j_1 & si \quad sinon \end{cases} \tag{3.10}$$

Selon (3.10), une fourmi k s'oriente vers sa nouvelle destination j à partir de i en basant dans un premier temps sur la valeur de q, et par la suite sur l'intensité $\tau_{ij}(t)$ ($i = 1, \ldots, n$ et $j = 1, \ldots,$ HMS) de phéromone déposé sur la piste la plus condensée ou sur une règle aléatoire uniforme.

Il est à noter que, l'importance du paramètre q_0 réside dans le fait, qu'il permet d'orienter l'algorithme vers trois états différents : vers une diversification forcée, vers une intensification forcée ou vers un état d'équilibre entre les deux. Ainsi, nous pouvons forcer l'intensification (exploitation des informations collectées par la colonie) en choisissant q_0 relativement grand ($q_0 \geq 0.75$) dans l'intervalle [0,1].

À travers la nouvelle méthode proposée, nous avons tenté de combiner, d'un côté, le processus d'accélération de la diversification, qui caractérise l'algorithme GHS et de l'autre côté, l'exploitation (intensification) forcée présentée par l'algorithme ACO après l'affectation d'une valeur ≥ 0.75 au paramètre q_0.

L'hybridation de C-ACO avec le GHS, permet d'ajouter une règle supplémentaire de MCSR au processus d'improvisation global d'algorithme GHS à travers la règle (3.10).

De plus, l'algorithme GHSACO est doté de deux mécanismes principaux ; (1) d'une procédure de commutation entre le GHS et le C-ACO, contrôlée par un paramètre $q_1 \in [0,1]$, (2) et d'un mécanisme global de mise-à-jour des pistes de phéromone.
En effet, au début de l'exécution de GHSACO deux scénarios différents, sont possibles :

- Le Premier scénario : si $q_1 = rand() > q_0$, dans ce cas le GHSACO se comporte comme un algorithme GHS.
- Le deuxième scénario : si $q_1 = rand() \leq q_0$, le GHSACO se comporte comme un algorithme C-ACO.

Dans ce second cas, un nouveau schéma de la matrice HM est introduit au processus de recherche global du GHSACO, sous forme d'un graphe constitué d'un ensemble de nœuds, où chaque nœud représente une valeur candidate.

Dans le HS, au début de l'exécution une matrice HM est générée avec un nombre d'harmonies égal à HMS. D'où chaque harmonie est représentée par un vecteur de n valeurs réelles. Par conséquent, le nombre total de nœuds dans le HM est égal à $n \times$ HMS. Or, chaque nœud dans la colonne $j_{C-1} \in \{1,2,3, \ldots . n-1\}$ est associé à tous les nœuds de la colonne suivante $j_C \in \{2,3, \ldots . n\}$ par l'intermédiaire de HMS pistes de phéromones. Le graphe qui modélise le HM est traversé par une seule fourmi chaque fois que la condition de passage au second scénario est satisfaite.

Débutant son parcourt à partir d'une position initiale (le nid), une fourmi se déplace à travers tous les nœuds du graphe et s'arrête à la fin de son déplacement à un nœud (final) qui représente l'une des valeurs candidats (appartient au HM) suivantes $\{x_1(n), x_2(n), \ldots . , x_{\mathrm{HMS}}(n)\}$. La représentation graphique de HM dans ce cas est illustrée par la figure 3.1, où le chemin réalisé par la fourmi est marquée par un trait gras. Les valeurs

candidates sélectionnées (nœuds), comme indiquer sur la figure 3.1, sont respectivement $\{x_2(1), x_3(2), \ldots., x_1(n)\}$.

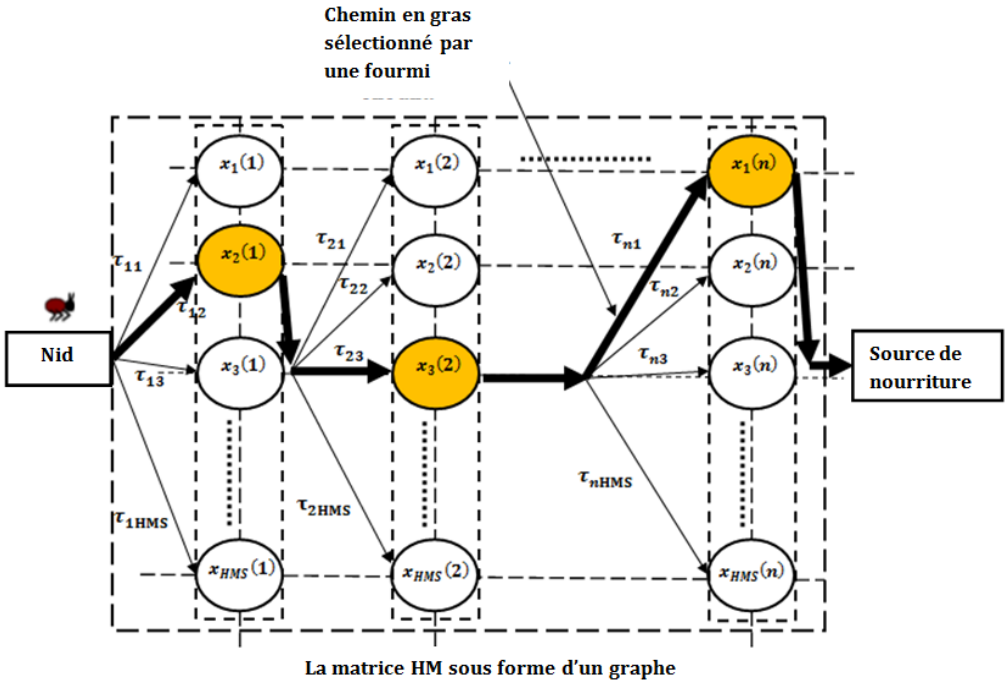

Figure 3.1 : Représentation graphique de la matrice HM, où le chemin effectué par une fourmi est marqué par un trait gras.

En exploitant cette nouvelle formulation de HM, nous pouvons ajouter une règle supplémentaire de MCSR à l'algorithme GHS. Par conséquent, chaque variable de décision $x_{new}(j)(j \in [1, n])$ est choisi à partir de n'importe quel vecteur d'harmonie $x_i(j)(i \in [1, \text{HMS}])$ basée sur la concentration des traces de phéromone τ_{ij} en utilisant (3.10).

Le processus de mise-à-jour de HM est effectuée juste après la satisfaction de la condition $q_1 = rand() \leq q_0$. Ce processus, est réalisé par un remplacement de la mauvaise harmonie existante dans le HM, par celle juste trouvée. Ceci après la satisfaction de la condition $\text{F}_{\text{obj}}(X_{new}) < \text{F}_{\text{obj}}(X_{\text{mauvaise}})$.

Finalement, la valeur $\text{F}_{\text{obj}}(X_{new})$ obtenue que ce soit dans le premier cas (GHS) ou dans le second cas (C-ACO) est utilisée pour mettre à jour la concentration des pistes de phéromone, en utilisant la règle suivante :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \frac{Q}{\text{F}_{\text{obj}}()} \tag{3.11}$$

Sachant que les paramètres ρ et Q sont respectivement le taux d'évaporation et un paramètre fixe. L'organigramme de GHSACO est représenté par figure 3.2.

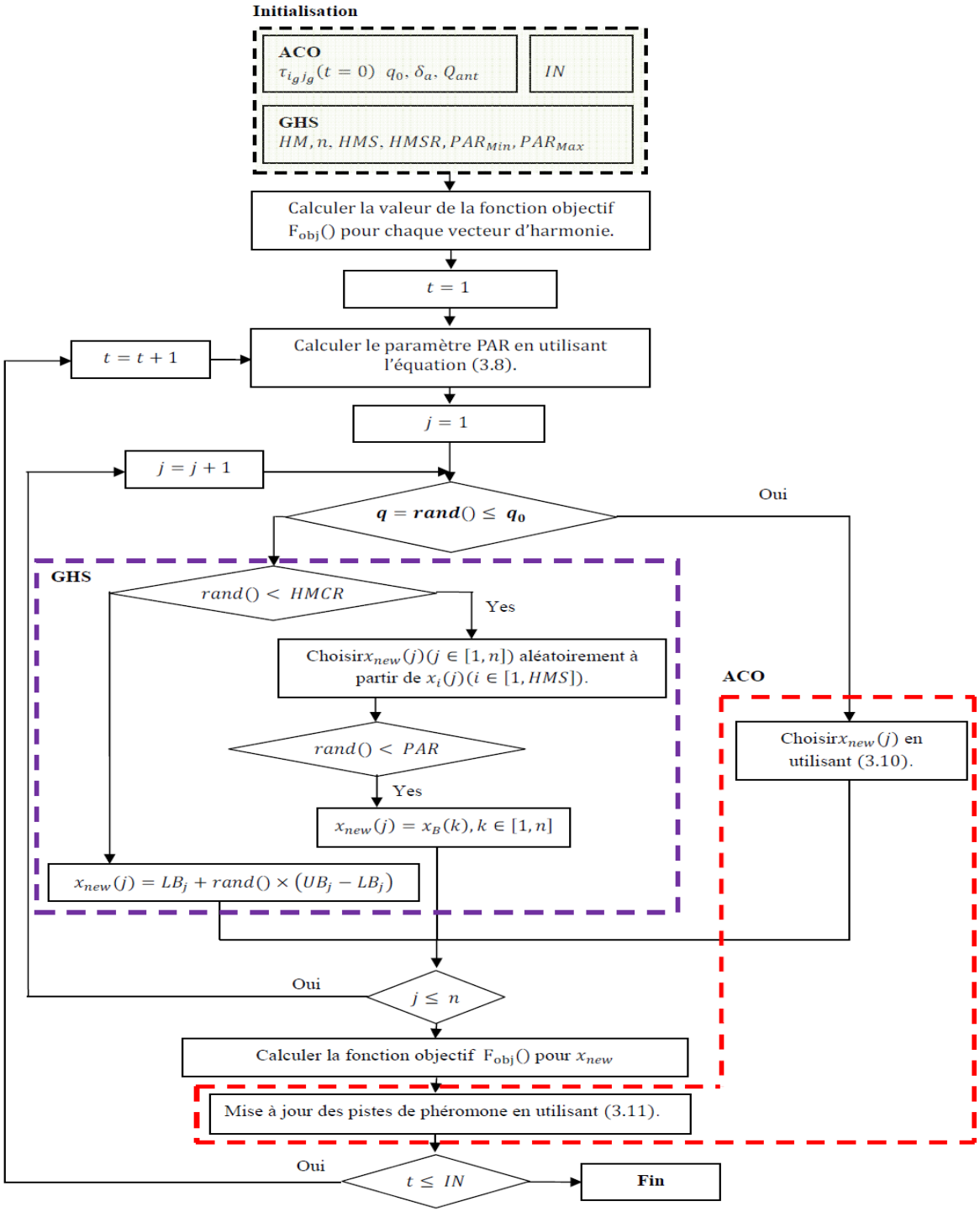

Figure 3.2 : Organigramme de l'algorithme GHSACO.

3.5 Résultats obtenus

3.5.1 Problèmes de test

Afin d'évaluer l'efficacité de la nouvelle variante hybride proposée, nous l'avons comparée aux algorithmes de base, HS, GHS et IHS sur un ensemble de fonctions-tests (*benchmark*). Les

fonctions F1 à F5 ne possèdent qu'un minimum global, tandis que les fonctions F6 à F10 sont des fonctions hautement multimodales, c.-à-d. le nombre de pics et de vallées augmente avec l'augmentation du nombre de dimensions *D*.

Les différentes caractéristiques de ces fonctions peuvent être vues dans le tableau 3.1.

Fonctions-test	S
Fonctions unimodales [Molg 05, Yao 99, Ali 05]	$[-100,100]^D$
Sphere function $f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-10,10]^D$
Schwefel problem 2.22 $f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-100,100]^D$
Step function $f_3(x) = \sum_{i=1}^{n} (x_i + 0.5)^2$	$[-100,100]^D$
Quartic function with noise $f_4(x) = \sum_{i=1}^{n} i x_i^4 + rand(0.1)$	$[-1.28, 1.28]^D$
Rotated hyper-ellipsoid function $f_5(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_i\right)^2$	$[-100,100]^D$
Fonctions multimodales [Molg 05, Yao 99, Ali 05]	$[-100,100]^D$
Rosenbrock function $f_6(x) = \sum_{i=1}^{n} \left(100\left(x_i - x_{i-1}^2\right)^2 + (x_{i-1} - 1)^2\right)$	$[-30,30]^D$
Rastrigin function $f_7(x) = \sum_{i=1}^{n} \left(x_i^2 - 10cos(2\pi x_i) + 10\right)$	$[-5.12, 5.12]^D$
Ackley function $f_8(x) = -20exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{30}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e$,	$[-32,32]^D$
Griewank function $f_9(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600,600]^D$
Generalized Schwefel Problem 2.26 $f_{10}(x) = 418.9829n - \sum_{i=1}^{n} \left(x_i sin\left(\sqrt{\lvert x_i \rvert}\right)\right)$	$[-500,500]^D$
Fonctions avec petite dimensions [Molg 05, Ali 05]	$[-100,100]^D$
Six-hump Camel-back function $f_{11}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	$[-5,5]^D$
Schaffer 1 problem (SF1) $f_{12}(x) = 0.5 + \frac{\left(sin\sqrt{x_1^2+x_2^2}\right)^2 - 0.5}{\left(1+0.001(x_1^2+x_2^2)\right)^2}$	$[-100,100]^D$

Tableau 3.1 : Caractéristiques des fonctions du benchmark utilisées dans notre étude, avec *S* correspond au domaine de définition de chaque fonction et *D* est le nombre de dimensions.

Pour chacune des fonctions présentées, le but est de trouver le minimum global $\vec{X}^*$ de F sur S telle que $F(\vec{X}^*) \leq F(\vec{X}), \forall x \in S; S \subseteq \mathbb{R}^D$ est un ensemble non vide et $\vec{X} \in S$ est le vecteur solution des n variables de décision $\vec{X} = [X_1, X_2, \dots, X_n]^n$ où chaque $X_1 (i = 1, \dots, n)$ est bornée par les limites inférieure et supérieure $L_i \leq X_1 \leq U_i$qui définissent l'espace de recherche S, D représente le nombre de dimensions de la fonction et $F(\vec{X}^*)$ désigne la valeur de la fonction à l'optimum global.

3.5.2 Paramétrage

Nous présentons dans cette sous-section la liste des paramètres des algorithmes GHSACO, GHS, IHS et HS utilisés dans cette étude. Le réglage de ces paramètres est effectué comme suit : le

paramètre HMS est choisi égal à 5 pour tous les algorithmes. Les paramètres de HS sont fixés de la même manière que ceux dans [Omra 08], tels que HMCR $=$ 0.9, PAR $=$ 0.3, and bw $=$ 0.01. Ce choix est motivé par le fait que l'algorithme HS avec ce paramétrage permet d'obtenir de bons résultats sur les différents tests réalisés.

Après plusieurs essaies, les paramètres de IHS sont définis comme suit : HMCR $=$ 0.95, PAR_{min} $=$ 0.35, PAR_{max}= 0.99, bw_{min} $= 1e-8$, bw_{max} $= (UB_i - LB_i)/20$, avec ces paramètre le IHS a permis d'obtenir de bons résultats. Pour l'algorithme C-ACO, les paramètres q_0, ρ et Q ont respectivement les valeurs 0.75, 0.1 et 1.

En outre, toutes les pistes de phéromone dans le graphe (HM dans le second cas) ont été initialisées avec les mêmes concentrations $\tau_{ij}(0)$, et $\tau_{ij} = \tau_{ij}(0) = 0.01 \times ones(n, \text{HMS})$. Pour le GHS, soit dans le cas, lorsque il est hybridé avec C-ACO ou dans le cas où il ne l'est pas,nous avons utilisé les valeurs suivantes : HMCR $= 0.9$, $\text{PAR}_{min} = 0.35$, et $\text{PAR}_{max} = 0.99$.

Les quatre algorithmes sont initialisés de la même manière de sorte que la comparaison soit la plus significative possible. Toutes les fonctions de test, à l'exception de f_{11} et f_{12} qui sont des fonctions bidimensionnelle, sont testées avec des dimensions égales à $D = 30$ et $D = 50$. Une série d'expérience de 30 essais chacune ont été effectuées pour ces fonctions de benchmark. Le critère d'arrêt des algorithmes est défini en fonction du nombre maximal d'évaluations (NI) de la fonction objectif. Ce paramètre est fixé à 50.000.

3.5.3 Résultats et discussions

Les valeurs moyennes des optimums globaux et l'écart type (*std*.) générés par les quatre algorithmes avec les deux dimensions imposées, sont illustrées dans les tableaux 3.2 et 3.4, respectivement. Pour plus de clarté, les solutions obtenues, qui sont de meilleure qualité, sont mises en gras.

De plus, les courbes d'évolution de sept fonctions (présentée ici comme des exemples) relatifs au meilleur test (parmi les 30) pour les quatre algorithmes, sont représentées respectivement dans les figures 3.3 jusqu'à 3.9.

Par ailleurs, pour chaque fonction du benchmark, nous avons voulu vérifier si les différences entre les solutions trouvées par GHSACO et les algorithmes GHS, IHS et HS étaient statistiquement significatives. À cet effet, nous avons réalisé le test de Wilcoxon (*Wilcoxon' rank sum test*) avec un niveau de signification égal à 5% pour GHSACO par rapport aux autres algorithmes.

Les valeurs h indiquées dans les tableaux 3.3 et 3.5, représentent les résultats de tous les tests effectués. En effet, une valeur de h égale à 1 ou -1, indique que les résultats obtenus par l'algorithme en question, sont meilleurs ou mauvais que ceux obtenus par son rival, alors qu'une

valeur de h égale à 0 signifie que les résultats obtenus par les deux algorithmes ne sont pas significativement différents.

Les résultats illustrés sur les deux tableaux 3.2 et 3.4 montrent clairement l'efficacité de cette nouvelle méthode, par rapport aux autres algorithmes. Or, nous observons que GHSACO permet d'obtenir 11 meilleurs résultats sur 12 fonctions de test pour le premier cas lorsque $D = 30$, et 09 bons résultats sur 11 fonctions de test pour le second cas $D = 50$.

Plus précisément, en se référant aux tableaux 3.3 et 3.5, on trouve que le GHSACO dépasse le HS dans tous les résultats obtenus. Ainsi, Il permet d'obtenir 11 bons résultats sur 12 par rapport à IHS, sauf dans le cas de la fonction *Schaffer 1 problem* (f_{12}), où les deux algorithmes présentent les mêmes résultats. Si on compare avec le GHS, on observe la supériorité de GHSACO dans 08 cas sur 12, et une ressemblance dans 04 cas qui sont respectivement les fonctions *Schwefel problem* 2.22 (f_2), *Rotated hyper-ellipsoid function* (f_5), *Six-hump Camel-back function* (f_{10}) et *Schaffer 1 problem* (f_{12}).

Une petite comparaison entre les résultats, indiqués dans les deux tableaux 3.2 et 3.4, montre une dégradation dans les performances de tous les algorithmes, cela est dû au fait que la difficulté des problèmes de test, agrandit lors d'une augmentation dans la dimension ***D***. Ainsi, les courbes d'évolution présentées dans les figures 3.3 jusqu'à 3.9, montrent que le **GHSACO** est meilleur que tous autres algorithmes en termes de rapidité de convergence et performances des résultats obtenus.

Fonction	Minimum Global	HS	IHS	GHS	GHSACO
f_1	0	0.03575054 (0.00889610)	1.99045445 (0.70988428)	0.01924310 (0.03533083)	**0.00173943** **(0.00365836)**
f_2	0	0.59417414 (0.06928637)	0.52417177 (0.09721411)	0.037602415 (0.034258166)	**0.01856376** **(0.02458498)**
f_3	0	0.03551193 (0.00801055)	1.922005087 (0.619447737)	0.01858366 (0.03684402)	**0.00288816** **(0.00466383)**
f_4	0	0.03740598 (0.01110534)	0.05509840 (0.01896946)	0.00914848 (0.00892074)	**0.00390394** **(0.00424018)**
f_5	0	3.8606285E+003 (1.1173112E+003)	4.2859834E+003 (1.1951669E+003)	2.3419564E+003 (3.3299571E+003)	**1.1580277E+003** **(1.6793734E+003)**
f_6	0	2.4171467E+002 (2.5281415E+002)	2.1004135E+002 (1.022655E+002)	60.04486897 1.19264913E+002	**18.15048088** **(17.63759615)**
f_7	0	20.24945296 (3.61538305)	0.99083536 (0.39770589)	0.004156309 (0.00927239)	**9.0581948E-004** **(0.00160881)**
f_8	0	1.77977943 (0.30980711)	0.67729801 (0.17719250)	0.03038325 (0.02829103)	**0.00784092** **(0.00855341)**
f_9	0	1.011631126 (0.062524235)	0.95743520 (0.09396111)	0.059356048 (0.146132524)	**0.00323038** **(0.00431834)**
f_{10}	0	9.10649688 (5.33577647)	5.85402392 (2.30540959)	0.04490093 (0.07782559)	**0.00562872** **(0.00723361)**
f_{11}	−1.0316285	-1.03162845 (6.3055671E-009)	-1.03151594 (1.4342794E-004)	-1.03146107 (1.691752E-004)	-1.03125078 (4.3700042E-004)
f_{12}	0	0.018237571 0.013867884	0.019116821 0.228230066	**0.012490576** 0.014830686	0.015640433 **0.012620615**

Tableau 3.2 : Comparaison entre GHSACO, GHS, IHS et HS en termes de valeur moyenne des minimum globaux et l'écart type (*std*.) obtenus durant les 30 essais, avec $D = 30$.

Problème	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
GHSACO,HS	1	1	1	1	1	1	1	1	1	1	1	1
GHSACO,IHS	1	1	1	1	1	1	1	1	1	1	1	0
GHSACO,GHS	1	0	1	1	0	1	1	1	1	1	0	0

Tableau 3.3 : Résultat du test Wilcoxon entre GHSACO et GHS, IHS et HS respectivement, avec $D = 30$.

Fonction	Minimum Global	HS	IHS	GHS	GHSACO
f_1	0	5.16837319 (1.28049749)	24.1286865 (7.04249540)	2.05065313 (3.5175047)	**0.00388146** **(0.0044600)**
f_2	0	6.45273115 (1.06540049)	2.19517586 (0.36315675)	0.41738291 (0.40740968)	**0.02000423** **(0.02361815)**
f_3	0	4.96754694 (1.14188237)	22.02971425 (9.70160628)	1.67825293 (1.99586362)	**0.00356498** **(0.00570966)**
f_4	0	0.329530323 (0.078275977)	0.178846900 (0.043878755)	0.070165920 (0.045750939)	**0.007051845** **(0.005827955)**
f_5	0	2.8611949E+004 (5.247521E+003)	**1.7544653E+004** **(4.2660720E+003)**	3.657472E+004 (2.625481E+004)	4.0856180E+004 (2.3849606E+004)
f_6	0	2.298493E+004 (9.436487E+003)	8.3305124E+002 (2.5277687E+002)	3.317152E+002 (5.102790E+002)	**46.9165244** **(71.8523465)**
f_7	0	74.84513039 (9.655168629)	8.4869152696 (2.2218119967)	0.488439589 (0.699219346)	**0.001073498** **(0.002135259)**
f_8	0	4.511300670 (0.389481555)	1.633577060 (0.127299219)	0.247796629 (0.258785313)	**0.008262857** **(0.006926495)**
f_9	0	6.279622471 (1.195714946)	1.21374677 (0.07518551)	0.69656004 (0.38610024)	**0.00644959** **(0.00929787)**
f_{10}	0	8.471330380 (1.728947836)	63.12745378 (16.76077466)	2.003519588 (2.940715554)	**0.012925598** **(0.026728745)**
f_{12}	0	0.017769196 (0.016844745)	0.014148773 (**0.010615135**)	**0.013250581** (0.011490605)	0.017114288 (0.016672930)

Tableau 3.4 : Comparaison entre GHSACO, GHS, IHS et HS en termes de valeur moyenne des minimum globaux et l'écart type (*std*.) obtenus durant les 30 essais avec $D = 50$.

Problème	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{11}	f_{12}
GHSACO,HS	1	1	1	1	1	1	1	1	1	1	1
GHSACO,HIS	1	1	1	1	1	1	1	1	1	1	0
GHSACO,GHS	1	1	1	1	0	1	1	1	1	1	0

Tableau 3.5 : Résultat du test Wilcoxon entre GHSACO et GHS, IHS et HS respectivement, avec $D = 50$.

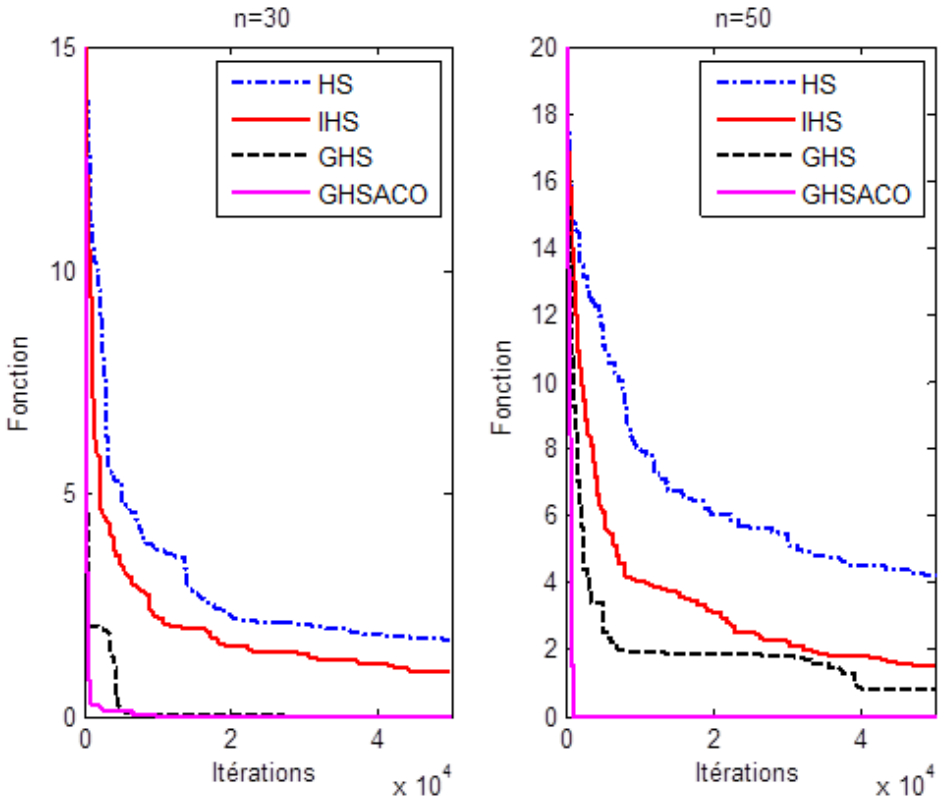

Figure 3.3 : Evolution des solutions obtenues pour la fonction *Ackley*.

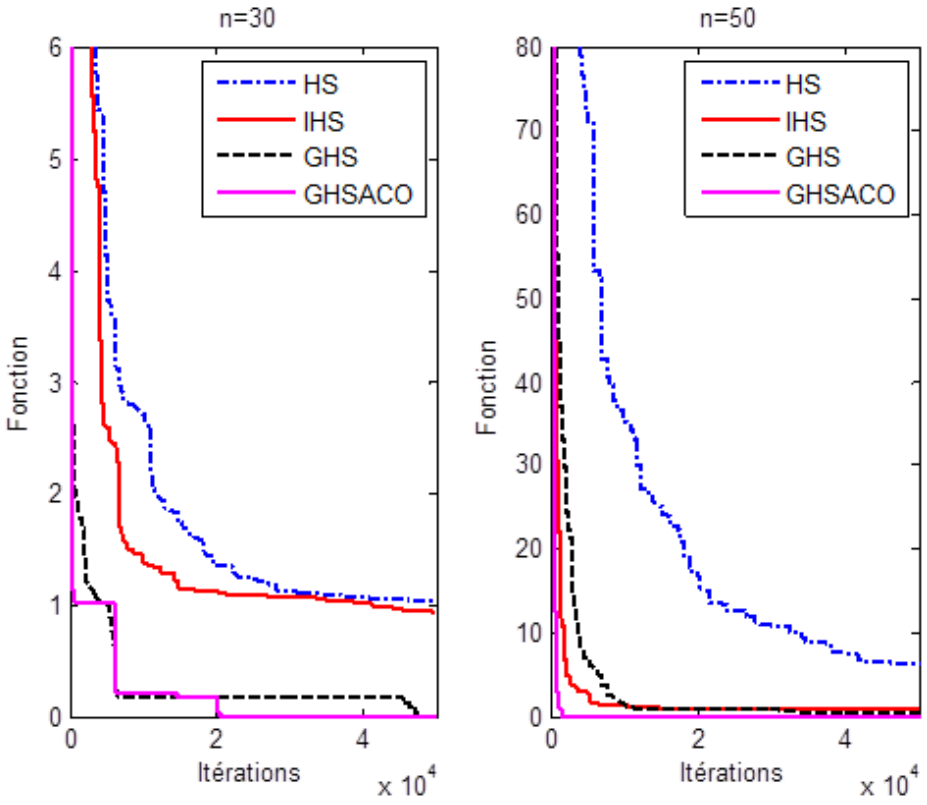

Figure 3.4 : Evolution des solutions obtenues pour la fonction *Griewank*.

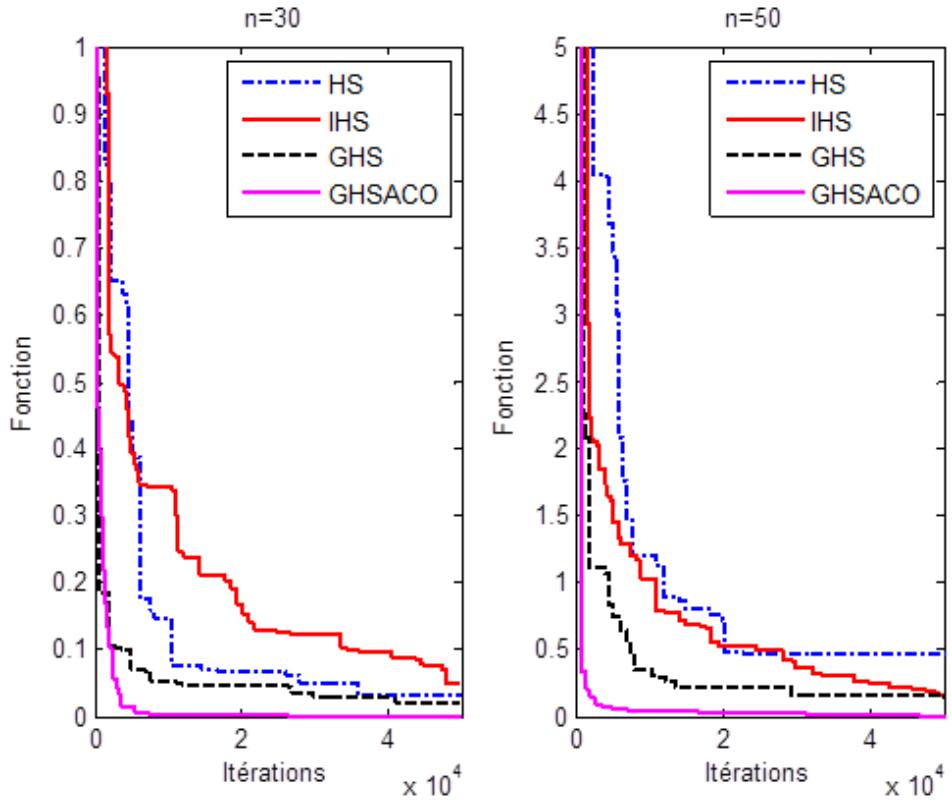

Figure 3.5 : Evolution des solutions obtenues pour la fonction *Quartic function with noise*.

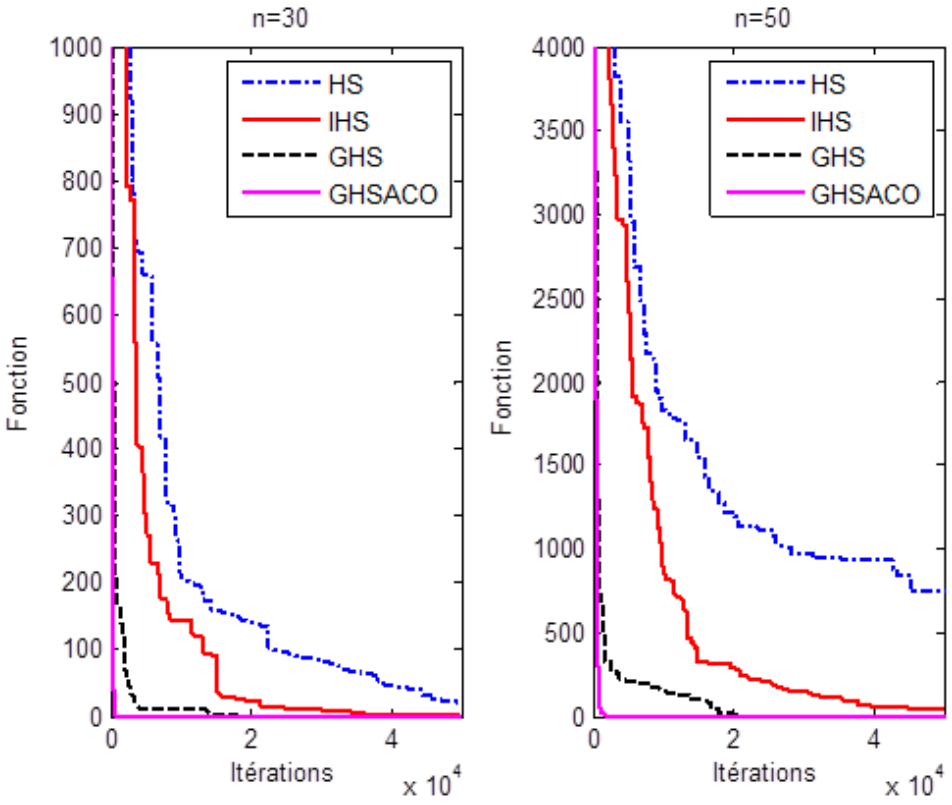

Figure 3.6 : Evolution des solutions obtenues pour la fonction *Generalized Schwefel Problem 2.26*.

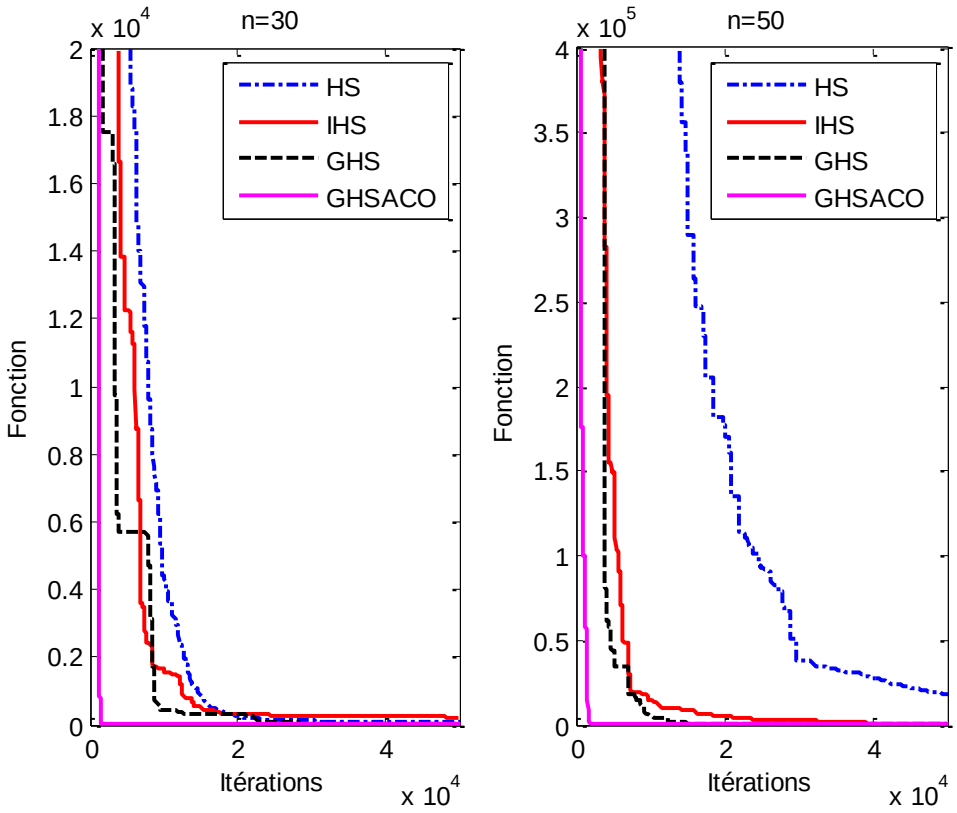

Figure 3.7 : Evolution des solutions obtenues pour la fonction *Rosenbrock*.

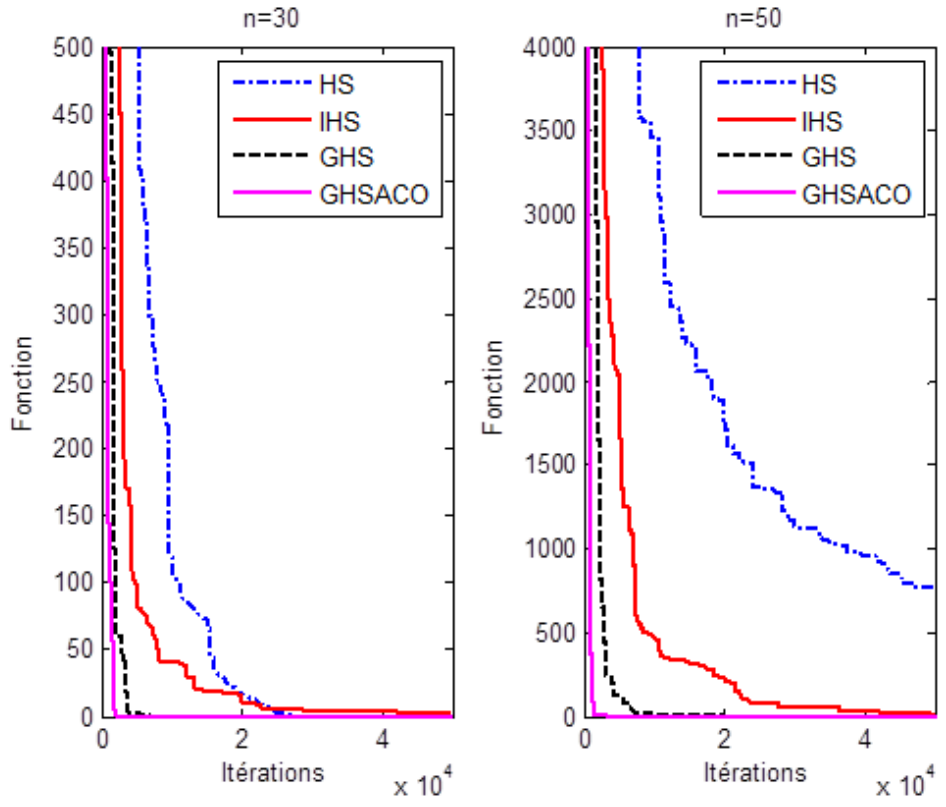

Figure 3.8 : Evolution des solutions obtenues pour la fonction *Step*.

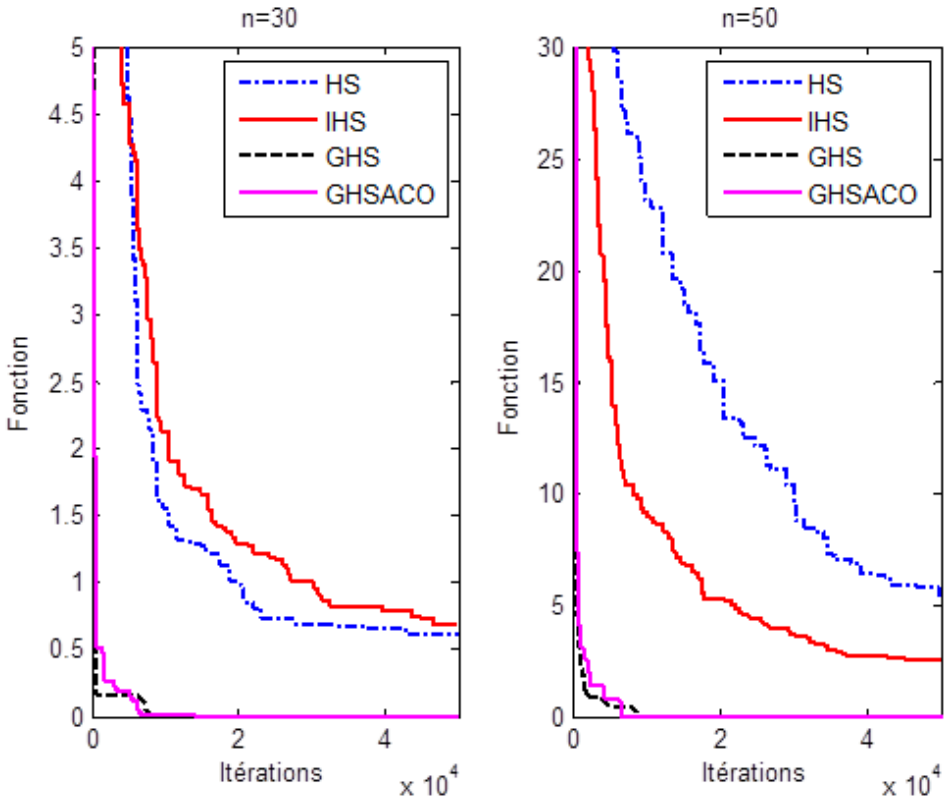

Figure 3.9 : Evolution des solutions obtenues pour la fonction *Schwefel problem* 2.22.

3.6 Conclusion

Dans ce chapitre, nous avons rappelé le principe de fonctionnement de l'algorithme HS, ainsi que ses variantes les plus célèbres IHS et GHS. Les caractéristiques principales de HS, ont été aussi présentées.

Une nouvelle variante de HS appelée GHSACO, basée sur l'hybridation entre le GHS et l'algorithme C-ACO, a été présentée. Ce nouvel algorithme a un mécanisme d'improvisation différent de celui de GHS dans les aspects suivants : (i) Il a une mémoire d'harmonie HM différente en structure et en conception, (ii) Caractérisé par une règle supplémentaire de MCSR, basée sur l'utilisation d'une règle aléatoire de transition avec l'intégration d'un mécanisme de mise-à-jour des pistes de phéromone, (iii) Il est équipé d'une procédure de commutation aléatoire entre le GHS et le C-ACO.

Les performances de l'algorithme proposé sont testées sur plusieurs fonctions de *benchmark* et confrontées aux algorithmes GHS, IHS et HS.

À partir de l'analyse et des résultats obtenus, nous constatons que GHSACO surpasse les autres variantes pour la plupart des problèmes testés. Nous pouvons conclure que les stratégies utilisées dans cette approche ont rendu HS plus robuste, et que les performances ont été

améliorées de manière significative. Le GHSACO sera utilisé dans le chapitre 5, dans un problème de réglage des paramètres d'une commande adaptative.

CHAPITRE 4

Application de l'algorithme ACO amélioré à l'optimisation des paramètres de la loi de commande par mode de glissement décentralisée

4.1 Introduction

Afin de montrer l'efficacité de l'algorithme ACO développé dans le deuxième chapitre, nous l'avons testé dans le calcul optimal des paramètres d'ajustement d'une loi de commande décentralisée par modes glissants appliquée aux systèmes non linéaires interconnectés. En effet, ce chapitre est consacré à l'élaboration d'une loi de commande décentralisée non linéaire optimisée par l'ACO ainsi développé.

La méthode de commande proposée vise permet de contourner un certain nombre de problèmes liés à la synthèse et à l'implémentation à savoir :

(i) Le la simplification de la synthèse ainsi que l'allègement du calcul au niveau de l'unité de commande par la décentralisation de la commande.

(ii) L'élimination complète du phénomène de broutement (*chattering phenomenon*) apparu au niveau du signal de commande généré par unité de commande locale. La solution proposée, consiste à remplacée, la partie discontinue (*commande corrective*) dans l'expression de la commande locale, par une approximation continue au voisinage de la surface de glissement à travers un système flou (*Fuzzy System* FS) [Zade 65] a une seule entrée et une seule sortie (*Single-Input Single-Output* 'SISO'). L'emploi de cette approximation, permet de lisser le signal de commande.

(iii) La résolution du problème de la détermination de commande équivalente, imposé, par le calcul mathématique très lourd et la nécessité d'une connaissance du modèle dynamique du système à régler. Afin d'attendre cet objectif, un calcul approximatif est effectué en utilisant un RFNN [Lee 00]. En effet, la méthode utilisée conduit à une minimisation importante dans le temps et la complexité de calcul.

Les deux systèmes FS et RFNN sont combinés, afin de construire des unités locales de commande par modes glissants. L'algorithme ACO amélioré proposé dans le second chapitre, est utilisé pour calculer d'une manière optimale quelques paramètres dans chaque loi de commande

locale. Cette approche montre son efficacité comme nous le verrons par la suite à travers les différents exemples présentés.

4.2 Commande par modes glissants décentralisée à base d'une approche coopérative et un algorithme ACO amélioré

4.2.1 Commande par modes glissants

4.2.1.1 Définitions et concepts de base

Le principe de la commande par modes glissants [Slot 91] est de pousser l'état du système à atteindre en temps fini une *hyper-surface* (dans l'espace d'état) donnée, pour ensuite y rester. Cette hyper-surface étant une relation entre les variables d'état du système, elle définit une équation différentielle, et donc détermine *totalement* la dynamique du système, pourvu qu'il reste sur cette hyper-surface. L'évolution d'un système soumis à une loi de commande qui le fait rester sur une hyper-surface donnée ne dépend donc plus du tout du système lui même ou des perturbations auxquelles il peut être soumis, mais uniquement des propriétés de cette hyper-surface. Le système bouclé n'est donc pas seulement *robuste* vis à vis des incertitudes et perturbations extérieures, mais totalement *insensible* à ces dernières.

Généralement, la synthèse d'une loi de commande par modes glissants se fait en deux étapes [Breg 10] :

1. Synthèse d'une hyper-surface en fonction des objectifs de commande et des propriétés statiques et dynamiques désirées pour le système bouclé. La dynamique exigée par l'hyper-surface doit être compatible avec l'amplitude de la commande ''utile'' disponible et la dynamique du système en boucle ouverte. Dans le cas contraire, le système ne pourra pas rester sur l'hyper-surface, et la propriété d'insensibilité aux perturbations sera perdue.
2. Synthèse d'une loi de commande discontinue de manière à contraindre les trajectoires d'état du système à atteindre cette hyper-surface en temps fini puis à y rester en dépit des incertitudes et des perturbations.

Considérons le système non linéaire incertain exprimé comme suit [Breg 10] :

$$\begin{aligned} &\dot{x}_1 = x_2 \\ &\dot{x}_2 = x_3 \\ &\ldots\ldots\ldots \\ &\dot{x}_n \in \psi(x,t) + [-C, C] + [\Gamma_{\mathrm{m}}, \Gamma_{\mathrm{M}}]u \\ &y = x_1 \end{aligned} \tag{4.1}$$

Avec ; $x = [x_1 \ldots\ldots x_2]^T \in \chi \subset \mathbb{R}^n$ représente l'état du système, $u \in \mathcal{U}$ est l'entrée de commande qui est une fonction éventuellement discontinue dépendante de l'état et du temps. $\psi(x,t)$, la

dynamique nominale du système, est une fonction définie sur χ. $[-C, C]$ est un terme additif inconnu, mais borné par $C > 0$ modélisant notamment :

- la différence entre les valeurs théoriques et réelles de paramètres mal connus du modèle,
- les phénomènes non modélisés ou négligés, à partir du moment où ils ne font pas varier le degré relatif du système,
- les perturbations extérieures.

$[\Gamma_m, \Gamma_M]$, avec $0 < \Gamma_m < \Gamma_M < \infty$, est un terme modélisant l'incertitude sur le gain du système vis à vis de la commande.

4.2.1.2 Synthèse de la surface de glissement

La synthèse d'une surface de glissement repose sur la définition d'une fonction $\sigma(x,t)$ appelée *variable de glissement* ou de *commutation*. Ainsi, une *surface de glissement* est définie par l'ensemble S comme suit :

$$S = \{x \in \chi \,|\sigma(x,t) = 0\} \tag{4.2}$$

La figure 4.1, montre l'exemple d'une surface de glissement.

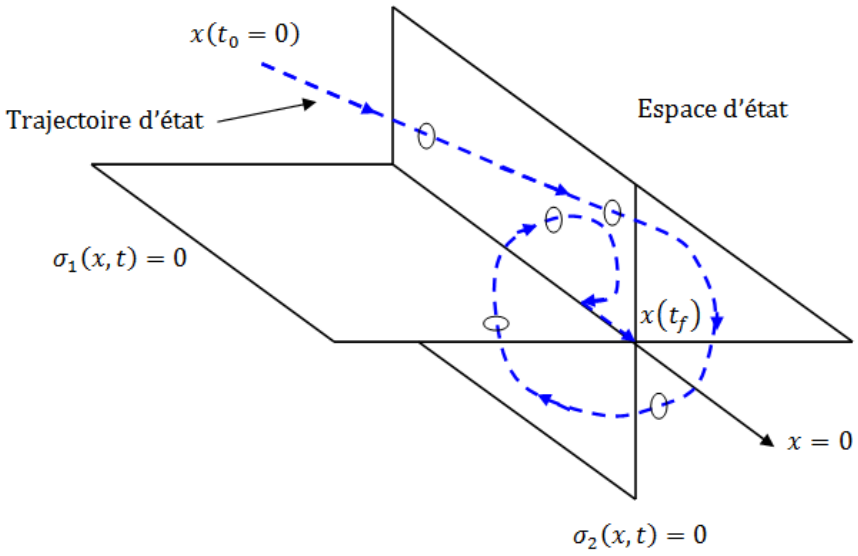

Figure 4.1 : Surface de glissement [Mend 02].

D'après Utkin [Utki 92], on dit qu'il existe un régime glissant idéal sur S s'il existe un temps fini t_f tel que la solution de (4.1) satisfait la condition $\sigma(x,t) = 0$ pour tout $t \geq t_f$.

En outre, d'après toujours [Utki 10], la condition nécessaire pour l'établissement d'un régime glissant est que la variable de glissement ait un degré relatif égal à 1 par rapport à la commande

u (Le degré relatif d'un système est le nombre minimum de fois qu'il faut dériver la sortie par rapport au temps pour faire apparaître l'entrée de manière explicite [Isid 95]).

Considérons la variable de glissement linéaire suivante :

$$\sigma(x,t) = e^{(n-1)} + \cdots + c_2\ddot{e} + c_1\dot{e} + c_0 e \tag{4.3}$$

d'où $e = y(t) - y_r(t)$ est les coefficients c_i $(0 \leq i \leq n-2)$ sont choisis tel que le polynôme $\lambda^{n-1} + \sum_{i=0}^{n-2} c_i\lambda^i$soit un polynôme d'Hurwitz.

Ainsi, lorsque la variable de glissement $\sigma(x,t)$ est forcée à zéro, l'erreur de poursuite e converge asymptotiquement vers zéro, avec une dynamique imposée par le choix des coefficients c_i. De plus, $\sigma(x,t)$ satisfait la condition sur le degré relatif puisque la commande u apparaît dans l'expression de sa première dérivée par rapport au temps, comme le montre l'équation (4.4), suivante [Breg 10] :

$$\dot{\sigma}(x,t) \in \psi(X,t) + [-C,C] + [\Gamma_{\mathrm{m}},\Gamma_{\mathrm{M}}]u - y_r^{(n)}(t) + \sum_{i=0}^{n-2} c_i e^{(i+1)} \tag{4.4}$$

4.2.1.3 Synthèse de la loi de commande

L'objectif de la loi de commande est de contraindre les trajectoires d'état du système (4.1) à atteindre et ensuite à rester sur la surface de glissement malgré la présence d'incertitudes sur le système. En d'autres termes, la loi de commande doit rendre la surface de glissement localement *attractive* (c.-à-d. au voisinage de la surface de glissement, toutes les trajectoires du système doivent être dirigées vers elle). Ainsi, la loi de commande doit être calculée en vérifiant une condition assurant la stabilité de $\sigma(x,t) = 0$. Une telle condition est appelée *condition d'attractivité*.

La méthode directe de *Lyapunov* permet de se prononcer quant à la stabilité d'un état d'équilibre sans avoir recours à la résolution de l'équation d'état du système. En supposant que l'état d'équilibre est 0, le signe d'une fonction $V(x)$, $(V(0) = 0;\ V(\infty) = \infty)$, appelée fonction de Lyapunov, et celui de sa dérivée temporelle $\dot{V}(x) = \frac{dV(x)}{dt}$ donnent une information sur la stabilité du système. $V(x)$ joue le rôle d'une fonction "énergie" fictive pour le système considéré. Si $V(x) > 0\ \forall x \neq 0$ et $\dot{V}(x) < 0$, le système est asymptotiquement stable.

Une classe de fonctions de Lyapunov classique pour la détermination de la condition d'attractivité est celle des fonctions quadratiques du type :

$$V(\sigma) = \frac{1}{2}\sigma^2 \tag{4.5}$$

Cette fonction est définie positive de manière évidente. Une condition nécessaire et suffisante pour que la variable de glissement $\sigma(x,t)$ tende vers zéro est que la dérivée de V soit définie négative, c.-à-d. $\dot{V} = \dot{\sigma}\sigma < 0$.

Cette relation est appelée *condition d'attractivité*. Néanmoins, elle n'est pas suffisante pour assurer une convergence en temps fini vers la surface. Ainsi, afin d'assurer une convergence de $\sigma(x,t)$ vers 0 en *temps fini*, une condition plus forte doit être respectée.

Dans le cas des modes glissants classiques, on utilise généralement la condition d'attractivité non-linéaire dite condition de η-*attractivité* [Utki 92], exprimée comme suit :

$$\dot{\sigma}\sigma \leq -\eta|\sigma| \quad \text{avec} \quad \eta > 0 \tag{4.6}$$

Cela revient, pour $\sigma \neq 0$, à considérer que $\dot{\sigma} \leq -\eta sgn(\sigma)$, $\eta > 0$. Ainsi, ce critère permet de garantir une convergence en temps fini, puisque sous les deux conditions suivantes $\sigma(0) > 0$, $\sigma(t) \leq \sigma(0) - \eta t$ et $\sigma(0) < 0$, $\sigma(t) \geq \sigma(0) + \eta t$, $\sigma(t)$ atteint 0 en un temps inférieur à $\frac{|\sigma(0)|}{\eta}$.

Le critère ci-dessus, est toujours satisfait si la commande est du type :

$$u = -u_M sgn(\sigma) \tag{4.7}$$

La grandeur (4.7) appelée commande corrective (*corrective control*). Ainsi, u_M est une grandeur, qui doit être choisie pour :

- compenser l'écart de dynamique entre le système réel et le système de référence (donné par la surface de glissement),
- compenser les perturbations coïncidentes.

D'après (4.4), u_M doit être réglée tel que [Breg 10] :

$$u_M \geq \max_{X,t}\left(\Gamma_{\mathrm{m}}^{-1}\left(\left|\psi(x,t) + \sum_{i=0}^{n-2} c_i e^{(i+1)} - y_r^{(n)}(t)\right| + C + \eta\right)\right) \tag{4.8}$$

Avec, le terme Γ_{m}^{-1} représente l'incertitude sur le gain, $\left|\psi(x,t) + \sum_{i=0}^{n-2} c_i e^{(i+1)} - y_r^{(n)}(t)\right|$ représente l'écart sur la dynamique et $(C + \eta)$ perturbation coïncidente. Lorsque u_M satisfait la condition (4.8), l'existence des modes glissants est entièrement garantie.

En pratique, on ne considère généralement que le cas où u_M est constant et assez grand pour l'ensemble du domaine physique dans lequel évolue le système.

Pour résumer, le comportement du système peut être décrit par deux phases :

(i) *Phase de convergence*, Cette phase correspond à l'intervalle de temps $t \in [0, t_f]$ pendant lequel les trajectoires d'état du système ne sont pas dans la surface de glissement S. Durant cette phase, le système reste sensible aux variations de paramètres. Sa durée peut être réduite en augmentant η, c.-à-d. en augmentant l'amplitude de la commande.

(ii) *Phase de glissement*, Cette phase correspond à l'intervalle de temps $t \in [t_f, \infty]$ durant laquelle les trajectoires d'état sont confinées dans la surface de glissement S. Durant cette phase, le comportement du système ne dépend plus du système d'origine ni des perturbations, mais est entièrement déterminé par le choix de la surface de glissement.

4.2.1.4 Commande équivalente

Pendant le régime glissant, la commande a une fréquence de commutation en théorie infinie ; autrement dit, la commande est discontinue à chaque instant. Ce cas de figure n'est absolument pas abordé par la théorie classique des équations différentielles, qui n'étudie que les équations du type $\dot{x} = f(x,t)$ où f est Lipschitzienne par rapport à x, c.-à-d., $\exists L > 0$, tel que $\forall\, x_a, x_b$;

$$\|f(x_a,t) - f(x_b,t)\| \leq L\|x_a - x_b\| \tag{4.9}$$

Ce n'est pas le cas si f est discontinue par rapport à x. Un exemple typique est un système soumis à une commande en boucle fermée discontinue du type de (4.7).
L'étude de systèmes soumis à de telles commandes est beaucoup plus complexe. La méthode théorique générale pour résoudre de telles équations différentielles discontinues est de les remplacer par des *inclusions différentielles* [Fili 88]. Aux points de discontinuité, on remplace la valeur (inconnue ou mal connue) du membre de droite par un ensemble de valeurs physiquement ou techniquement possibles. Par exemple, on peut prendre pour représentation de la fonction $sign$:

$$sign(x) = \begin{cases} -1 & \text{si} \quad x < 0 \\ [-1;1] & \text{si} \quad x = 0 \\ 1 & \text{si} \quad x > 0 \end{cases} \tag{4.10}$$

Cette méthode se justifie par exemple par le fait qu'un signal physique est rarement discontinu, mais présente souvent une dérivée extrêmement élevée aux points considérés comme discontinus, ce qui fait qu'il passera (très rapidement) par toutes les valeurs intermédiaires entre la valeur à l'instant précédent et celle à l'instant suivant la discontinuité.

Cette technique permet aussi de traiter les incertitudes et perturbations inconnues, mais bornées, par le même outil mathématique. Cependant, il faut souvent faire appel à d'autres arguments pour trouver la solution de l'inclusion différentielle. De plus, si le système n'est pas affin en la commande, la transformation de l'équation différentielle en inclusion différentielle n'est pas unique, donc la solution obtenue dépend du détail de la méthode utilisée [Fili 88].

Une autre approche, plus concrète, donne une solution particulière. C'est la méthode de *régularisation* des équations [Utki 92, Utki 99]. Il s'agit ici d'ajouter une petite imperfection pour faire en sorte que la commande soit continue, ou en tous cas, qu'elle ne commute qu'en des points isolés. Cet ensemble des instants de commutation n'a aucune influence sur l'intégrale, donc la théorie classique s'applique. La solution de l'équation d'origine est alors la limite quand l'imperfection artificiellement ajoutée tend vers 0. L'imperfection peut être une dynamique rapide d'actionneur, un retard pur sur la commande,… La solution obtenue dépend dans le cas général du type d'imperfection considéré et de la manière de la faire tendre vers 0. Par contre, pour un système affin en la commande, toutes ces méthodes donnent le même résultat.

La méthode de régularisation la plus classique pour les modes glissants est la *commande équivalente* [Utki 99]. Quand le système est sur la surface de glissement S, $\sigma(x,t) = 0\ \forall t$, donc $\dot{\sigma}(x,t) = 0$. D'après (4.4), cette dernière condition donne :

$$u = \frac{-\psi(X,t) - \sum_{i=0}^{n-2} c_i e^{(i+1)} + y_r^{(n)}(t) - [-C,C]}{[\Gamma_m, \Gamma_M]} \tag{4.11}$$

Cette valeur particulière de u est appelée *commande équivalente*, car le système a exactement la même évolution si on lui applique cette commande fictive à la place de la commande réelle discontinue (4.7).

Cependant, la méthode de régularisation pose plusieurs problèmes, lors de son application. Elle, nécessite une connaissance très parfonde du modèle dynamique et des paramètres du système à régler, elle repose en outre sur un calcul mathématique généralement lord et complexe. Toutes ces contraintes, nous amenons à utiliser dans le calcul de la commande équivalente, une méthode approximative basée sur l'emploi des réseaux de neurones flous récurrents RFNNs. Nous présentons en détaille, cette méthode dans la section 4.2.2.4.

4.2.1.5 Le phénomène du *chattering*

Un régime glissant idéal requiert une commande pouvant commuter à une fréquence infinie. Certainement, pour une utilisation pratique, seule une commutation à une fréquence finie est possible, ce qui cause un retard entre la mesure de la sortie et le calcul de la commande, qui peut être amplifié si le système présente naturellement des retards ou des dynamiques négligées.

Cela conduit le système à quitter la surface de glissement sans que la commande ne puisse réagir, puis, une fois le signe de la commande inversé, à revenir sur cette surface et passer de l'autre côté, et ainsi de suite.

Ainsi, durant le régime glissant, les discontinuités appliquées à la commande peuvent entraîner des oscillations haute fréquence de la trajectoire du système autour de la surface de glissement (figure 4.2), un phénomène appelé *broutement* ou *chattering* en anglais. Les principales raisons à l'origine de ce phénomène sont [Youn 99] :

- les retards purs en série avec le système en boucle ouverte (retards inhérents au système, échantillonnage,….),
- les dynamiques non modélisées des capteurs et observateurs, qui retardent le moment où le régulateur prend conscience qu'il faut inverser la commande,
- les dynamiques non modélisées des actionneurs et autres dynamiques rapides du système, qui retardent le moment où la commande est suffisamment forte pour rapprocher le système de la surface de glissement.

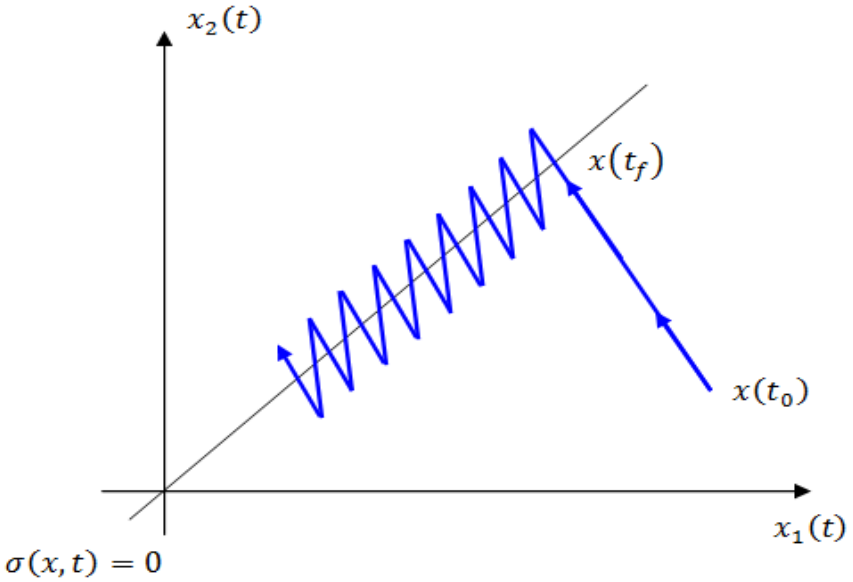

Figure 4.2 : Phénomène de *chattering* [Mend 02].

Tous ces phénomènes ont globalement l'effet de retarder l'application effective de la commande permettant de ramener le système sur la surface de glissement à partir du moment où il l'a quittée.

On parle aussi de chattering pour désigner l'oscillation haute fréquence de la commande (et non plus de la variable de glissement). Une autre cause de chattering, notamment sur la commande, est le bruit de mesure. En effet, une erreur de mesure quand l'état est très proche de la surface de glissement peut entraîner une erreur de signe de la commande, car cette dernière

croit à tort que le système se trouve de l'autre côté de la surface. Ce phénomène est augmenté par la nécessité d'avoir des observateurs ou dérivateurs rapides, donc filtrant peu la mesure.

Les phénomènes de chattering peuvent être si pénalisants que l'utilisation d'une loi de commande par modes glissants peut, dans certaines applications, être à proscrire, vu que son utilisation peut dégrader les performances, voire conduire à l'instabilité à cause du chattering sur la sortie. Le chattering de la commande, quant à lui, peut entraîner une usure prématurée des actionneurs ou de certaines parties du système à cause de trop fortes sollicitations. En excitant les modes propres des dynamiques non modélisées ou des fréquences de résonance du système correspondant aux retards de commutation, cette commande peut provoquer sur les systèmes mécaniques un bruit haute fréquence et des oscillations préjudiciables à leur structure. Sur des systèmes autres que mécaniques, les oscillations engendrées peuvent poser d'autres problèmes (réduction de précision, créations d'ondes électromagnétiques néfastes, ou autres ondes amplifiées par le système,…).

De nombreuses solutions ont été proposées, dont le but est de réduire ou d'éliminer complètement ce phénomène. Il existe des méthodes comme celle de la couche limite (*boundary layer*) figure 4.3, qui consiste à remplacer la fonction *sign* de la loi de commande par une approximation continue à gain élevé dans un proche voisinage de S [Burt 86, Utki 92], et saturée en dehors de ce voisinage. Le régime glissant qui en résulte n'est plus confiné dans S, mais dans un proche voisinage de celui-ci. Dans ce cas, le système est dit en régime *pseudo-glissant*.

Ces méthodes réduisent la robustesse de la commande. Elles sont paramétrées par une constante positive ϕ (voir figure 4.3) réglée pour avoir un bon compromis entre réduction du chattering et conservation de la robustesse.

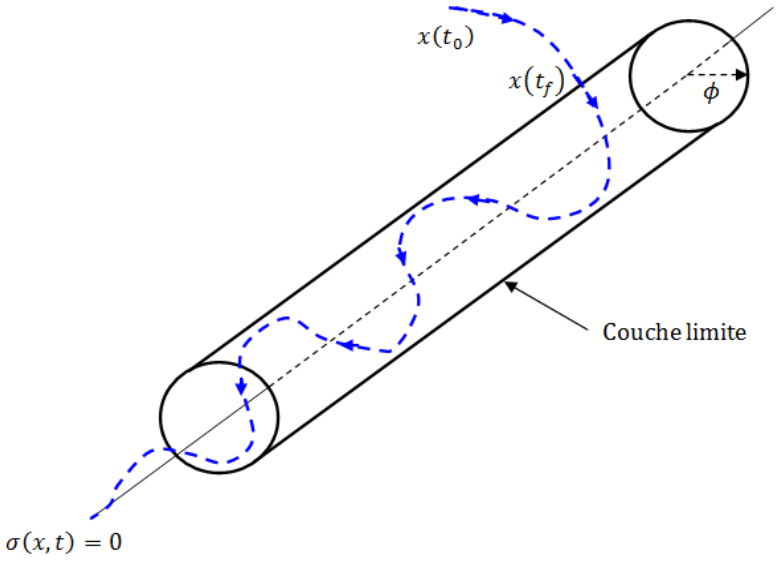

Figure 4.3: Une couche limite enveloppe une surface de glissement [Mend 02].

Dans ce travail de thèse et afin d'éliminer ce phénomène nous proposons l'utilisation d'un FS de type SISO afin d'approximer la commande corrective (4.7). Nous détaillons cette méthode, dans la section 4.2.2.4.

4.2.2 Méthode de commande proposée

4.2.2.1 Classe de systèmes interconnectés

Considérons le système non-linéaire interconnecté S composé de N sous-systèmes non-linéaires s_i définie par :

$$\begin{aligned}
&\dot{x}_{i,1} = x_{i,2}\\
&\dot{x}_{i,2} = x_{i,3}\\
&\ldots\ldots\\
&\dot{x}_{i,n_i} = f_i(\bar{x}_i) + g_i(\bar{x}_i)u_i(t) + z_i(\bar{x}_s) + d_i(\bar{x}_i)
\end{aligned} \tag{4.12}$$

D'où $\bar{x}_i = \left[x_{i,1} \ldots x_{i,n_i}\right]^T \in \mathbb{R}^{n_i}$, $\bar{x}_s = \left[\bar{x}_1^T \ldots \bar{x}_{\mathrm{N}}^T\right]^T \in \mathbb{R}^{\mathrm{N}}$,$n = \sum_{i=1}^{\mathrm{N}} n_i$ sont respectivement le vecteur d'état du sous-système s_i et le vecteur d'état du système global S, $u_i(t) \in \mathbb{R}$ la commande du sous-système s_i. $f_i(\bar{x}_i)$ est une fonction continue inconnue, $g_i(\bar{x}_i) \in \mathbb{R}^{n_i}$ et $z_i(\bar{x}_s)$: $\mathbb{R}^n \to \mathbb{R}$ et $d_i(\bar{x}_i)$ sont respectivement ; la fonction de commande qui est inconnue est bornée, les incertitudes, les perturbations externes et les effets d'interconnections des autres sous-systèmes.

Les hypothèses présentées ci-dessous, sont exprimées selon la structure du système S définie par (4.12).

Hypothèse 4.1. Chaque vecteur d'état $\bar{x}_i, i \in \mathrm{N}$, est mesurable,

Hypothèse 4.2. Pour $t \in [0, +\infty]$, la sortie désirée x_{id} ainsi que ses dérivées $\left[x_{id}^{(1)}, \ldots, x_{id}^{(n)}\right]$ sont bornées, c.à.d. , $\left|x_{jd}^{(i)}\right| \leq h_j, i = 1, \ldots, n_i, j = 1, \ldots, \mathrm{N}$, avec h_j est une constante positive,

Hypothèse 4.3. $\forall \bar{x}_i \in R^{n_i}, \exists k_i > 0$, elle existe $|g_i(\bar{x}_i)| \geq k_i > 0, i = 1, .., \mathrm{N}$.

4.2.2.2 Synthèse de la commande

L'approche de commande que nous allons développer consiste à synthétiser pour chaque sous-système une loi de commande locale ne nécessitant ni la mesure des grandeurs des autres sous-systèmes, ni l'utilisation du modèle dynamique non linéaire du système global.

L'objectif de cette commande est donc de forcer les états existant donnés par le vecteur $\bar{x}_i = \left[x_{i,1}, \ldots, x_{i,n_i}\right]^T$ à suivre celles du vecteur des états désirés $\bar{x}_{id} = \left[x_{id}, \ldots, x_{id}^{(n_i-1)}\right]^T$.

Définissons l'erreur de poursuite locale $e_i = \bar{x}_i - \bar{x}_{id} = \left[e_{i,1}, .., e_{i,n_i}\right]^T, i = 1, \dots, \mathrm{N}$.

Le problème traité ici est le suivant ; Connaissant le vecteur des consignes $\bar{x}_{id}$ déterminer une variable de glissement locale stable σ_i et une commande u_i de manière que l'erreur de poursuite locale $e_i$tend asymptotiquement vers zéro quant t tend vers ∞.

On choisit comme variable de glissement locale, la variable définie par :

$$\sigma_i = \Lambda_i^{\mathrm{T}} e_i \tag{4.13}$$

où, $\Lambda_i^{\mathrm{T}} = \left[\ C_{i,2}\ \ C_{i,3}, \dots \dots, C_{i,(n_i-1)}, 1\right] \in \mathbb{R}^{n_i}$ avec $C_{(.)}$ sont des constantes.

En régime de glissement idéal, la variable de glissement doit être nulle $\sigma_i = 0$. Ainsi, la dynamique de l'erreur est donnée par l'équation suivante :

$$\sigma_i(e_i) = C_{i,1}e_{i,1} + C_{i,2}e_{i,2} + \cdots + C_{i,(n_i-1)}e_{i,(n_i-1)} + e_{i,n_i} \tag{4.14}$$

Avec : $e_{i,1} = x_{i,1} - x_{id}, e_{i,2} = \dot{x}_{i,1} - \dot{x}_{id}, \dots., e_{i,n_i} = x_{i,1}^{(n_i-1)} - x_{id}^{(n_i-1)}, i = 1, \dots, \mathrm{N},$

Nous avons aussi :

$$\dot{\sigma}_i = \sum_{j=1}^{n_i-1} C_{i,j}\, e_{i,(j+1)} + \dot{e}_{i,n_i} = \sum_{j=1}^{n_i-1} C_{i,j}\, e_{i,(j+1)} + f_i(\bar{x}_i) + g_i(\bar{x}_i)u_i(t) + z_i(\bar{x}_s) + d_i(\bar{x}_i) - x_{id}^{(n_i)} =$$

$$u_i^* + \Delta u_i^* + g_i(\bar{x}_i)u_i(t) \tag{4.15}$$

D'où $u_i^* = \sum_{j=1}^{n_i-1} C_{i,j}\, e_{i,(j+1)} - x_{id}^{(n_i)}$, et $\Delta u_i^* = f_i(\bar{x}_i) + z_i(\bar{x}_s) + d_i(\bar{x}_i)$

Les hypothèses suivantes sont indispensables, lors de la synthèse d'une commande par modes glissants ;

Hypothèse 4.4. $|f_i(\bar{x}_i, t)| \leq F_i(\bar{x}_i), i = 1, \dots, N$ avec F_i est une fonction positive,

Hypothèse 4.5. $|d_i(\bar{x}_i, t)| \leq D_i(\bar{x}_i), i = 1, \dots, N$ avec D_i est une fonction positive,

Hypothèse 4.6. Les interconnexions sont limitées par un polynôme d'états d'ordre p, c.-à-d. il existe des nombres non négatif ξ_{ij}^k tel que $|z_i(\bar{x}_s)| \leq \sum_{k=0}^{P} \sum_{j=1}^{N} \xi_{ij}^k \left\|\bar{x}_{s_j}\right\|^k$.

L'expression de la commande locale u_i est exprimée comme suit :

$$u_i = \frac{1}{g_i}(-u_i^* - \Delta u_i') \tag{4.16}$$

Avec : $\Delta u_i' = \left(F_i + \sum_{k=0}^{P} \sum_{j=1}^{N} \xi_{ij}^k \left\|\bar{x}_{s_j}\right\|^k + D_i + \psi_i\right) sgn(\sigma_i)$ et ψ_i est une constante positive.

Remarque 4.1. Les hypothèses 4 à 6 indiquent que la synthèse d'une commande par modes glissants classique implique la connaissance des limites des différentes fonctions, ce qui n'est pas nécessaire dans la méthode de commande proposée,

Remarque 4.2. La partie discontinue dans (4.16), confirme l'apparition du phénomène de chattering,

4.2.2.3 Schéma global de la commande décentralisée proposée

La structure globale de la commande proposée, comprend N sous-contrôleurs RFNN-FSMC$_i$ (l'abréviation RFNN-FSMC$_i$ est utilisée, pour indiquer qu'il s'agit d'un sous-contrôleur ou un contrôleur local par modes glissants basé sur un RFNN et un FS). En effet, chaque RFNN-FSMC a deux parties principale ; la première a pour but de calculer le terme correctif exprimé par (4.7). Cette fonction est assurée par un FS. Tandis que, la seconde est sert pour le calcul de la commande équivalente à travers un RFNN.

La commande locale u_i générée par chaque sous-contrôleur est le résultat direct de la somme de ces deux termes. De plus, le terme correctif généré par FS est utilisé pour la mise-à-jour des paramètres (poids) de RFNN associé à ce FS. Le premier avantage de l'utilisation de ce schéma de commande, comme le montrent les résultats de simulation est la suppression complète du phénomène de chattering.

Ainsi, l'algorithme ACO proposé est utilisé pour le réglage des paramètres suivants : Les taux d'apprentissage utilisés par l'algorithme de rétro-propagation du gradient (*Back-Propagation Algorithm* BPA) [Werb 90], dans le processus de mise-à-jours des paramètres de RFNN, les coefficients $C_{(.)}$ des variables de glissements $\sigma_i()$ ainsi que les paramètres des fonctions d'appartenances du FS (Fonction d'entrée et de sortie).

Les deux figures ci-dessous (figure 4.4 et figure 4.5), montrent respectivement le schéma global de la commande décentralisée, ainsi que la structure interne du sous-contrôleur.

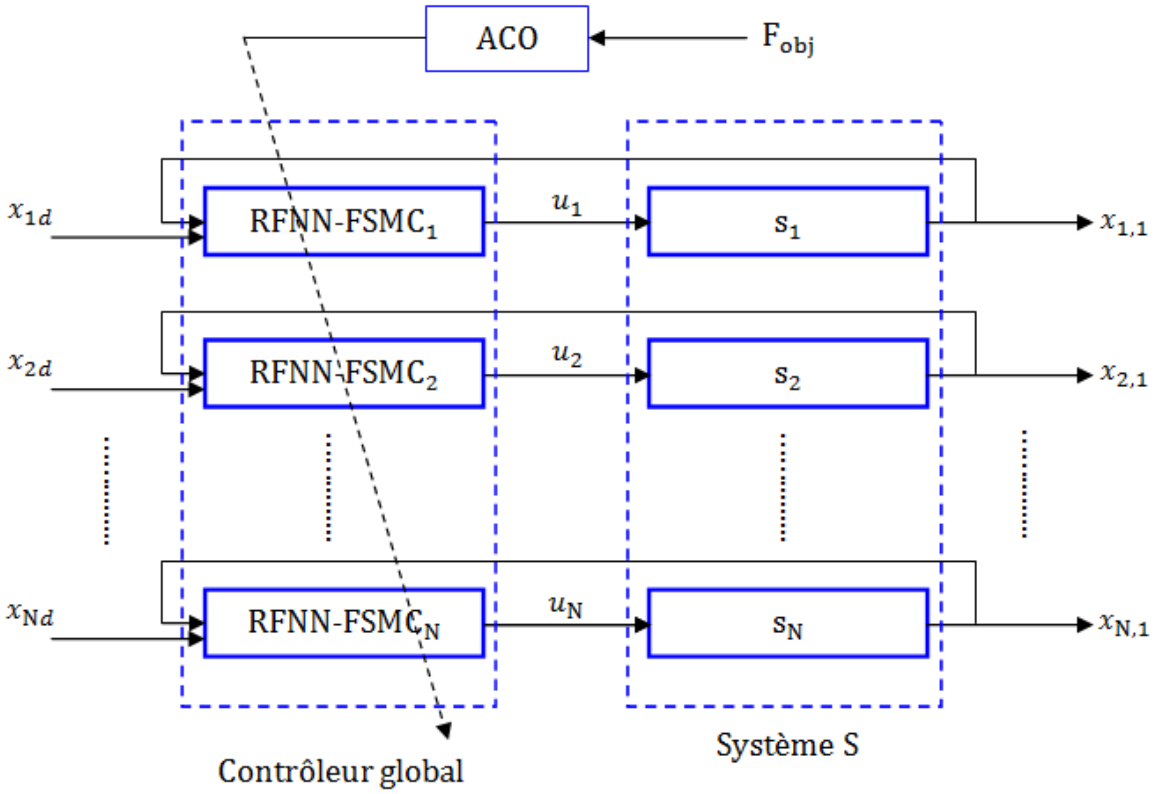

Figure 4.4 : Schéma global de la commande décentralisée.

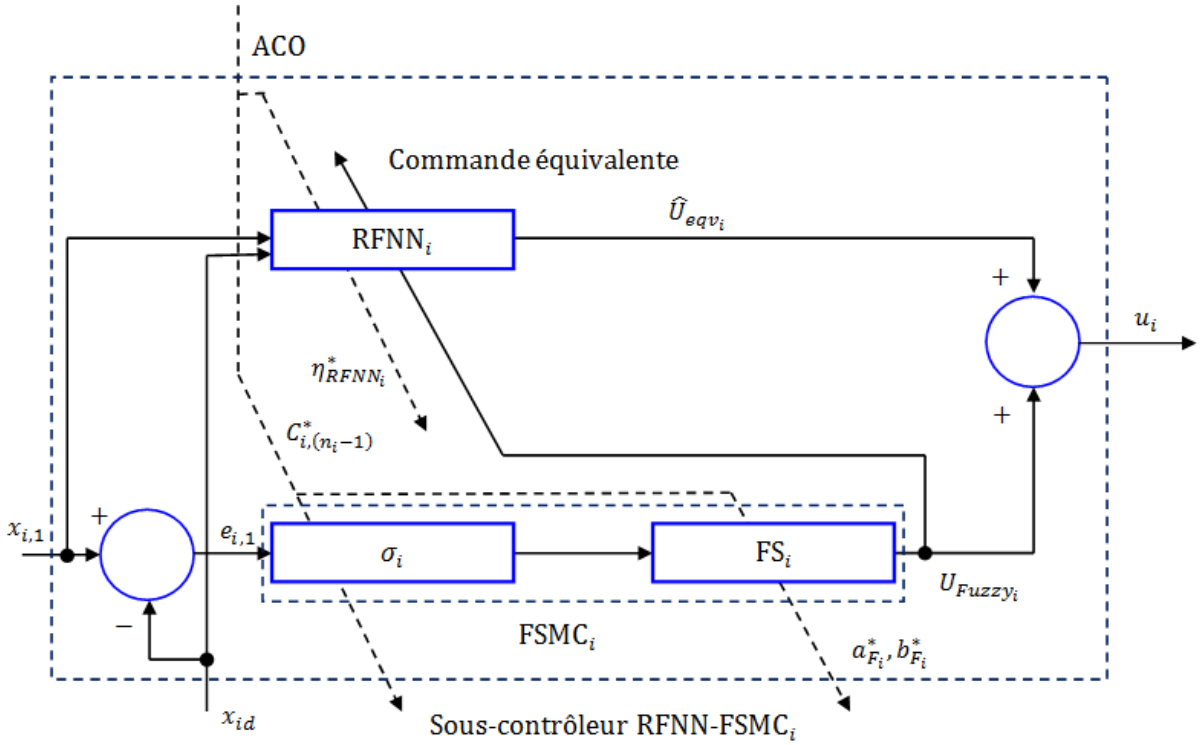

Figure 4.5 : Configuration interne de chaque sous-contrôleur.

4.2.2.4 Calcul de la commande équivalente

Comme nous l'avons dit, chaque sous-contrôleur RFNN-FSMC$_i$ génère une commande locale u_i, qui est la somme d'une commande équivalente $\hat{u}_{eqv\,i}$ et un terme correctif.

Le calcul de $\hat{u}_{eqv\,i}$ au cœur de chaque RFNN-FSMC$_i$ est assuré par un RFNN qui a la configuration illustrée par la figure 4.6. Cette dernière, regroupe quatre couches différentes, qui sont respectivement :

Couche 1 (Couche d'entrée) : Les nœuds (neurones) dans cette couche sont équipés d'une rétroaction pondérée, permet d'ajouter l'historique instantané (à chaque instant k) relatif à chaque sortie dans cette couche. Pour chaque nœud i, la relation entrée/sortie est donnée comme suit :

$$net_i^1(k) = x_i^1(k) + w_i^1 y_i^1(k-1), y_i^1(k) = net_i^1(k) \tag{4.17}$$

Avec ; x_i^1 $(i = 1{,}2)$ représente l'entrée i au nœud i, w_i^1 le poids de rétroaction pour le nœud i ainsi $y_i^1(k)$ est la sortie du nœud i à l'instant k. Le nombre de nœuds dans cette couche est égal à n.

Couche 2 (Couche de Fuzzification) : Cette couche a m nœuds, chacun assure la *fuzzifiaction* de son entrée à travers une fonction d'appartenance de type Gaussien comme suit :

$$net_{ij}^2(k) = \frac{\left(x_i^2 - a_{ij}\right)^2}{2\left(b_{ij}\right)^2}, y_{ij}^2 = exp\left(net_{ij}^2(k)\right) \tag{4.18}$$

Avec a_{ij} et b_{ij} $(i = 1{,}2$ and $j = 1{,}2 \ldots, m = 5$ dans nôtre cas) sont les centres et les poids de toutes les fonctions d'appartenances dans cette couche.

Les deux indices ij indiquent respectivement la sortie du nœud i dans la première couche associe au nœud j dans la deuxième couche.

Couche 3 (Couche de règles floues) : Cette couche forme la base de règles floues. En effet, chaque nœud dans cette couche correspond à une règle floue. Une règle floue q générée au niveau de cette couche, peut être décrite comme suit :

$$\text{Règle floue } q : if\ x_1\ is\ A_1^q, \ldots\ldots, x_n\ is\ A_n^q\ , then\ y_1\ is\ B_1^q, \ldots, y_p\ is\ B_p^q \tag{4.19}$$

Avec A_i^q est l'ensemble flou de l'entrée i dans la règle q, ainsi B_j^q est l'ensemble flou de la sortie j dans la règle q.

La relation entrée/sorite pour le nœud q est décrite comme suit :

$$net_q^3(k) = \prod y_{iq_i}^2, y_q^3 = net_q^3(k), q = 1, \dots., l = 5. \quad (4.20)$$

Où le terme $y_{iq_i}^2$ représente la sortie d'un nœud associe à une entrée i et relatif à une règle q avec $q_i \in \{1,2, \dots., m\}$.

Couche 4 (Couche de Défuzzification) : dite aussi couche de sortie. La relation entrée/sorite pour le nœud o dans cette couche est exprimée comme suit :

$$net_o^4(k) = \sum w_{oq}^4 \times y_q^3, y_o^4(k) = net_o^4(k) \quad (4.21)$$

Avec w_{oq}^4 $(o = 1, .., p = 1)$ est les poids de pondération associes au nœud o connecté à q règles.

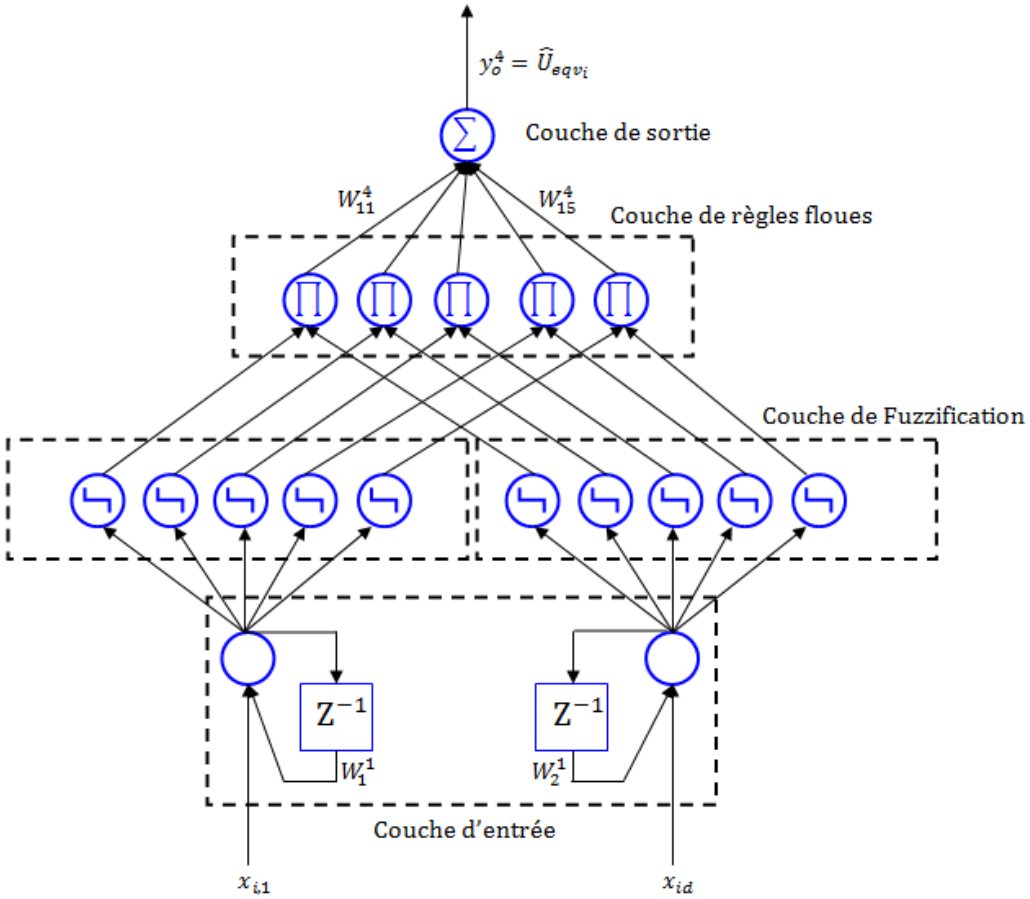

Figure 4.6 : Structure du RFNN, utilisé pour le calcul de la commande équivalente.

Un RFNN nécessite une opération d'apprentissage pour qu'il puisse générer la sortie désirée quelque soit sa nature. L'algorithme BPA [Werb 90] est l'outil algorithmique classique le plus utilisée pour cette opération. Il permet une adaptation itérative des paramètres du réseau par la minimisation de l'erreur (*Mean Square Error* MSE) entre la sortie désirée (estimée) et la sortie du réseau (réelle). Dans nôtre cas, cette erreur exprimée par (4.22) est obtenue par la différence

entre la valeur de la commande équivalente désirée u_{eqv_d} et celle de la commande équivalente estimée $\hat{u}_{eqv}$ générée par le RFNN.

$$E_{eqv}(k) = \frac{1}{2}\left[u_{eqv_d}(k) - \hat{u}_{eqv}(k)\right]^2 \tag{4.22}$$

Les paramètres ajustables du RFNN, sont réglés par l'algorithme BPA en utilisant la formule suivante :

$$W_{RFNN}(k+1) = W_{RFNN}(k) + \Delta W_{RFNN}(k) = W_{RFNN}(k) + \eta_{RFNN}\left(-\frac{\partial E_{eqv}(k)}{\partial W_{RFNN}(k)}\right) \tag{4.23}$$

Avec η_{RFNN} représente le taux d'apprentissage et W_{RFNN} les poids de RFNN à régler, qui sont respectivement w_i^1, a_{ij}, b_{ij}, et w_{oq}^4. Les formules de mise-à-jour de ces paramètres, sont les suivantes :

$$w_{oq}^4(k+1) = w_{oq}^4(k) - \eta_{RFNN}(u_{eqv}(k) - \hat{u}_{eqv}(k))y_q^3 \tag{4.24}$$

$$a_{ij}(k+1) = a_{ij}(k) - \eta_{RFNN}(u_{eqv}(k) - \hat{u}_{eqv}(k))w_i^1 y_q^3 \frac{y_i^1 - a_{ij}}{(b_{ij})^2} \tag{4.25}$$

$$b_{ij}(k+1) = b_{ij}(k) - \eta_{RFNN}(u_{eqv}(k) - \hat{u}_{eqv}(k))w_i^1 y_q^3 \frac{(y_i^1 - a_{ij})^2}{(b_{ij})^3} \tag{4.26}$$

$$w_i^1(k+1) = w_i^1(k) + \eta_{RFNN}\left(u_{eqv}(k) - \hat{u}_{eqv}(k)\right)w_i^1 y_q^3 \frac{y_i^1 - a_{ij}}{(b_{ij})^2} y_i^1(k-1) \tag{4.27}$$

L'équation (4.22) porte le terme $u_{eqv_d}(k)$, qui représente la commande équivalente désirée. Néanmoins, ce terme est inconnu, ce qui nous pose un problème. Dans ce cas, et afin de surmonter ce problème l'expression $\left(u_{eqv_d}(k) - \hat{u}_{eqv}(k)\right)$ dans (4.22) est remplacée par le terme correctif u_{Fuzzy}. En réalité, selon [Tsai 04] ce changement est possible puisque les deux termes ont la même nature.

Remarque 4.3. Le taux d'apprentissage η_{RFNN} est commun pour tous les paramètres à ajuster, il est réglé par optimisation à travers l'algorithme ACO proposé.

4.2.2.4 Calcul de la commande corrective

Le chattering est un phénomène fortement indésirable, à cause de ses effets nuisibles. À cet effet, la minimisation ou l'élimination compète de ce dernier devient indispensable. Dans le présent travail, et afin d'éliminer entièrement ce phénomène, nous avons utilisé un FS continu de type SISO en se référant au travail de Ahcene Boubakir [Bouba 09]. Le FS utilisé permet

d'approximer le terme correctif dans l'expression (4.16). Ainsi, il a comme entrée la variable de glissement σ_i et comme sortie la commande corrective $u_{Fuzzy\,i}$.

L'univers de discours de la variable d'entrée σ_i est partagé en cinq sous-ensembles flous comme le montre la figure 4.7 et l'équation ci-dessous [Kim 95, Liu 04] :

$$T(\tilde{\sigma_i}) = \{NB, NM, ZR, PM, PB\} = \{\tilde{F}_{\sigma_i}^1, \ldots., \tilde{F}_{\sigma_i}^5\} \tag{4.28}$$

Même chose pour la variable de sortie $u_{Fuzzy\,i}$, son univers de discours est partagé aussi en cinq sous-ensembles flous (figure 4.8) comme suit :

$$T(\tilde{u}_{Fuzzy\,i}) = \{NB, NM, ZR, PM, PB\} = \{\tilde{F}_{u_{Fuzzy\,i}}^1, \ldots, \tilde{F}_{u_{Fuzzy\,i}}^5\} \tag{4.29}$$

D'où, NB, NM, ZR, PM et PB sont respectivement; Negative Big, Negative Medium, Zero, Positive Medium, et Positive Big,

Chaque sous-ensemble flou dans les deux cas entrée/sortie est représenté par une fonction d'apparence de type Gaussien définie par :

$$\mu(x) = exp\left(\frac{-(x-a_F)^2}{2(b_F)^2}\right) \tag{4.30}$$

Avec a_F et b_F sont le centre et le poids de cette fonction d'appartenance, ils sont réglés pour chaque sous-ensemble flou par optimisation en utilisant le ACO.

Remarque 4.4. Le terme ϕ indiqué sur la figure 4.7 représente la couche limite (Boundary Layer) qui enveloppe la variable de glissement.

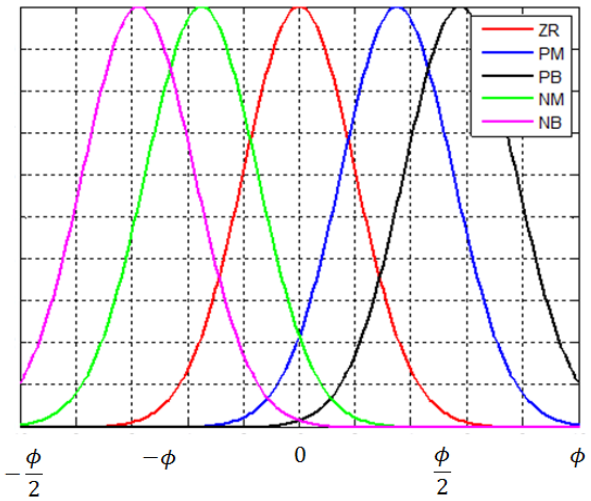

Figure 4.7 : Fuzzification de la variable σ_i.

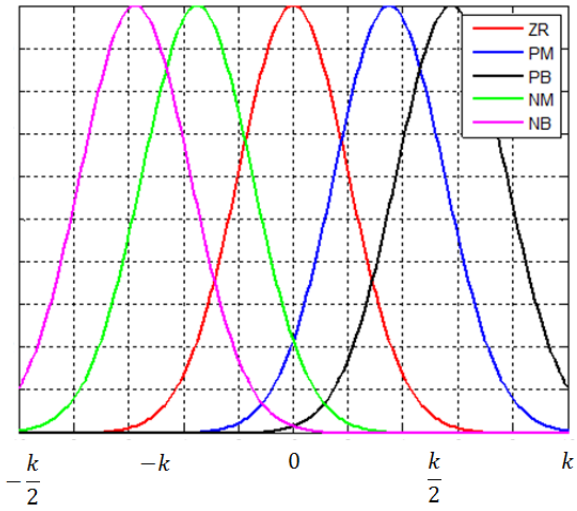

Figure 4.8 : Fuzzification du terme correctif $u_{Fuzzy\,i}$.
Chaque FS_i a la base de règles floues suivante [Kim 95] :

Règle 1: If σ is NB, then u_{Fuzzy} is PB,

Règle 2: If σ is NS, then u_{Fuzzy} is PS,

Règle 3: If σ is ZR, then u_{Fuzzy} is ZR,

Règle 4: If σ is PS, then u_{Fuzzy} is NS,

Règle 5: If σ is PB, then u_{Fuzzy} is NB.

La sortie de chaque FS_i est exprimée comme suit :

$$u_{fuzzy\,i} = -k_i \times sig(\sigma_i/\phi_i) \tag{4.31}$$

Avec: $sig(z) = \begin{cases} -1 & z < -1 \\ z & -1 \le z \le 1 \\ 1 & z > A \end{cases}$

4.2.2.5 Preuve de la stabilité

Considérant la fonction de Lyapunov globale candidate $V = \sum_{i=1}^{N} V_i$ avec ;

$$V_i = V_{RFNN_i} + V_{FSMC_i} \Rightarrow \dot{V}_i = \dot{V}_{RFNN_i} + \dot{V}_{FSMC_i} \tag{4.32}$$

D'où, $V_{RFNN_i} = \frac{1}{2}\left(e_{RFNN_i}(k)\right)^2 = \frac{1}{2}\left(E_{eqv\,i}(k)\right)^2$ est la fonction de Lyapunov relative à $RFNN_i$ avec est $e_{RFNN_i}(k)$ l'erreur d'apprentissage et V_{FSMC_i} la fonction de Lyapunov pour la partie (FS_i + modes glissants).

Nous avons aussi, $\dot{V}_{RFNN_i} \cong \frac{\Delta V_{RFNN_i}(k)}{\Delta T}$, représente la valeur de la dérivée de V_{RFNN_i}obtenue par approximation. Selon les auteurs de [Lee 00] $\Delta V_{RFNN_i}(k) < 0$ est vérifiée si les taux d'apprentissage, vérifient la condition suivante :

$$\eta_{w_{oq}^4, RFNN_i} \simeq \eta_{w_i^1, RFNN_i} = \eta_{a_{ij, RFNN_i}} = \eta_{b_{ij}, RFNN_i} = \gamma \times \left[\frac{1}{\left| w_{i,RFNN_i,max}^1 \right| \left(\frac{2}{b_{ij,RFNN_i,min}} \right)} \right]^2 \quad (4.33)$$

Avec $\gamma = \frac{1}{R_{RFNN_i} \times N_{RFNN_i,max}}$, et R_{RFNN_i} et $N_{RFNN_i,max}$ sont respectivement le nombre des règles et le nombre maximal des sous-ensembles flous dans le RFNN$_i$ (nombre de nœuds dans la seconde couche correspond à un nœud d'entrée).

Proposition 1. Pour le système interconnecté définie par (4.12), le FS$_i$ conçu, permet de garder l'état du sous-système i dans une région limitée $B_i(t)$. D'où, $B_i(t) = \{\bar{x}_i, |\sigma_i(\bar{x}_i, t)| \leq \phi_i\}$ et $\phi_i > 0$ si on choisi K_i comme suit ;

$$k_i \geq F_i + \sum_{k=0}^{P} \sum_{j=1}^{N} \xi_{ij}^k \left\| \bar{x}_{s_j} \right\|^k + D_i + \psi_i \quad (4.34)$$

Avec ψ_i est une constante postive.

Démonstration.

Définir la fonction de Lyapunov candidate suivante ;

$$V_{FSMCi} = \frac{1}{2}\sigma_i^2 \quad (4.35)$$

Nous avons ;

$$\begin{aligned} \dot{V}_{FSMCi} &= \sigma_i \left(\sum_{j=1}^{n_i-1} C_{i,1}\, e_{i,(j+1)} + f_i(\bar{x}_i) + g_i(\bar{x}_i) u_i(t) + z_i(\bar{x}_s) + d_i(\bar{x}_i) - x_{id}^{(n_i)} \right) \\ &= \sigma_i \left(f_i(\bar{x}_i) + z_i(\bar{x}_s) + d_i(\bar{x}_i) - K_i \times sig(\sigma_i/\Phi_i) \right) \end{aligned} \quad (4.36)$$

Si $|\sigma_i| > \phi_i$,

$$\begin{aligned} \dot{V}_{FSMCi} &\leq \sigma_i \left(F_i + \sum_{k=0}^{P} \sum_{j=1}^{N} \xi_{ij}^k \left\| \bar{x}_{s_j} \right\|^k + D_i - K_i sig(\sigma_i/\Phi_i) \right) \\ &\leq \sigma_i \left(F_i + \sum_{k=0}^{P} \sum_{j=1}^{N} \xi_{ij}^k \left\| \bar{x}_{s_j} \right\|^k + D_i \right) - K_i |\sigma_i| \\ &\leq -\psi_i \times |\sigma_i| < 0 \end{aligned} \quad (4.37)$$

À partir de (4.33) et (4.37), la condition suivante est vérifiée ;

$$\dot{V}_i = \dot{V}_{RFNN_i} + \dot{V}_{FSMC_i} < 0 \tag{4.38}$$

On conclu que le système globale interconnecté composé de N sous-système est donc asymptotiquement stable en boucle fermée avec la commande décentralisée (4.16).

4.2.3 Régalage des paramètres de la commande décentralisée par l'algorithme ACO amélioré

Afin d'améliorer les performances de la commande proposée, plusieurs paramètres sont calculés en utilisant la variante de ACO proposée, afin d'augmenter les capacités d'approximation des RFNN_i et des FS_i, ainsi que d'améliorer la vitesse de convergence vers les surfaces de glissement σ_i.

Le problème d'optimisation traité ici, est de type combinatoire. Dans ce cas, l'ensemble de toutes les variables recherchées, dont chacune représente une sous-solution, est représenté par un vecteur $\mathrm{P_S}$ exprimé comme suit :

$$\begin{aligned}\mathrm{P_S} = [&C_{11}, \dots\dots, C_{i,1}, C_{1,2}, \dots\dots, C_{i,2}, \dots\dots, C_{1,(n_i-1)}, \dots\dots, C_{i,(n_i-1)}, \eta_{Net\ 1}, \eta_{Net\ 2}, \dots\dots, \eta_{Net\ i},\\ &a_{input\ F_1}, b_{input\ F_1}, a_{input\ F_2}, b_{input\ F_2}, \dots\dots, a_{input\ F_i}, b_{input\ F_i}, a_{output\ F_1}, b_{input\ F_1},\\ &\dots\dots, a_{output\ F_i}, b_{output\ F_i}] = [\mathrm{p}^{i_p}]\end{aligned} \tag{4.39}$$

Avec $i_p = 1, \dots, N_p$ et N_p est le nombre total de tous les paramètres à régler. En outre, chaque sous contrôleur $\mathrm{RFNN\text{-}FSMC}_i$ a $\big((n_i - 1) + 4\big)$ paramètre à optimiser, en somme nous avons $N_p = N \times \big((n_i - 1) + 4\big)$ paramètres.

Le processus de réglage global comprend dans son début les démarches suivantes, en plus de toutes les étapes d'exécution de l'algorithme ACO qui viennent par la suite (Algorithme 2.2 : Algorithme ACO amélioré):

- Affectation des distributions aléatoires uniformes pour chaque paramètre à régler selon la méthode décrite dans la section 2.3.2 du second chapitre,
- Constriction d'un graphe complet comme un modèle équivalent au vecteur $\mathrm{P_S}$, semblable à celui présenté dans la figure 2.5, composé de $N_p \times J$ nœuds (paramètres),
- Formulation d'une fonction objectif $\mathrm{F_{obj}}()$, utilisée pour évaluer la qualité des solutions trouvées (vecteur $\mathrm{P_S}(t)$).

La fonction objectif exprimée comme suit, est celle utilisée :

$$\mathrm{F_{obj}} = \int\big(\big|e_{1,1}(t) + , \dots\dots, + e_{i,1}(t)\big|\big) + (|u_1(t) + , \dots\dots, + u_i(t)|) \tag{4.40}$$

Avec $e_{i,1}(t)$ est l'erreur de poursuite relative au sous-système i et $u_i(t)$ est la commande locale générée par le sous-contrôleur RFNN-FSMC$_i$.
Le processus d'optimisation est achevé d'une façon indirecte (*offline*), pendent t_{max} itérations.

4.3 Simulations

Afin de prouver l'efficacité de la méthode de commande ainsi que la variante de ACO préposée, deux exemples de systèmes interconnectés fortement non-linéaires seront présentés. Ils s'agit d'un simulateur de vol d'hélicoptère (*Twin Rotor Mimo system* TRMS) [Feed 98] et un système a double pendule inversés connectés entre eux [Yeh 97]. Dans ce qui suit, nous présentons, les configurations des structures de commandes adaptées aux systèmes en question ainsi que tous les résultats de simulation obtenus, accompagnés d'un ensemble de commentaires et d'analyses.

4.3.1 Exemple 1 : Commande d'un TRMS

Le TRMS comme le montre la figure 4.9, est un prototype aérodynamique conçu pour le développement et l'implémentation de nouvelles lois de commandes. Son comportement ressemble à celui d'un hélicoptère. De point de vue de commande, le TRMS est présenté comme un système d'ordre élevé fortement non linéaire fortement couplé. Une grande variété de lois de commandes ont été testées sur ce système en utilisant ; les modes glissants classiques [Allo 07], les modes glissants fusionnés avec les systèmes flous [Allo 12d], avec FNNs [Allo 12c], avec RFNNs [Allo 12e]. On trouve aussi, la commande prédictive dans le travail de Rahideh [Rahi 12], un régulateur PID dans [Juan 08a], la commande floue décentralisée dans [Rahi 06] et autres [Juan 08b, Tao 10].

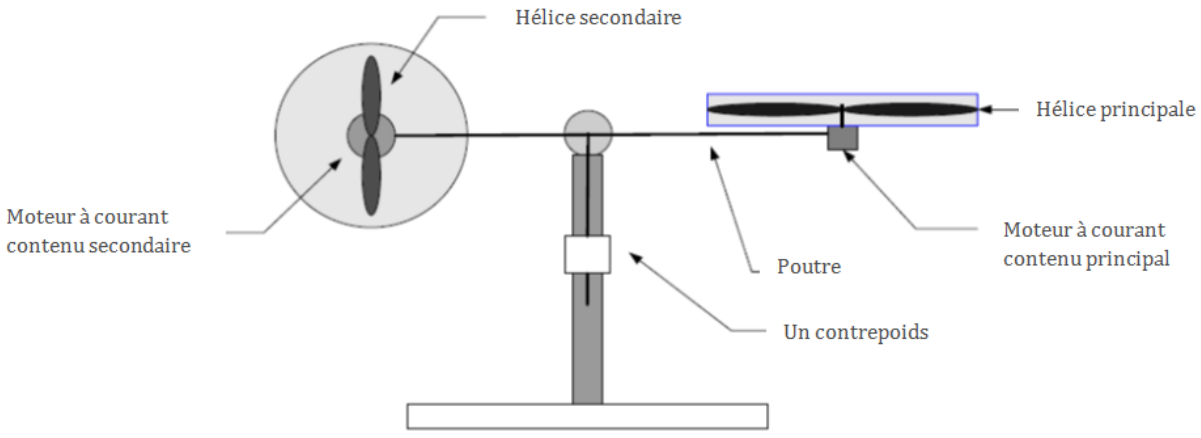

Figure 4.9 : Le TRMS.

L'objectif de la commande de ce système est de faire pivoter sa poutre principale autour des deux axes (horizontal et vertical) rapidement et avec précision en variant les forces

aérodynamiques des deux propulseurs. Dans un hélicoptère normal la force aérodynamique est commandée en changeant l'angle d'attaque. Cependant, dans le TRMS cet angle est fixé, et les forces aérodynamiques sont commandées en changeant la vitesse des propulseurs.

Le TRMS a deux modes de fonctionnement, mode couplé et mode découplé. En effet, le premier mode est le plus difficile en terme d'augmentation de degrés de libertés (2 degrés) ainsi la présence des effets de couplages. Pour plus de détailles sur le modèle et le principe de fonctionnement du TRMS, on conseille les lecteurs de consulter les références suivantes [Feed 98, Juan 08a].

Le TRMS a comme sorties l'angle d'élévation α_v (*Pitch angle*) et l'angle d'azimut α_h (*Yaw angle*) et comme entrées les deux commandes u_v et u_h.

La figure ci-dessous, illustre la structure de commande décentralisée synthétisée pour le TRMS.

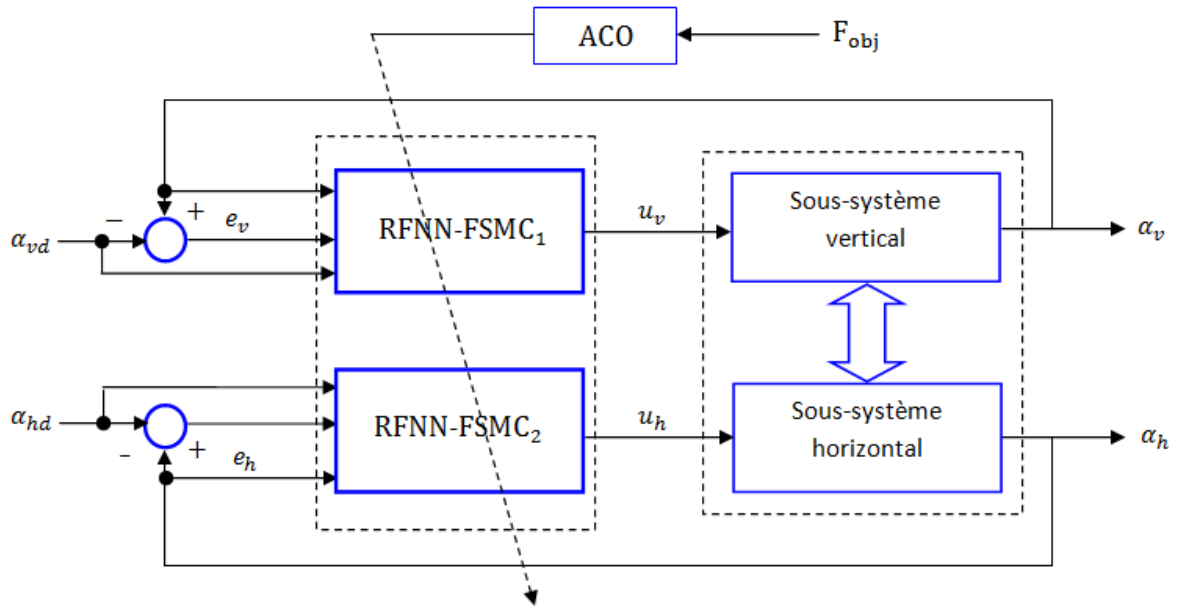

Figure 4.10 : Structure de commande décentralisée optimisée par ACO pour le TRMS.

Il est à noter, que la structure de commande globale est composée de deux sous-contrôleurs RFNN-FSMC$_{i=1,2}$, l'un pour le sous-système horizontal et l'outre pour le sous-système vertical. Chaque RFNN-FSMC$_{i=1,2}$ reçoit à son entrée comme le montre la figure 4.10 l'angle, sa valeur désirée et l'erreur de poursuite c.-à-d. $[\alpha_v, \alpha_{vd}, e_v]$ et $[\alpha_h, \alpha_{hd}, e_h]$.

Les paramètres de ces deux sous-contrôleurs sont fixés comme suit :

(i) Pour les deux RFNN$_{i=1,2}$, leurs paramètres ont au début de simulation les valeurs suivantes :

$a_{1\,ij} = a_{2\,ij} = 100 \times \begin{bmatrix} -1.5 & -0.8 & 0 & 0.8 & 1.5 \\ -1.5 & -0.8 & 0 & 0.8 & 1.5 \end{bmatrix}$, $b_{1\,ij} = b_{2\,ij} = 1.21$, $w^1_{1/i} = w^1_{2/i} = 0.010$.

$w^4_{1/oq} = w^4_{2/oq} = 0.0018$.

(ii) Les expressions de σ_1 et σ_2 sont données comme suit :

$$\sigma_1(e_1) = C_{1,1}e_{1,1} + C_{1,2}e_{1,2} + C_{1,3}e_{1,3} = C_{1,1}e_{1,1} + C_{1,2}\dot{e}_{1,1} + C_{1,3}\ddot{e}_{1,1} \quad (4.41)$$

$$\sigma_2(e_2) = C_{2,1}e_{2,1} + C_{2,2}e_{2,2} + C_{2,3}e_{2,3} = C_{2,1}e_{2,1} + C_{2,2}\dot{e}_{2,1} + C_{2,3}\ddot{e}_{2,1} \quad (4.42)$$

Les paramètres ϕ_1 et ϕ_2 qui représentent les couches limites pour les deux sous contrôleur RFNN-FSMC$_{i=1,2}$ ont tous les deux la valeur 120, ainsi k_1 et k_2 ont la valeur 20.

Pour cet exemple, le vecteur P_S qui porte tous paramètres à régler est exprimé comme suit :

$$P_S = \left[C_{i,j}, \eta_{RFNNi}, a_{inputF_i}, b_{inputF_i}, a_{outputF_i}, b_{outputF_i}\right] \quad (4.43)$$

avec $i = 1,2$ and $j = 1,.,3$

En somme, nous avons 16 paramètres, dont leurs bornes sont données par le tableau 4.1. Pour simplifier, les indices $i = 1,2$ indiquent le nombre du sous-système, $i = 1$ signifie le sous-système vertical et $i = 2$ signifie le sous-système horizontal.

Paramètre	Min	Max
$C_{1,1}, C_{1,2}, C_{1,3}$	49,14,0.7	196,28,1
$C_{2,1}, C_{2,2}, C_{2,3}$	49,14,0.7	196,28,1
$\eta_{RFNN1}, \eta_{RFNN2}$	0.004	0.05
$a_{inputF1}, a_{inputF2}$	10	120
$b_{inputF1}, b_{inputF2}$	10	21
$a_{outputF1}, a_{outputF2}$	2.5	20
$b_{outputF1}, b_{outputF2}$	2	4

Tableau 4.1 : Définition des bornes de paramètres.

L'algorithme ACO est exécuté avec le paramétrage suivant ; $m = 10$, $S_{seuil} = 3.5$, $r_0 = 0.25$, $t_{max} = 50$, $\rho = 0.1$, $Q = 0.01$ et J = 200, toutes les pistes de phéromone sont initialisées avec une concentration $\tau_0 = 0.1$.

De plus, l'algorithme est exécuté 10 fois pour 50 tours de fourmis (itérations). A chaque exécution l'algorithme converge à une solution acceptable après moins de 20 itérations. Parmi ces dix exécutions, le meilleur résultat correspondant à la valeur minimale de la fonction F_{obj}. La figure 4.11 montre l'évolution en fonction du nombre de tours de la fonction F_{obj} pour l'algorithme ACO proposé ainsi que le C-ACO. L'algorithme proposé converge vers une valeur de $F_{obj} = 0.8013$ qui correspond aux valeurs finales des paramètres données dans le tableau 4.2.

Paramètre	**Valeur**
$C_{1,1}, C_{1,2}, C_{1,3}$	58.8141, 16.5025, 0.9668
$C_{2,1}, C_{2,2}, C_{2,3}$	83.3116, 16.1106, 0.9834
$\eta_{RFNN1}, \eta_{RFNN\ 2}$	0.0269, 0.0301
$a_{inputF1}, a_{inputF2}$	50.9045, 103.9698
$b_{inputF1}, b_{inputF2}$	19.8945,13.7035
$a_{outputF1}, a_{outputF2}$	11.9975,15.7789
$b_{outputF1}, b_{outputF2}$	2.8141, 2.5729

Tableau 4.2 : Les valeurs finales des paramètres.

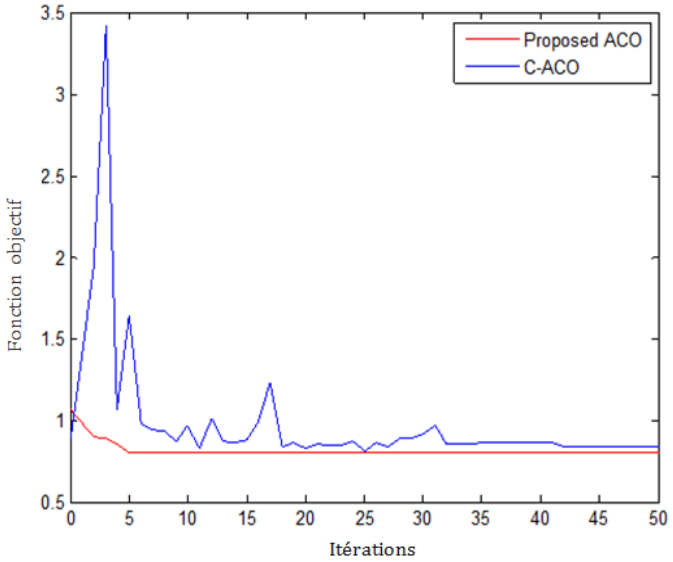

Figure 4.11 : Evolution de la fonction objectif pour l'algorithme ACO proposée et le C-ACO.

La meilleure valeur, la moyenne, la mauvaise et l'écart type obtenues pour la fonction objectif F_{obj} liées aux algorithmes C-ACO et celui proposé, sont illustrées dans le tableau 4.3.

Valeur	L'algorithme proposé	C-ACO
Meilleure	0.8013	0.8155
Moyenne	0.8114	0.9730
Mauvaise	1.0689	3.4177
Ecart type (*std.*)	0.0420	0.4038

Tableau 4.3 : Comparaison entre l'algorithme ACO proposé et le C-ACO.

Des tests de poursuite de trajectoires sinusoïdales et de stabilisation en mode couplé, ont été réalisés. En effet, les deux sous-systèmes vertical et horizontal, ont reçus dans le premier cas deux consignes différentes données par les équations (4.44) et (4.45). Les résultats de simulation obtenus sont illustrés dans les figure 4.12, figure 4.13 et figure 4.14.

$$\alpha_{vd}(t) = 0.3[\cos(0.23\pi t - 0.015\pi) + 0.3\sin(0.45\pi t)] \ (rad) \quad (4.44)$$

$$\alpha_{hd}(t) = 0.2[\cos(0.55\pi t - 0.015\pi) + 0.2\sin(0.45\pi t)] \ (rad) \quad (4.45)$$

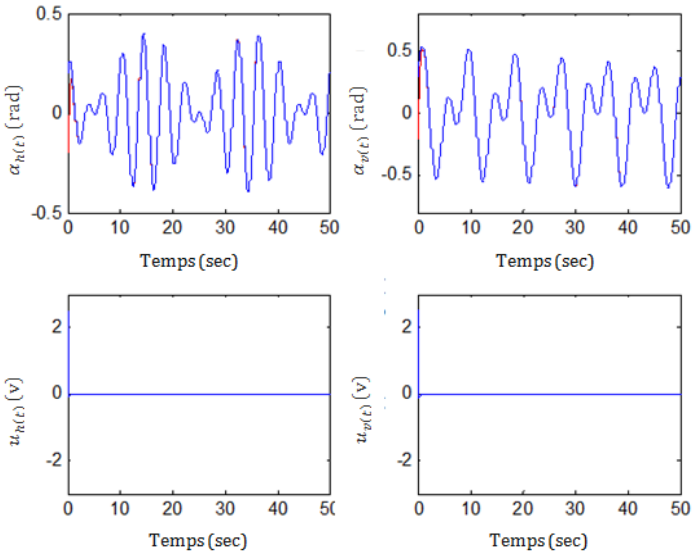

Figure 4.12 : Poursuite d'une trajectoire sinusoïdale ; réponses du TRMS (en bleu réponse du système et en rouge la trajectoire désirée) ; commandes générées $u_{i=1,2}$.

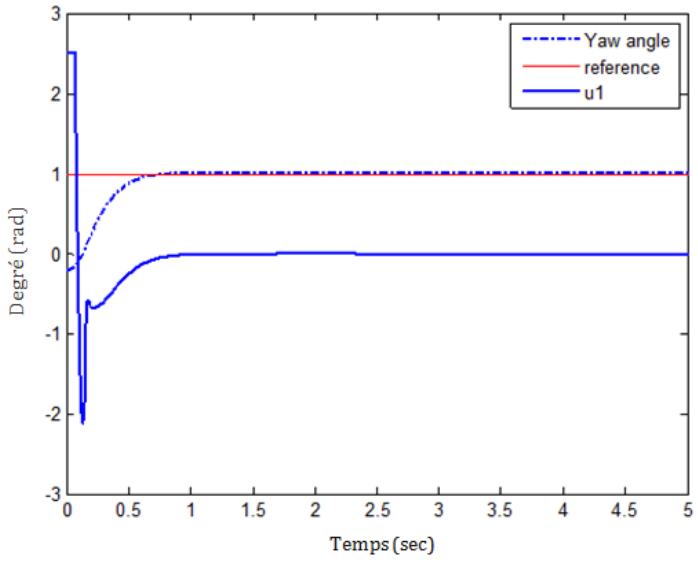

Figure 4.13 : Test de stabilisation du TRMS à un angle $\alpha_{hd} = 1\ rad$ (en bleu discontinu réponse réelle et en rouge valeur désirée) ; commande u_1.

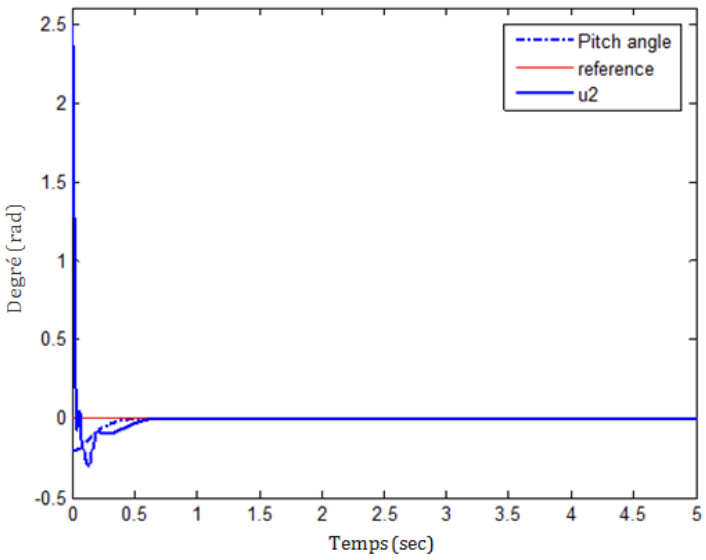

Figure 4.14 : Test de stabilisation du TRMS à un angle $\alpha_{vd} = 0\ rad$ (en bleu discontinu réponse réelle et en rouge valeur désirée) ; commande u_1.

De plus, le même test de poursuite des trajectoires données par (4.44) et (4.45), a été réalisé en utilisant une commande décentralisé par modes glissants classiques, les résultats de ce test sont illustrés dans la figure 4.15.

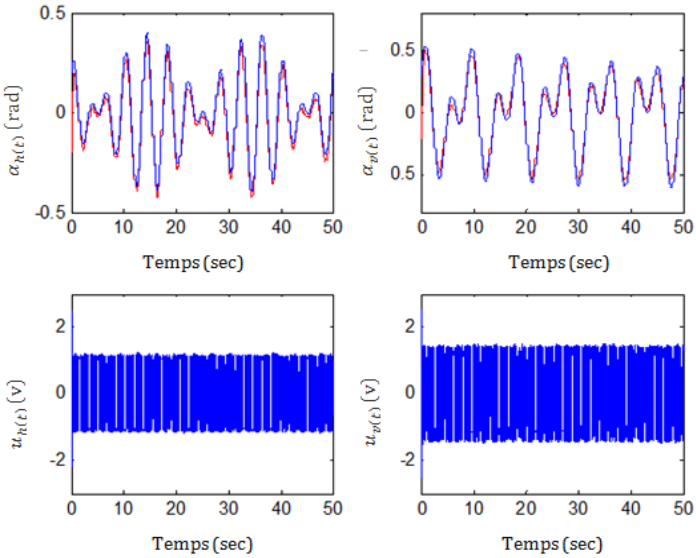

Figure 4.15 : Poursuite d'une trajectoire sinusoïdale avec commande décentralisée par modes glissants classiques ; réponses du TRMS (en bleu réponse du système et en rouge la trajectoire désirée) ; commandes générées $u_{i=1,2}$.

En outre, afin de tester la robustesse du contrôleur décentralisé global vis-à-vis des perturbations externes, nous avons changé la position des deux rotors (secondaire et principal)

en même temps, avec un angle égal à 1.5 rad à l'instant de simulation $t = 10s$ pendant le test de stabilisation. Les résultats obtenus, sont illustrés dans les figure 4.16, figure 4.17.

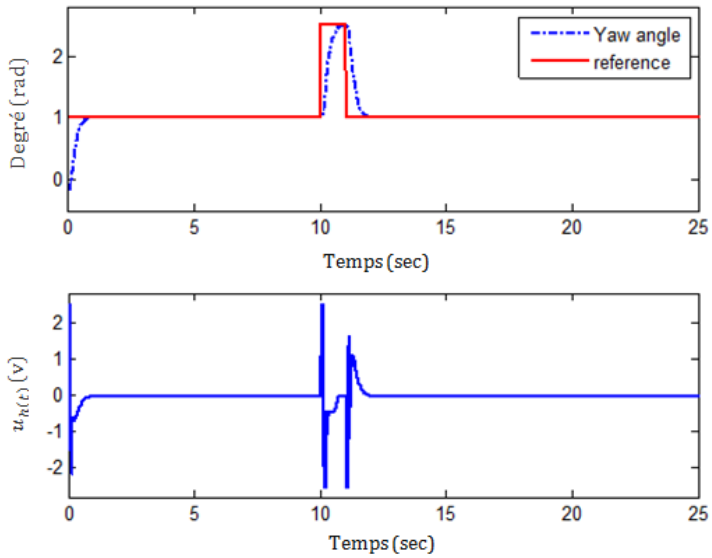

Figure 4.16 : Angle α_h avec la présence d'une perturbation externe ($\alpha_{p_h} = 1.5\ rad$) appliquée à l'instant $t = 10s$; commande u_1 générée.

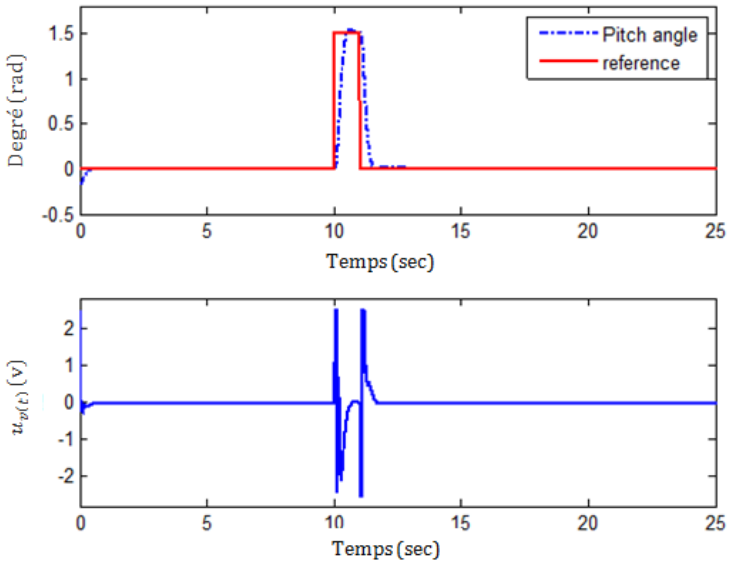

Figure 4.17 : Angle α_v avec la présence d'une perturbation externe ($\alpha_{p_v} = 1.5\ rad$) appliquée à l'instant $t = 10s$; commande u_2 générée.

4.3.2 Interprétation des résultats

Il est calcaire, comme le montrent la figure 4.11 et le tableau 4.3, que l'algorithme ACO proposé donne de meilleurs résultats par rapport au C-ACO en termes de rapidité de

convergence et de précision. Ceci est dû principalement aux déférentes améliorations, que nous avons introduite sur le C-ACO.

En terme de commande, les résultats de simulations obtenus à travers les différents tests ; suivi de trajectoires imposées, stabilisation et stabilisation avec la présence des perturbations externes ainsi que ceux obtenus en utilisant une commande décentralisée par modes glissants classiques, ont montrés l'efficacité de la méthode de commande proposée. On remarque que les réponses du TRMS dans tous les cas (tests) ont un temps de réponse très court, aussi la robustesse remarquable vis-à-vis du changement dans les états du système pendent le processus de commande. De plus, le phénomène de chattering a été complètement éliminé sans aucune dégradation dans la qualité des réponses du système, ce qui démontre fortement l'efficacité de la méthode.

Il est à noter, que l'implémentation pratique sur le prototype réel du TRMS est possible, une fois le processus de régalage des paramètres de la de commande globale est fait. Néanmoins, un réglage on-line est presque impossible, puisque le processus d'entrainement par une métaheuristique est généralement très gourmand en temps calcul. De plus, il y a le risque de tomber dans un état d'instabilité au début du réglage par une sélection prématuré aléatoire des paramètres en question. De ce fait, la présence d'un modèle très proche au système réel est très indispensable pour réussir la procédure.

4.3.3 Exemple 2 : Commande d'un système double pendule inversé

Nous considérons un autre exemple de systèmes interconnectés non-linaires, il s'agit du système double pendule inversé. Comme le montre la figure 4.18, ce système est composé de deux pendules inversés liées entre eux par un ressort fixé sur eux à une position $a(t)$ par rapport aux points d'articulation.

Ce système a été utilisé pour la première fois par Shi et Singh [Shi 92], il peut être considéré comme un système complexe composé de deux sous-systèmes interconnectés. Chaque sous-système peut être commandé par une unité de commande locale $u_i(t)$ $i = 1,2$, qui n'a accès qu'aux variables disponibles localement et n'agit que sur les variables de commande relatives au sous-système en question.

La commande de ce système sert à amener chaque pendule, à suivre une trajectoire désirée avec une mobilité permanente des deux chariots. Plusieurs travaux ont été réalisés en utilisant ce système comme un exemple, les références suivantes présentent de bons exemples [Lam 00, Zhan 01, Da 00, Yeh 99].

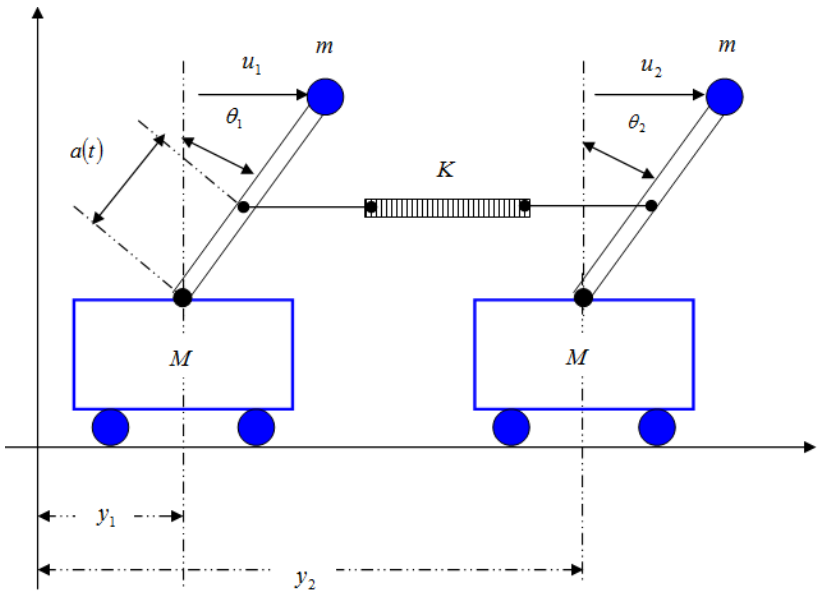

Figure 4.18 : Système double pendule inversé.

Le modèle mathématique qui caractérise le comportement dynamique de ce système, est issu de la littérature [Da 00, Yeh 99]. Il est exprimé sous forme de deux sous-systèmes S_1 et S_2 interconnectés, chacun a ses propres variables d'état, de commande et de sortie locales. Les équations d'état non linéaires sont donnés par :

$$S_1 \begin{cases} \dot{x}_{11} = x_{12} \\ \left(\frac{g}{cl} - \frac{ka(t)(a(t)-cl)}{cml^2}\right) x_{11} + \frac{1}{cml^2} u_1 + \frac{ka(t)(a(t)-cl)}{cml^2} x_{21} - B_1 x_{12}^2 - \frac{k(a(t)-cl)}{cml^2}(y_1 - y_2) \end{cases} \quad (4.46)$$

$$S_2 \begin{cases} \dot{x}_{21} = x_{22} \\ \left(\frac{g}{cl} - \frac{ka(t)(a(t)-cl)}{cml^2}\right) x_{21} + \frac{1}{cml^2} u_2 + \frac{ka(t)(a(t)-cl)}{cml^2} x_{11} - B_2 x_{22}^2 - \frac{k(a(t)-cl)}{cml^2}(y_2 - y_1) \end{cases} \quad (4.47)$$

Tel que ; S_1 et S_2 sont les deux sous systèmes, avec $x_{11} = \theta_1$ et $x_{12} = \dot{\theta}_2$ ainsi que $x_{21} = \theta_2$ et $x_{22} = \dot{\theta}_2$. En a aussi :

$y_1 = sin(2t)$, $y_1 = L + sin(3t)$, $B_i = \frac{m}{M} sin(\theta_i)$ telle que $\|B_i\| \leq \frac{m}{M}, i = 1,2$, $c = \frac{m}{m+M}$, $a(t) = sin(5t)$.

L'expression des termes d'interconnexion peut être tirée à partir de ce modèle d'état. Elles sont exprimées par :

$$Z_1(x_{11}, t) = a(t)\left(a(t) - \frac{1}{2}\right)(\theta_2 - \theta_1) - \frac{m}{2} B_1 \dot{\theta}_1^2 - \left(a(t) - \frac{1}{2}\right)(y_1 - y_2) \quad (4.48)$$

$$Z_2(x_{21},t) = a(t)\left(a(t)-\frac{1}{2}\right)(\theta_1-\theta_2) - \frac{m}{2}B_2\dot{\theta}_2^2 - \left(a(t)-\frac{1}{2}\right)(y_2-y_1) \quad (4.49)$$

Ces deux termes sont caractérisés par des expressions non-linéaires, comme indiquent les deux équations ci-dessus.

Le schéma de commande de ce système est similaire à celui présenté sur la figure 4.10 puisque, le nombre de ses sous-systèmes est le même que celui du TRMS. De plus, les deux $\text{RFNN}_{i=1,2}$ ainsi que les $\text{FS}_{i=1,2}$ ont les mêmes configurations que ceux dans la structure de commande présentée dans le premier exemple, à part les centres des fonctions d'apparences des deux $\text{FS}_{i=1,2}$ qui ont les valeurs suivantes $a_{1\,ij} = a_{2\,ij} = 10^{-4} \times \begin{bmatrix} -1.5 & -0.8 & 0 & 0.8 & 1.5 \\ -1.5 & -0.8 & 0 & 0.8 & 1.5 \end{bmatrix}$. De même pour les variables de glissement $\sigma_1()$ et $\sigma_2()$, les paramètres à optimiser ainsi que la fonction objectif F_{obj}.

Dans cet exemple, l'algorithme ACO proposé est simulé avec le paramétrage suivant : $m = 10$, $\text{S}_{\text{seuil}} = 200$, $r_0 = 0.45$, $t_{max} = 50$,$\rho = 0.1$, $Q = 0.01$ et $\text{J} = 200$, toutes les pistes de phéromone sont initialisées avec une concentration $\tau_0 = 0.01$. En outre, les paramètres à optimiser ont les mêmes intervalles de recherches avec les mêmes bornes, sauf les deux taux d'apprentissage η_{RFNN1}et η_{RFNN2} qui doivent être recherchés dans l'intervalle [0.95, 1.1].

Les résultats de simulations issus de l'application de la méthode de commande proposée, ainsi que les modes glissants classiques sur ce système, sont illustrés dans les figures 4.19 jusqu'à 4. 23. Ils comprennent respectivement, l'évolution en fonction du nombre d'itérations (t) de la fonction F_{obj} pour la variante ACO proposée ainsi que le C-ACO, les réponses du système et les signaux de commandes générés par les deux sous-contrôleurs en poursuite de trajectoires (carrées) et en stabilisation.

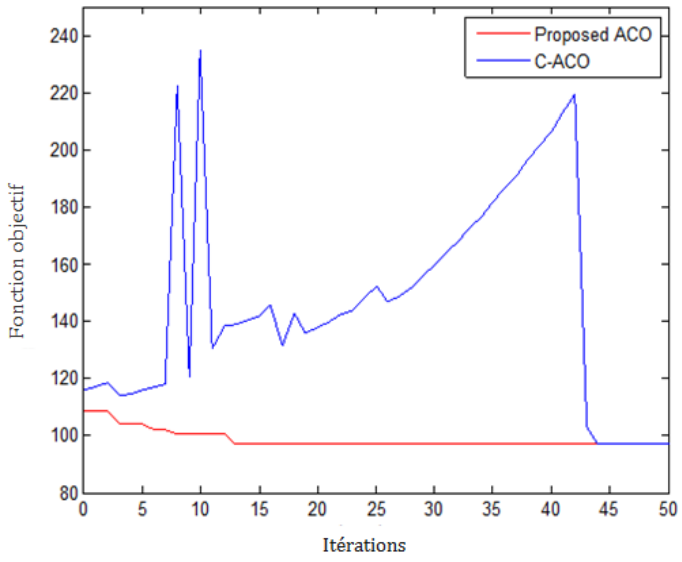

Figure 4.19 : Evolution de la fonction objectif pour l'algorithme ACO proposée et le C-ACO.

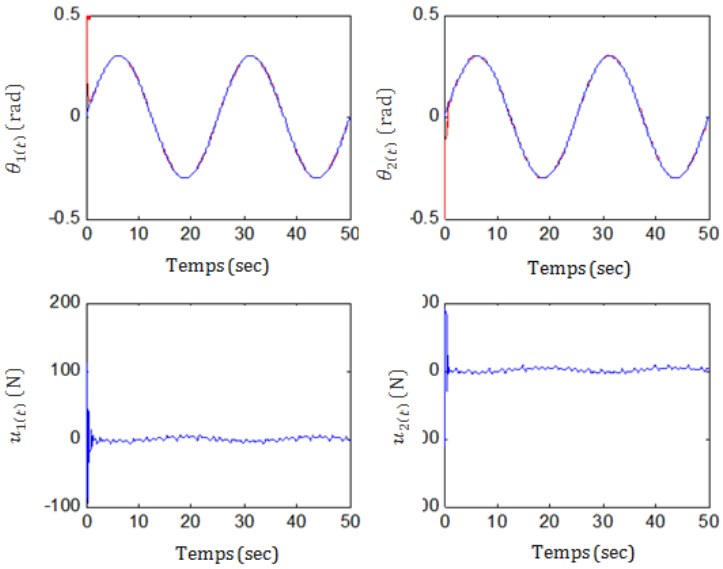

Figure 4.20 : Poursuite d'une trajectoire sinusoïdale ; réponses du système double pendule inversé (en rouge réponse du système et en bleu la trajectoire désirée) ; commandes générées $u_{i=1,2}$.

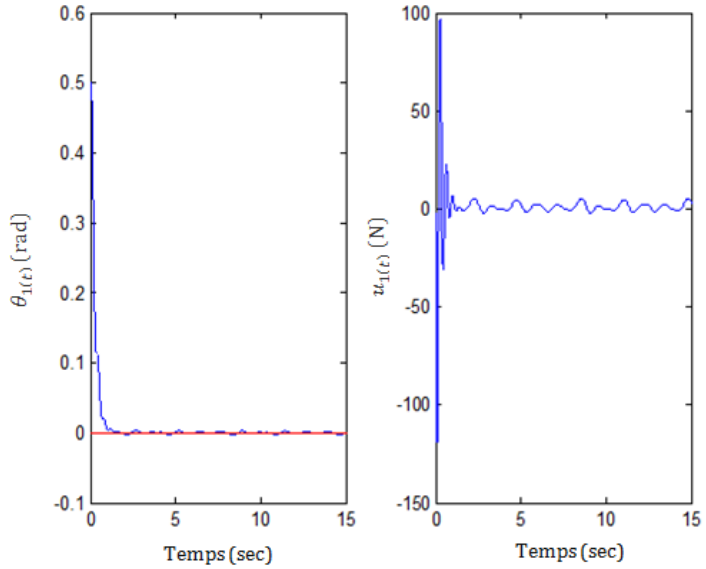

Figure 4.21 : Test de stabilisation du système double pendule inversé à un angle $\theta_1 = 0\ rad$ (en bleu réponse réelle et en rouge valeur désirée) ; commande u_1.

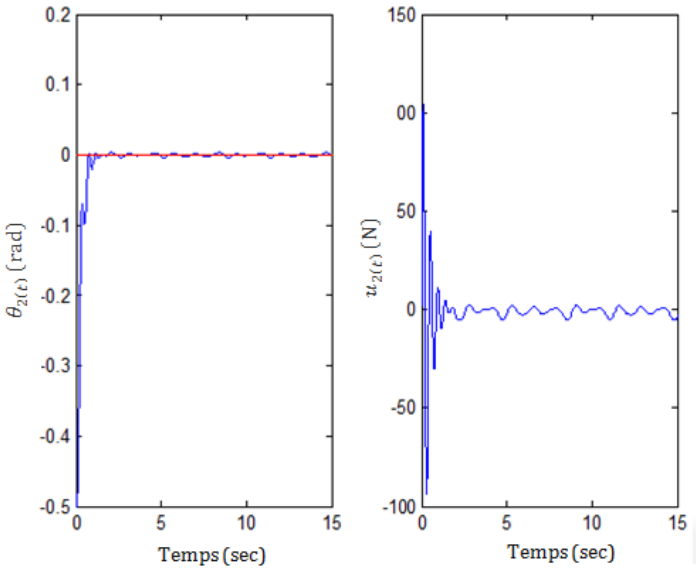

Figure 4.22 : Test de stabilisation du système double pendule inversé à un angle $\theta_2 = 0\ rad$ (en bleu réponse réelle et en rouge valeur désirée) ; commande u_1.

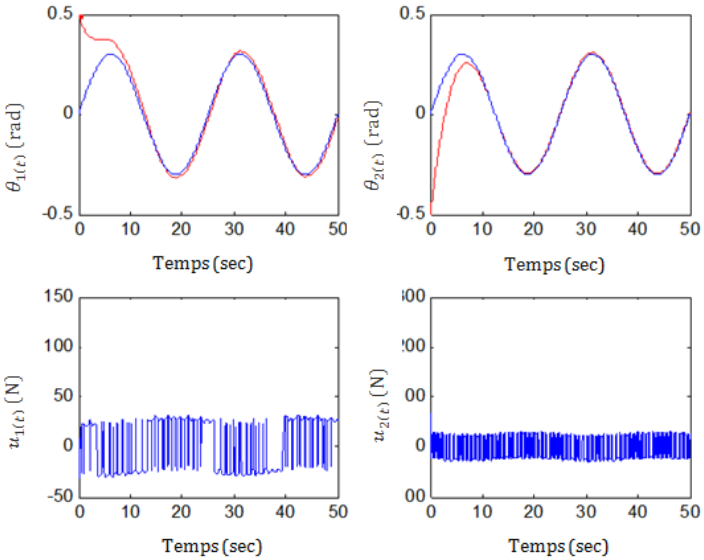

Figure 4.23 : Poursuite d'une trajectoire sinusoïdale avec une commande décentralisée par modes glissants classiques ; réponses du système double pendule inversé (en rouge réponse du système et en bleu la trajectoire désirée) ; commandes générées $u_{i=1,2}$.

À partir de tous ces résultats, nous pouvons remarquer dans un premier temps que l'algorithme d'optimisation proposé est meilleur par rapport au **C-ACO**, toujours en terme de rapidité de convergence et de précision. En outre, les deux angles $\boldsymbol{\theta_1}$et $\boldsymbol{\theta_2}$ du système sont stabilisés aussi bien en régulation qu'on poursuite de trajectoires. Ainsi, les commandes générées sont complètement filtrées des oscillations à haute fréquence (phénomène de

chattering). Toutes ces constations, prouvent l'efficacité de la méthode de commande proposée ainsi que la l'algorithme d'optimisation proposée.

4.4 Conclusions

En somme, nous pouvons conclure que l'approche de commande présentée, basée sur la combinaison entre deux techniques issues de l'intelligence artificielle, à savoir : les RFNNs et les FSs et une métaheuristique d'optimisation ainsi que la commande décentralisée, permet d'offrir les avantages suivants :

- La simplicité de la commande due principalement à l'utilisation d'une structure décentralisée. Cette dernière présente un certain nombre d'avantages à savoir : la minimisation du volume d'informations traités par les unités de commande, la simplicité des lois de commande élaborées par rapport au cas centralisé ainsi que la fiabilité de transfère de données en utilisant uniquement l'information locale.
- Le calcul de la commande équivalente (locale) ne nécessite pas, comme dans le cas des modes glissants classique la connaissance de l'équation dynamique de chaque sous-système,
- Le phénomène de chattering ainsi que tous ses effets, sont complètement éliminés sans aucune dégradation dans les performances des commandes synthétisées,
- L'utilisation de l'algorithme ACO avec des améliorations dans ses capacités dans le réglage de quelques paramètres de chaque sous-contrôleur permet d'améliorer significativement les performances du système,
- L'emploi des RFNN et les FSs comme des approximateurs dans le calcul des composantes des lois de commandes locales rend non indispensable la connaissance des bornes des interconnections entre les sous-systèmes ainsi que les incertitudes existantes. Ceci est très utile dans le cas des implémentations pratiques où l'estimation des limites de ces grandeurs devient très difficile.

CHAPITRE 5

Application de l'algorithme GHSACO au problème d'apprentissage des RFNNs en commande adaptative

5.1 Introduction

Dans ce chapitre, une commande adaptative indirecte, basée sur les RFNNs, d'une classe de systèmes non linéaires sera présentée.

Selon les travaux de Ching-Hung Lee [Lee 00], les RFNNs peuvent être considérés comme des approximateurs universels avec une précision souvent meilleure que les FNNs et les NNs, en raison de leur structure dynamique.

Afin d'exploiter cette propriété, nous avons utilisé les RFNNs dans l'estimation des fonctions caractérisant la dynamique du système à commander [Ioan 01, Park 04a, Park 04b]). De plus, la commande adaptative proposée, est munie d'un contrôleur superviseur [Wang 94], intégré afin de garantir la présence des états du système commandé dans des régions sécurisées. Tous les paramètres (poids) du RFNN seront ajustés *offline* en utilisant l'algorithme GHSACO en évitant l'emploi des méthodes d'entrainement classiques, telles que l'algorithme BPA [Werb 90] et ses variantes classiques p. ex. [Guia 98] qui présentent certains défauts en terme de ; stagnation dans des minimums locaux, de leur vitesse de convergence très lente, de la nécessité d'une adaptation et une initialisation appropriées de leurs paramètres et la difficulté de calcul rencontrée lorsque l'architecture du réseau entrainé devient complexe [Krem 01].

Les capacités de la méthode de commande proposée sont évaluées en utilisant un système pendule inversé [Lo 98]. L'approche introduite (GHSACO + commande adaptative) montre son efficacité comme nous le verrons par la suite à travers les différents résultats obtenus.

5.1 Commande adaptative indirecte basée sur un contrôleur superviseur et les RFNNs

5.1.1 Objectifs de la commande proposée

On considère le système non-linaire d'ordre n défini comme suit :

$\dot{x}_1 = x_2$

$\dot{x}_2 = x_3$

……

$\dot{x}_n = f(x_1, \ldots\ldots, x_n) + g(x_1, \ldots\ldots, x_n)u$ (5.1)

$y = x_1$

Avec $u \in \mathbb{R}$ et $y \in \mathbb{R}$ sont respectivement, l'entrée et la réponse du système, $x = [x_1, x_2, \dots., x_n]^T$ est le vecteur d'états mesurables. Nous supposons également que f et g sont des fonctions continues incertaines, et g sans perte de généralité, est une fonction strictement positive. Selon [Isid 89], ces systèmes ont une forme normale avec un degré relatif égal à n.

L'objectif de la commande est de concevoir une commande adaptative indirecte basé sur les RFNNs de telle sorte que la sortie du système $y(t) = x_1$ suive une référence bornée $y_m(t)$.

On définit l'erreur de poursuite $e(t) = y_m(t) - y(t)$, avec $\delta = \left[e, \dot{e}, \dots., e^{(n-1)}\right]^T$ et $k = [k_n, \dots., k_1]^T$, telles que toutes les racines du polynôme $h(s) = s^n + k_1 s^{n-1} + \cdots + k_n$ soient dans le demi-plan gauche.

Connaissant les deux fonctions f et g, une loi de commande optimale peut être définie comme suit :

$$u^* = \frac{1}{g(x)}\left[-f(x) + y_m^{(n)} + k^T\delta\right] \tag{5.2}$$

À partir de (5.1) et (5.2), nous obtenons :

$$e^{(n)} + k_1 e^{(n-1)} + \cdots + k_n e = 0 \tag{5.3}$$

Cependant, puisque les deux fonctions f et g sont généralement inconnues ou mal connues, la loi de commande (5.2) ne peut pas être obtenue. À cet effet, nous avons utilisé les RFNNs comme approximateurs permettant d'approximer ces fonctions.

Remarque 5.1.

Les RFNNs utilisés dans ce chapitre ont la même structure que ceux utilisés dans le chapitre 4.

5.1.2 Synthèse de la commande adaptative indirecte

Tout d'abord, nous remplaçons les deux fonctions f et g dans (5.2) par leurs expressions approximatives $\hat{f}\left(x\middle|w_i^1, a_{ij}, b_{ij}, w_{oq}^4\right)$ et $\hat{g}\left(x\middle|w_i^1, a_{ij}, b_{ij}, w_{oq}^4\right)$. Par conséquent, l'expression de la loi de commande (5.2) devient :

$$u_c = \frac{1}{\hat{g}}\left[-\hat{f} + y_m^{(n)} + k^T\delta\right] \tag{5.4}$$

Après cette modification, (5.2) devient connue [Sast 11]. En substituant, maintenant (5.4) dans (5.1) et après quelques manipulations, on obtient l'équation d'erreur suivante :

$$e^{(n)} = -k^T\delta + \left[\hat{f} - f(x)\right] + [\hat{g} - g(x)]u_c \tag{5.5}$$

L'équation (5.5) est équivalente à :

$$\dot{\delta} = \Lambda_c\delta + b_c\left[\left[\hat{f} - f(x)\right] + [\hat{g} - g(x)]u_c\right] \tag{5.6}$$

Avec;

$$\Lambda_c = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ -k_n & -k_{n-1} & \cdots & \cdots & \cdots & \cdots & -k_1 \end{bmatrix} \text{ and } b_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ 1 \end{bmatrix} \tag{5.7}$$

Puisque la matrice Λ_c est stable, par effet de $\left(|sI - \Lambda_c| = s^{(n)} + k_1 s^{(n-1)} + \cdots + k_n\right)$, on peut extraire une matrice P symétrique positive de dimension $n \times n$ qui satisfait l'équation de Lyapunov [Slot 91]:

$$\Lambda_c^{\mathrm{T}} P + P\Lambda_c = -Q \tag{5.8}$$

Avec Q est une matrice positive.

Définissant l'équation $V_\delta = \frac{1}{2}\delta^T P\delta$

En se basant sur (5.6) et (5.8) nous obtenons :

$$\dot{V}_\delta = \frac{1}{2}\dot{\delta}^T P\delta + \frac{1}{2}\delta^T P\dot{\delta} = -\frac{1}{2}\delta^T Q\delta + \delta^T P b_C\left[\left[\hat{f} - f(x)\right] + [\hat{g} - g(x)]u_c\right] \tag{5.9}$$

Hypothèse 5.1.

Les grandeurs suivantes $f^U(x), g^U(x)$ et $g_L(x)$ qui satisfont les conditions suivantes ; $|f(x)| \leq f^U(x), g_L(x) \leq g(x) \leq g^U(x)$ avec $f^U(x) < \infty$ et $g_L(x) > 0$ pour $x \in U_c$ mesurables.

5.1.3 Contrôleur superviseur

Dans ce travail, nous proposons l'utilisation d'un contrôleur superviseur qui sert à assurer les états du système dans des régions sécurisées. La commande générée par ce dernier est notée u_s. En effet, la commande globale u est la somme de u_s et de u_c définie par (5.4).

$$u = u_c + u_s \tag{5.10}$$

La présence de u_s dans (5.10) dépend de la valeur de V_δ. En effet, u_s n'intervient pas dans (5.10) si la condition $V_\delta \leq p_c$ est vérifie, avec p_c est une constante positive définie par le concepteur. Dans le cas contraire ($V_\delta > p_c$), le contrôleur superviseur sera mis en marche.
Si nous substituons (5.10) dans (5.1), l'équation (5.6) devient :

$$\dot{\delta} = \Gamma_c \delta + b_c \left[[\hat{f} - f(x)] + [\hat{g} - g(x)]u_c - g(x)u_s \right] \tag{5.11}$$

En utilisant (5.11) et (5.8), on obtient :

$$\dot{V}_\delta = -\frac{1}{2}\delta^T Q\delta + \delta^T P b_C \left[[\hat{f} - f(x)] + [\hat{g} - g(x)]u_c - g(x)u_s \right]$$

$$\leq -\tfrac{1}{2}\delta^T Q\delta + |\delta^T P b_C| \left[[|\hat{f}| + |f(x)|] + |[\hat{g} + g(x)]u_c| \right] - \delta^T P b_C g(x) u_s \tag{5.12}$$

En se basant sur l'hypothèse 5.1 et (5.12), l'expression de u_s devient :

$$u_s = S_w sgn(\delta^T P b_C) \tfrac{1}{g_L(x)} \left[[|\hat{f}| + |f^U(x)|] + |[\hat{g} + g^U(x)]u_c| \right] \tag{5.13}$$

Avec S_w=1 si $V_\delta > p_c$, $sgn()$ est la fonction $sign$. Nous avons aussi le cas, S_w=0 si $V_\delta \leq p_c$.

En substituant (5.13) dans (5.12), avec la condition $V_\delta > p_c$ on obtient :

$$\dot{V}_\delta = -\frac{1}{2}\delta^T Q\delta + \delta^T P b_C \left[[\hat{f} + f(x)] + [\hat{g} - g(x)]u_c - \frac{g(x)}{g_L(x)} \times \left[[|\hat{f}| + |f^U(x)|] \right. \right.$$
$$\left. \left. + |[\hat{g} + g^U(x)]u_c| \right] \right]$$

$$\leq -\tfrac{1}{2}\delta^T Q\delta \leq 0 \tag{5.14}$$

On peut donc conclure que le système (5.1) est asymptotiquement stable en boucle fermée en utilisant la commande (5.10).

5.2 Apprentissage des RFNNs intégrés dans une structure de commande adaptive par algorithme GHSACO

Afin d'éviter les problèmes présentés par l'algorithme BPA et ses variantes classiques [Werb 90, Guia 98] lors de l'apprentissage des RFNNs, nous proposons l'utilisation de l'algorithme GHSACO développé dans le chapitre 4.

En effet, la structure de commande globale comprend deux RFNNs, l'un pour approximer la fonction f noté RFNN$_{\hat{f}}$ et l'autre pour approximer la fonction g noté RFNN$_{\hat{g}}$. Chaque réseau a l'ensemble de poids caractérisés par le vecteur $V_{\hat{f}\ et\ \hat{g}} = \left[w_i^1, a_{ij}, b_{ij}, w_{oq}^4 \right]_{\hat{f}\ et\ \hat{g}}$.

Tous ces paramètres sont obtenues automatiquement (*offline*) par leur codage en tant que partie d'un vecteur de solutions, c.-à-d. chacun de ces paramètres constitue une variable de décision. Ceci permet de déterminer la structure optimale de chaque réseau RFNN$_{\hat{f}}$ et RFNN$_{\hat{g}}$.

Le processus d'apprentissage (optimisation) par algorithme GHSACO des paramètres des RFNNs de la commande adaptative indirecte synthétisée comprend les cinq étapes suivantes :

Etape 1. *Définition du problème*

Le problème d'optimisation consiste à minimiser une fonction objectif $F_{obj}(\underline{P})$, ici $\underline{P}$ est un vecteur de solutions candidates, exprimé par $\underline{P} = [p_1, p_2, p_3, \dots\dots, p_n]^T \in \mathbb{R}^n$, avec $p_i \in \widehat{P}_i$ et $i = 1,2, \dots\dots, n$ et $\widehat{P}_i = [LB_i \ UB_i]$ représente l'univers de discours de la variable p_i, avec LB_i et UB_i représentent respectivement la valeur minimale et maximale de p_i.

La fonction F_{obj} est utilisée ici pour évaluer la qualité des réponses obtenues. Dans le présent travail, nous avons utilisé la fonction ''*intégrale de la valeur absolue de l'erreur de poursuite* $e(t)$'' (*Integral Absolute Error* IAE) donnée par ; $\sum_k^{ST}|e(k)|\Delta t_c$ avec ; ST est le temps de simulation, et Δt_c est le pas de simulation.

Spécifiquement, pour le problème traité ici, le vecteur de $\underline{P}$ peut être exprimé comme suit :

$$\underline{P} = \left[w_{11}^1, w_{12}^1, .. w_{1m}^1\middle|, w_{21}^1, w_{22}^1, .. w_{2m}^1\middle|, \dots\dots, w_{i1}^1, w_{i2}^1, .. w_{im}^1\middle|, a(1,1), a(1,2), \dots, a(1,j), a(2,1), a(2,2), \dots, a(2,j), \dots . a(i,j)|b(1,1), b(1,2) \dots, b(1,j), b(2,1), b(2,2), \dots, b(2,j), \dots . b(i,j)|w_1^4, w_2^4, \dots, w_q^4\right]^T \quad (5.15)$$

Étape 2. Produire un nombre de vecteurs (harmonies) $\underline{P}(j)(j \in [1,n])$ égal à HMS, qui portent des solutions candidates aléatoires. Tous ces vecteurs sont regroupés (stockés) dans la matrice HM. Un élément p_j dans un vecteur $\underline{P}$ parmi les autres HMS vecteurs est généré comme suit :

$$\underline{P}(j) = LB_j + rand() \times (UB_j - LB_j) \quad (5.16)$$

Étape 3. Déterminer la structure de chaque réseau RFNN$_{\hat{f}}$ et RFNN$_{\hat{g}}$ en utilisant les variables de décision p_j de l'harmonie $\underline{P}$.

Étape 4. Calculer la valeur de la fonction objectif F_{obj} pour chaque harmonie $\underline{P}$

Étape 5. Pour chaque itération $t = 1, \dots, NI$, l'algorithme GHSACO fonctionne soit comme un algorithme GHS, soit comme un algorithme ACO, selon un mécanisme de commutation contrôlé par un paramètre aléatoire $q_1 = rand\ ()$.

- Un nouveau vecteur $\underline{P}_{new}$ (nouvelle harmonie) produit contenant les paramètres optimaux de chaque réseau ($RFNN_{\hat{f}}$ et $RFNN_{\hat{g}}$),
- Évaluer la fonction objectif F_{obj} pour cette nouvelle harmonie, si $F_{obj}(\underline{P}_{new}) = min\left(F_{obj}(\underline{P}^1), F_{obj}(\underline{P}^2), \ldots., F_{obj}(\underline{P}^{HMS})\right)$, la mauvaise harmonie dans matrice HM doit être remplacée par cette nouvelle harmonie. Réorganiser la matrice HM en triant tous les vecteurs $\underline{P}_i(j)(i \in [1, \text{HMS}])$ de décision en fonction des valeurs de leur fonction objectif.
- Utiliser la nouvelle meilleure valeur de $F_{obj}(\underline{P}_{new})$ obtenue, afin de mettre à jour la concentration des pistes de phéromone (matrice $\tau_{ij}(t)$, $i \in [1, n], j \in [1, \text{HMS}]$), en utilisant la règle (3.11).

Étape 6. Arrêter la recherche si $i_{it} \geq NI$.

Le schéma global de principe de la commande proposée, qui comprend le contrôleur adaptative indirect IARFNNC, le contrôleur superviseur et l'algorithme GHSACO est représentée par la figure 5.1.

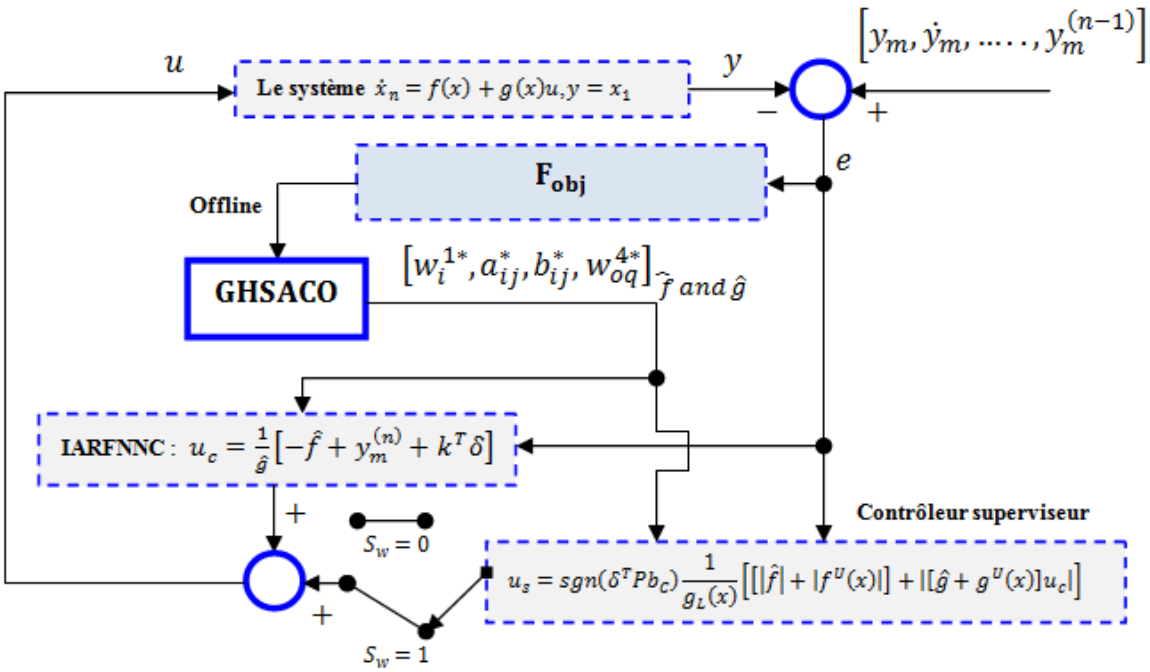

Figure 5.1 : Structure de commande globale proposée.

5.3 Simulations

Afin de prouver l'efficacité de la méthode de commande proposée, nous l'avons appliquée à un problème de stabilisation d'un pendule inversé [Lo 98]. Le système pendule inversé est composé comme indique la figure 5.2 d'un chariot mobile en translation sur un axe horizontal et

d'un pendule libre fixé verticalement sur le chariot. En exerçant une force horizontale $u(t)$ sur le chariot, il résulte une rotation de $\theta(t)$ radian du pendule. La commande de ce système doit réaliser la régulation de l'angle $\theta(t)$, en maintenant la position $\theta(t)$ à un angle de référence θ_d, pour des conditions initiales quelconques.

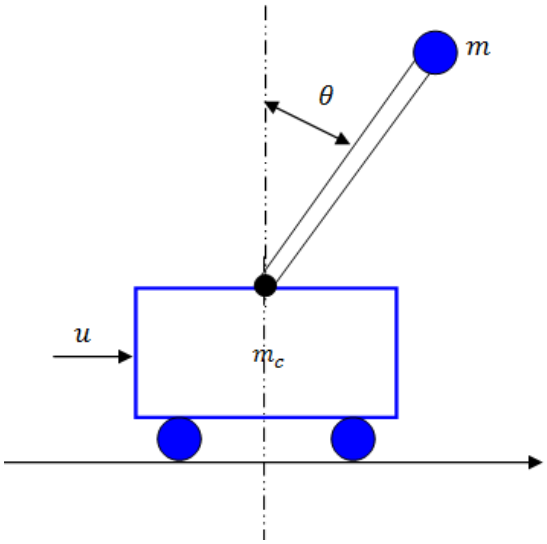

Figure 5.2 : Système simple pendule inversé.

Le modèle dynamique non linéaire du système pendule inversé est donné par :

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{gsin(x_1) - \frac{mlx_2^2 cos(x_1) sin(x_1)}{m_c+m}}{l\left(\frac{4}{3} - \frac{mcos^2(x_1)}{m_c+m}\right)} + \frac{\frac{cos(x_1)}{m_c+m}}{l\left(\frac{4}{3} - \frac{mcos^2(x_1)}{m_c+m}\right)} u = f(x_1, x_2) + g(x_1, x_2)u \qquad (5.17)$$

D'où $x_1 = \theta(t)$ et $x_2 = \dot{\theta}(t)$ sont respectivement, la position angulaire et la vitesse angulaire du pendule. Les paramètres de ce modèle sont choisis comme suit ;
$g = 9.8\ m/s^2$,$m_c = 1\ kg$, $m = 0.1\ kg$, and $l = 0.5\ m$.

L'objectif de commande ici, est de stabiliser le pendule dans une position $x_1 \in (-\pi/6\,, \pi/6)$. Les deux RFNN$_{\hat{f}}$ et RFNN$_{\hat{g}}$ utilisés pour approximer respectivement les fonctions $\hat{f}$ et $\hat{g}$, ont la même configuration. Ainsi, chaque réseau comporte deux entrées $n = 2$, qui sont respectivement x_1 et x_2. Chaque entrée est associée à $m = 5$ nœuds dans la couche de fuzzification. En outre, la couche de règles floues comprend $l = 5$ nœuds. Enfin, la couche de défuzzification est composée d'un seul nœud $o = 1$.

Les valeurs initiales des paramètres à régler $\left[w_i^1, a_{ij}, b_{ij}, w_{oq}^4\right]_{\hat{f}\ and\ \hat{g}}$, sont choisies aléatoirement dans des intervalles spécifiques.

Nous avons imposé au système un angle de référence θ_d égal à $0.1sin(t)$. Les états initiaux à partir desquels le système débute ses mouvements sont $x(0) = [\pi/6\,, 0]$.
Les valeurs des f^U, g^U, and g_L sont déterminées comme suit ;

$$|f(x_1,x_2)| = \left|\frac{gsin(x_1) - \frac{mlx_2^2 cos(x_1)sin(x_1)}{m_c+m}}{l\left(\frac{4}{3} - \frac{mcos^2(x_1)}{m_c+m}\right)}\right| \leq \frac{9.8+\frac{0.025}{1.1}x_2^2}{\frac{2}{3}-\frac{0.05}{1.1}} = 15.946 + 0.036970x_2^2 = f^U(x_1,x_2)$$

$$|g(x_1,x_2)| = \left|\frac{\frac{cos(x_1)}{m_c+m}}{l\left(\frac{4}{3} - \frac{mcos^2(x_1)}{m_c+m}\right)}\right| \leq \frac{1}{1.1\left(\frac{2}{3}-\frac{0.01}{1.1}\right)} = 1.4792 = g^U(x_1,x_2)$$

Puisque nous cherchons à obtenir une position $|x_1| \leq \pi/6$, alors, $g(x_1,x_2)$ doit vérifier ;

$$|g(x_1,x_2)| \geq \frac{cos(\pi/6)}{1.1\left(\frac{2}{3}-\frac{0.05cos^2(\pi/6)}{1.1}\right)} = 1.12 = g_L(x_1,x_2)$$

Les paramètres $k = [k_1\ \ k_2]$, Q et p_c sont choisis égales à $k = [7\ \ 3.5]$, $Q = diag(10{,}10)$ et $p_c = 0.01$. La résolution de l'équation (5.8) permet de calculer la valeur de la matrice P, on obtient après résolution $P = \begin{bmatrix} 15 & 5 \\ 5 & 5 \end{bmatrix}$.

De plus, et afin de mettre en évidence les performances de l'algorithme GHSACO, par rapport à d'autres algorithmes de la même famille, à savoir ; le HS, IHS, GHS, PSO et GAs, nous avons utilisé chacun de ces algorithmes dans le problème traité en vue d'une comparaison. Ainsi, le paramétrage de ces algorithmes est effectué comme suit ;

Le paramètre HMS est choisi égal à 100 pour tous les algorithmes (le GHSACO, le HS est ses variantes le IHS et GHS). Les paramètres de HS sont fixés de la même manière que ceux dans [Omra 08], tels que HMCR $= 0.9$, PAR $= 0.3$, and $bw = 0.01$.

Après plusieurs essaies, les paramètres de IHS sont définis comme suit : HMCR $= 0.95$, $\text{PAR}_{min} = 0.35$, PAR_{max}= 0.99, $bw_{min} = 1e-8$, $bw_{max} = (UB_i - LB_i)/20$, avec ces paramètre le IHS a permet d'obtenir de bons résultats. Pour l'algorithme C-ACO, ses paramètres q_0, ρ et Q ont respectivement les valeurs 0.75, 0.1 et 1.

En outre, toutes les pistes de phéromone dans le graphe (HM dans le second cas) ont été initialisées avec les mêmes concentrations $\tau_{ij}(0)$, et $\tau_{ij} = \tau_{ij}(0) = 1 \times ones(n, \text{HMS})$.
Pour le GHS, soit dans le cas, lorsque il est hybridé avec C-ACO ou dans le cas où il travail seul, nous avons utilisé les valeurs suivantes : HMCR $= 0.9$, $\text{PAR}_{min} = 0.35$, and $\text{PAR}_{max} = 0.99$.

Les coefficients de l'algorithme PSO, ont été choisis comme suit ; les constantes $c_1 = 1$, $c_2 = 0.1$, $w = 0.61$, la taille de l'essaim "nombre de particules" est choisi égal à 100. Concernant

les GAs, ils sont utilisés avec un paramétrage semblable à celui dans [Allo 13]. Une étude comparative entre les six cas de commandes adaptatives synthétisées (chaque cas est relatif à un algorithme d'optimisation), a été faite en se basant sur les deux critères de performance J_1 and J_2 données par (5.18) et (5.19), avec p_d étant le nombre des échantillons du signal mesuré.

$$J_1 = \sum_{i=1}^{P_d} e_i \tag{5.18}$$

$$J_2 = \sum_{i=1}^{P_d} u_i \tag{5.19}$$

Les résultats de simulation pour chaque critère sont illustrés dans le tableau 5.1.
Les six commandes adaptatives synthétisées, ont été testées 20 fois. De plus, chaque algorithme d'optimisation est simulé avec un nombre maximal d'itération NI égal à1000.

Le tableau 5.1 porte aussi, les valeurs moyennes, mauvaises et meilleures des optimums globaux et l'écart type (*std*.) générés par les six algorithmes d'optimisation. Les meilleures solutions obtenues sont indiquées en gras.
De plus, les courbes d'évolution pour la fonction objectif F_{obj}, relatives au meilleur test (parmi les 20) pour les six algorithmes, sont représentées dans la figure 5. 6.

Les réponses du système simple pendule inversé ($x_1 = \theta(t)$ et $x_2 = \dot{\theta}(t)$), obtenues en appliquant les six commandes adaptatives, basées respectivement sur les algorithmes HS, IHS, GHS, GHSACO, PSO et GAs, sont illustrées respectivement dans les figure 5. 3 et figure 5. 4 de (a) à (f). Ainsi, les signaux de commande générés $u(t)$, sont aussi illustrés dans la figure 5. 5.

	Meilleure	**Moyenne**	**Mauvaise**	***std*.**	**J_1**	**J_2**
GHSACO-IARFNNC	**0.005740256**	**0.031759901**	**0.054135168**	**0.010596189**	**0.007463702**	1.645989112E+004
GHS-IARFNNC	0.017055825	0.036200746	0.062043317	0.010941248	0.023198614	**8.064244674E+003**
IHS-IARFNNC	0.046208065	0.079498893	0.100079648	0.010976485	0.037812696	9.728294530E+005
HS-IARFNNC	0.079174790	0.109473446	0.138840162	0.016998117	0.062444252	5.073870747E+005
PSO-IARFNNC	0.032414590	0.047225338	0.071863350	0.021482143	0.041371294	4.661866750E+005
GA-IARFNNC	0.056386666	0.060554015	0.064629408	0.026178405	0.049687775	3.475512064E+005

Tableau 5.1 : Comparaison entre GHSACO, HS, IHS, GHS, PSO et GAs.

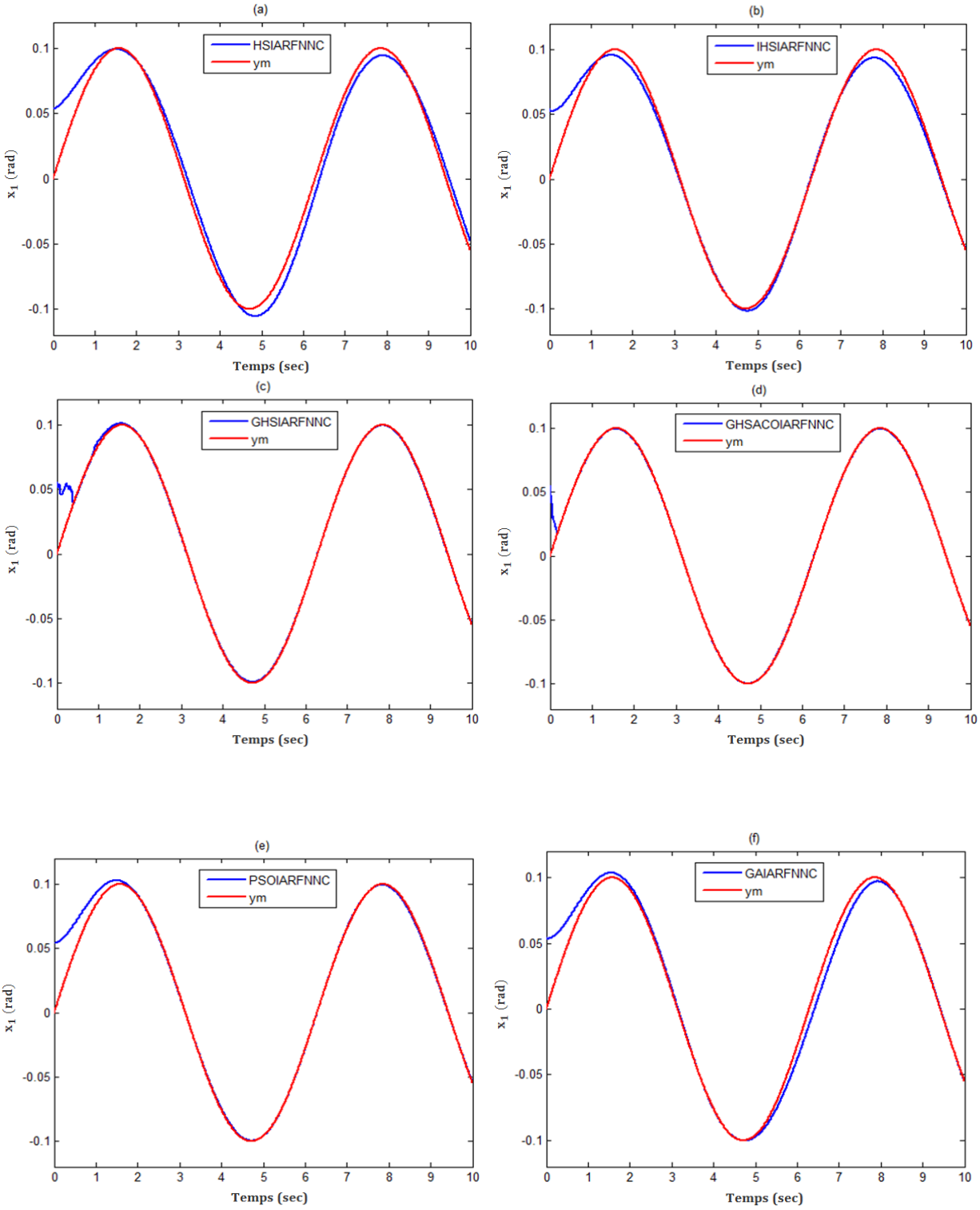

Figure 5.3 : Réponses du système (positions angulaire du pendule) $(y(t) = x_1)$ en bleu et trajectoire désirée y_m en rouge, (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs.

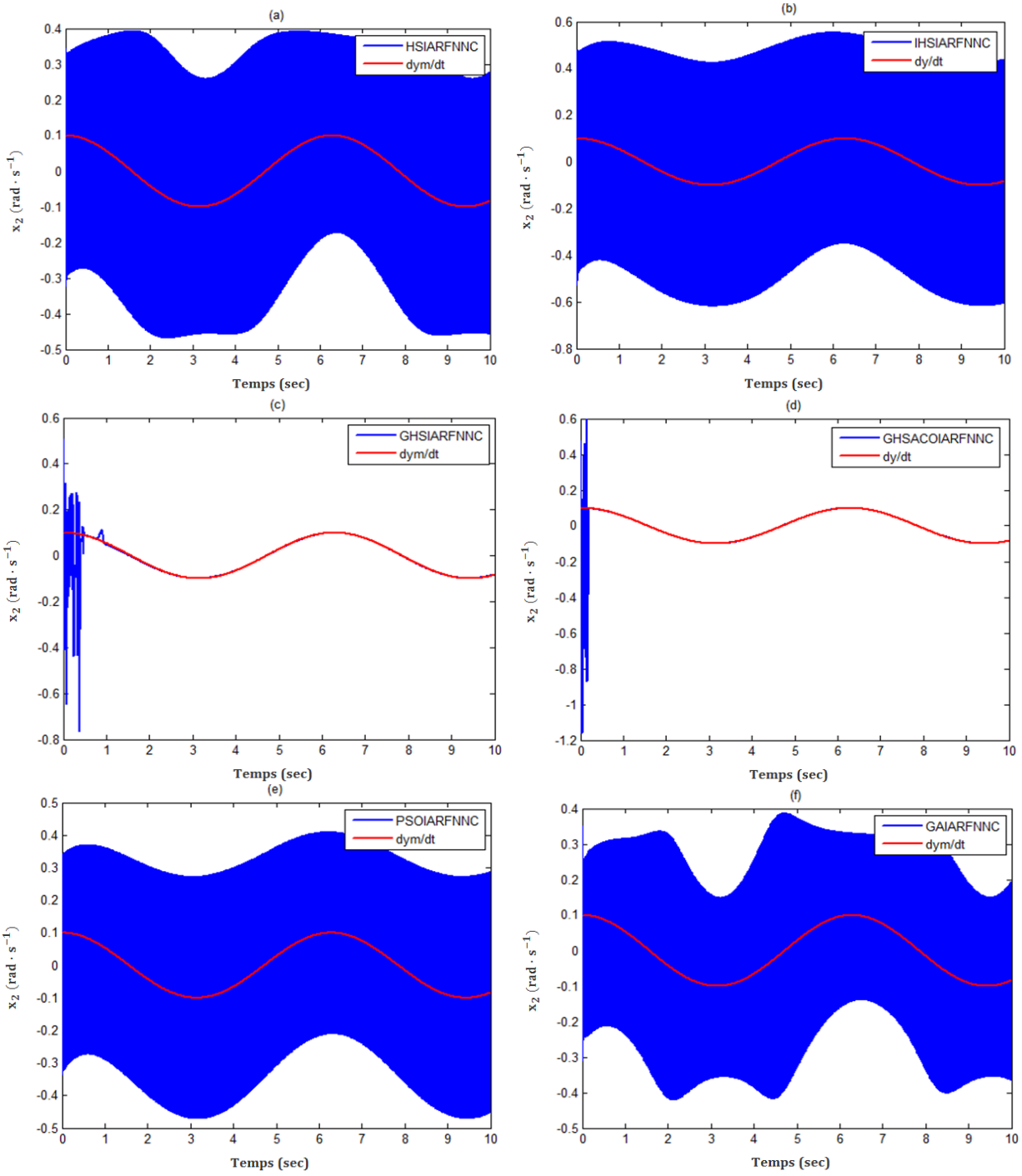

Figure 5.4 : Réponses du système (vitesse angulaire du pendule) x_2 en bleu et trajectoire désirée $\dot{y}_m$ en rouge, (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs.

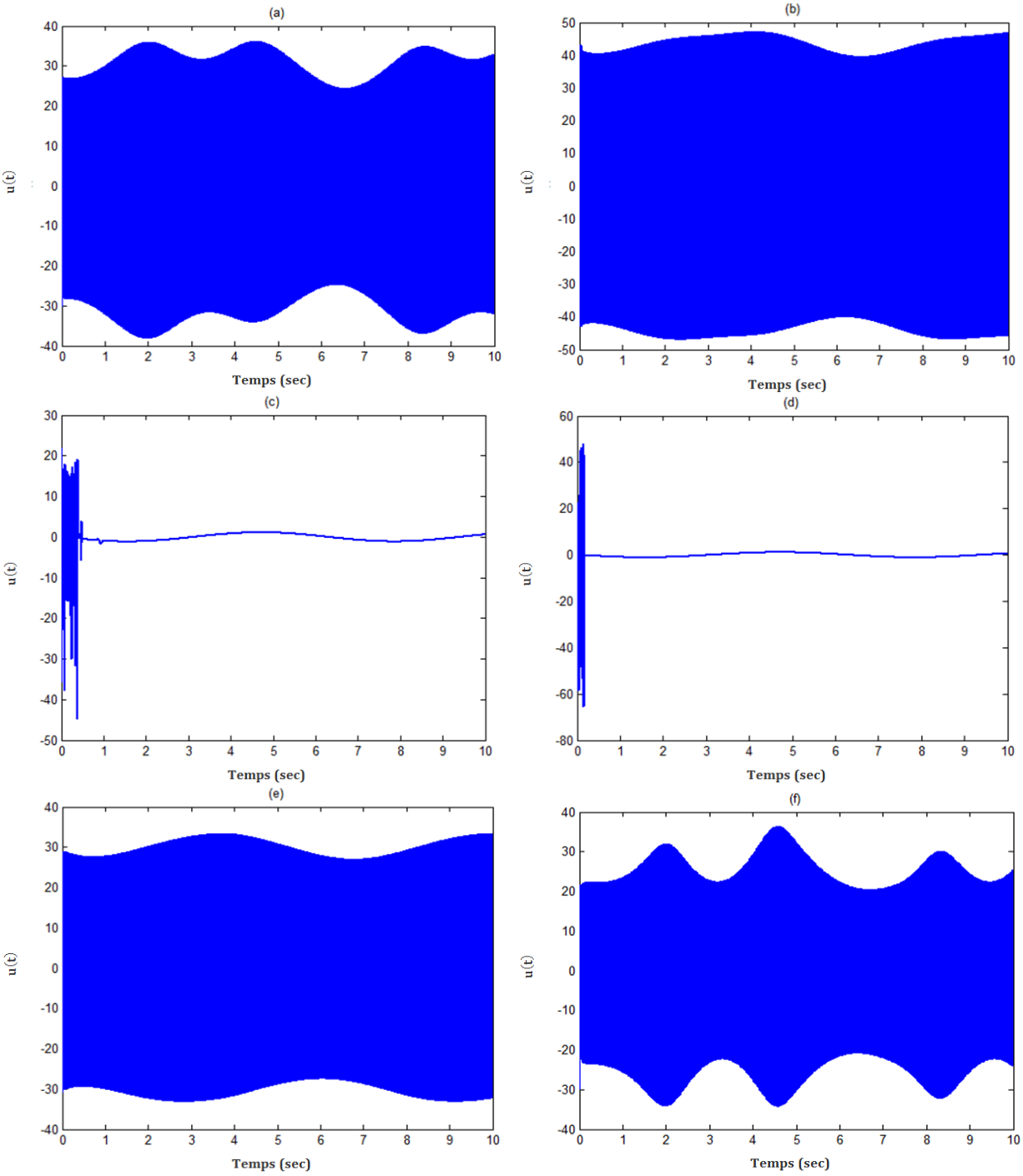

Figure 5.5 : Signaux de commandes générés relatifs aux commandes adaptatives synthétisées, basées respectivement sur les algorithmes (a)HS, (b)IHS, (c) GHS, (d) GHSACO, (e) PSO et (f) GAs.

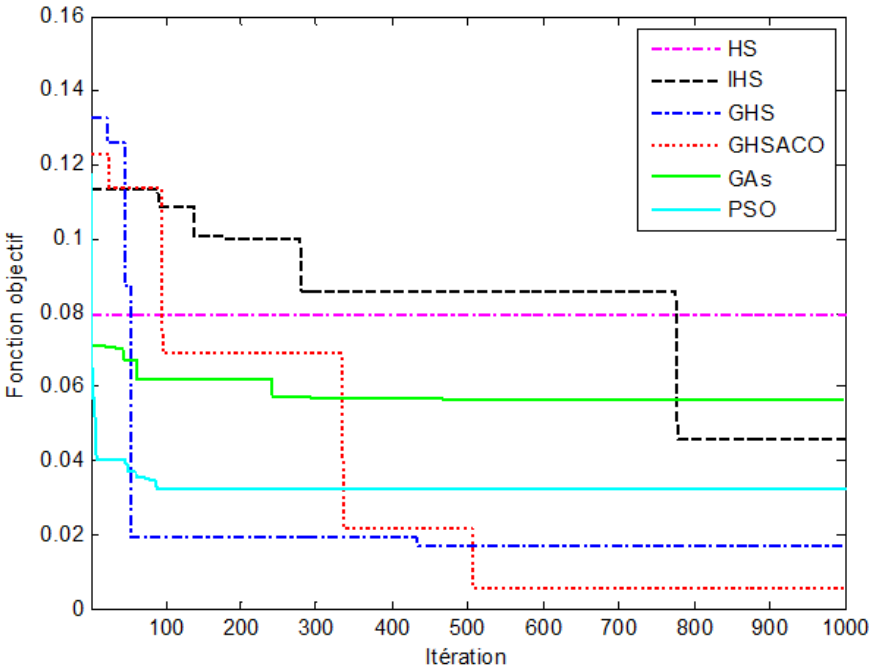

Figure 5.6 : Evolution de la fonction objectif F_{obj} en fonction des itérations, pour les algorithmes,GHSACO, HS, IHS, GHS, PSO et GAs respectivement.

Comme le montrent les figures 5.3 (a) à (f), toutes les commandes adaptatives, basées sur les six algorithmes d'optimisation, peuvent conduire le pendule du système à suivre la trajectoire désirée, mais avec des différentes précisions. Par ailleurs, les résultats illustrés dans le tableau 5.1, ainsi que les courbes de réponse du système (la position de pendule $x_1 = \theta(t)$ et sa vitesse angulaire $x_2 = \dot{\theta}(t)$) présentées respectivement, dans les figures 5.3 et 5.4 de (a) à (f), montrent clairement la supériorité de la commande adaptative synthétisée à base de l'algorithme GHSACO par rapport aux autres commandes en terme de performances de suivi de trajectoires.

Il apparaît aussi, d'après ces figures que la commande adaptative synthétisée à base de l'algorithme GHS, permet d'obtenir de bons résultats, aussi bien pour la position $x_1 = \theta(t)$ que pour la vitesse angulaire $x_2 = \dot{\theta}(t)$.

Ainsi, la figure 5. 5 (d), montre que le signal de commande généré, ne présente pas d'oscillations à haute fréquence (phénomène de chattering), il apparaît que ces oscillations ont disparu après les 0.2 secondes de simulation, sans aucune dégradation dans les performances de la commande. Cependant, ce phénomène indésirable persiste dans tous les autres signaux de commandes (figures 5. 5 (a, b, e, f)), à part le cas de la commande adaptative basée sur l'algorithme GHS, dont ces oscillations ont aussi disparu (figure 5. 5 (c)).

De plus, les courbes d'évolution de la fonction de performance F_{obj} en fonction des itérations successives, relatives aux six algorithmes d'optimisation, illustrées dans la figure 5. 6, montrent que le GHSACO dépasse tous les autres algorithmes en terme de qualité des solutions obtenues et de vitesse de convergence.

5.4 Conclusions

Dans ce chapitre, nous avons appliqué l'algorithme GHSACO, proposé dans le chapitre 3 dans le problème d'apprentissage de RFNNs dans une structure de commande adaptative indirecte. En effet, l'aspect indirect de la commande adaptative présentée, implique comme nous l'avons signalé, une connaissance des fonctions caractérisant la dynamique du système à commander. Par ailleurs, les RFNNs ont montrés à travers un grand nombre d'études [Lee 00, Lin 04, Wang 04], sont efficacité dans l'estimation des fonctions inconnues. Cependant, leur apprentissage par algorithme de type BPA ainsi que ses variantes [Werb 90, Guia 98], pose souvent des problèmes, tels que ; leur convergence vers des optimums locaux, leur vitesse de convergence très lente ainsi que la complexité de leur calcul généralement liée à la structure du réseau entrainé [Song 07, Lee 09]. Ainsi, et afin de remédier aux inconvenants suscités en bénéficiant des capacités de l'algorithme GHSACO proposé, nous l'avons utilisé comme un outil d'apprentissage des RFNNs utilisés dans la structure de commande adaptive ainsi synthétisée.

La méthode de commande proposée, a été testée par simulation sur le modèle d'un pendule inversé pour des problèmes de poursuite de trajectoires. Par ailleurs et afin de mettre en évidence l'efficacité de l'algorithme d'optimisation proposé nous avons comparé les résultats obtenus par ce dernier, et ceux obtenus, par les algorithmes HS, IHS, GHS, PSO et les GAs.

À travers les différents résultats de simulation obtenus, nous pouvons conclure que l'algorithme GHSACO développé à montré son efficacité comme bon outil d'apprentissage des RFNNs.

Conclusion générale

Les travaux de recherche présentés dans cette thèse concernent le développement et l'application de nouvelles méthodes d'optimisation de type combinatoires mono-objectifs s'appuyant sur les algorithmes métaheuristiques. Nous avons focalisé nos recherches sur ce type d'algorithmes destinés à résoudre des problèmes d'optimisation difficiles.

Nous avons présenté, dans un premier temps, un état de l'art sur les métaheuristiques à population de solutions dédiées aux problèmes d'optimisation combinatoires mono-objectifs. Les deux grandes familles de ces méthodes ainsi que leurs algorithmes, les plus cités dans la littérature, sont présentés.

Parmi les métaheuristiques illustrées, un intérêt particulier a été donné aux méthodes d'optimisation par colonies de fourmis (ACO) et recherche d'harmonie (HS). Les versions primitives de ces algorithmes, présentent comme toutes les méthodes appartenant à cette classe, des insuffisances majeures. Entre autres, nous citons le problème de convergence prématurée, qui peut conduire ces algorithmes à stagner dans des optimums locaux ainsi que leur vitesse de convergence relativement lente.

Des contributions pour l'amélioration de ces deux méthodes ont été proposées dans les domaines considérés. En premier lieu, nous avons proposé un nouvel algorithme, relatif aux ACO, afin d'atteindre les deux objectifs suivants : (1) le renforcement de l'intensification de l'algorithme C-ACO et l'évitement de la convergence prématurée vers des régions sous-optimales (obtenue par une ou plusieurs fourmis défaillantes): (i) l'introduction d'une modification au niveau de la règle aléatoire de transition (celle utilisée dans la variante ACS), (ii) l'emploi d'un mécanisme sélectif lors de la mise-à-jour des pistes de phéromone. (2) l'accélération de convergence de l'algorithme par l'ajout d'un processus de stockage de bonnes solutions qui vient juste après l'opération de mise-à-jour des pistes de phéromone.

Cet algorithme a montré ses performances, en terme de vitesse de convergence et qualité de la solutions. En effet, il permet d'obtenir des résultats qui dépassent de manière très significative ceux obtenus par C-ACO lors de leur application à un problème d'ajustement des paramètres d'une commande décentralisée par modes glissants.

L'idée d'hybridations de plusieurs métaheuristiques nous a semblé intéressante dans la mesure où on peut tirer profil de l'avantage de chacune. Nous avons, donc, choisi d'hybrider une version de l'algorithme HS appelé algorithme de recherche d'harmonie globale (*Global Best Harmony Search* GHS) et l'algorithme C-ACO. Cette hybridation est implémentée dans un nouvel

algorithme, dénommé GHSACO. Ce dernier, est basé sur: (i) une mémoire d'harmonie HM différente en structure et en conception, (ii) une règle supplémentaire de considération de la mémoire (*memory consideration selection rule* MCSR) basée sur l'utilisation d'une règle aléatoire de transition avec l'intégration d'un mécanisme de mise-à-jour des pistes de phéromone, (iii) une procédure de commutation aléatoire entre le GHS et le C-ACO. À partir des analyses et des expérimentations faites sur de nombreuses fonctions de benchmark mono-objectifs, nous avons conclu que les améliorations introduites par l'hybridation des deux algorithmes ont pu limiter le phénomène de convergence prématurée de l'algorithme HS. Elles ont également amélioré les performances de HS et sa variante GHS d'une manière significative pour la plupart des fonctions de test.

Après avoir présenté la variante ACO proposée et validé l'algorithme GHSACO en utilisant des fonctions de benchmark analytiques, nous les avons appliqués à des problèmes d'ajustement des paramètres de lois de commande. Le développement des techniques proposées a donné lieu à deux types d'applications, présentées dans les chapitres 4 et 5.

La première application, concerne le développement d'une commande décentralisée par modes glissants basée sur une approche coopérative, sert à exploiter les capacités des systèmes flous (Fuzzy systems FSs) ainsi que les RFNN*s*, afin de synthétisée des lois de commande locales par modes glissants, simples et permettent de générées des signaux de commande sans broutement (chattering).

L'algorithme ACO proposé a été utilisé pour optimiser plusieurs paramètres dans chaque unité de commande locale. Plus L'objectif est d'augmenter les capacités d'approximation des RFNN*s* et les FSs, ainsi que d'amélioration de la vitesse de convergence sur les surfaces de glissement locales. Nous avons montré à travers les résultats obtenus que les objectifs visés par l'application de cette méthode de commande soient atteint.

Nous avons également abordé le problème de l'apprentissage des RFNNs dans une structure de commande adaptative indirecte, basée sur un contrôleur superviseur, d'une classe de systèmes non-linéaires. Dans cette application, nous constatons l'efficacité de l'algorithme GHSACO.

En perspective, une analyse expérimentale plus poussée reste à faire. Les paramètres des algorithmes proposés peuvent également être auto-adaptés, afin de faciliter leur utilisation. De plus, les algorithmes proposés dans cette thèse sont mono-objectifs ne pouvant optimiser qu'une seule fonction objectif à la fois. Néanmoins, de nombreux problèmes de calcul des paramètres dans certaines structures de commande nécessitent l'optimisation de plusieurs fonctions objectifs simultanément.

Références bibliographiques

[Aart 97] E. H. L. Aarts, and J. K. Lenstra. "Local Search in Combinatorial Optimization". John Wiley & Sons, Inc., 1997.

[Adle 66] J Adler, "Chemotaxis in bacteria," Science, vol. 153, pp. 708–716, 1966.

[Ali 05] M.M. Ali, C. Khompatraporn, and Z.B. Zabinsky. ''A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems''. Journal of Global Optimization, Vol. 25, No. 4, pp. 635–672, 2005.

[Allo 07] **F. Allouani**, D. Boukhetala, F. Farah, F. Hachid, and M. Tadjine. "Commande par mode de glissement d'un simulateur de vol l'hélicoptère," 5éme Conférence nationale en génie électrique, Ecole Militaire Polytechnique, Alger, Algérie, 16-17 Avril, 2007.

[Allo 12a] **F. Allouani**, D. Boukhetala, and F. Boudjema. "Decentralised sliding mode controller design based on hybrid approach for interconnected uncertain non-linear systems". International Journal Instrumentation Technology, Vol. 1, No. 2, pp. 155–187, 2012.

[Allo 12b] **F. Allouani**, D. Boukhetala, and F. Boudjema. "Comparison between Ant Colony and Genetic Algorithms for Neuro-fuzzy-sliding Mode Controller Optimization". in Proc. 4th International Conference on Electrical Engineering, USTHB, Algiers, Algeria, 07-09 May 2012.

[Allo 12c] **F. Allouani**, D. Boukhetala, and F. Boudjema. " A PSO Based decentralized Neuro-Fuzzy-Sliding Mode Controller for the Twin Rotor MIMO system," Journal of Automation & Systems Engineering, Vol. 6, No. 2, pp. 133-148, 2012.

[Allo 12d] **F. Allouani**, D. Boukhetala, and F. Boudjema. "Particle swarm optimization Based Fuzzy Sliding Mode Controller for the Twin Rotor MIMO system". In Proceedings of the 16th IEEE Mediterranean Electrotechnical Conference, Medina, Yasmine Hammamet, Tunisia, 25-28 March 2012, pp. 1063-1066.

[Allo 12e] **F. Allouani**, D. Boukhetala, F. Boudjema and K. Zenger "A Hybrid Controller for the Twin Rotor MIMO system Based on a Bacterial Optimization algorithm". In Proceedings of the First International Conference on Electrical Engineering and Control Applications ICEECA'12 Khenchela, Algeria, 20-22 November 2012.

[Allo 13] **F. Allouani**, D. Boukhetala, and F. Boudjema. "Decentralized Sliding Mode Controller Based on Genetic algorithm and a Hybrid approach for Interconnected Uncertain Nonlinear Systems ". International Journal of Control and Automation, Vol. 6, No. 1, pp. 61–86, 2013.

[Allo 15a] **F. Allouani**, D. Boukhetala, and F. Boudjema, Z. Kai, and G. Xiao-Zhi. "A Novel Global Harmony Search Method based on Ant Colony Optimization Algorithm". Journal of Experimental and Theoretical Artificial Intelligence, Paper accepted for publication in 07-01-2015.

[Allo 15b] **F. Allouani**, D. Boukhetala, and F. Boudjema, and G. Xiao-Zhi. "A Novel Global Harmony Search Method based tuning of RFNN for adaptive control of uncertain nonlinear systems". International Journal of Intelligent Computing and Cybernetics, Vol. 08, No. 1, 2015.

[Amin 13] F. Amini, and P. Ghaderi. "Hybridization of Harmony Search and Ant Colony Optimization for optimal locating of structural dampers." Applied Soft Computing, Vol. 13, No. 5, pp. 1–9, 2013.

[Bäck 00] T. Bäck, D.F. Fogel, and Z. Michalewicz. "Evolutionary Computation 1 Basic Algorithms and Operators". Institute of Physics Publishing, PA, USA, 2000.

[Bäck 97a] T. Bäck, U. Hammel, and F. P. Schweffel. "Evolutionary Computation : Comments on the history and current state". IEEE transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 3-17,1997.

[Bäck 97b] T. Bäck, D.F. Fogel, and Z. Michalewicz. "Handbook of Evolutionary Computation". IOP Publishing Ltd and Oxford University Press, PA, USA, 1997.

[Baoj 07] Z. Baojiang, L. Shiyong. "Ant colony optimization algorithm and its application to Neuro-Fuzzy controller design". Journal of Systems Engineering and Electronics, Vol. 18, No. 3, pp. 603–610, 2007.

[Beck 89] R. Beckers, J. L. Deneubourg, and S. Goss. "Trails and UTurns in the Selection of a Path by the Ant Lasius Niger". Journal of Theoretical Biology, 159 :397–415, 1992.

[Bena 10] L. Benameur. " Contribution à l'optimisation complexe par des techniques de swarm intelligence". Thèse de Doctorat, Université Mohammed V – Agdal, Faculté des sciences, Rabat, Maroc, 2010.

[Bilch 95] G. Bilchev, I.C. Parmee. "The ant colony metaphor for searching continuous design spaces". Lecture Notes in Computer Science, Vol. 993, pp. 25–39. Springer, Berlin, 1995.

[Bona 99] E. Bonabeau. M. Dorigo, and G. Theraulaz. "Swarm intelligence : from natural to artificial systems". Oxford University Press, Inc., New York, NY, USA, 1999.

[Bouba 09] A. Boubakir, F. Boudjema, and S. Labiod. ''A Neuro-fuzzy-sliding Mode Controller Using Nonlinear Sliding Surface Applied to the Coupled Tanks System''. International Journal of Automation and Computing, Vol. 06, No. 1, pp. 72–80, 2009.

[Boube 09] H. Boubertakh, M. Tadjine, P.-Y. Glorennec, and S. Labiod. "Tuning Fuzzy PID Controllers using Ant Colony Optimization". IEEE 17th Mediterranean Conference on Control & Automation, Makedonia Palace, Thessaloniki, Greece, pp. 13–18, June 24 - 26, 2009.

[Bous 13a] I. Boussaïd, J. Lepagnot, and P. Siarry. "A survey on optimization metaheuristics". Information Sciences, Vol. 237, pp. 8–117, 2013.

[Bous 13b] I. Boussaïd. "Perfectionnement de métaheuristiques pour l'optimisation continue". Thèse de Doctorat, université des sciences et de la technologie Houari boumediene (USTHB), Algérie, 2013.

[Breg 10] V. Bregeault. "Quelques contributions à la théorie de la commande par modes glissants". Thèse de Doctorat, Ecole doctorale, sciences et technologies de l'information et des mathématiques, Ecole Centrale de Nantes, 2010.

[Bull 99] B. Bullnheimer, R. Hartl, and C. Strauss. "A New Rank-based Version of the Ant System: a Computational Study". Central European Journal for Operations Research and Economics, Vol. 7, No. 1, pp. 25–38, 1999.

[Burt 86] J. A. Burton, and A. S. I. Zinober. "Continuous approximation of variable structure control". International Journal of Systems Science, Vol. 17, No. 6, pp. 875–885, 1986.

[Cast 00] L. N. De Castro, and F. Von Zuben. "Artificial Immune Systems : Part II : A Survey

of Applications". Technical Report DCA-RT 02/00, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil, 2000.

[Cast 02] L. N. De Castro, and J. Timmis. "Artificial immune systems: a novel paradigm for pattern recognition". Artificial Neural Networks in Pattern Recognition, In: Corchado, Juan Manuel and Alonso, Luis and Fyfe, Colin, eds. Artificial Neural Networks in Pattern Recognition. University of Paisley, pp. 67-84, 2002.

[Cast 15] O. Castillo, H. Neyoy, J. Soria, P. Melin, and F. Valdez. "A new approach for dynamic fuzzy logic parameter tuning in Ant Colony Optimization and its application in fuzzy control of a mobile robot". Applied Soft Computing, Vol. 28, pp. 150–159, March 2015.

[Cast 99] L. N. De Castro, and F. Von Zuben. "Artificial Immune Systems : Part I : Basic Theory and Applications". Technical Report TR-DCA 01/99, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil, 1999.

[Chen 09] H. Chen, Y. Zhu, and K. Hu. "Cooperative Bacterial Foraging Optimization". Discrete Dynamics in Nature and Society, Vol. 2009, pp. 1–17, 2009.

[Chen 12] J. Chen, Q. K. Pan, L. Wang, and J. Q. Li. "A hybrid dynamic harmony search algorithm for identical parallel machines scheduling". Engineering Optimization, Vol. 44, No. 2,pp. 209–224, 2012.

[Chen 13] B. Chen, L. Chen and Y. Chen. "Efficient ant colony optimization for image feature selection". Signal Processing, Vol. 93, No. 6, pp. 1566–1576, June 2013.

[Chen 14] L. Chen, X. Huang, J. Tian, and X. Fu. "Blind noisy image quality evaluation using a deformable ant colony algorithm". Optics & Laser Technology, Vol. 57, pp. 265–270, April 2014.

[Cheng13] B. Chenga, Q. Wanga, S. Yanga, and X. Hua. "An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes". Applied Soft Computing, Vol. 13, pp. 765–772, 2013.

[Colo 91] A. Colorni, M. Dorigo, and V. Maniezzo. "Distributed Optimization by Ant Colonies". In : Proceedings of the First European Conference on Artificial Life, pp. 134–142, MIT Press, Paris, France, December 1991.

[Comb 98] W.E. Combs. "Combinatorial rule explosion eliminated by a fuzzy rule configuration". IEEE Transactions on Fuzzy Systems, Vol. 6, No. 1, pp. 1 – 11, 1998.

[Cong 13] Z. Cong, B. De Schutter, and R. Babuska. "On the convergence of Ant Colony Optimization with stench pheromone". IEEE Congress on Evolutionary Computation, pp. 1876–1883, Cancun, 20–23 June 2013.

[Cord 00] O. Cordon, I. D. Viana, F. Herrera, and L. Moreno. "A New ACO Model Integrating Evolutionaly Computation Concepts : the Best-Worst Ant System". In : Proceedings of the Second International Workshop on Ant Algorithms, IEEE Transaction on Evolutionary Computation, pp. 22–29, Brussels, Belgium, September 2000.

[Da 00] F. Da.''Decentralized Sliding Mode Adaptive Controller Design Based on Fuzzy Neural Networks for Interconnected Uncertain Non-linear Systems''. IEEE Transactions on Neural Networks, Vol. 11, No. 6, pp. 1471–1479, 2000.

[Darw1859] C. Darwin. "On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life". J. Murray, June 1859.

[Dege 12] S.O. Degertekin. "Improved harmony search algorithms for sizing optimization of

truss structures“. Computers & Structures, Vol. 92–93, pp. 229–241, February 2012.

[Dene 90] J. L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels. “The self-organizing exploratory pattern of the argentine ant“. Journal of Insect Behavior, Vol. 3, No. 2, pp. 159–168, March 1990.

[Ding 12] Q. Ding, X. Hu, L. Sun, and Y. Wang. “An improved ant colony optimization and its application to vehicle routing problem with time windows“. Neurocomputing, Vol. 98, pp. 101–107, 2012.

[Dori 03] M. Dorigo, and T. Stützle. “Handbook of Metaheuristics“. Vol. 57 of International series in operations research and management science, chapter The Ant Colony Optimization Metaheuristics : Algorithms, Applications and Advances. Kluwer Academic Publishers, Boston Hardbound, 2003.

[Dori 05] M. Dorigo, and C. Blum. “Ant colony optimization theory: A survey“. Theoretical Computer Science, Vol. 344, No. 2-3, pp. 243–278, November, 2005.

[Dori 91] M. Dorigo, V. Maniezzo, and A. Colorni. “Positive feedback as a search strategy“. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[Dori 92] M. Dorigo. “Optimization, Learning and Natural Algorithms“. PhD thesis, Politecnico di Milano, Italy, 1992.

[Dori 96] M. Dorigo, V. Maniezzo, and A. Colorni. “Ant system: optimization by a colony of cooperating agents“. IEEE Transactions on Systems, Man, and Cybernetics, Part B, Vol. 26 No. 1, pp. 29–41, 1996.

[Dori 97a] M. Dorigo and L. Gambardella. “Ant Colony System : a Cooperative Learning Approach to the Travelling Salesman Problem”. IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53–66, 1997.

[Dori 97b] M. Dorigo, and L. M. Gambardella. “Ant Colonies for the Traveling Salesman Problem“. BioSystems, Vol. 43, pp. 73–81. 1997.

[Dréo 03] J. Dréo, A. Pétrowski, P. Siarry, and É. Taillard. “Métaheuristiques pour l'optimisation difficile“. Éditions Eyrolles, 2003.

[Dréo 04] J. Dréo. “Adaptation de la méthode des colonies de fourmis pour l'optimisation en variables continues. Application en génie biomédical“. Thèse de doctorat en sciences, Spécialité : Génie Biologique et Médical, Option : Optimisation, Université Paris 12, Val de Marne, UFR de Sciences, 2004.

[Duan 09] H. b. Duan, X. Y. Zhang, J. Wu, and G. J. Ma. “Max-Min Adaptive Ant Colony Optimization Approach to Multi-UAVs Coordinated Trajectory Replanning in Dynamic and Uncertain Environments“. Journal of Bionic Engineering, Vol. 6, No. 2, pp. 161–173, June 2009.

[Durh 94] W. H. Durham. “Co-evolution : Genes, Culture, and Human Diversity“. Stanford University Press, Stanford, California, 1994.

[Eibe 04] A.E. Eiben, and J.E. Smith, "Introduction to Evolutionary Computing", Assembly Automation, Vol. 24, No. 3, pp. 324–324, 2004.

[Enge 10] A. P. Engelbrecht. “Computational Intelligence; An Introduction“, Second Edition, John Wiley, University of Pretoria, South Africa, 2007.

[Esta 14] M. J. Estahbanati. “Hybrid probabilistic-harmony search algorithm methodology in generation scheduling problem“. Journal of Experimental & Theoretical Artificial .Intelligence, Vol. 26, No. 2,pp. 283–296, 2014.

[Feed 98] Feedback Instruments Ltd. ‘’Twin Rotor MIMO System‘’. 33-220 User Manual,

1998.

[Feo 89] T. A. Feo, and M. G. C. Resende. “A probabilistic heuristic for a computationally difficult set covering problem“. Operations Research Letters, Vol. 8, No. 2, pp. 67–71, 1989.

[Feo 95] T. A. Feo, and M. G. C. Resende. “Greedy randomized adaptive search procedures“. Journal of Global Optimization, Vol. 6 No. 2, pp. 109–133, 1995.

[Fili 88] A. F. Filippov. “Differential Equations with Discontinuous Righthand Sides“. Kluwer, 1988.

[Foge 62] L. J. Fogel. “Autonomous automata”. Industrial Research Magazine, Vol. 4, No. 2, pp. 14–19, 1962.

[Foge 66] L. J. Fogel, A. J. Owens, and M. J.Walsh. “Artificial Intelligence through Simulated Evolution“. John Wiley & sons, first edition, 1966.

[Fors 07] M. Forster, B. Bickel, B. Hardung, and G. Kokai. “Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, pp 1991–1998, 2007.

[Gao 10] X. Z. Gao, T. Jokinen, X. Wang, S.J. Ovaska, and A. Arkkio. “A New Harmony Search method in optimal wind generator design“. IEEE International Conference on Electrical Machines, pp. 1–6,2010.

[Geem 01] Z. W. Geem, J. H. Kim, and G. V. Loganathan. “A new heuristic optimization algorithm : harmony search“. Simulation, Vol. 76, No. 2, pp. 60–68, 2001.

[Geem 05] Z. W. Geem, K. S. Lee, and Y. Park. “Application of harmony search to vehicle routing“. American Journal of Applied Sciences, Vol. 2, No. 12, pp. 1552–1557, 2005.

[Geem 06] Z.W. Geem. “Optimal cost design of water distribution networks using harmony search“. Engineering Optimization, Vol. 38, No. 3, pp. 259–280, 2006.

[Geem 08] Z. W. Geem. “Harmony Search Applications in Industry“. Soft Computing Applications in Industry, Studies in Fuzziness and Soft Computing, Vol. 226, pp. 117-134, 2008.

[Gen 00] M. Gen, and R. Cheng. “Genetic Algorithms and engineering optimization“. Wiley inter-science, 2000.

[Glov 86] F. Glover. “Future paths for integer programming and links to artificial intelligence“. Computers and Operations Research, Vol. 13, No. 5, pp. 533–549, 1986.

[Goss 89] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. “Self-organized shortcuts in the Argentine ant”. Naturwissenschaften, Vol. 76, No. 12, pp. 579–581, 1989.

[Guia 98] Z. Guian, and S. Jennie. “Advanced neural network training algorithm with reduced complexity based on Jacobian deficiency”. IEEE Transactions on Neural Networks, Vol. 9, No. 3, pp. 448–453, 1998.

[Gutj 00] W. J. Gutjahr. “A Graph-based Ant System and its convergence“. Future Generation Computer Systems, Vol. 16, No. 8, pp. 873–888, 2000.

[Herr 03] F. Herrera, M. Lozano, and A.M. Sanchez. “A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study“. International Journal of Intelligent Systems, Vol. 18, pp. 309–338, 2003.

[Holl 75] J. H. Holland. “Adaptation in Natural and Artificial Systems“. The University of Michigan Press, 1975.

[Ioan 01] P. A. Ioannou, and J. Sun, ’’Robust Adaptive Control’’. Prentice-Hall, Englewood

Cliffs, NJ, 1996.

[Isid 95] A. Isidori. "Nonlinear Control Systems". Springer-Verlag, London, 1995.

[Jeba 09] K. Jebari, A. Bouroumi, and A. Ettouhami. "An experimental study of parent selection operators for genetic algorithms". In Proceedings of International Conference on Software, Knowledge Information Management and Applications, pp. 56–61, Fez, Morocco, 2009.

[Jing 11] X. Jing, and L. LiangPing. "A hybrid ant colony optimization for continuous domains". Expert Systems with Applications, Vol. 38, No. 9, pp. 11072–11077, September 2011.

[Jova 11] R. Jovanovica, and M. Tubab. "An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem". Applied Soft Computing, Vol. 11, pp. 5360–5366, 2011.

[Juan 08a] J. G. Juang, M.T. Huang, and W.K. Liu. "PID control using presearched genetic algorithms for a MIMO system". IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, Vol. 38, No. 5, pp.716–727. 2008.

[Juan 08b] J. G. Juang, R. W. Lin, and W. K. Liu. "Comparison of classical control and intelligent control for a MIMO system".Applied Mathematics and Computation, Vol. 205, pp. 778–791, 2008.

[Kara 07] D. Karaboga, and B. Basturk. "A powerful and efficient algorithm for numerical function optimization: artificial bee colony algorithm".Journal of Global Optimization. Vol. 3, pp. 9459–9471, 2007.

[Kefa 15] M. Kefayat, A. Lashkar Ara, and S.A. Nabavi Niaki. "A hybrid of ant colony optimization and artificial bee colony algorithm for probabilistic optimal placement and sizing of distributed energy resources". Energy Conversion and Management, Vol. 92, pp. 149–161, 2015.

[Kenn 95] J. Kennedy. and R. C. Eberhart. "Particle Swarm Optimization". In : Proceedings of the IEEE International Conference on Neural Networks IV, pp. 1942–1948, Perth, Australia, November, 1995.

[Khal 14] M. S. Khalid, and A. F. Alex. "Classification with cluster-based Bayesian multi-nets using Ant Colony Optimisation". Swarm and Evolutionary Computation, Vol. 18, pp. 54–70, October 2014.

[Kim 95] S. W. Kim, and J. J. Lee. ''Design of a fuzzy controller with fuzzy sliding surface''. Fuzzy Sets and Systems, Vol. 71, No. 3, pp.359–367. 1995.

[Kirk 83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. "Optimization by simulated annealing". Science, Vol. 220, No. 4598, pp. 671–680, 1983.

[Koza 89] J. R. Koza. "Hierarchical genetic algorithms operating on populations of computer programs". pp. 768–774, Morgan Kaufmann, Detroit, Michigan, USA, 1989.

[Koza 90] J. R. Koza. "Genetic programming : A paradigm for genetically breeding populations of computer programs to solve problems". Technical Report, STANCS-90-1314, Stanford University, Department of Computer Science, 1990.

[Krem 01] S.C. Kremer, and J. F. Kolen. "A Field Guide to Dynamical Recurrent Networks". Wiley-Blackwell, New York, USA, 2001.

[Kudi 11] S. Kudikala. "Performance Study of Harmony Search Algorithm for Analog Circuit Sizing". IEEE International Symposium on Electronic System Design, Kochi, Kerala, pp. 12–17, 19–21 Dec 2011.

[Lam 00] H. K. Lam, F. H. F. Leung, and P. K. S. Tam. '' Stable and robust Fuzzy Control for Uncertain Non-linear systems''. IEEE Transactions on systems Man and

Cybernetics, Part A, Sys and Humans, Vol. 30, No. 6, pp. 825–840, 2000.

[Lamp 99] J. Lampinen, and I. Zelinka. "Mixed Integer-Discrete-Continuous Optimization By Differential Evolution - Part 1: the optimization method". In : Proceedings of 5th International Mendel Conference on Soft Computing, 1999.

[Lee 00] C. H. Lee, and C.C. Teng. ''Identification and control of dynamic systems using recurrent fuzzy neural networks''. IEEE Transactions on Fuzzy Systems, Vol. 8, No. 4, pp. 349–366. 2000.

[Lee 09] C. H. Lee and M. H. Chiu. "Recurrent neuro fuzzy control design for tracking of mobile robots via hybrid algorithm", Expert Systems with Applications, Vol. 36, pp. 8993–8999, 2009.

[Lin 04] F. J. Lin, C. H. Lin, and Huang, P. K. "Recurrent fuzzy neural network controller design using sliding-mode control for linear synchronous motor drive". IEE Proceedings Control Theory and Applications, Vol. 151, No. 4, pp. 407–416. 2004.

[Liu 04] J. Z. Liu, W. J. Zhao, L. J. Zhang. ''Design of Sliding Mode Controller Based on Fuzzy Logic''A. In Proceedings of the 3rd IEEE Conference on Machine Learning and Cybernetics, IEEE Press, Shanghai, pp. 616–619, 2004.

[Lo 98] J.C. Lo, and Y. H. Kuo. ''Decoupled fuzzy sliding-mode control''. IEEE transactions on Fuzzy Systems, Vol. 6, No. 3, pp. 426–435, 1998.

[Lobo 05] F. G. Lobo, and C. F. Lima. "A review of adaptive population sizing schemes in genetic algorithms". In Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation, pp. 228–234, 2005.

[Mahd 07] M. Mahdavi, M. Fesanghary, and E. Damangir. "An improved harmony search algorithm for solving optimization problems". Applied Mathematics and Computation. Vol. 188, pp. 1567–1579, 2007.

[Manj 13] D. Manjarres. I. Landa-Torres, S. Gil-Lopez, J. DelSer, M. N. Bilbao, S. Salcedo-Sanz, and Z.W.Geem. "A survey on applications of the harmony search algorithm". Engineering Applications of Artificial Intelligence, Vol. 26, pp. 1818–1831, 2013.

[Mart 07] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. "Classification with ant colony optimization". IEEE Transactions on Evolutionary Computation, Vol. 11, No. 5, pp. 651–665, 2007.

[Mend 02] M. F. Mendes, W. Kraus Jr, and E. R. de Pieri. "Variable structure position control of an industrial robotic manipulator". Journal of the Brazilian Society of Mechanical Sciences, Vol. 24, No. 3, 2002.

[Mlad 95] N. Mladenovic. "A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization". In Abstracts of papers presented at Optimization Days, pp. 112, Montréal, Canada, May 1995.

[Mlad 97] N. Mladenovic, and P. Hansen. "Variable neighborhood search". Computers and Operations Research, Vol. 24, pp. 1097–1100, 1997.

[Moh'd 11] O. Moh'd Alia, R. Mandava. "The variants of the harmony search algorithm: an overview ". Artificial Intelligence Review. Vol. 36, pp. 49–68, 2011.

[Molg 05] M. Molga, and C. Smutnicki. ''Test functions for optimization needs''. 2005. http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

[Mühl 97] H. Mühlenbein, and G. Paass. "From recombination of genes to the estimation of distributions i. binary parameters". In Proceedings of the 4th International Conference on Parallel Problem Solving from Nature, pp. 178–187, London, UK, 1996. Springer-Verlag.

[Omra 08] M. G. H. Omran, and M. Mahdavi. "Global-best harmony search". Applied

Mathematics and Computation, Vol. 198 No. 2, pp. 643–656, 2008.

[Osha 15] A.S. Oshaba, E.S. Ali and S.M. Abd Elazim. “ACO based speed control of SRM fed by photovoltaic system“. International Journal of Electrical Power & Energy Systems, Vol. 67, pp. 529–536, May 2015.

[Panc 09] A. Panchal. “Harmony Search in Therapeutic Medical Physics“. Music-Inspired Harmony Search Algorithm, Studies in Computational Intelligence, Vol. 191, pp. 189–203, 2009.

[Papa 82] C. H. Papadimitriou, and K. Steiglitz. “Combinatorial optimization: algorithms and complexity“. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[Park 04a] C. W. Park, Y. W. Cho. ’’T–S model based indirect adaptive fuzzy control using online parameter estimation’’. IEEE Transactions on Systems, Man, and Cybernetics, Part B, Vol. 34 No. 6, pp. 2293–2302, 2004.

[Park 04b] J. W. Park, R.G. Harley, G. K. Venayagamoorthy, ’’Indirect adaptive control for synchronous generator: comparison of MLP/RBF neural networks approach with Lyapunov stability analysis’’. IEEE Transactions on Neural Networks, Vol. 15, No. 2, pp. 460–464, 2004.

[Pass 02] K. M. Passino. “Biomimicry of bacterial foraging for distributed optimization and control“. IEEE Control Systems Magazine, Vol. 22, No. 3, pp. 52–67, 2002.

[Qigu 09] D. Qiguo. “An effective initialization strategy of pheromone for ant colony optimization“. IEEE Fourth International Conference on Bio-Inspired Computing, pp. 1–4, Beijing, 2009.

[Rahi 06] A. Rahideh, and M. H. Shaheed. "Hybrid Fuzzy-PID-based Control of a Twin Rotor MIMO System". In Proceedings of the 32nd Annual IEEE Conference on Industrial Electronics, pp. 48 – 53, Paris 2006.

[Rahi 12] A. Rahideh, M. H. Shaheed. "Constrained output feedback model predictive control for nonlinear systems". Control Engineering Practice, Vol. 20, No. 4, pp. 431–443, 2012.

[Rech 65] I. Rechenberg. “Cybernetic solution path of an experimental problem”. Technical report, Library translation 1122, Ministry of Aviation, Royal Air Force Establishment (UK), 1965.

[Reev 02] C. R. Reeves, and J. E. Rowe. “Genetic algorithms: Principles and Perspectives“. Kluwers Academic Publishers, NY, USA, 2002.

[Reyn 94] R. G. Reynolds. “An introduction to cultural algorithms“. In A. V. Sebald & L. J. Fogel (Eds.), Proceedings of the Third Annual Conference on Evolutionary Programming, Singapore, pp. 131-139. 1994.

[Sast 11] S. Sastry, and M. Bodson, “Adaptive Control: Stability, Convergence, and Robustness“. Prentice-Hall, Englewood Cliffs, NJ, 2011.

[Schi 02] L. Schindler, D. Kerrigan, and J. Kelly. “Understanding the Immune System“. Science Behind the News - National Cancer Institute, 2002.

[Scho 97] M. Schoenauer, and Z. Michalewicz. “Evolutionary Computation. Control and Cybernetics“. Vol. 26, No. 3, pp. 307-338. 1997.

[Shar 10] K. Das-Sharma, A. Chatterjee, and A. Rakshit. “Design of a Hybrid Stable Adaptive Fuzzy Controller Employing Lyapunov Theory and Harmony Search Algorithm“. IEEE Transactions on Control Systems Technology, Vol. 18, No. 6, pp. 1440–1447, 2010.

[Shar 13] K. Das-Sharma. “Stable fuzzy controller design employing group improvisation based harmony search algorithm“. International Journal of Control, Automation

and Systems, Vol. 11, No. 5, pp. 1046–1052, October 2013.

[Shi 92] L. Shi, and S. K. Singh. ''Decentralized adaptive controller design for large-scale systems with higher order interconnections''. IEEE Transactions on Automatic Control, Vol. 37, pp. 1106–1118, 1992.

[Slot 91] J. J. Slotine, and W. Li. "Applied Nonlinear Control". Englewood Cliffs, Prentice-Hall, 1991.

[Soch 08] K. Socha, M. Dorigo. "Ant colony optimization for continuous domains". European Journal of Operational Research, Vol. 185, No. 3, pp. 1155–1173, 2008.

[Song 07] Y. Song, Z. Chen, and Z. Yuan. "New chaotic PSO-based neural network predictive control for nonlinear process", IEEE Transactions on Neural Networks, Vol. 18, No. 2, pp. 595–600, 2007.

[Storn 97] R. M. Storn, and K. V. Price. "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces". Journal of Global Optimization, Vol. 11, No. 4, pp. 341–359, December 1997.

[Stüt 00] T. Stützle and H. Hoos. "MAX-MIN Ant System". Future Generation Computer Systems, Vol. 16, No. 8, pp. 889–914, 2000.

[Stüt 97] T. Stützle and H. Hoos. "Improvements on Ant System : Introducing MAX-MIN Ant System". In : Proceedings of the Third International Conference of Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, Norwich, UK, April 1997.

[Stüt 98] T. Stützle. "Local Search Algorithms for Combinatorial Problems : Analysis, Improvements, and New Applications". Phd thesis, Darmstadt University of Technology, 1998.

[Sun 05] W. Sun, and Y. Wang. "An Adaptive Control for AC Servo System Using Recurrent Fuzzy Neural Network". Advances in Natural Computation, Lecture Notes in Computer Science, Vol. 3611, pp. 190–195, 2005.

[Tail 01] E. D. Taillard, L. M. Gambardella, M. Gendreau, and J. Y. Potvin. "Adaptive Memory Programming : A Unified View of Meta-Heuristics". European Journal of Operational Research, Vol. 135, No. 1, pp. 1–16, 2001.

[Talb 97] E. Talbi. "Metaheuristics : from design to implementation". John Wiley and Sons, NJ, USA, 2009.

[Tao 10] C. W. Tao, J. S. Taur, Y. C. Chen."Design of a parallel distributed fuzzy LQR controller for the twin rotor multi-input multi-output system". Fuzzy Sets and Systems, Vol. 161, pp. 2081–2103, 2010.

[Tian 14] L. Tianjun , S. Thomas , M. O. Marco, and M. Dorigo. "A unified ant colony optimization algorithm for continuous optimization". European Journal of Operational Research, Vol. 234, No. 3, 1, pp. 597–609, May 2014.

[Tsai 04] C. H. Tsai, H. Y. Chung, and F. M. Yu. ''Neuro-sliding Mode Control with Its Applications to Seesaw Systems''. IEEE Transactions on Neural Networks, Vol. 15, No. 1, pp. 124–134, 2004.

[Utki 92] V. Utkin. "Sliding Modes in Control Optimization". Springer, Berlin, 1992.

[Utki 99] V. Utkin, J. Guldner, and S. Jingxin. "Sliding Mode Control in Electromechanical Systems". Systems and Control. Taylor & Francis, London, 1999.

[Wang 04] Y. C. Wang, C. J. Chien, and C. C. Teng. "Direct Adaptive Iterative Learning Control of Nonlinear Systems Using an Output-Recurrent Fuzzy Neural Network". IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics, Vol. 34, No. 3, pp.1348–1359, 2004.

[Wang 13a] H. Wang, X. Yuan, Y. Wang and Y. Yang. "Harmony search algorithm-based fuzzy-

PID controller for electronic throttle valve". Neural Computing and Applications, Vol. 22, No. 2, pp 329–336, 2013.

[Wang 13b] L. Wang, R. Yang, P. M. Pardalos, L. Qian, and M. Fei. "An adaptive fuzzy controller based on harmony search and its application to power plant control". International Journal of Electrical Power & Energy Systems, Vol. 53, pp. 272–278, 2013.

[Wang 94] L. X. Wang, ''A supervisory controller for fuzzy control systems that guarantees stability, IEEE Transactions on Automatic Control''. Vol. 39, No. 9, pp. 1845–1847, 1994.

[Werb 90] P.J. Werbos. ''Backpropagation through time: what it does and how to do it''. In proceedings of IEEE, Vol. 78, No. 10, pp. 1550–1560, 1990.

[Willi 12] C. Williams. "A View of Life in an Ant Colony," Colonial PAST CONTROL INC, 2007.

[Wort 01] N.M. Worthy, and W. M. Spears. "Foundations of Genetic Algorithms". Academic Press, London, UK, 2001.

[Yang 10] J. G. Yang, and Y. B. Zhuang. "An improved ant colony optimization algorithm for solving a complex combinatorial optimization problem". Applied Soft Computing, Vol. 10, pp. 653–660, 2010.

[Yao 99] X. Yao, Liu, Y., & Lin, G. ''Evolutionary programming made faster''. IEEE Transactions on Evolutionary Computation. Vol. 3, No.2, pp. 82–102, 1999.

[Yeh 97] Z. M. Yeh. ''Adaptive multivariable fuzzy logic controller''. Fuzzy Sets and Systems, Vol. 86, No. 1, pp.43–60, 1997.

[Yeh 99] Z. M. Yeh. ''A Systematic Method for Design of Multivariable Fuzzy Logic Control Systems''. IEEE Transactions on Fuzzy systems, Vol. 7, pp. 741–752,1999.

[Youn 99] K.D. Young, V.I. Utkin, and Ö. Özgüner. "A control engineer's guide to sliding mode control". IEEE Transactions on Control Systems Technology, Vol. 7, No.3, pp. 328–342, 1999.

[Yunq 12] Z. Yunqiang, T. Ying, S. Chaoli, and Z. Jianchao. ''A Hybrid Intelligent Algorithm for Mixed-Variable Optimization Problems''. Future Communication, Computing, Control and Management Lecture Notes in Electrical Engineering, Vol. 141, pp 249-256, 2012.

[Zade 65] L. A. Zadeh. "Fuzzy sets". Information and Control, Vol. 8, pp.338-353, 1965.

[Zhan 01] T. P. Zhang. ''Stable Adaptive Fuzzy sliding mode control of interconnected systems''. Fuzzy Sets and Systems, Vol. 122, pp. 5–19, 2001.

[Zhan 09] R. Zhang. "Iterative Multiuser Detection and Channel Decoding for DS-CDMA Using Harmony Search". IEEE Signal Processing Letters, Vol. 16, No. 10, pp. 917–920, 2009.

Résumé

ملخص- في هذه الأطروحة، قمنا باقتراح بديلين بالنسبة لكل من خوارزميات مستعمرات النمل (*Ant colony optimization* ACO) و وخوارزمية البحث عن الانسجام (*Harmony Search* HS). الهدف من هذه المساهمة هو تحسين هذه الخوارزميات بغية تجنب مشكلة التقارب المبكر و سرعة التقارب البطيئة. الطريقة الاولى المقترحة والتي تخص خوارزميات مستعمرات النمل ACO، تعتمد على إدخال بعض التغييرات في الخوارزمية الاصلية C-ACO *Conventional ACO*. الطريقة الثانية المقترحة والتي تسمى GHSACO، نشأت من عملية تهجين بين إصدار من خوارزمية HS تدعى خوارزمية البحث عن الانسجام العام (GHS *Global Best Harmony search*) و خوارزمية C-ACO. بعد ذلك، قمنا باستخدام الخوارزميات المقترحة (ACO المحسن و GHSACO) على التوالي في التطبيقين التاليين، الاولى في عملية تعديل وسائط بنية مراقبة لا مركزية تعتمد على طريقة وضع الانزلاق (modes glissants) والموجهة لمراقبة الأنظمة الغير خطية المترابطة، اما الخوارزمية الثانية فقد استغلت في عملية تعديل غير مباشرة (offline) للإعدادات الشبكات العصبونية الضبابية المتكررة (*Recurrent Fuzzy Neural Networks* RFNN*s*) المندمجة في بنية مراقبة متكيفة والموجهة لمراقبة قسم من الأنظمة الغير خطية. نتائج المحاكاة المتحصل عليها أظهرت فعالية جميع الطرق المقترحة

كلمات مفتاحية : خوارزميات مستعمرات النمل، خوارزمية البحث عن الانسجام، تحكم لا مركزي انزلاقي، تحكم تلاؤمي، الشبكات العصبونية الضبابية المتكررة.

Résumé- Dans cette thèse, deux variantes de *métaheuristiques* sont proposées pour les algorithmes de colonies de fourmis (*Ant Colony Cptimization* ACO) et pour l'algorithme de recherche d'harmonie (*Harmony Search* HS) respectivement. En effet, le but de cette contribution est d'améliorer la vitesse convergence et d'éviter le problème de convergence prématurée afin d'éviter les optimums locaux. La première méthode, relative aux algorithmes ACO, est basée sur l'introduction de quelques modifications dans la structure algorithmique d'algorithme ACO classique (*Conventional ACO* C-ACO). La seconde méthode proposée, dite,GHSACO, est issue d'une hybridation entre une variante d'algorithme HS, appelée algorithme de recherche d'harmonie globale (*Global Best Harmony Search* GHS) et un algorithme de type C-ACO. Les deux algorithmes développés, sont appliqués respectivement au problème de calcul des paramètres d'une commande par mode de glissement décentralisée des systèmes non linéaires interconnectés et au problème d'apprentissage des réseaux de neurones flous récurrents (*Recurrent Fuzzy Neural Networks* RFNN*s*) intégrés dans une structure de commande adaptative d'une classe de systèmes non linéaires. Les résultats de simulations, obtenus montrent l'efficacité de l'ensemble des méthodes proposées.

Mots clés : Algorithmes de colonies de fourmis, Algorithme de recherche d'harmonie, Commande par mode de glissement décentralisée, Commande adaptative, Réseaux de neurones flous récurrents.

Abstract- In this thesis, two variants of *metaheuristics* are proposed for the *Ant Colony Optimization* ACO algorithms and for the *Harmony Search* HS method respectively. Indeed, the aim of this contribution is to improve the speed of convergence and to overcome the premature convergence problem. The first method, relating to ACO algorithms, is based on the introduction of some improvement in the basic algorithmic structure of the *Conventional ACO* C-ACO, the second proposed method, so-called GHSACO, is obtained by merging the Global-Best Harmony Search (GHS) and C-ACO together. The two developed algorithms are used, respectively, to optimize the parameters values of a decentralized sliding mode controller of nonlinear interconnected systems and in the training problem of *Recurrent Fuzzy Neural Networks* RFNN*s* used in adaptive control structure, of a class of nonlinear systems. The obtained simulation results show clearly the effectiveness of all proposed methods.

Key words: Ant colony optimization algorithms, Harmony Search algorithm, Decentralized sliding mode controller, Adaptive control, Recurrent Fuzzy Neural Networks.